Arc-Based Dynamic Discretization Discovery for Continuous-Time Service Network Design

Alexander Helber
Chair of Operations Research, RWTH Aachen University, helber@or.rwth-aachen.de

November 17, 2026

Abstract: In the *continuous-time service network design problem*, a freight carrier decides the path of each shipment in their network and the dispatch times of the vehicles transporting the shipments. State-of-the-art algorithms for this problem are based on the dynamic discretization discovery framework. These algorithms solve a relaxation of the problem using a sparse discretization of time at each network node and iteratively refine the discretization. We introduce a novel relaxation for this framework, with independently chosen time discretizations for each arc, allowing smaller time-expanded networks than previous approaches. We also adapt existing acceleration strategies to the new relaxation and propose a new one that anticipates refinements that may be necessary in future iterations. Our computational experiments demonstrate that our approach matches the performance of existing approaches on standard benchmark sets, and outperforms them on instances with specific topologies. A case study on instances from SBB Cargo, the freight branch of the Swiss Federal Railways, also demonstrates improvements in running times and solution quality on real-life topologies.

Keywords: service network design, dynamic discretization discovery, arc-based discretization

1 Introduction

The (continuous-time) service network design problem is an important optimization problem arising in the context of consolidation-based transportation. Companies in this sector need to cost-efficiently organize the transportation of shipments that are generally smaller than the capacity of the used vehicles (Crainic and Hewitt 2021). The companies operate consolidation terminals at which shipments can be consolidated and dispatched together, improving vehicle utilization and reducing cost. Overall, the problem consists of deciding a path for each shipment through the company's network, and which shipments are consolidated and dispatched together at which time, such that overall cost is minimized.

Commonly, this service network design problem is modeled using time-expanded networks based on a discretization of time. Time-expanded networks contain nodes representing the terminals of the physical network at specific times. These nodes are connected by arcs representing either dispatches between terminals at a specific time, or holding shipments at a terminal. The problem can then be stated as finding paths for each shipment in the time-expanded network, consolidating shipments traversing the same timed dispatch arc. The chosen time discretization has a large impact on the quality of obtained solutions and computational tractability (Boland et al. 2019). This motivated Boland et al. (2017) to develop a dynamic

discretization discovery approach that solves a relaxed problem based on a very small time-expanded network. The network is then iteratively refined until a provably optimal solution in continuous time (i.e., with arbitrary time resolution) is found. Their approach, paired with various later improvements (Hewitt 2019; Marshall et al. 2021; Van Dyk and Koenemann 2024; Shu et al. 2025), is capable of solving instances in seconds or minutes for which just constructing an integer program based on the classical time-expanded networks would already take longer.

In the previously mentioned approaches, for each commodity a time-expanded network is constructed based on a shared discretization of time at each node. This introduces timed arc copies that may be unnecessary for some commodities, increasing the size and possibly solving time of the relaxations. Marshall et al. (2021) and Shu et al. (2025) propose preprocessing rules to remove some of these unnecessary timed arc copies. Van Dyk and Koenemann (2024) instead work towards an arc-based discretization of time, where arcs leaving the same node can receive individual time discretizations in certain cases. These cases arise in some practical applications, e.g., when each commodity can only be dispatched along a single fixed path in the network. But when commodities have overlapping routing options, several outgoing arcs may need to share a time discretization.

We aim to directly construct smaller time-expanded networks for each commodity, to further reduce the relaxation size. This begs the question of which commodities can be consolidated. Previously, commodities traversing the same timed arc copy could be consolidated, but this is not possible if every commodity has an individual time-expansion. We therefore introduce a shared discretization of time on arcs to decide which commodities can consolidate. Based on this idea, we make the following contributions:

- We introduce a new relaxation with individual time-expanded networks for each commodity, based on a shared discretization of time on arcs and individual discretizations of time on nodes. This relaxation generalizes that of Van Dyk and Koenemann (2024), which can only give arcs individual time discretizations in specific cases. We demonstrate a potentially unbounded network-size advantage over existing approaches at the same tight lower bound. For a given arc discretization, we derive a minimum size node discretization that permits exactly the same transfers between timed arc copies as a full node discretization.

- We adapt previously introduced concepts to obtain strong initial relaxations and refine our discretization. We also propose a new algorithmic enhancement that anticipates refinements that may become necessary in future iterations.

- We conduct computational experiments and a case study on instances based on real-world freight networks. Our approach matches the state-of-the-art on commonly used benchmark instances with random network topologies. On wheel networks with a central hub and flexible routing, it solves 16 percentage points more instances than the next-best approach. On the real-world instances, our approach solves smaller instances about three times faster than the state-of-the-art and achieves smaller gaps at the time limit on larger instances.

The remainder of the paper is organized as follows. In Section 2, we review relevant literature. In Section 3, we present a formal description of the problem that we study. In Section 4, we introduce our new arc-based relaxation of the problem, show how to refine it, and how our algorithm works. In Section 5, we describe our computational study and in Section 6, we summarize our findings and discuss potential future work.

2 Related Work

The dynamic discretization discovery framework was first introduced by Boland et al. (2017). Their work also contains an overview of prior work that advanced similar concepts but in the context of different

problems and utilizing different algorithmic approaches. We focus on algorithmic contributions to solving network design problems with dynamic discretization discovery approaches that have been published since their seminal work and relate them to our work.

Boland et al. (2017) introduce the *continuous-time service network design problem* (CTSNDP) and the dynamic discretization discovery framework for solving it. They demonstrated how to construct an integer program on a time-expanded network which is a relaxation of the CTSNDP (i.e., provides a lower bound on the optimal solution value). In this relaxed problem, travel times of commodities are underestimated, allowing consolidations that are not physically possible. They also propose an algorithm that, given a solution to the integer program, iteratively refines the time-expanded network such that the lower bound converges towards the optimal solution value. Together with a primal heuristic that repairs the solutions of the integer program, this forms the basis of an exact algorithm for solving the CTSNDP. Hewitt (2019) show that a stronger initial relaxation can be obtained by adding additional time points obtained by solving the linear programming relaxation of the integer program. They also adapt the concept of valid inequalities to the dynamic discretization discovery framework to improve the lower bounds obtained from the integer program. Marshall et al. (2021) propose a novel relaxation of the CTSNDP utilizing an interval-based interpretation of time discretization. They also introduce a new refinement strategy which not only prevents the same relaxation solution from occurring again but also prevents the same impossible consolidations from occurring again. This can reduce the number of iterations until the algorithm converges. Van Dyk and Koenemann (2024) propose a method by which certain nodes can be split into multiple copies with only a subset of outgoing arcs each. Specifically, if the arcs leaving a node can be partitioned so that each commodity can traverse only arcs in one part of the partition, the node can be split. Then, not all arcs leaving an (original) node need to share the same time discretization. This can lead to smaller relaxed problems and faster solving times. Shu et al. (2025) improve the refinement strategy of Marshall et al. (2021), show how to obtain a much stronger initial relaxation by adding so-called significant time points, and develop a new primal heuristic.

In our work we utilize the same dynamic discretization discovery framework as Boland et al. (2017) but propose a different relaxation than prior works. Our relaxation is based on an arc-based discretization of time which is conceptually inspired by Van Dyk and Koenemann (2024) and an interval-based interpretation of this time discretization which is conceptually inspired by Marshall et al. (2021). The key difference is that arcs can be assigned independent time discretizations even when the commodities' routing options overlap. We also adapt the state-of-the-art initialization and refinement methods of Shu et al. (2025) to our approach.

3 Problem Description

For notational convenience, we refer to some sets of integer numbers as follows: For $n_1, n_2 \in \mathbb{N}$, we say $[n_1] = \{1, \dots, n_1\}$, $[n_1) = [n_1 - 1]$, $[n_1, n_2] = \{n_1, \dots, n_2\}$ if $n_2 \geq n_1$ and $[n_1, n_2] = \emptyset$ otherwise, and $[n_1, n_2) = [n_1, n_2 - 1]$. Recurring notation used throughout this work is introduced in the text and listed in Table 1 for reference.

We study a generalization of the *(continuous-time) service network design problem*, the *(continuous-time) service network design problem with restricted routes* (SND-RR) as introduced by Van Dyk and Koenemann (2024). In an instance of this problem we are given a network $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, also called the *flat-network*, and a set of commodities $\mathcal{K}$. The nodes $\mathcal{V}$ represent terminals and the arcs $\mathcal{A}$ connections between terminals along which shipments can be dispatched. For each arc $a \in \mathcal{A}$ we are given a transit time $\tau_a \in \mathbb{N}$, a capacity $u_a > 0$, and a fixed cost $f_a \geq 0$ for vehicles dispatched on this arc. Each commodity $k \in \mathcal{K}$ has a quantity $q^k > 0$ that needs to be transported along a single path within its (sub)network $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{A}^k)$ (with $\mathcal{V}^k \subseteq \mathcal{V}, \mathcal{A}^k \subseteq \mathcal{A}$) from its source node $o^k \in \mathcal{V}^k$ to its destination node $d^k \in \mathcal{V}^k$. A commodity k becomes available at its source node at its release time $r^k \in \mathbb{N}$ and needs to arrive at its destination node no

Symbol	Description
Instance of SND-RR	
$\mathcal{K}$	Set of commodities
$\mathcal{G} = (\mathcal{V}, \mathcal{A})$	Flat-network with terminals $\mathcal{V}$ and arcs $\mathcal{A}$
$\mathcal{G}^k = (\mathcal{V}^k, \mathcal{A}^k)$	Flat-network of commodity k, a subnetwork of $\mathcal{G}$
τ_a, u_a, f_a	Transit time, vehicle capacity, and fixed cost per vehicle on arc $a \in \mathcal{A}$
c_a^k	Flow cost for transporting commodity k along arc a
q^k	Quantity of commodity k
o^k, d^k	Source and destination node of commodity k
r^k, ℓ^k	Release time and deadline of commodity k
Derived parameters	
τ_{uv}^k	Length (with respect to τ) of a shortest u-v-path in $\mathcal{G}^k$
$\underline{t}_{kv}, \bar{t}_{kv}$	Earliest arrival and latest departure time of commodity k at node $v \in \mathcal{V}^k$
$\underline{t}_a, \bar{t}_a$	Earliest and latest time at which any commodity can be dispatched on arc a
Solutions to SND-RR	
p^k, $P = (p^k)_{k \in \mathcal{K}}$	Flat-path of commodity k and collection of flat-paths
$t^k = (t_a^k)_{a \in p^k}$, T	Dispatch times of commodity k and collection of dispatch times
$S = (P, T)$, $c(S)$	Solution to SND-RR and its cost
Time discretization and relaxed problem	
$\mathcal{T} = ((\mathcal{T}_{kv})_{k \in \mathcal{K}, v \in \mathcal{V}^k}, (\mathcal{T}_a)_{a \in \mathcal{A}})$	Time discretization
$\mathcal{T}_{kv}$, n_{kv}	Partition of $[\underline{t}_{kv}, \bar{t}_{kv}]$ into n_{kv} node intervals
$\mathcal{T}_a$, n_a	Partition of $[\underline{t}_a, \bar{t}_a]$ into n_a arc (dispatch) intervals
$I^k = (I_i^k)_{i \in [n]}$, $\mathcal{I}$	Interval in which commodity k is dispatched on the i-th arc of p^k; collection
$W = (P, \mathcal{I})$, $c(W)$	Solution to R-SND-RR($\mathcal{T}$) and its cost
$I^-(k, a, I), I^+(k, a, I)$	Relaxed dispatch/arrival interval (Definition 4 and 5)
Refinement of the discretization	
(P, C)	Flat-solution given by flat-paths P and consolidations C
$\mathscr{G}(P, C) = (\mathscr{V}, \mathscr{A})$	Dispatch-node graph of the flat-solution (P, C)
ρ_a, $\rho(p)$	Length of an arc $a \in \mathscr{A}$ and of a path p in the dispatch-node graph
$\mathscr{P}$	Set of paths in $\mathscr{G}(P, C)$ starting at some node (k, o^k)
k_m^p, v_m^p, a_m^p	Commodity, node, and flat-arc at the m-th position of a too-long path p
t_m^p	Time point added for the m-th position of a too-long path p (Theorem 3)

Table 1: Reference for recurring notation used throughout this work

later than its deadline $\ell^k \in \mathbb{N}$. Transporting a commodity k along arc $a \in \mathcal{A}^k$ incurs flow costs of $c_a^k \geq 0$.

A solution consists of paths from source node to destination node for each commodity, as well as dispatch times detailing when each commodity starts traversing each arc in its path. The goal is to minimize the sum of flow costs, which depend on the chosen paths, and fixed costs, which depend on the timing of dispatches. At any time that one or multiple commodities are dispatched on an arc a, enough capacity (in integer multiples of u_a) needs to be purchased to transport those commodities. To define a solution more formally, we adapt the notation and terminology introduced by Marshall et al. (2021). For every commodity $k \in \mathcal{K}$, we want to find an o^k-d^k-path p^k in its flat-network $\mathcal{G}^k$, which we call a *flat-path*. For readability, we will give paths as sequences of nodes, but say $a \in p^k$ to refer to an arc traversed by a flat-path. Note that we assume in this work (without loss of optimality) that all flat-paths in an optimal solution are simple (i.e., repeat no vertices). Additionally, for every commodity we seek dispatch times $t^k = (t_a^k)_{a \in p^k}$ that denote the time at which the commodity starts its transport over each arc in its flat-path. A solution $S = (P, T)$ to SND-RR consists of flat-paths $P = (p^k)_{k \in \mathcal{K}}$ and dispatch times $T = (t^k)_{k \in \mathcal{K}}$ for each commodity. We introduce the following concepts to define a feasible solution:

Definition 1 (k-feasible). *A flat-path p^k is k-feasible if the commodity k can traverse it within the time window from its release time to its deadline, i.e., $r^k + \sum_{a \in p^k} \tau_a \leq \ell^k$.*

Definition 2 (Consistent dispatch times). *Let p^k denote a flat-path for commodity k traversing n arcs, and a_i its i-th arc. We say dispatch times t^k for p^k are consistent if $t^k_{a_1} \geq r^k$, $t^k_{a_n} + \tau_{a_n} \leq \ell^k$, and $t^k_{a_i} + \tau_{a_i} \leq t^k_{a_{i+1}}$ for all $i \in [n)$.*

A solution is feasible if, for all $k \in \mathcal{K}$, the dispatch times t^k are consistent for the flat-path p^k. Note that this also implies that p^k is k-feasible. To compute the cost of a solution, we define for each arc $a \in \mathcal{A}$ the set of time points at which commodities are dispatched on that arc as $\Theta(a) = \bigcup_{k \in \mathcal{K}:\ p^k \ni a} \{t^k_a\}$. Then, for each such time point $t \in \Theta(a)$ the set of commodities that will be consolidated on arc a is

$$\kappa(a, t) = \{k \in \mathcal{K} \mid p^k \ni a, t^k_a = t\}.$$

The cost of a solution S is computed as

$$c(S) = \sum_{k \in \mathcal{K}} \sum_{a \in p^k} c^k_a + \sum_{a \in \mathcal{A}} \sum_{t \in \Theta(a)} f_a \left\lceil \frac{\sum_{k \in \kappa(a,t)} q^k}{u_a} \right\rceil$$

where the first term sums the flow costs for each flat-path and the second term the costs for purchasing transport capacity. To solve SND-RR is to find a feasible solution of minimum cost.

3.1 Running Example

We introduce a toy example instance of SND-RR that will be used as a running example. The flat-network with its travel times is depicted in Figure 1. The four commodities' origin and destination nodes are marked by writing their release time and deadline next to the respective node. All commodities $k \in \mathcal{K}$ have a quantity of $q^k = 1$. All arcs $a \in \mathcal{A}$ have a capacity of $u_a = 2$, fixed cost of $f_a = 1$, and flow costs of $c^k_a = 0$. Note that each commodity only has a single path available in the flat-network. Thus, in any feasible solution we have paths $p^{k_1} = (v_1, v_2, v_3), p^{k_2} = (v_1, v_2), p^{k_3} = (v_2, v_3)$, and $p^{k_4} = (v_2, v_4)$.

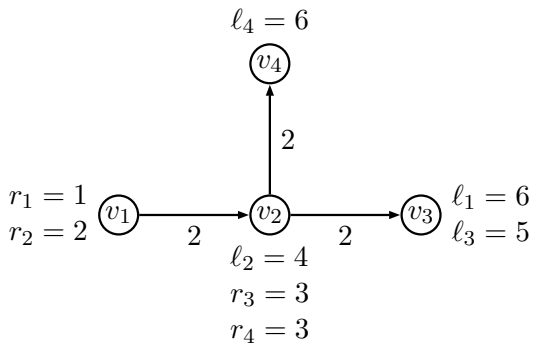


Figure 1: Flat-network $\mathcal{G}$ with transit time τ_a for each arc (example instance)

This instance is constructed in such a way that k_1 can be consolidated with either k_2 or k_3. No other consolidations are possible. For example, consider a solution S where k_1 and k_2 are consolidated on arc (v_1, v_2). To achieve consolidation of k_1 and k_2, we can select the dispatch times $t^{k_1}_{(v_1,v_2)} = t^{k_2}_{(v_1,v_2)} = 2$, $t^{k_3}_{(v_2,v_3)} = t^{k_4}_{(v_2,v_4)} = 3$, and $t^{k_1}_{(v_2,v_3)} = 4$. Then, we achieve costs of $c(S) = 4$ for the four vehicle dispatches: one on arc (v_1, v_2) at time 2, one each on arcs (v_2, v_3) and (v_2, v_4) at time 3, and one on arc (v_2, v_3) at time 4. So we have to dispatch k_1 late if it is to be consolidated with k_2, in which case it arrives too late to be also consolidated with k_3. Still this solution is optimal since fewer than four dispatches are not possible.

3.2 A Mixed-Integer Programming Model

A common way of modeling and solving such problems is to use a time-indexed integer programming formulation over a time-expanded network that explicitly models every possible time at which a commodity could leave or arrive at a node. We provide one such formulation as an alternative problem description, identical to that of Van Dyk and Koenemann (2024) except for notation.

To exclude unnecessary variables, we first determine the first and last time point at which a commodity can be at a node. Denote by $\tau^k_{uv} \in \mathbb{N}_0 \cup \{\infty\}$ the length (with respect to τ_a) of a shortest path between two nodes $u, v \in \mathcal{V}^k$ in the commodity's flat-network $\mathcal{G}^k$ if such a path exists (and $\tau^k_{uv} = \infty$ otherwise). Then we denote with $\underline{t}_{kv} = r^k + \tau^k_{o^k v}$ the earliest time that the commodity k can arrive at node $v \in \mathcal{V}^k$ and with $\bar{t}_{kv} = \ell^k - \tau^k_{vd^k}$ the latest time at which it can depart that node. We assume that $\underline{t}_{kv} \leq \bar{t}_{kv}$. Otherwise, we could delete node v from the commodity's subnetwork without losing feasible solutions.

We now define for each commodity $k \in \mathcal{K}$ its fully time-expanded network $G^k = (V^k, A^k)$. It contains copies of each node in the flat-network for each time point in the time window for that node, i.e., $V^k = \{(v,t) \mid v \in \mathcal{V}^k, t \in [\underline{t}_{kv}, \bar{t}_{kv}]\}$. The set of arcs $A^k = A^k_{\mathrm{t}} \cup A^k_{\mathrm{h}}$ consists of transport arcs A^k_{t} and holding arcs A^k_{h}. The set of transport arcs contains copies of arcs for each allowed time point, i.e., $A^k_{\mathrm{t}} = \{((u,t),(v,t+\tau_{(u,v)})) \mid (u,v) \in \mathcal{A}^k, t \in [\underline{t}_{ku}, \bar{t}_{ku}] \colon t + \tau_{(u,v)} \in [\underline{t}_{kv}, \bar{t}_{kv}]\}$. The set of holding arcs contains arcs representing that a commodity waits at its current location, i.e., $A^k_{\mathrm{h}} = \{((v,t),(v,t+1)) \mid v \in \mathcal{V}^k, t \in [\underline{t}_{kv}, \bar{t}_{kv})\}$. For ease of notation, we also define the complete set of transport arcs as $A_{\mathrm{t}} = \bigcup_{k \in \mathcal{K}} A^k_{\mathrm{t}}$.

To continue with our running example, Figure 2 depicts the time-expanded network $G = (V, A)$ which is obtained as the union of all commodities' time-expanded networks, i.e., $V = \bigcup_{k \in \mathcal{K}} V^k$ and $A = \bigcup_{k \in \mathcal{K}} A^k$. A solution now consists of a path from (o^k, r^k) to (d^k, ℓ^k) in G^k for each commodity k. For example, for commodity k_1 we need to select a path from $(v_1, 1)$ to $(v_3, 6)$.

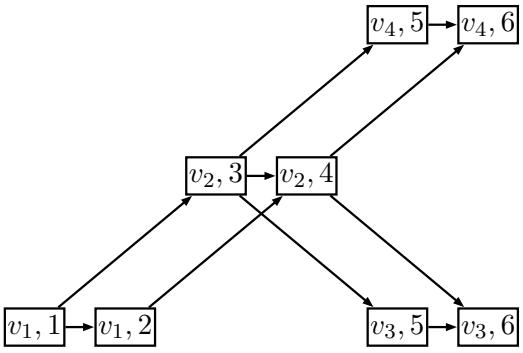


Figure 2: Fully time-expanded network G for the toy instance

We define parameters for arcs in the time-expanded network analogously to the flat-network. For a timed arc $\hat{a} = ((u,t),(v,t')) \in A_{\mathrm{t}}$ we have fixed costs $f_{\hat{a}} = f_{(u,v)}$ and capacities $u_{\hat{a}} = u_{(u,v)}$. Further, for each commodity $k \in \mathcal{K}$ and arc $\hat{a} = ((u,t),(v,t')) \in A^k_{\mathrm{t}}$ we have flow costs $c^k_{\hat{a}} = c^k_{(u,v)}$ for all $k \in \mathcal{K}$. We denote the out- and ingoing arcs of a node $\hat{v} \in V^k$ with $\delta^+_k(\hat{v})$ and $\delta^-_k(\hat{v})$. With this, we state the following

integer program for SND-RR:

$$\min \quad \sum_{k \in \mathcal{K}} \sum_{\hat{a} \in A_{\mathrm{t}}^k} c_{\hat{a}}^k x_{\hat{a}}^k + \sum_{\hat{a} \in A_{\mathrm{t}}} f_{\hat{a}} y_{\hat{a}} \tag{1a}$$

$$\text{s.t.} \sum_{\hat{a} \in \delta_k^+(\hat{v})} x_{\hat{a}}^k - \sum_{\hat{a} \in \delta_k^-(\hat{v})} x_{\hat{a}}^k = \begin{cases} 1 & \text{if } \hat{v} = (o^k, r^k) \\ -1 & \text{if } \hat{v} = (d^k, \ell^k) \\ 0 & \text{else} \end{cases} \quad \forall k \in \mathcal{K}, \forall \hat{v} \in V^k \tag{1b}$$

$$\sum_{k \in \mathcal{K}:\ \hat{a} \in A_{\mathrm{t}}^k} q^k x_{\hat{a}}^k \leq u_{\hat{a}} y_{\hat{a}} \qquad \forall \hat{a} \in A_{\mathrm{t}} \tag{1c}$$

$$x_{\hat{a}}^k \in \{0, 1\} \qquad \forall k \in \mathcal{K}, \forall \hat{a} \in A^k \tag{1d}$$

$$y_{\hat{a}} \in \mathbb{N}_0 \qquad \forall \hat{a} \in A_{\mathrm{t}} \tag{1e}$$

Variable $x_{\hat{a}}^k$ takes value 1 if commodity k takes timed arc $\hat{a} = ((u,t),(v,t'))$. If this is a transport arc, i.e., if $u \neq v$, this also means that arc (u,v) is selected to be part of the commodity's flat-path p^k and the dispatch time of that arc is $t_{(u,v)}^k = t$. Variable $y_{\hat{a}}$ for a timed transport arc $a = ((u,t),(v,t'))$ indicates how many units of capacity need to be purchased on the arc (u,v) to transport the commodities dispatched at time t. Constraint (1b) ensures flow conservation, i.e., that every commodity flows along a path in its time-expanded network. Constraint (1c) ensures that sufficient capacity is purchased to cover all transports that take place. As has been discussed by Boland et al. (2017), directly solving this integer program is often not a promising approach.

4 Dynamic Discretization Discovery

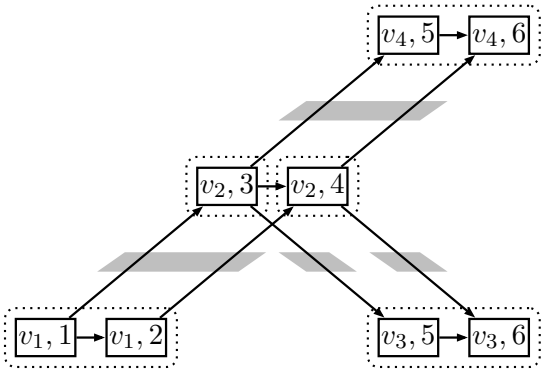
We propose an algorithm to solve SND-RR that fits into the dynamic discretization discovery framework introduced by Boland et al. (2017). As in their framework, we create a relaxation of SND-RR, which can be formulated as an integer program on a time-expanded network. Solving the relaxed problem provides a lower bound on the optimal value of SND-RR. We also use a primal heuristic that tries to obtain a feasible solution to SND-RR with the same value as the solution of the relaxed problem. If this succeeds, we have found an optimal solution. If not, our algorithm determines how to modify the relaxed problem such that the same solution does not appear again. This process is iterated until an optimal solution is found. The main difference in our approach is that we construct the relaxed problem differently from previous approaches and in a way that results in smaller integer programs that can often be solved faster.

We first motivate and formally introduce our relaxed problem. We prove that every solution to SND-RR has a corresponding solution of equivalent or lower cost in the relaxed problem and propose an integer program to solve it. Then, we compare our relaxed problem with those previously proposed in the literature. Finally, we show how to select the commodity-node discretizations, refine the arc discretizations, and combine these components into a dynamic discretization discovery algorithm.

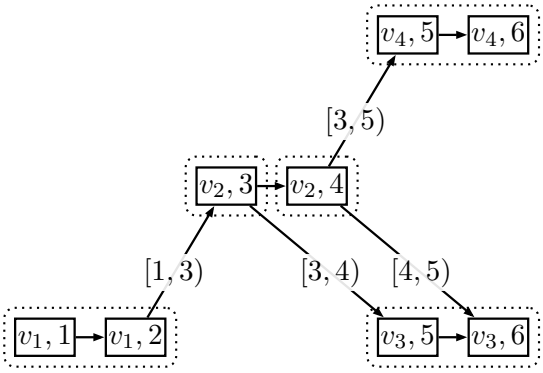
4.1 Relaxed Problem

Conceptually, a relaxed version of the problem can be derived by aggregating timed nodes and arcs in the fully time-expanded network in such a way that two properties hold: first, every path in the fully time-expanded network needs to have a corresponding path in the aggregated network with the same (or lower) flow costs. Second, all commodities that traveled over the same timed arc in the fully time-expanded network also need to travel over the same timed arc in the aggregated network. This ensures that all consolidations that are possible in the actual problem are also possible in the relaxed problem. Then, solving

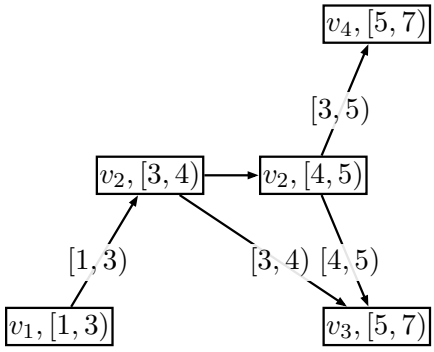
program (1) on such an aggregated network will provide a lower bound on the optimal objective value of SND-RR. With this idea in mind, we give an example for how to appropriately aggregate a network before we formalize the idea.



(a) Fully time-expanded network with partition of time horizon indicated for nodes (dotted lines) and arcs (shaded areas)



(b) Timed arcs aggregated into interval-arcs



(c) Timed arcs and nodes aggregated

Figure 3: A possible aggregation of the time-expanded network for the running example

4.1.1 Idea: Aggregating Time-Expanded Networks

We partition the time horizon into intervals, separately for each node and each arc. Then, all timed copies of nodes (or arcs) inside the same interval are replaced with a single copy. We demonstrate the concept on our example instance, for which Figure 3a displays a possible partition of the time horizon for nodes and arcs.

First, we explain how to aggregate timed arcs, which results in the network depicted in Figure 3b. The arc (v_2, v_4) has a single interval $[3, 5)$ in its partition. So the two timed arcs $((v_2, 3), (v_4, 5))$ and $((v_2, 4), (v_4, 6))$ are replaced with a single interval-arc in the aggregated network. This interval-arc will represent dispatching on *either* of these two timed arcs. Recall that we want to represent each path from the fully time-expanded network in the aggregated network. So we still need to be able to get from $(v_2, 3)$ to $(v_4, 5)$ and from $(v_2, 4)$ to $(v_4, 6)$. Therefore, the interval-arc leaves from the latest copy of the tail node and arrives at the earliest copy of the head node. Similarly, the two timed copies of arc (v_1, v_2) are replaced with a single arc from $(v_1, 2)$ to $(v_2, 3)$. The two timed copies of arc (v_2, v_3) each receive a corresponding interval-arc, since this arc has a finer partition into two intervals.

Similarly, we can also aggregate timed node copies into interval-nodes. A commodity traversing an interval-node represents that it is at this node at some point in the interval. For our example, we obtain the

network depicted in Figure 3c. Note that this merging of timed nodes may create parallel arcs.

We now assume that commodities dispatched on an arc in the same interval can be consolidated. Thus, each solution of SND-RR also has a representation in this aggregated network, but potentially at a lower cost. For example, we could send commodity k_1 along the path from $(v_1, [1, 3))$ via $(v_2, [3, 4))$ to $(v_3, [5, 7))$. Then, we could consolidate it with commodity k_2 on the first arc and with commodity k_3 on the second arc. Together with the dispatch for commodity k_4, this solution would only cost 3, less than the optimal solution for SND-RR.

So the aggregated networks lead to a lower bound on the real optimal value. Specifically, they are overly optimistic in two regards: First, they allow consolidating commodities that cannot be consolidated in the original problem. For example, two commodities dispatched on arc (v_1, v_2) at time 1 and 2 cannot actually be consolidated, but in the aggregated network they can, leading potentially to lower costs. Second, travel times are underestimated, allowing commodities to be dispatched earlier than physically feasible. For example, a commodity dispatched at node v_1 at time 2 could still arrive at node v_2 in the interval $[3, 4)$ in the aggregated network. So it could also be dispatched at node v_2 at time 3, even though this is not feasible in the original problem. These observations explain why solving the network design problem on such an aggregated network is a relaxation of the original problem.

4.1.2 Formal Description of the Relaxed Problem

We now introduce our time discretization and then use it to define our relaxed problem. Formally, a time discretization $\mathcal{T} = ((\mathcal{T}_{kv})_{k\in\mathcal{K}, v\in\mathcal{V}^k}, (\mathcal{T}_a)_{a\in\mathcal{A}})$ is a tuple of time intervals for each node in each commodity's network as well as time intervals for each arc in the flat-network.

For each commodity $k \in \mathcal{K}$ and node $v \in \mathcal{V}^k$ the discretization contains a set of $n_{kv} \in \mathbb{N}$ time intervals $\mathcal{T}_{kv} = \{[t_1^{kv}, t_2^{kv}), \ldots, [t_{n_{kv}}^{kv}, t_{n_{kv}+1}^{kv})\}$. These intervals partition $[\underline{t}_{kv}, \bar{t}_{kv}]$, such that $t_1^{kv} = \underline{t}_{kv}$ and $t_{n_{kv}+1}^{kv} = \bar{t}_{kv} + 1$. So they collectively represent possible times at which commodity k could be dispatched at this node. As an initial time discretization we can simply use the full time horizon, i.e., $\mathcal{T}_{kv} = \{[\underline{t}_{kv}, \bar{t}_{kv}]\}$ for each commodity and node. Note that this definition is slightly more general than what we demonstrated in the previous example (Figure 3). There, for ease of exposition, we used the same node discretization for each commodity. For example, at node v_2 we used the discretization $\mathcal{T}_{kv_2} = \{[3, 4), [4, 5)\}$ for $k \in \{k_1, k_4\}$.

For the arcs, we partition the time horizon into intervals in which commodities may be dispatched. For each arc $a = (u, v) \in \mathcal{A}$, denote by $\underline{t}_a = \min_{k\in\mathcal{K}:\, a\in\mathcal{A}^k} \underline{t}_{ku}$ the earliest time at which a commodity could be dispatched on it. Similarly, denote by $\bar{t}_a = \max_{k\in\mathcal{K}:\, a\in\mathcal{A}^k} \bar{t}_{kv} - \tau_a$ the latest time at which a commodity could be dispatched on it. Then, for each arc $a \in \mathcal{A}$ we denote by $\mathcal{T}_a = \{[h_1^a, h_2^a), \ldots, [h_{n_a}^a, h_{n_a+1}^a)\}$ a partition of $[\underline{t}_a, \bar{t}_a]$ into n_a intervals, such that $h_1^a = \underline{t}_a$ and $h_{n_a+1}^a = \bar{t}_a + 1$. As an initial partition, we can again use the full time horizon, i.e., $\mathcal{T}_a = \{[\underline{t}_a, \bar{t}_a]\}$ for all $a \in \mathcal{A}$. In the example in Figure 3, we set the arc discretizations as $\mathcal{T}_{(v_1,v_2)} = \{[1, 3)\}, \mathcal{T}_{(v_2,v_3)} = \{[3, 4), [4, 5)\}$, and $\mathcal{T}_{(v_2,v_4)} = \{[3, 5)\}$.

The relaxed problem R-SND-RR($\mathcal{T}$) receives as input an instance of SND-RR as well as a time discretization $\mathcal{T}$. Just as in SND-RR, we want to find an o^k-d^k-path p^k in $\mathcal{G}^k$ for every commodity $k \in \mathcal{K}$. Instead of determining for each arc in a flat-path the exact dispatch time, we determine the time interval in which the commodity is dispatched on that arc. Specifically, we determine the arc intervals $I^k = (I_i^k)_{i\in[n]}$ where $I_i^k \in \mathcal{T}_{a_i}$ is the interval in which commodity k is dispatched on its i-th arc a_i and n is the number of arcs traversed by p^k. We assume that commodities dispatched on the same arc in the same interval can be consolidated. A solution to R-SND-RR($\mathcal{T}$) is now given by a tuple $W = (P, \mathcal{I})$ of flat-paths $P = (p^k)_{k\in\mathcal{K}}$ and arc intervals $\mathcal{I} = (I^k)_{k\in\mathcal{K}}$. To describe our notion of a feasible solution to R-SND-RR($\mathcal{T}$), we introduce some additional concepts.

Definition 3 (Feasible arc interval)**.** *For commodity $k \in \mathcal{K}$ and arc $a = (u, v) \in \mathcal{A}^k$, we call an arc interval $I \in \mathcal{T}_a$ feasible if there exists a time point $t \in I$ such that both $t \in [\underline{t}_{ku}, \bar{t}_{ku}]$ and $t + \tau_a \in [\underline{t}_{kv}, \bar{t}_{kv}]$.*

Said another way, an arc interval is feasible for a commodity if we can dispatch it at some point inside this interval and still arrive on time. In our example in Figure 3, the interval $[1, 3)$ on arc (v_1, v_2) is feasible for k_2, since it could be dispatched at time 2 and still arrive on time. In contrast, the interval $[4, 5)$ on arc (v_2, v_3) is not feasible for k_3, since it would arrive too late if dispatched at time 4.

Note that the feasibility of an arc interval can be checked in constant time once $[\underline{t}_{kv}, \bar{t}_{kv}]$ has been precomputed for all nodes in each commodity's subnetwork. Definition 3 also ensures that a commodity cannot be dispatched from its source node in an interval that lies before its release time or be dispatched on an arc to its destination node in an interval where the earliest arrival time is after its deadline. We require that for each arc $a_i \in p^k$, the interval I_i^k is feasible.

Next, we describe our relaxation of the concept of consistent dispatch times (Definition 2) which required that a commodity can only be dispatched from a node at or after the time at which it arrives at that node. Recall the example of an aggregated network we constructed in Figure 3c. There, we saw that a commodity can be dispatched on an arc in an interval $[h, h')$ from the last node whose interval still overlaps the arc interval. For example, a commodity can still be dispatched on arc (v_2, v_4) in interval $[3, 5)$ from the interval-node $(v_2, [4, 5))$. However, dispatching from a node interval $[t, t')$ on a given arc (u, v) is not allowed if $t + \tau_{(u,v)} > \bar{t}_{kv}$, i.e., dispatching in this interval would lead to a deadline violation. For the sake of notation, we define the $\max$ (and $\min$) operator over a set of intervals to select the interval with the largest (smallest) endpoint. Then, we can define the last node interval from which a commodity can dispatch in a given arc interval as follows:

Definition 4 (Relaxed dispatch interval)**.** *For a commodity $k \in \mathcal{K}$, arc $a = (u, v) \in \mathcal{A}^k$, and feasible arc interval $I = [h, h') \in \mathcal{T}_a$, we call the node interval $I^-(k, a, I) = \max\{[t, t') \in \mathcal{T}_{ku} \mid t < \min\{h', \bar{t}_{kv} - \tau_a + 1\}\}$ the relaxed dispatch interval.*

Conversely, we saw that a commodity dispatched on an arc a in an interval $[h, h')$ arrives at the node whose interval contains the earliest time point $h + \tau_a$ at which the commodity could arrive. In the example, a commodity dispatched on arc (v_1, v_2) in interval $[1, 3)$ arrives at node v_2 in the node interval $[3, 4)$ since $1 + 2 = 3 \in [3, 4)$. Formally, we specify the earliest node interval in which a commodity can arrive as follows:

Definition 5 (Relaxed arrival interval)**.** *For a commodity $k \in \mathcal{K}$, arc $a = (u, v) \in \mathcal{A}^k$, and feasible arc interval $I = [h, h') \in \mathcal{T}_a$, we call the node interval $I^+(k, a, I) = \min\{[t, t') \in \mathcal{T}_{kv} \mid t' > \max\{h, \underline{t}_{ku}\} + \tau_a\}$ the relaxed arrival interval.*

Note that if an arc interval is feasible, the relaxed dispatch and arrival intervals always exist. Based on these relaxed dispatch and arrival intervals we introduce our relaxed notion of consistent dispatch times. Essentially, we require that a commodity is not dispatched from a node interval that is earlier than the interval in which it arrived at the node. To simplify notation, we say $I < I'$ for two intervals I, I' if the endpoint of I is lower than that of I'. Correspondingly, we say $I \leq I'$ exactly if $I < I'$ or $I = I'$.

Definition 6 (Relaxed time consistent arc intervals)**.** *We call arc intervals I^k for a flat-path p^k traversing n arcs relaxed time consistent if the interval for each arc is feasible and for each $i \in [n)$ the relaxed arrival interval of arc a_i is no later than the relaxed dispatch interval of arc a_{i+1}, i.e., $I^+(k, a_i, I_i^k) \leq I^-(k, a_{i+1}, I_{i+1}^k)$.*

We say a solution $W = (P, \mathcal{I})$ to R-SND-RR($\mathcal{T}$) is feasible if for all $k \in \mathcal{K}$ the dispatch time intervals I^k are relaxed time consistent for p^k. The cost of a feasible solution W is the sum of the flow costs for each commodity as well as the fixed costs for dispatched vehicles. For each arc $a \in \mathcal{A}$ and interval $I \in \mathcal{T}_a$ we collect the set of consolidated commodities $\bar{\kappa}(a, I) = \{k \in \mathcal{K} \mid \exists i \colon a \text{ is the } i\text{-th arc in } p^k, I_i^k = I\}$. With this, we can state the cost formally as

$$c(W) = \sum_{k \in \mathcal{K}} \sum_{a \in p^k} c_a^k + \sum_{a \in \mathcal{A}} \sum_{I \in \mathcal{T}_a} f_a \left\lceil \frac{\sum_{k \in \bar{\kappa}(a,I)} q^k}{u_a} \right\rceil .$$

Note that we can get non-simple flat-paths (i.e., that repeat vertices and even arcs) as part of our relaxation solutions. Such paths can without loss of optimality be assumed to not exist in optimal solutions to SND-RR. But in optimal solutions to R-SND-RR($\mathcal{T}$) they can occur since they allow traveling back in time. We could require only simple paths in feasible solutions to R-SND-RR($\mathcal{T}$), which could give stronger bounds, but preliminary experiments indicated that this made the relaxed problem computationally harder to solve.

We now show that R-SND-RR($\mathcal{T}$) is indeed a relaxation of SND-RR:

Proposition 1. *Given an instance of SND-RR and a time discretization $\mathcal{T}$, for every feasible solution $S = (P, T)$ to SND-RR there is a corresponding feasible solution $W(S) = (P(S), \mathcal{I}(S))$ to R-SND-RR($\mathcal{T}$) of equal or lower cost.*

Proof. Given a feasible solution S to SND-RR, we first show how to construct a feasible solution $W(S) = (P(S), \mathcal{I}(S))$ and then show that $c(W(S)) \leq c(S)$. The relaxed solution will utilize the same flat-paths for each commodity, i.e., $P(S) = P$. To determine the arc intervals, we select $I_i^k \ni t_{a_i}^k$, i.e., the interval containing the time point at which a commodity is dispatched in the solution S. These arc intervals are feasible because S is feasible and therefore $t_{a_i}^k \in [\underline{t}_{ku}, \bar{t}_{ku}]$ and $t_{a_i}^k + \tau_{a_i} \in [\underline{t}_{kv}, \bar{t}_{kv}]$. We still need to show that they are also relaxed time consistent. Recall that they are relaxed time consistent if for any two consecutive arcs $a_i = (u, v), a_{i+1} = (v, w)$ in p^k it holds that $I^+(k, a_i, I_i^k) \leq I^-(k, a_{i+1}, I_{i+1}^k)$. For brevity, we denote the corresponding node intervals I^+ and I^-. First, consider the relaxed dispatch interval I^- for arc a_{i+1}, which is the latest node interval in $\mathcal{T}_{kv}$ that overlaps with the arc interval I_{i+1}^k chosen for a_{i+1}. Since I_{i+1}^k contains $t_{a_{i+1}}^k$, the node interval I^- either also contains $t_{a_{i+1}}^k$ or is later than this time point. Now consider the relaxed arrival interval I^+ for arc a_i. Since the arc interval contains $t_{a_i}^k$, this node interval either contains $t_{a_i}^k + \tau_{a_i}$ or is earlier than this time point. Finally, since S is a feasible solution we know that $t_{a_i}^k + \tau_{a_i} \leq t_{a_{i+1}}^k$ and therefore $I^+ \leq I^-$. Thus, $W(S)$ is feasible for R-SND-RR($\mathcal{T}$).

We still need to show that this solution does indeed give a lower bound on the cost of S. With regard to the flow costs, both solutions are identical since the flow costs are computed the same for both problems and $P = P(S)$. With regard to the fixed costs, we note that commodities that could previously be consolidated (since they were dispatched at the same time) can still be consolidated in the relaxed problem (since they are now dispatched in the same interval). Additionally, some commodities that cannot be consolidated in SND-RR may be consolidated if their dispatch times lie in the same interval, potentially leading to lower costs. More formally, let us consider the set of commodities $\bar{\kappa}(a, I)$ dispatched on arc a in interval $I = [t', t'')$ in solution $W(S)$. The cost contribution of this set in the relaxed problem is $f_a \left\lceil \frac{\sum_{k \in \bar{\kappa}(a,I)} q^k}{u_a} \right\rceil$ while in the original problem it would be $\sum_{t=t'}^{t''-1} f_a \left\lceil \frac{\sum_{k \in \kappa(a,t)} q^k}{u_a} \right\rceil$. Since $\bar{\kappa}(a, I) = \bigcup_{t=t'}^{t''-1} \kappa(a, t)$, the first term is evidently no larger than the second. Therefore, the fixed costs for the solution $W(S)$ are either identical to or an underestimation of the fixed costs for a solution S. □

If we add sufficiently many time points to the discretization the lower bound eventually becomes tight, i.e., an optimal solution to SND-RR and R-SND-RR($\mathcal{T}$) have the same value. This can be seen if we construct a full time-discretization, i.e., using intervals of length 1 only.

Proposition 2. *Given an instance for SND-RR, there exists a time discretization $\mathcal{T}$ such that the value $c(S)$ of an optimal solution S to SND-RR is equal to the value $c(W)$ of an optimal solution W to R-SND-RR($\mathcal{T}$).*

Proof. We can select the full time discretization $\mathcal{T}$ with intervals containing single integer time points, i.e., $\mathcal{T}_{kv} = \{\{\underline{t}_{kv}\}, \{\underline{t}_{kv} + 1\}, \ldots, \{\bar{t}_{kv}\}\}$ for all $k \in \mathcal{K}, v \in \mathcal{V}^k$ and $\mathcal{T}_a = \{\{\underline{t}_a\}, \{\underline{t}_a + 1\}, \ldots, \{\bar{t}_a\}\}$. Since for each time point a unique interval exists on each arc and at each node, the relaxed problem R-SND-RR($\mathcal{T}$) does not underestimate any travel times. Specifically, if a commodity is dispatched on an arc a in an interval $[t, t + 1)$, the relaxed dispatch interval is $[t, t + 1)$ and the relaxed arrival interval is $[t + \tau_a, t + \tau_a + 1)$, i.e.,

they are identical to the actual times. So relaxed time consistent arc intervals in the solution to R-SND-RR($\mathcal{T}$) can be mapped directly to consistent dispatch times, and we can derive a feasible solution S to SND-RR from a feasible solution W to R-SND-RR($\mathcal{T}$). These solutions also have identical costs, since only commodities dispatched at exactly the same time on an arc can be consolidated in both cases. Therefore, each solution to R-SND-RR($\mathcal{T}$) corresponds directly to a solution of SND-RR with identical cost. □

4.1.3 Solving the Relaxed Problem

To actually solve R-SND-RR($\mathcal{T}$), we utilize an integer program formulated over a time-expanded network. We will first describe how to construct this time-expanded network and then introduce the integer program. Given a time discretization $\mathcal{T} = ((\mathcal{T}_{kv})_{k\in\mathcal{K},v\in\mathcal{V}^k}, (\mathcal{T}_a)_{a\in\mathcal{A}})$, we can construct what we call a *relaxed time-expanded network* $G^k_{\mathcal{T}} = (V^k_{\mathcal{T}}, A^k_{\mathcal{T}})$ for each commodity k. The set of nodes contains one copy of each node in the commodity's flat-network for each time interval in the discretization, i.e., $V^k_{\mathcal{T}} = \{(v, I) \mid v \in \mathcal{V}^k, I \in \mathcal{T}_{kv}\}$. The set of arcs $A^k_{\mathcal{T}} = A^k_{\mathcal{T}\mathrm{t}} \cup A^k_{\mathcal{T}\mathrm{h}}$ contains transport and holding arcs. The transport arcs correspond to copies of arcs in the flat-network for each feasible time interval, $A^k_{\mathcal{T}\mathrm{t}} = \{(a, I) \mid a \in \mathcal{A}^k, I \in \mathcal{T}_a \colon I \text{ is feasible for } k\}$. Note that the arcs are given as tuples of flat-arc and time interval. This is because the time-expanded networks may contain parallel arcs, which are not uniquely identified by their tail and head nodes. We instead explicitly specify the head and tail nodes of each arc. Consistent with our definition of the relaxed arrival and dispatch intervals, we say for a timed arc $\hat{a} = (a, I) \in A^k_{\mathcal{T}\mathrm{t}}$ that $\mathrm{head}(\hat{a}) = (v, I^+(k, a, I))$ and $\mathrm{tail}(\hat{a}) = (u, I^-(k, a, I))$. The holding arcs are constructed as usual, linking timed copies of a node. Let I^{kv}_q denote the q-th interval in $\mathcal{T}_{kv}$. We add one holding arc for each timed copy except the last: $A^k_{\mathcal{T}\mathrm{h}} = \{(v, I^{kv}_q) \mid \forall v \in \mathcal{V}^k, \forall q \in [n_{kv})\}$. As before, we define for $\hat{a} = (v, I^{kv}_q) \in A^k_{\mathcal{T}\mathrm{h}}$ that $\mathrm{head}(\hat{a}) = (v, I^{kv}_{q+1})$ and $\mathrm{tail}(\hat{a}) = (v, I^{kv}_q)$. Lastly, we refer to the in- and outgoing arcs of a timed node $\hat{v} \in V^k_{\mathcal{T}}$ as $\delta^-_k(\hat{v}) = \{\hat{a} \in A^k_{\mathcal{T}} \mid \hat{v} = \mathrm{head}(\hat{a})\}$ and $\delta^+_k(\hat{v}) = \{\hat{a} \in A^k_{\mathcal{T}} \mid \hat{v} = \mathrm{tail}(\hat{a})\}$.

We define parameters for interval arcs in the relaxed time-expanded network analogously to the flat-network. So for timed transport arcs $\hat{a} = (a, I) \in A^k_{\mathcal{T}\mathrm{t}}$ we have fixed cost $f_{\hat{a}} = f_a$, capacity $u_{\hat{a}} = u_a$, travel time $\tau_{\hat{a}} = \tau_a$, and flow cost $c^k_{\hat{a}} = c^k_a$. For each commodity k we refer to the timed node copies of its origin and destination as $\hat{o}^k = (o^k, \min \mathcal{T}_{ko^k})$ and $\hat{d}^k = (d^k, \max \mathcal{T}_{kd^k})$. We also denote the full set of interval-arcs by $A_{\mathcal{T}\mathrm{t}} = \{(a, I) \mid a \in \mathcal{A}, I \in \mathcal{T}_a\}$. An integer program to solve R-SND-RR($\mathcal{T}$) can then be stated as follows.

$$\min \quad \sum_{k\in\mathcal{K}} \sum_{\hat{a}\in A^k_{\mathcal{T}\mathrm{t}}} c^k_{\hat{a}} x^k_{\hat{a}} + \sum_{\hat{a}\in A_{\mathcal{T}\mathrm{t}}} f_{\hat{a}} y_{\hat{a}} \tag{2a}$$

$$\text{s.t.} \sum_{\hat{a}\in\delta^+_k(\hat{v})} x^k_{\hat{a}} - \sum_{\hat{a}\in\delta^-_k(\hat{v})} x^k_{\hat{a}} = \begin{cases} 1 & \text{if } \hat{v} = \hat{o}^k \\ -1 & \text{if } \hat{v} = \hat{d}^k \\ 0 & \text{else} \end{cases} \quad \forall k \in \mathcal{K}, \forall \hat{v} \in V^k_{\mathcal{T}} \tag{2b}$$

$$\sum_{k\in\mathcal{K}\colon \hat{a}\in A^k_{\mathcal{T}}} q^k x^k_{\hat{a}} \le u_{\hat{a}} y_{\hat{a}} \qquad \forall \hat{a} \in A_{\mathcal{T}\mathrm{t}} \tag{2c}$$

$$x^k_{\hat{a}} \in \{0, 1\} \qquad \forall k \in \mathcal{K}, \forall \hat{a} \in A^k_{\mathcal{T}} \tag{2d}$$

$$y_{\hat{a}} \in \mathbb{N}_0 \qquad \forall \hat{a} \in A_{\mathcal{T}\mathrm{t}} \tag{2e}$$

A variable $x^k_{\hat{a}}$ takes value 1 if commodity k is transported along timed arc $\hat{a}$. If that arc is a transport arc, i.e., $\hat{a} = (a, I) \in A^k_{\mathcal{T}\mathrm{t}}$, it also means that the flat-arc a should be in the flat-path p^k of the commodity and the corresponding arc interval is I. Variable $y_{\hat{a}}$ for $\hat{a} = (a, I)$ indicates how many vehicles are dispatched on arc a in the time interval I. Constraint (2b) ensures that the selected arcs for each commodity form a path in the time-expanded network. Constraint (2c) ensures that enough vehicles are dispatched on each arc in each time interval to transport the commodities traveling over that arc in that time interval.

4.1.4 Comparison with Other Relaxations

We compare our arc-based relaxation with the original node-based one proposed by Boland et al. (2017), the interval-based one proposed by Marshall et al. (2021), and the arc-dependent one proposed by Van Dyk and Koenemann (2024). In essence, we claim that our relaxation is a generalization, of which the other three relaxations are special cases. To substantiate this claim, we show that we can always construct a relaxed problem that leads to a MIP of the same or smaller size than the other relaxations while preserving the same lower bound. In the following, we focus on a conceptual discussion of the differences between the relaxations.

In Appendix B we formally show how to reproduce the other relaxations with ours. Here, we give a concrete example where our relaxation leads to a smaller model with the same lower bound. For the sake of presentation, we use a small example with small size differences and disregard variable-reduction rules proposed in the literature. In Appendix F we show that the difference in network size can grow arbitrarily large, even considering these variable-reduction rules. Specifically, we construct a family of instances with $\mathcal{O}(n)$ commodities for which an arc-based relaxation with $\mathcal{O}(n)$ timed arc copies yields a tight lower bound, while the other approaches require $\Omega(n^2)$ copies. We also investigate the size benefits empirically in our computational study.

Comparison with the Node-Time-Based Relaxation Interpreted through the lens of our relaxation, the original node-based relaxation proposed by Boland et al. (2017) can be reproduced as follows.

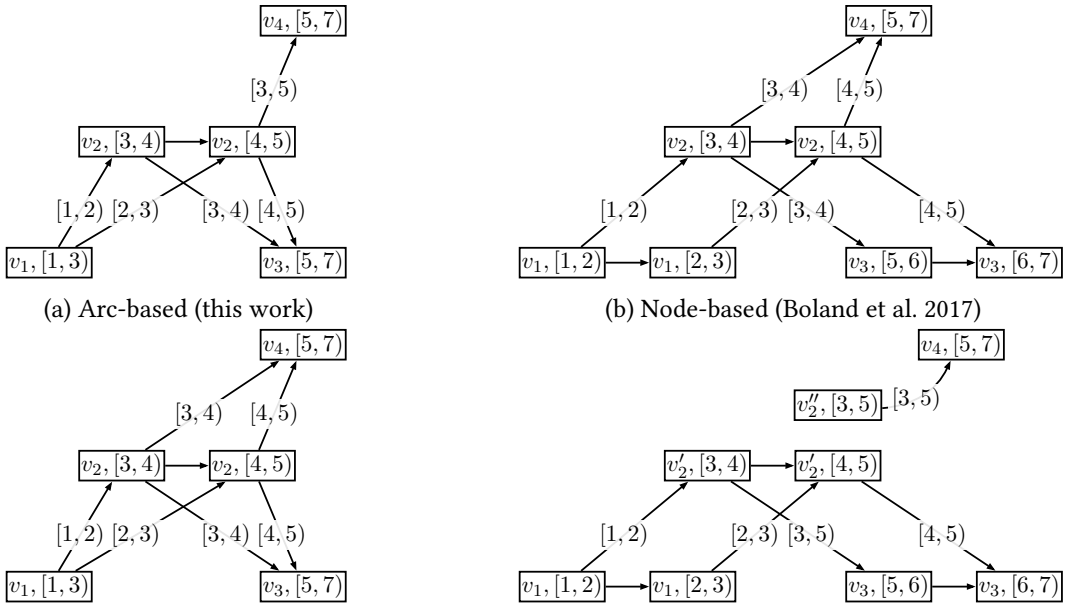
- They specify a single discretization $\mathcal{T}'_v$ for each node that is shared by each commodity (i.e., $\mathcal{T}_{kv} = \mathcal{T}'_v$ for all k, up to clamping to the time window $[\underline{t}_{kv}, \bar{t}_{kv}]$).
- The discretization of each arc is equivalent to that of its tail node (i.e., $\mathcal{T}_{(u,v)} = \mathcal{T}'_u$ for all $(u, v) \in \mathcal{A}$).
- They require that at each commodity's source node o^k (sink node d^k) an interval has to start at the release time r^k (deadline ℓ^k). (This increases the size of the aggregated network substantially.)

Evidently, our approach offers more flexibility, for two reasons: first, because we can discretize time on each arc and at each node individually. Second, because the interval-based interpretation naturally does not require discretizing time at commodities' release times and deadlines. Note that the second advantage is already present in the interval-based relaxation of Marshall et al. (2021).

As an example that our relaxation can lead to smaller MIPs while maintaining the bound quality, consider again our running example. If we want to recover a tight lower bound (i.e., the optimal solution value of SND-RR), we need to prevent commodity k_1 from consolidating first with k_2 and then with k_3, as is possible in the relaxed network in Figure 3c. So dispatching k_1 together with k_2 needs to lead to arrival of k_1 at v_2 at time 4, at which time it cannot leave on the same timed arc copy as k_3 anymore. To represent this, we change the arc-based discretization to $\mathcal{T}_{(v_1,v_2)} = \{[1, 2), [2, 3)\}$, leading to the relaxed network in Figure 4a. Compare this to the relaxed network depicted in Figure 4b, which is the smallest possible using the node-based relaxation. Evidently, the need to differentiate between time 3 and 4 at node v_2 leads to an additional and unnecessary timed copy of arc (v_2, v_4).

Comparison with the Node-Interval-Based Relaxation Marshall et al. (2021) build upon the node-based relaxation but interpret timed nodes to represent intervals of time instead of distinct time points. That allows them to discard the timed copies of nodes for each commodity's origin and destination.

If we want to obtain a tight lower bound with their approach for our example, the network in Figure 4c results. Note that while they can merge the timed copies of nodes v_1 and v_3, saving two holding arcs, they still create two timed copies of arc (v_2, v_4). They propose an additional preprocessing rule that would be



(a) Arc-based (this work)

(b) Node-based (Boland et al. 2017)

(c) Interval-based (Marshall et al. 2021)

(d) Arc-dependent (Van Dyk and Koenemann 2024)

Figure 4: Relaxed networks leading to a tight lower bound for the example instance

able to merge these two copies, but the more complex example in Appendix F demonstrates a size advantage even when applying this rule.

Comparison with the Arc-Dependent Relaxation Van Dyk and Koenemann (2024) also build upon the node-based relaxation, but they make use of the observation that time points added to prevent illegal consolidations on one arc are not necessarily useful for other arcs. Specifically, they observe that in some instances the arcs leaving a node can be partitioned in such a way that each commodity can only traverse arcs in one part of the partition. As an example, consider our toy instance: We can partition $\delta^+(v_2) = \{(v_2, v_3), (v_2, v_4)\}$ into the two sets $\{(v_2, v_3)\}$ and $\{(v_2, v_4)\}$ since commodities k_1 and k_3 can only traverse the first arc and commodity k_4 can only traverse the second arc. In such cases, they propose to split the node into multiple copies for each part of the partition. This has the benefit that each copy can receive its own discretization of time. In our example, node v_2 would be split into the two nodes v_2' (for the arc onwards to v_3) and v_2'' (for the arc onwards to v_4). (Note that technically they also add another copy for commodities ending at that node, but for the sake of presentation we suppress this node here.) For our example, this results in a network as shown in Figure 4d. So up to holding arcs (which could be removed by combining this discretization with the interval-based one), the network is equivalent to ours (Figure 4a).

But, crucially, the two approaches are *only* equivalent if a partition of the arcs as described is possible. If no such non-trivial partition exists, our approach is *strictly* more flexible and allows us to construct smaller networks yielding the same bound. To demonstrate this, consider the following modification to our toy instance: we add a node v_5 reachable from both v_3 and v_4, resulting in the network $\mathcal{G}'$ as depicted in Figure 5. We also move the destination of k_1 to this new node and extend its deadline to 7. In this case, all arcs can be traversed by k_1 and thus a partition of the arcs in $\delta^+(v_2)$ is no longer permissible. Thus, on this modified instance, the approach by Van Dyk and Koenemann (2024) cannot split node v_2 into copies with distinct outgoing arcs. As a consequence, their network would contain two timed copies of arc (v_2, v_4).

Van Dyk and Koenemann (2024) demonstrate that there are important practical instance classes of CTSNDP in which such partitions are possible. But, the more flexibly commodities can be routed through a

network, the less fine-grained the partitions one can find. In general, for any instance where their approach has a benefit compared to the node-based approaches, introducing a single commodity that can traverse all arcs in the network prevents them from finding such partitions. In contrast, our proposed arc-based approach can always discretize time at each arc individually.

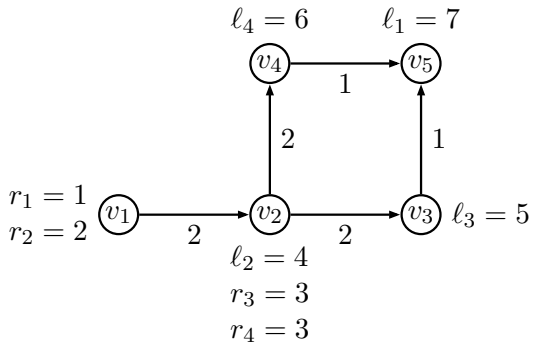


Figure 5: Flat-network $\mathcal{G}'$ with transit time τ_a for each arc (modified example instance)

4.2 Selecting the Commodity-Node Discretizations

The discretization of time at a node for each commodity influences the number of holding arcs and corresponding variables. But it also has an impact on the strength of the relaxation, by permitting more or fewer timed paths for a commodity:

Proposition 3. *Let $\mathcal{T}$ be a discretization, k a commodity, and $I_1 = [h_1, h_1') \in \mathcal{T}_{(u,v)}$, $I_2 = [h_2, h_2') \in \mathcal{T}_{(v,w)}$ feasible arc intervals for k, where $(u,v),(v,w) \in \mathcal{A}^k$. Denote their earliest arrival and latest departure at v by*

$$e = \max\{h_1, \underline{t}_{ku}\} + \tau_{(u,v)}, \qquad s = \min\{h_2' - 1, \bar{t}_{kw} - \tau_{(v,w)}\}.$$

Then,

$$I^+(k,(u,v),I_1) \leq I^-(k,(v,w),I_2) \iff \nexists [t,t') \in \mathcal{T}_{kv} \text{ with } s < t \leq e.$$

Proof. By definition, the relaxed arrival interval contains e, and the relaxed dispatch interval contains s. The former is later than the latter exactly when a node interval begins strictly after s and no later than e. □

So if the node discretization contains a time point $t \in (s,e]$, a commodity cannot dispatch on (u,v) in I_1 and then on (v,w) in I_2 by Definition 6. This can of course only be the case if $s < e$. Otherwise, the two arc-intervals are always compatible. We call a node discretization $\mathcal{T}_{kv}$ *strict* if it contains a $t \in (s,e]$ for every pair of feasible incoming and outgoing arc intervals with $s < e$. Thus, a strict node discretization permits as few arc transfers as possible. Evidently, a full node discretization (i.e., $\mathcal{T}_{kv} = \{[t,t+1) \mid t \in [\underline{t}_{kv}, \bar{t}_{kv}]\}$) is strict, potentially improving relaxation strength, but also yields many holding arcs. Thus, we propose the following sparse node discretization that nonetheless is just as restrictive as a full discretization:

Theorem 1. *For a given arc discretization $(\mathcal{T}_a)_{a\in\mathcal{A}}$, commodity k, and node $v \in \mathcal{V}^k$, denote by*

$$\begin{aligned} E_{kv} =& \{\max\{h, \underline{t}_{ku}\} + \tau_{(u,v)} \mid (u,v) \in \mathcal{A}^k, [h,h') \in \mathcal{T}_{(u,v)} \text{ feasible for } k\}, \\ L_{kv} =& \{\min\{h' - 1, \bar{t}_{kw} - \tau_{(v,w)}\} \mid (v,w) \in \mathcal{A}^k, [h,h') \in \mathcal{T}_{(v,w)} \text{ feasible for } k\} \end{aligned}$$

the sets of earliest arrival and latest departure times associated with the feasible incoming and outgoing arc intervals, respectively. Choose $\mathcal{T}_{kv}$ by partitioning $[\underline{t}_{kv}, \bar{t}_{kv}]$ at the time points in

$$\{\min\{t \in E_{kv} \mid t > s\} \mid s \in L_{kv}\}, \tag{3}$$

omitting empty minima. Then, $\mathcal{T}_{kv}$ is strict and has the minimum number of intervals among all strict node discretizations.

Proof. For any $s \in L_{kv}$ and $e \in E_{kv}$ with $s < e$, the construction inserts the first arrival time after s, which lies in $(s, e]$. Thus, the partition is strict.

For each distinct selected boundary t, choose $s_t \in L_{kv}$ such that $t = \min\{e \in E_{kv} \mid e > s_t\}$. Every strict partition must contain a boundary in $(s_t, t]$. These intervals are pairwise disjoint: if $t < t'$, then $s_{t'} \geq t$, since otherwise t' would not be the first arrival after $s_{t'}$. Consequently, every strict partition needs at least one internal boundary for each selected point. The constructed partition therefore has the minimum number of intervals. □

A sparser node discretization may still yield the same solution value for a particular instance, while resulting in fewer holding arcs. We conducted preliminary experiments with alternative strategies for determining the node-discretizations, but they yielded no performance improvements. Therefore, unless stated otherwise, for the remainder of this work we only specify arc discretizations $(\mathcal{T}_a)_{a \in \mathcal{A}}$ and use the node discretizations of Theorem 1.

We note that for problem variants where commodity handling at nodes is modeled in more detail, the flexibility afforded by commodity-specific node discretizations may be used to more effect. For example, if commodities need to be handled (sorting, customs, loading, …) and there is a limited handling capacity and commodity-specific handling durations, some commodities might require finer node discretizations. Of course, such variants might require specific procedures for deriving (and refining) the node discretizations.

4.3 Refining the Discretization

Solving the relaxation R-SND-RR($\mathcal{T}$) only gives us a lower bound on the objective value of SND-RR. After obtaining a solution W to R-SND-RR($\mathcal{T}$), we need to determine if it is possible to convert W into a feasible solution S to SND-RR with $c(W) = c(S)$, in which case S would be optimal. If that is not the case, we want to refine the discretization so that we do not receive the same relaxation solution again and repeat the process. We use some concepts introduced by Marshall et al. (2021) adapted to our relaxation for both these steps. Note that for simplicity of presentation and notation, in the following we will assume that the paths in solutions obtained from the relaxation are simple, i.e., do not repeat nodes.

A *consolidation* (a, κ) is a tuple of an arc $a \in \mathcal{A}$ as well as a set of commodities $\kappa \subseteq \mathcal{K}$ with the interpretation that these commodities are transported together along this arc. Given a solution $W = (P, \mathcal{I})$ to R-SND-RR($\mathcal{T}$), we define its set of consolidations as $C = C(W) = \{(a, \bar{\kappa}(a, I)) \mid a \in \mathcal{A}, I \in \mathcal{T}_a \colon |\bar{\kappa}(a, I)| \geq 1\}$ and call the pair (P, C) a *flat-solution*. Given such a flat-solution, we want to determine if it can be converted into a feasible solution to SND-RR, in which case we call it *implementable:*

Definition 7 (Implementable (adapted from Marshall et al. 2021))**.** *A flat-solution (P, C) is implementable if there exist dispatch times T such that the solution $S = (P, T)$ to SND-RR is feasible and $t_a^{k_1} = t_a^{k_2}$ for all $(a, \bar{\kappa}) \in C, k_1, k_2 \in \bar{\kappa}$.*

If a flat-solution is not implementable, we also don't want it to occur again in the relaxation, i.e., we want to make it not *representable*:

Definition 8 (Representable (adapted from Marshall et al. 2021))**.** *A flat-solution (P, C) is called representable with respect to a discretization $\mathcal{T}'$ if there exist dispatch time intervals $\mathcal{I}'$ in $\mathcal{T}'$ such that $W' = (P, \mathcal{I}')$ is a feasible solution for R-SND-RR($\mathcal{T}'$) and the resulting set of consolidations is identical, i.e., $C(W') = C$.*

To detect if a flat-solution is implementable and if not to refine the discretization, we adapt the approach proposed by Shu et al. (2025) to our relaxation. They introduce a *dispatch-node graph* which is

constructed based on the flat-solution. If that graph contains no so-called *too-long paths*, the flat-solution is implementable. On the other hand, if it contains a too-long path, adding certain time points based on the path to the discretization is sufficient to ensure that the flat-solution is no longer representable. We first reproduce from their work how the dispatch-node graph is constructed, how a too-long path is defined, and how they can be found. Then we present a theorem that details which time points need to be added to our time discretization to make a nonimplementable flat-solution also not representable.

Definition 9 (Dispatch-node graph (Shu et al. 2025)). *Given a flat-solution (P, C), its dispatch-node graph is defined as a directed graph $\mathscr{G}(P, C) = (\mathscr{V}, \mathscr{A})$ with a set of dispatch (and arrival) nodes $\mathscr{V} = \{(k, v) \mid k \in \mathcal{K}, v \in p^k\}$ and an arc set $\mathscr{A} = \{((k_1, u), (k_2, v)) \mid ((u, v), \bar{\kappa}) \in C, k_1, k_2 \in \bar{\kappa}\}$. Each arc $a = ((k_1, u), (k_2, v)) \in \mathscr{A}$ has a length of $\rho_a = \tau_{(u,v)}$.*

As an example, Figure 6 shows the dispatch-node graph for a flat-solution obtained for our toy instance with the discretization from Figure 3. On the right, the paths for each commodity as well as the consolidations ($C = \{c_1, c_2, c_3\}$) in the solution are specified. Note that all arcs $a \in \mathscr{A}$ in this graph have a length of $\rho_a = 2$.

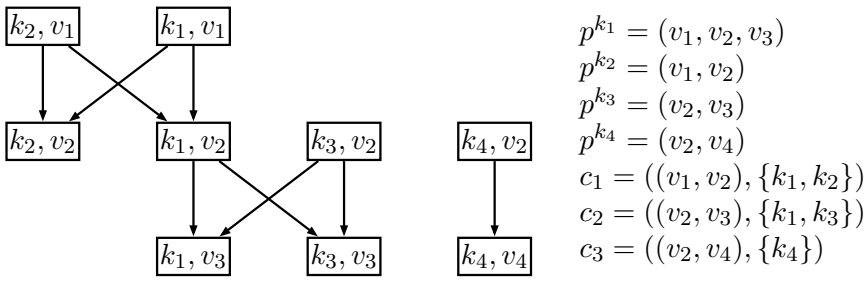


Figure 6: Dispatch-node graph for an optimal solution to R-SND-RR($\mathcal{T}$) for the toy instance

We denote for any path p in $\mathscr{G}$ its length with respect to ρ as $\rho(p) = \sum_{a \in p} \rho_a$. Let $\mathscr{P}$ denote the set of paths in $\mathscr{G}(P, C)$ that start at any node (k, o^k) representing the source node of some commodity $k \in \mathcal{K}$. Shu et al. (2025) showed that the length of a path $p \in \mathscr{P}$ starting from (k, o^k) to any other node (k', v') gives a lower bound on the earliest dispatch time of the commodity k' at node v'. Specifically, p certifies that with the consolidations C, commodity k' cannot be dispatched from v' before time $r^k + \rho(p)$. This motivates the following:

Definition 10 (Too-long path (Shu et al. 2025)). *Given a flat-solution (P, C), a (not necessarily simple) path $p \in \mathscr{P}$ from some node (k, o^k) to another node (k', v') is a too-long path if $r^k + \rho(p) > \bar{t}_{k'v'}$.*

Theorem 2 (Shu et al. 2025). *A flat-solution (P, C) is nonimplementable if and only if its dispatch-node graph $\mathscr{G}(P, C)$ contains a too-long path.*

For our toy example, the path $p = ((k_2, v_1), (k_1, v_2), (k_3, v_3))$ is too-long since $r^{k_2} + \rho(p) = 2 + 4 = 6 > 5 = \bar{t}_{k_3 v_3}$. So as discussed before, a solution that consolidates k_1 with k_2 and then k_1 with k_3 is not feasible for the original problem, since k_3 would arrive too late at its destination.

Shu et al. (2025) explain that a too-long path can contain a partial path that is also too-long. They show that if the discretization is adjusted so that the partial path can no longer occur in the dispatch-node graphs of representable flat-solutions, the original too-long path can also no longer occur. This motivates the following definition:

Definition 11 (Minimal too-long path (Shu et al. 2025)). *A too-long path p is minimal if every partial path of p that starts from the initial node of p, except p itself, is not a too-long path.*

To find minimal too-long paths, Shu et al. (2025) propose a label-propagating algorithm that enumerates all paths starting from each commodity's source node in the dispatch-node graph. Since all arcs have positive lengths, a path either eventually becomes too-long, or reaches some commodity's sink node and is not further extended. We direct the reader to their work for a detailed description of the algorithm.

In their Theorem 3, Shu et al. (2025) demonstrate how a nonimplementable solution can be made not representable once minimal too-long paths have been identified. Consider a too-long path p visiting n nodes. For $m \in [n)$, we write (k^p_m, v^p_m) for the node at the m-th position of p, and $a^p_m = (v^p_m, v^p_{m+1})$ for its m-th flat-arc. Then p tells us that if k^p_1 and k^p_2 are consolidated on a^p_1, k^p_2 cannot arrive at node v^p_2 before time $t^p_2 = r^{k^p_1} + \tau_{a^p_1}$. If we then also consolidate k^p_2 and k^p_3 on arc a^p_2, k^p_3 cannot arrive at node v^p_3 before time $t^p_3 = t^p_2 + \tau_{a^p_2}$. By splitting arc intervals at those time points, we can ensure that these consolidations lead to the same earliest arrival times in the relaxed problem, since the derived strict node discretizations exclude transfers to earlier dispatch intervals. Since the last commodity would be too late, this refinement prevents the same consolidations from occurring again in the relaxed problem.

We formalize the discretization refinement idea in the following theorem.

Theorem 3. *Let p denote a too-long path visiting n nodes in the dispatch-node graph $\mathscr{G}(P, C)$ of a flat-solution (P, C). Let $\mathcal{T}'$ be a discretization with strict commodity-node discretizations. For each $m \in [n)$, set $t^p_m = r^{k^p_1} + \sum_{m'=1}^{m-1} \tau_{a^p_{m'}}$. If, for every $m \in [n)$ there exists an interval $[t^p_m, h') \in \mathcal{T}'_{a^p_m}$, then the flat-solution is not representable with respect to $\mathcal{T}'$.*

Proof. Suppose the flat-solution is representable, and let $I_m = [h_m, h'_m) \in \mathcal{T}'_{a^p_m}$ be the interval selected for the consolidation involving k^p_m and k^p_{m+1}. We show by induction that $h_m \geq t^p_m$ for every $m \in [n)$. Since I_1 is feasible for k^p_1 at its source, it contains a dispatch time at least $r^{k^p_1} = t^p_1$. As no interval crosses t^p_1, this implies $h_1 \geq t^p_1$.

For $m \in [2, n)$, the earliest arrival of k^p_m on a^p_{m-1} in I_{m-1} is at least $h_{m-1} + \tau_{a^p_{m-1}} \geq t^p_m$. By strictness and Proposition 3, the latest feasible departure of k^p_m in I_m must be at least this arrival time. Thus $h'_m - 1 \geq t^p_m$, and since no interval crosses t^p_m, we obtain $h_m \geq t^p_m$.

Consequently, the earliest arrival of k^p_n in I_{n-1} is at least $t^p_{n-1} + \tau_{a^p_{n-1}} = r^{k^p_1} + \rho(p) > \bar{t}_{k^p_n v^p_n}$. This contradicts feasibility of I_{n-1} for k^p_n. Therefore, the flat-solution is not representable. □

In the case of our running example, to eliminate the too-long path $p = ((k_2, v_1), (k_1, v_2), (k_3, v_3))$ we would need to split the arc interval $[1, 3)$ on the first arc (v_1, v_2) at time point $r^{k_2} = 2$, resulting in the intervals $[1, 2)$ and $[2, 3)$. All other required intervals are already split as required in Figure 3c. The resulting arc discretizations are those used in Figure 4a. With the derived strict node discretizations, it is not possible to consolidate k_1 with k_2 and then k_1 with k_3. Solving the resulting relaxed problem gives an implementable (and thus optimal) flat-solution.

As noted by Shu et al. (2025), refining the discretization in the spirit of Theorem 3 is sufficient to eliminate a too-long path, but not necessary. When refining the discretization to eliminate some too-long path p, it can happen that some other too-long path p' also becomes eliminated. In that case, it is unnecessary to refine the discretization to eliminate p'. Therefore, it is useful to determine if a given discretization already eliminates a too-long path. A too-long path p' can occur exactly if the relaxation permits a feasible solution leading to the consolidations implied by p'. Note that two successive arcs in the too-long path also correspond to two flat-arcs traversed by a single commodity in its flat-path. For the relaxation solution to be feasible, the arc intervals chosen for two successive consolidations also need to be consistent. We formalize these requirements for a too-long path to occur:

Theorem 4. *Let p denote a too-long path visiting n nodes in the dispatch-node graph $\mathscr{G}(P, C)$ of a flat-solution (P, C). If the flat-solution is representable with respect to a discretization $\mathcal{T}'$, then we can find for each $m \in [n)$ an arc-interval $I_m \in \mathcal{T}'_{a^p_m}$ such that the following conditions hold:*

1. *for each* $m \in [n)$ *it holds that* I_m *is feasible on arc* a^p_m *for* k^p_m *and* k^p_{m+1} *and*

2. *for each* $m \in [2, n)$ *it holds that* $I^+(k^p_m, a^p_{m-1}, I_{m-1}) \leq I^-(k^p_m, a^p_m, I_m)$.

Proof. If the flat-solution is representable, there is some solution W' to R-SND-RR($\mathcal{T}'$) that achieves the same consolidations $C(W') = C$. Note that an arc $((k^p_m, v^p_m), (k^p_{m+1}, v^p_{m+1}))$ only exists in the dispatch-node graph if there exists a consolidation $(a, \kappa) \in C(W')$ with $a = a^p_m$ and $k^p_m, k^p_{m+1} \in \kappa$. By construction of $C(W')$, both k^p_m and k^p_{m+1} must have been dispatched together on a in the same interval I_m. Since W' is feasible, I_m must be feasible on a for both commodities, fulfilling the first condition. For an internal node (k^p_m, v^p_m) of the too-long path, with $m \in [2, n)$, we know that k^p_m used the interval I_{m-1} on arc a^p_{m-1} and interval I_m on arc a^p_m. Again by feasibility of W' we know that for each commodity the arc dispatch intervals are relaxed-time consistent, giving us the second condition. □

We use the contraposition of this condition: if no such arc intervals exist, then the flat-solution is not representable. Practically, we can use Algorithm 1 to determine if a path p is eliminated. It attempts to find arc intervals fulfilling the conditions in Theorem 4 by always selecting the earliest possible interval, which leaves the greatest flexibility downstream. If no fitting interval can be found, p is already eliminated by $\mathcal{T}$. Note that this condition is sufficient, not necessary.

Algorithm 1 Determining if $\mathcal{T}$ eliminates p

Require: Instance of R-SND-RR($\mathcal{T}$), too-long path p
Ensure: Returns whether p is eliminated

```
for all m ∈ [1, n) do
    I ← {I ∈ T_{a^p_m} | I is feasible for k^p_m and k^p_{m+1}, m = 1 or I⁺(k^p_m, a^p_{m-1}, I_{m-1}) ≤ I⁻(k^p_m, a^p_m, I)}
    if I = ∅ then
        return True
    end if
    I_m ← min I
end for
return False
```

Using this elimination check, we can adapt the refinement algorithm of Shu et al. (2025) to our relaxation, resulting in Algorithm 2. The approach always starts with the current discretization. It then constructs the dispatch-node graph and finds a set of minimal too-long paths. To make use of the observation that some too-long paths are eliminated if others are eliminated, we iteratively refine as follows: First, sort all the pairs of arcs and time points at which the discretization would need to be refined by non-decreasing order of time. Then, consider these pairs (t, a) in order. If a too-long path p for some m would introduce time point t^p_m (according to Theorem 3) on arc $a^p_m = a$, we first check if it is already eliminated. If not, we split the interval for the corresponding arc and reconstruct the derived commodity-node discretizations by Theorem 1 before the next elimination check.

4.4 Cycles in the Dispatch-Node Graph

Note that the dispatch-node graph $\mathscr{G}(P, C)$ associated with a flat-solution (P, C) may contain cycles. Such a cycle represents a cyclic dependency, where one dispatch cannot happen before another and vice versa, making the flat-solution always nonimplementable. As an example, consider the instance with two commodities depicted in Figure 7a. If we solve the relaxed problem with a coarse discretization for this instance, we would consolidate the two commodities on the shared arcs (v_1, v_2) and (v_3, v_4), resulting in the dispatch-node graph depicted in Figure 7b. The dispatch-node graph contains for example the too-long

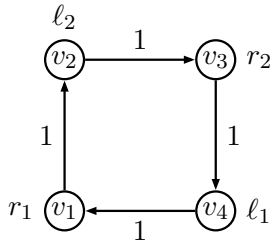
Algorithm 2 Refinement procedure

Require: Time discretization $\mathcal{T}$, relaxation solution W
Ensure: Time discretization $\mathcal{T}'$ such that flat-solution corresponding to W is not representable

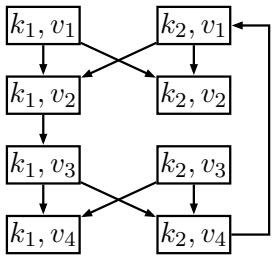
$\mathcal{T}' \leftarrow \mathcal{T}$
Determine flat-solution (P, C) corresponding to W
Construct dispatch-node graph $\mathscr{G}(P, C)$
Find set $\mathscr{P}$ of minimal too-long paths
Determine set $\Delta T \subseteq \mathbb{N} \times \mathcal{A}$ of arc time points to add, sorted by time
for all $(t, a) \in \Delta T$ **do**
    for all $p \in \mathscr{P}$ and m with $a^p_m = a$, $t^p_m = t$ **do**
        if p is not eliminated by $\mathcal{T}'$ **then**
            SplitInterval($\mathcal{T}'_a, t$)
            Derive commodity-node discretizations from $(\mathcal{T}'_a)_{a \in \mathcal{A}}$ by Theorem 1
        end if
    end for
end for

function SplitInterval($\mathcal{T}_a, t$)
    if there exists $I = [t', t'') \in \mathcal{T}_a$ with $t' < t < t''$ **then**
        $\mathcal{T}_a \leftarrow (\mathcal{T}_a \setminus \{I\}) \cup \{[t', t), [t, t'')\}$
    end if
end function

path p that starts with $((k_1, v_1), (k_1, v_2))$ and then repeats the cycle c in the graph, stopping at the first node (k', v'), where $r^{k_1} + \rho(p) > \bar{t}_{k'v'}$. To eliminate this path, a number of time points needs to be added on each arc that is (up to a difference of one) equal to the number of repetitions of the cycle in p. This may massively blow up the discretization size if $\rho(c)$ is small compared to $\bar{t}_{k'v'} - r^{k_1}$.



(a) Instance with two commodities



(b) Dispatch-node graph if all commodities traversing the same arc are consolidated

Figure 7: Example for cycles in dispatch-node graph

This problem was already observed by Marshall et al. (2021) for their *solution graph*, which is conceptually similar to the dispatch-node graph. Both node- and arc-based discretizations suffer from this inflation of the discretization size if cycles occur. But arc-based discretization only adds timed copies of flat-arcs implicated in the cycle. Contrast this with node-based approaches, which add timed copies of all flat-arcs incident to nodes implicated in the cycles. If these nodes have a large out-degree, an even larger part of the time-expanded network is inflated.

Both in theory and especially in practice we observe that initializing the discretization, such as with

the significant time points of Shu et al. (2025) (described in Section A.1), reduces the occurrence of such cycles. Yet, if commodities have very wide time windows (i.e., $\ell^k - r^k \gg \tau^k_{o^k d^k}$) few or no such time points can be found. This can lead to an even worse problem than the blow-up in relaxation size. Assume the instance in Figure 7a would contain a third commodity k_3 with $o^{k_3} = v_1$ and $d^{k_3} = v_3$. If we consolidate this commodity with k_1, k_2 on its first arc and with k_1 on its second arc, we have two cycles available in the dispatch-node graph. Now when enumerating minimal too-long paths, we can switch between the two cycles at will. This results in a number of minimal too-long paths that is exponential in the number of times that the cycles have to be walked to make any of the paths too-long. Obviously, enumerating an exponential number of anything can quickly overtake available time or memory resources.

Since we observed this effect in practice, we suggest a simple work-around that effectively limits the number of too-long paths enumerated: at each node in the dispatch-node graph we store the highest time value seen so far, where the time value is the release time of the commodity at the start of a too-long path plus the length of the partial path to the node. Now whenever a label would be extended to a dispatch-node, if its time value is not higher than the value stored at the node, the label is not extended. Effectively, this prunes some too-long paths that cause less delay than some other too-long path. This allows us to still find and eliminate some minimal too-long paths, which is necessary for our algorithm to eventually converge.

4.5 Dynamic Discretization Discovery Algorithm

We combine the previously introduced concepts into Algorithm 3 to solve SND-RR. It iteratively solves the relaxation R-SND-RR($\mathcal{T}$) and uses the heuristic from Marshall et al. (2021) (see Section A.2) to try and find a primal solution of identical value. This will succeed if the flat-solution corresponding to the relaxation solution is implementable. In that case, the algorithm has found an optimal solution and can terminate. Otherwise, the flat-solution is not implementable, and we add time points to the discretization to make it not representable and repeat the process.

Algorithm 3 Arc-based Dynamic Discretization Discovery Algorithm for SND-RR

Require: An instance of SND-RR
Ensure: S is an optimal solution
1: Initialize arc discretizations $(\mathcal{T}_a)_{a\in\mathcal{A}}$ and derive $\mathcal{T}$ by Theorem 1
2: Upper bound $\overline{z} \leftarrow \infty$, lower bound $\underline{z} \leftarrow -\infty$, solution $S \leftarrow \emptyset$
3: **while** $\overline{z} > \underline{z}$ **do**
4: Find an optimal solution W to R-SND-RR($\mathcal{T}$), $\underline{z} \leftarrow c(W)$
5: Obtain solution S' to SND-RR from primal heuristic
6: **if** $\overline{z} > c(S')$ **then**
7: $\overline{z} \leftarrow c(S'), S \leftarrow S'$
8: **end if**
9: **if** $\overline{z} > \underline{z}$ **then**
10: Refine discretization according to Algorithm 2
11: **end if**
12: **end while**

We remark on some implementation details: As proposed by Boland et al. (2017), we add the inequality $\sum_{\hat{a}\in A^k_{\mathcal{T}_t}} \tau_{\hat{a}} x^k_{\hat{a}} \leq \ell^k - r^k$ for all $k \in \mathcal{K}$, which ensures that paths are k-feasible. Otherwise, the heuristic may fail to find any feasible solutions. The relaxation model also contains the strong linking valid inequalities as proposed by Marshall et al. (2021) in their Section 5.4, as they have been shown to be generally beneficial. If more than 1000 too-long paths are enumerated in a dispatch-node graph or enumeration takes longer than $60\,\mathrm{s}$, we activate the pruning described in Section 4.4. The algorithm allows early termination if the

gap $\frac{\overline{z}-\underline{z}}{\overline{z}}$ reaches some pre-defined target gap value. By default, we solve the relaxation only to a gap of $98\,\%$ of the target gap value and then use the lower bound obtained by the solver as $\underline{z}$ instead of $c(W)$. We terminate the solving process if the solver for the relaxation obtains a lower bound that is good enough to reach the target gap value.

4.6 Anticipatory Refinement

As described in Section 4.1.4, node-based relaxations can be interpreted as arc-based approaches that choose to share a discretization for all arcs leaving a node. So when eliminating the same too-long paths, the node-based approaches generate a finer discretization than is technically necessary. Preliminary experiments showed that node-based approaches on average require fewer iterations to converge than arc-based ones, indicating that some of these additional time points avoid conflicts that have not been observed yet, but arise in future iterations.

We make use of this observation and try to anticipate too-long paths that could arise in solutions that are similar to one we obtained from solving the relaxation. The refinement approach described previously prevents some consolidations from reoccurring; we construct too-long paths that could arise if commodities were consolidated differently or would slightly change their path.

We first remove the last arcs of the enumerated minimal too-long paths from $\mathscr{G}$, since the consolidation chain leading to them will be broken up in any case. If the resulting graph is acyclic, which is often the case, we perform the following procedure: We determine for each dispatch node the highest lower bound on the dispatch time implied by any path leading to it, i.e.,

$$\underline{t}'(k,v) = \max\{r^{k'} + \rho(p) \mid p \text{ is a path from } (k', o^{k'}) \text{ to } (k,v)\}$$

and similarly the last time at which it needs to be dispatched for all following consolidations to succeed

$$\overline{t}'(k,v) = \min\{\overline{t}_{k'v'} - \rho(q) \mid q \text{ is a path from}(k,v) \text{ to } (k', v')\}.$$

Since the graph is acyclic, we can determine these values in linear time by processing the nodes in (reverse) topological order. Then, if we would add an arc between two dispatch nodes (k,v) and (k',v') with $\underline{t}'(k,v) + \tau_{(v,v')} > \overline{t}'(k',v')$, the graph would contain a new too-long path: a prefix p that achieves the maximum in the definition of $\underline{t}'(k,v)$ concatenated with the new arc and the suffix q that achieves the minimum in the definition of $\overline{t}'(k',v')$. Then, we can add this hypothetical too-long path (or rather the minimal too-long path it contains) into the set used for the refinement procedure (Algorithm 2).

Of course, not every potential connection between two dispatch nodes could plausibly occur in a relaxation solution. We only consider them if the hypothetical arc $((k,v),(k',v'))$ did not already exist in the original dispatch-node graph and both commodities could be consolidated on this arc, i.e., $(v,v') \in \mathcal{A}^k \cap \mathcal{A}^{k'}$ and $[\underline{t}_{kv}, \overline{t}_{kv'} - \tau_{(v,v')}] \cap [\underline{t}_{k'v}, \overline{t}_{k'v'} - \tau_{(v,v')}] \neq \emptyset$.

5 Computational Study

Recall the motivation for introducing arc-based discretization: it does not create unnecessary timed copies of arcs, thus reducing the size of the relaxation problem and potentially the computational effort to solve it. The goal of our study is to evaluate if and to which degree these effects are empirically observable.

5.1 Compared Algorithms

We implemented four basic variants of Algorithm 3 that differ only in how the time-expanded networks in the relaxation are constructed. `node-time` builds its networks according to the conditions in Shu et al. (2025)

based on a node-time-based discretization as proposed by Boland et al. (2017). `partition` uses the same conditions, adapted to the arc-dependent discretization of Van Dyk and Koenemann (2024). `node-interval` builds its network based on a node-interval-based discretization as specified by Marshall et al. (2021). Lastly, `arc` uses the construction detailed in this paper. We then apply the same variable-reduction rules to all networks: first, we replace a timed transport-arc copy if another copy of the same arc supports every eligible commodity, and, in each commodity's subnetwork, the replacement tail node is reachable from the original tail node and the original head node is reachable from the replacement head node (using only holding arcs). This subsumes the reduction rules presented by Marshall et al. (2021) and Shu et al. (2025). Then, we remove all unnecessary holding arcs and capacity variables as proposed by Shu et al. (2025).

For refinement they all use Algorithm 2, adapted to their relaxations. As in the original refinement algorithm of Shu et al. (2025), the node-based approaches `node-time` and `node-interval` add time points at nodes along the too-long paths. Approach `partition` does the same, but only adds the time point to the copy of the node corresponding to the flat-arc implicated in the too-long path. Otherwise, the algorithms are exactly the same. To validate that our implementations of the alternative discretizations provide a fair baseline, we compare `node-interval` and `partition` against the publicly available implementations of Marshall et al. (2021) and Van Dyk and Koenemann (2024) in Appendix D using the identical hardware and MIP solver.

To achieve state-of-the-art performance, various speed-up techniques from the literature are necessary. Since these might affect each discretization approach differently, we introduce variants `node-time+`, `node-interval+`, `partition+`, and `arc+` that include the following enhancements:

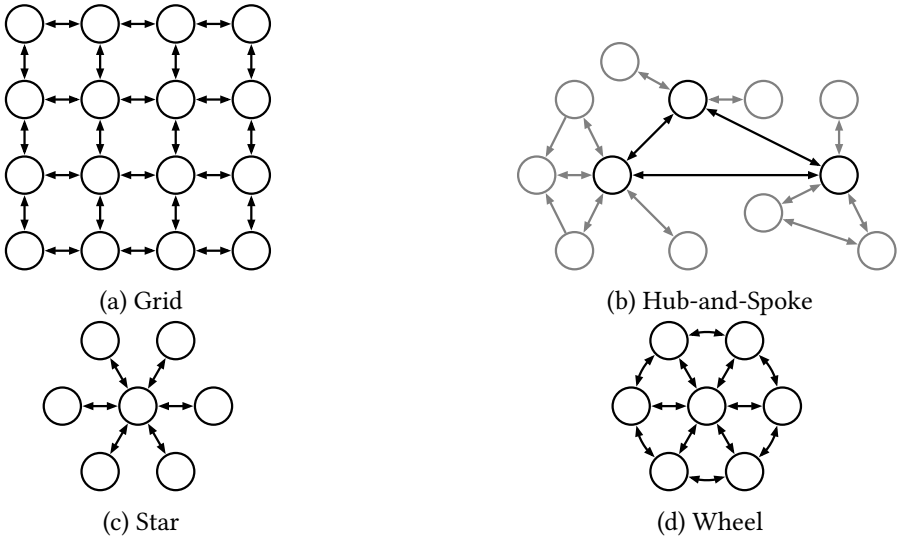
- The adaptive gap procedure proposed by Marshall et al. (2021), where the gap to which the relaxation is solved depends on the current value of $\frac{\overline{z}-\underline{z}}{\overline{z}}$.
- As proposed by Shu et al. (2025), we obtain too-long paths from all solutions the solver finds for the relaxation and not just the best solution. We also call the repair heuristic on all of these solutions.
- Initialize the relaxation with the initial time points of Shu et al. (2025). (See Section A.1 for the adaptation to arc-based discretization.)
- Use the newly proposed anticipatory refinement (Section 4.6) to predict up to 1000 hypothetical too-long paths and refine them as well.

All algorithms are implemented in Python 3.12.2 and use Gurobi 12.0.1 (Gurobi Optimization, LLC 2025) to solve the relaxation and the linear program in the primal heuristic. All instances were solved with a time limit of one hour and a target gap of $1\,\%$ on machines with a Xeon L5630 Quad Core $2.13\,\mathrm{GHz}$ processor.

5.2 Benchmark Instance Sets

We use two instance sets from the literature as well as a new one that we propose to investigate networks with differing topologies. The first set comprises instances from Boland et al. (2017). Note that in these instances, the commodities' networks are not restricted, so in principle they are equivalent to the full flat-network. In a preprocessing step, we construct subnetworks for each commodity k containing only the nodes and arcs that can be part of a k-feasible path. The instances in this set were classified by Marshall et al. (2021) depending on the tightness of the release-deadline time windows and the relation of variable to fixed costs. They classify an instance as having low flexibility (LF) if $\min_{k\in\mathcal{K}} \ell^k - r^k - \tau^k_{o^k,d^k} < 227$ and high flexibility (HF) otherwise. An instance also has a low cost ratio (LC) if $\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \frac{f_a}{c_a u_a} < 0.175$ and a high cost ratio (HC) otherwise. For our analysis, we report the results only for the instance classes *HC/LF* and *HC/HF*, since the instances in the other classes are trivial to solve (on average less than one second). The remaining 360 instances have 20 to 30 nodes, 228 to 686 arcs, and 100 to 400 commodities.

The second set comprises the 576 instances proposed by Van Dyk and Koenemann (2024) that were designed to showcase the benefit of arc-based discretizations of time. Specifically, in (near-)optimal solutions to these instances, some nodes will have dispatches on many outgoing arcs at many time points. So if all arcs receive timed copies for each commodity that can traverse them at all time points for this node, the relaxation should become difficult to solve quickly. Their instances have 20 or 30 nodes, 45 to 480 arcs, and 100 to 300 commodities. They divide their instances into three classes of 192 instances each, which we call *designated path*, *critical times*, and *hub-and-spoke (easy)*. In designated path instances, each commodity's subnetwork $\mathcal{G}^k$ consists only of one origin-destination path and just the dispatch times need to be decided. In critical times instances, release times and deadlines of commodities are modified such that there are just a handful of time points per node to simulate warehouse shifts. Hub-and-spoke instances represent regional networks that are connected by hub nodes such that commodities can only travel between two regions via the hubs.



(a) Grid (b) Hub-and-Spoke (c) Star (d) Wheel

Figure 8: Examples for the four topology classes in the new instance set

We propose a new third set that is designed to showcase how the different algorithms perform on networks with varying topologies. As in Van Dyk and Koenemann (2024), our assumption is that arc-based discretization has a benefit versus node-based discretization if the network contains some hub-like nodes which have a high outdegree and are traversed by many commodities in near-optimal solutions. Therefore, we create 100 instances each based on four different topologies with a varying degree of how important a few hub-like nodes are. For a detailed description of the generation process for the instances see Appendix C. The instances have 12 to 30 nodes, 40 to 134 arcs, and 50 to 200 commodities. So they are rather small compared to the instances in the other sets, but they are still reasonably hard due to high fixed costs for dispatches and flexible time windows for the commodities, making smart consolidation imperative for high-quality solutions. We next describe the underlying topologies.

In the class *grid* the network is a four by four grid where all arcs also have the same travel time, see Figure 8a for an example. Therefore, there are no candidates for hub-like nodes: no node is significantly better connected than any other. We would expect the arc-based approach to have very little or no benefit on these instances.

The class *hub-and-spoke (hard)* contains instances with the same topology as those in hub-and-spoke (easy). Some nodes are designated hubs and are interconnected, all other nodes are assigned to a hub and only connected to the hub and possibly other nodes assigned to the same hub, see Figure 8b for an example. Due to the choice of the commodity parameters, these instances are significantly harder to solve than the

ones in the easy set. By design, a few nodes will serve as hubs which most commodities need to traverse, therefore we expect the arc-based approaches to perform better on this set.

The classes *wheel* and *star* are similar in that they feature one dominant node to which all other nodes are connected, so it has a very high out-degree and many commodities will traverse this node due to the excellent consolidation opportunities. Therefore, we expect the benefit of the arc-based discretization to be the most pronounced on these sets of instances.

5.3 Benchmark Performance and Algorithmic Enhancements

We first investigate the relative performance of different discretizations on the benchmark instance set of Boland et al. (2017). The results for the base algorithms are reported in Table 2. We state the average solving time (including time-outs as $1\,\mathrm{h}$), number of refinement iterations, network size in the first and last iteration, and percentage of instances solved to a gap of $\leq 1\,\%$ within the time limit. When we speak about network size, we refer to the proportion

$$\frac{\sum_{k\in\mathcal{K}} |A^k_{\mathcal{T}\mathrm{t}}|}{\sum_{k\in\mathcal{K}} |A^k_{\mathrm{t}}|}$$

i.e., the number of timed transport arcs in the relaxed time-expanded networks for each commodity divided by the number in the full time-discretization (Section 3.2). This metric considers not only the size of the time-expanded networks but also how many commodities can traverse each arc, and thus how many arc-flow variables the relaxed problems contain.

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
HC/HF	`arc`	999.73	9.5	0.25	0.69	80.8
	`node-time`	584.54	6.5	0.45	0.83	92.1
	`node-interval`	486.73	6.2	0.25	0.89	94.9
	`partition`	662.58	7.1	0.33	0.88	89.3
HC/LF	`arc`	122.67	7.5	0.82	1.67	98.4
	`node-time`	65.00	4.6	1.45	2.22	100.0
	`node-interval`	42.24	4.5	0.82	2.30	100.0
	`partition`	61.48	5.2	1.11	2.26	100.0

Table 2: Results of base algorithms on HC/* instances, $1\,\%$ gap limit, $1\,\mathrm{h}$ time limit

Recall that `arc` only refines time on arcs implicated in too-long paths, while the others also refine time on arcs incident to nodes in the too-long path. As expected, `arc` produces smaller networks. In the first iteration, it always starts with an equivalent network to `node-interval`, but at termination its networks stay smaller (e.g., $0.69\,\%$ versus $0.89\,\%$ for HC/HF). But the other relaxations also profit from refining time on additional arcs: doing so may prevent illegal consolidations in future iterations. And indeed we observe that `arc` requires more iterations to termination, indicating that its smaller networks also lead to worse dual bounds. For the base algorithms, we see that the increase in iterations is not sufficiently offset by easier to solve relaxation problems and `arc` performs the worst on average.

To investigate the relationship between network size and relaxation strength, we plot one versus the other for each iteration in Figure 9. We measure the relaxation strength as the dual gap $\frac{\mathrm{LB}_i - \mathrm{LB}^*}{\mathrm{LB}^*}$ where LB_i is the lower bound in iteration i and LB^* is the best lower bound obtained by any solver over all iterations. A trend emerges: `arc` generally has many more iterations with weak lower bounds than `node-interval`, but also usually terminates with smaller networks. The marked representative instance demonstrates the

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
HC/HF	`arc+`	170.78	3.9	0.54	0.79	99.4
	`node-time+`	167.08	3.0	0.90	1.07	99.4
	`node-interval+`	183.02	2.9	0.86	1.19	99.4
	`partition+`	177.25	3.1	0.84	1.06	99.4
HC/LF	`arc+`	18.07	2.7	1.53	2.01	100.0
	`node-time+`	19.00	2.1	2.59	2.89	100.0
	`node-interval+`	21.05	2.1	2.39	2.95	100.0
	`partition+`	17.85	2.1	2.39	2.77	100.0

Table 3: Results of enhanced algorithms on HC/* instances, 1 % gap limit, 1 h time limit

typical pattern: in the first iteration, both approaches yield the same network and lower bound. But already in the second iteration, `node-interval`'s network is larger than `arc`'s in its fourth iteration. Generally, `arc` can achieve the same lower bound with smaller networks, but requires more iterations to get there. The same pattern emerges when comparing `arc` to `node-time` or `partition`, thus the comparison plots are omitted here.

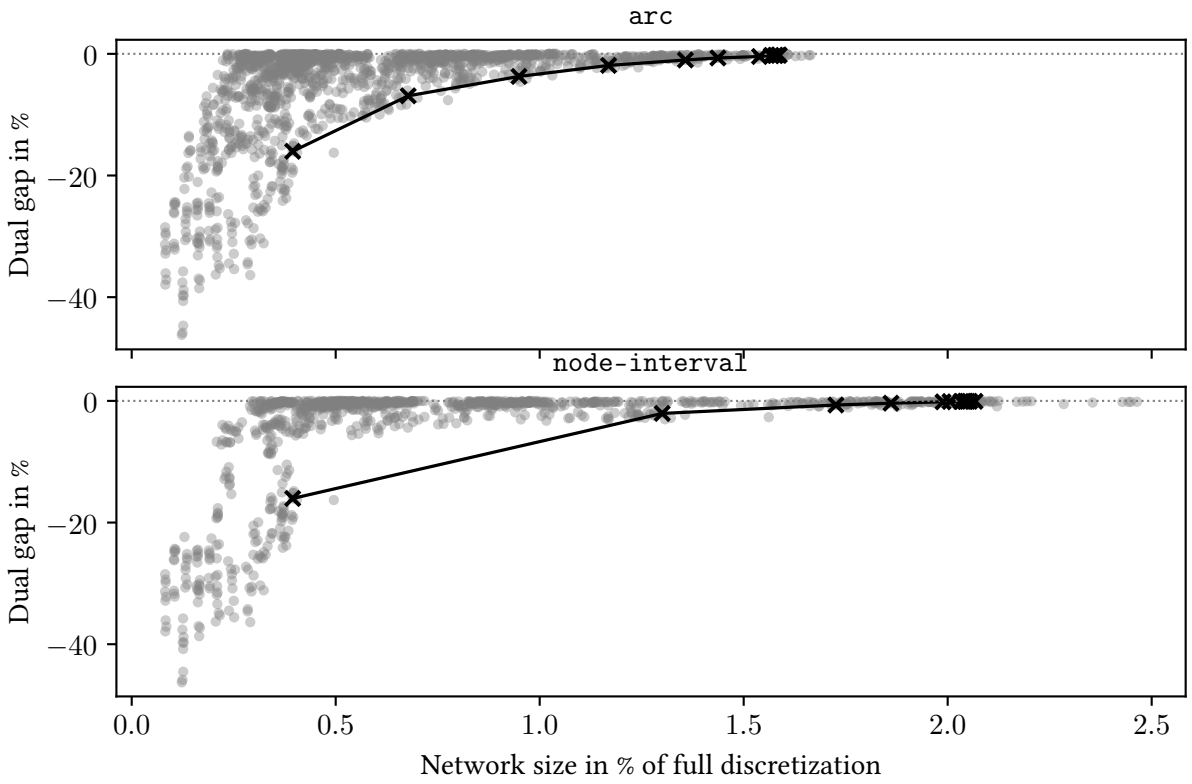


Figure 9: Lower bound quality versus relaxed-network size at each iteration on HC/HF instances. Crosses indicate successive iterations for one representative instance

Figure 10 shows how the time spent on each iteration changes over the iterations, normalized from 0 (first iteration) to 1 (last iteration). We observe that `arc` has the lowest median per-iteration time. Compare

the growth in time per iteration between `arc` and `node-interval`. They start with exactly the same relaxation in the first iteration and thus have near-identical solving times. But as the iterations progress, the median solving time of `arc` grows less than that of `node-interval`. So the additional growth in network size of the node-based approaches does also correspond to an increase in relaxation solving times. For the node-time-based approaches (`node-time` and `partition`), the many additional time points in the first iteration lead to longer relaxation solving times from the beginning.

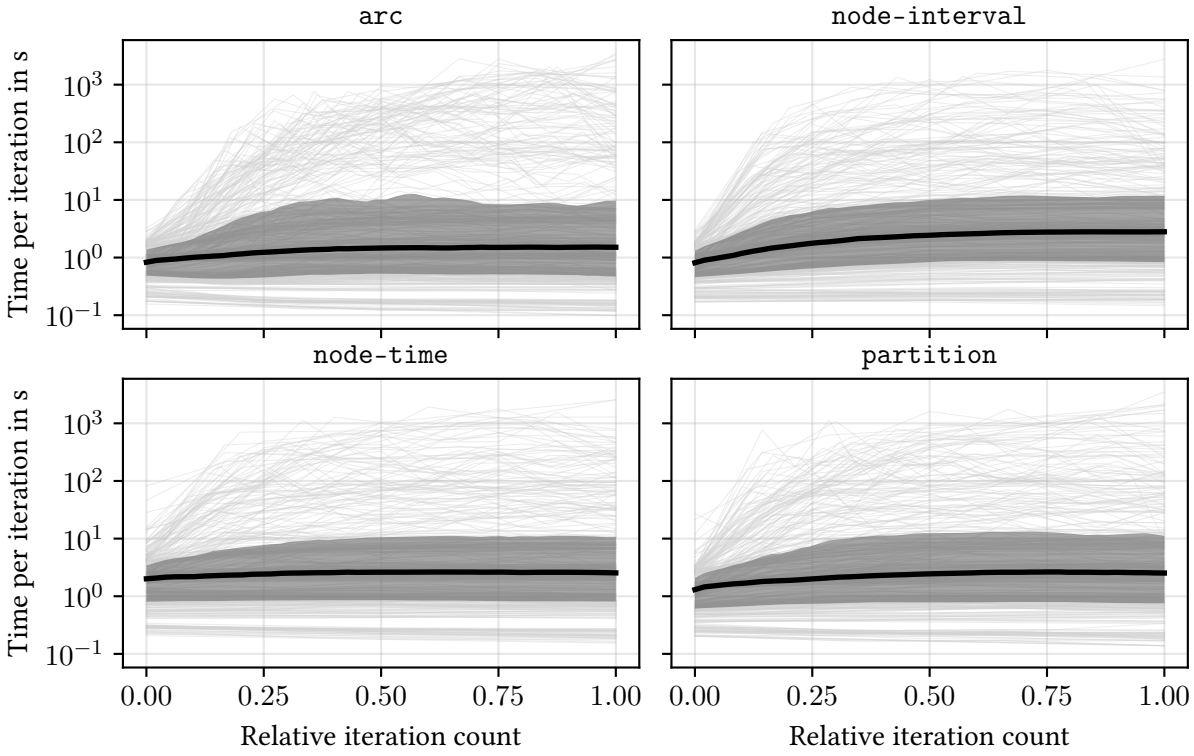


Figure 10: Time per iteration over iterations, normalized by number of iterations to termination. Gray lines represent single instances, dark gray area 25th to 75th percentiles, and the black line the median value

So if `arc` generally produces smaller networks, but suffers from weak lower bounds, one might expect that it gains more from algorithmic enhancements that lead to more time points in the discretization. Indeed, the four speed-up techniques implemented for the enhanced algorithms all fall in this category. The adaptive gap procedure stops the relaxation solving process earlier, obtaining weaker lower bounds, and thus generally leads to more iterations. But more iterations also mean more refinement rounds, increasing the size of the discretization. Similarly, refining all solutions obtained from the relaxation solver instead of just the best also leads to larger discretizations. The initial time points and the anticipatory refinement evidently also add more time points. The results in Table 3 confirm this: all enhanced approaches generate larger networks compared to the basic approaches (in Table 2), both in the first and last iteration. They also require fewer iterations on average and solve quicker. With these enhancements, all algorithms actually perform quite similarly. Compared to the basic approaches, this means that the arc-based approach benefitted by far the most.

To determine how much impact each of the enhancements has for each relaxation, we also conducted an ablation experiment, where we run each approach with all but one of the enhancements. Table 4 reports the results. We can observe that all features help every relaxation on average, with the one exception of

node-interval+ that actually profits from disabling the adaptive gap on the HC/LF instances. Interestingly, the anticipatory refinement helps all approaches, with nice performance improvements for a simple to implement enhancement. Most notably though, the arc-based approach profits strongly from the initial time points. Disabling this feature leads to 75 % (90 %) more iterations on HC/HF (HC/LF) instances compared to arc+, and the 166 % (152 %) increase in running time seen in Table 4. So as anticipated, adding more time points allows us to compensate for the weaker arc-based relaxations. But overall, on this benchmark set arc+ can only match, not beat, the performance of existing relaxations.

		Disabled enhancement			
Class	Relaxation	Adapt. gap	Anticip. Ref.	Init. time points	Ref. & rep. all sol.
HC/HF	arc+	+41.81	+28.36	+166.15	+69.70
	node-time+	+34.07	+35.38	+36.80	+44.82
	node-interval+	+22.14	+25.57	+17.47	+33.68
	partition+	+31.85	+36.42	+30.28	+44.54
HC/LF	arc+	+13.84	+32.51	+152.35	+10.43
	node-time+	+36.10	+37.14	+30.08	+27.95
	node-interval+	-3.65	+5.94	+22.39	+9.62
	partition+	+20.41	+35.02	+51.42	+32.77

Table 4: Ablation results: Percentage change in mean running time when the feature is disabled versus the fully-featured approach

5.4 Benefit of Arc-Based Discretization on Different Topologies

The instances in the previous section are based on randomly generated networks. Thus, they do not have any specific topology that real-life logistics networks might have. We therefore investigate next if an arc-based discretization can yield a computational advantage on instances with specific topologies. For this purpose we compare the performance of the enhanced algorithms on the dataset by Van Dyk and Koenemann (2024) and our newly generated instances. Results for the base algorithms can be found in Appendix E, we discuss here the results for the enhanced algorithms.

The results on the instances by Van Dyk and Koenemann (2024) are reported in Table 5. We observe that these instances are no match for state-of-the-art algorithms, as all methods solve them in a few seconds on average. On these instances, the additional iterations required by arc+ are not sufficiently offset by the faster per-iteration solving times. We would also like to highlight that these instances are quite peculiar: the too-long path based refinement actually takes longer on average than solving the relaxation. For example, on the hub-and-spoke instances, arc+ spends on average 17.3 s in refinement, so about three quarters of the total running time. Similarly, node-interval+ spends 12.3 s in refinement. Compare this to the HC/HF instances, where arc+ and node-interval+ spend 1.67 % and 0.93 % of their time in refinement. Additional investigations showed that this long time in refinement comes from some relaxation solutions where many commodities are consolidated, and enumerating too-long paths in the dispatch graph takes a long time. This is especially the case in later iterations, where near-feasible solutions require enumerating many paths that never become too-long.

Our newly generated instances are significantly harder to solve than the other classes, see the results in Table 6. On these harder instances, arc+ solves more instances and solves them faster on average than the other approaches for all classes except star, where partition+ is fastest. We now would like to investigate the hypothesis that the benefit of arc-based discretization is the largest on instances with

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
Des. path	arc+	13.68	2.6	15.28	29.51	100.0
	node-time+	14.08	2.2	24.81	36.78	100.0
	node-interval+	11.34	2.2	20.63	35.20	100.0
	partition+	12.09	2.4	17.20	30.49	100.0
Crit. time	arc+	10.59	2.5	32.80	38.30	100.0
	node-time+	11.59	2.1	40.64	44.99	100.0
	node-interval+	9.54	2.0	38.47	45.88	100.0
	partition+	10.33	2.2	38.34	43.69	100.0
H&S (easy)	arc+	21.76	2.8	19.04	26.73	100.0
	node-time+	20.73	2.5	25.71	32.54	100.0
	node-interval+	17.50	2.4	22.62	31.45	100.0
	partition+	20.62	2.6	21.68	29.87	100.0

Table 5: Results on instances from Van Dyk and Koenemann (2024), 1 % gap limit, 1 h time limit

hub-like nodes. The grid instances have no such nodes, and we see that arc+ outperforms the others only slightly, solving only about 7 percentage points more of the instances. Interestingly, the number of solved instances on the hub-and-spoke instances varies even less, though both arc+ and partition+ achieve faster solving times than the others. Once we move to the instances with a single hub node (wheel and star), both arc+ and partition+ can solve more instances, and faster, than the two node-based approaches. Note that on the star instances, node-interval+ is almost as bad as node-time+, indicating that here the benefit of moving to an arc-based/dependent discretization is much larger than the benefit of moving to an interval-based construction.

Table 6 also demonstrates that the arc-based discretization used by arc+ is more widely applicable than the arc-dependent one used by partition+. On the grid instances, where partition+ can rarely if ever find a non-trivial partition of the outgoing arcs of a node, it performs essentially the same as node-time+. But on the wheel and star instances, where partition+ can more often find such partitions, it does indeed construct slightly smaller networks and solves more instances faster. In contrast, arc+ can construct smaller networks on all network topologies. It performs the best, relative to the others, on the wheel instances, which aligns with our expectations: there is a single central hub, making arc-based discretization attractive, but partition+ cannot find fine-grained partitions of the arcs.

5.5 Case Study on Real-Life Topologies

We extend our study to networks arising in an industrial application: single wagon load transportation at the freight branch of the Swiss Federal Railways (SBB Cargo). They have graciously provided us with instances from their planning process. These instances exhibit a topology with a few hubs representing shunting yards, which most commodities traverse. We refer to Moldenhauer et al. (2025) for a more detailed description of the application context and a visualization of the transport network, which is very similar to the one underlying the instances we consider. The network structure is similar to the hub-and-spoke instances, though the regional networks follow mostly a tree structure (due to the Swiss topography), with some additional interconnections between regions. Note that the industrial context at SBB Cargo is complex, requiring multiple pre- and postprocessing steps and various operational constraints. However, the fundamental mathematical problem that drives the computational complexity of the employed models is scheduled service network design. Hence, it is interesting to study how our algorithms scale on this type

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
H&S (hard)	arc+	1394.19	3.8	6.10	7.66	74.0
	node-time+	1567.36	3.0	9.56	10.97	71.0
	node-interval+	1743.39	3.0	9.71	12.51	66.0
	partition+	1474.15	3.2	8.70	10.49	73.0
Grid	arc+	1752.80	3.1	16.57	24.72	64.0
	node-time+	1997.45	2.7	22.77	29.88	57.0
	node-interval+	2075.65	2.7	23.41	35.44	55.0
	partition+	2012.91	2.8	21.75	30.68	55.0
Wheel	arc+	1316.07	3.6	4.12	5.55	90.0
	node-time+	2026.06	3.0	6.97	7.94	69.0
	node-interval+	2218.38	2.9	6.70	9.02	63.0
	partition+	1907.61	3.0	6.54	7.79	74.0
Star	arc+	31.91	3.4	2.13	3.32	100.0
	node-time+	1271.49	2.9	4.81	5.30	79.0
	node-interval+	1403.10	2.9	3.77	5.45	74.0
	partition+	28.39	3.1	2.62	3.39	100.0

Table 6: Results on new instances, 1 % gap limit, 1 h time limit

of network topology, despite the fact that we only approximate the real-life application. The corresponding problems at SBB Cargo are currently solved with an approach based on the arc-based discretization presented in this paper.

We keep one of the operational modifications since it is of interest to a wider audience, requires minor changes to the presented algorithm, and gives another motivation for arc-specific discretization. This modification is commodity-specific processing times at each node, representing a time buffer for shunting operations at SBB Cargo. It can easily be modeled by replacing the transit time τ_a by τ_{ak}, which represents the combined transit time plus handling duration before a commodity k becomes available for dispatch at the head node of a. We use this duration when computing the relaxed dispatch and arrival intervals and significant time points, and in the repair heuristic. In the dispatch-node graph, we set the length of an arc $((k, v), (k', v'))$ to $\tau_{(v,v'),k'}$.

We investigate two sets of instances, *sbbc-small* and *sbbc-large*. The set sbbc-small consists of 30 instances with 20 to 35 nodes, 45 to 348 arcs, and 79 to 302 commodities. The commodities have on average 1.5 to 6 paths available in their subnetwork, though individual commodities have up to 69 paths available. Importantly, if we disregard direct arcs from origin to destination that are present in some instances, each instance has nodes that most commodities need to traverse. These are large shunting yards (e.g., in Zürich or Basel), and depending on the instance the most trafficked node *has* to be visited by about 65 % to 85 % of commodities. These instances thus resemble our wheel instances in that a single hub is traversed by most commodities.

The set sbbc-large consists of three instances. One of them we call *base*, it is a large scale instance obtained from a network-wide planning process. Due to the enormous size (209 nodes, 484 arcs, 1967 commodities), most commodities are restricted to a single promising path, and on average commodities have 1.087 paths available. Similarly to the small instances, there is one important hub node that has to be traversed by about 72 % of commodities. The other two instances *base+1* and *base+2* are derived from this instance to investigate the benefit of relaxing this limited routing. In them, we add to each commodity's

subnetwork the arcs on the shortest (w.r.t. travel time) or two shortest paths. Note that to the sbbc-large instances we apply some trivial preprocessing, for example removing commodities that share no arcs with any other commodity. Due to the added routing flexibility, the preprocessing can remove fewer nodes, arcs, and commodities in the base+1 and base+2 instances. Therefore, they have slightly higher counts of nodes (211 and 212), arcs (508 and 514), and commodities (1977 and 1978).

On the sbbc-small instances, we ran each solver three times with different seeds, to account for the low number of instances. In Table 7 we give mean results over all instance-seed pairs. All approaches could solve all instances quite fast, except `node-interval+`, which failed to solve one instance-seed pair. Observe that `arc+` requires essentially the same number of iterations as the other approaches, but its final networks are about one third smaller, resulting in significantly faster solving times. So for these instances, with a strong focus on a single hub node, our relaxation is very beneficial.

Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
`arc+`	35.88	2.8	0.51	0.64	100.00
`node-time+`	115.75	2.7	0.73	1.01	100.00
`node-interval+`	137.94	2.8	0.67	1.07	98.89
`partition+`	119.49	2.6	0.69	0.95	100.00

Table 7: Results on sbbc-small instances, $1\,\%$ gap limit, $1\,\mathrm{h}$ time limit

Since the sbbc-large set contains only three instances, we solve each of them with 10 seeds. This also allows us to observe the spread in performance due to performance variability (Lodi and Tramontani 2014). We also activate the pruning during too-long path enumeration from the start, since otherwise most of the time is spent on refinement in many runs, yielding no improvements after the first iteration.

For these instances, we are interested not only in how the algorithms scale with more flexible commodity routing, but also how additional flexibility improves the achievable objective values. We therefore set the target gap to $0.1\,\%$, to get better solutions where possible. This gap is not reached by any solver on any instance, but for the base instance, `arc+` reaches an average gap of $0.50\,\%$ and `partition+` of $0.64\,\%$. Figure 11 shows the distribution of the gap values over the seeds for each instance and approach. We can observe that all perform quite well on the base instance. But once more routing options become available, the final gaps increase for all approaches. Yet, `arc+` performs the best on average, yielding an average gap of $3.09\,\%$ on the base+1 instance, while the second best approach, `partition+`, yields an average gap of $6.38\,\%$. Overall we can observe that on these large scale instances with a favourable structure, the arc-based discretizations significantly outperform the node-based discretizations.

Interestingly, despite the larger remaining gaps, allowing more routing options can yield better primal solutions than for the base instance. Consider the primal solution quality reported in Table 8. Note that the solution quality here is with respect to the instance before preprocessing, to obtain comparable values. The best primal solution found across all algorithms improves as more flexibility is permitted. Thus, even if these instances cannot be solved to optimality, running the solver with different seeds in parallel can yield better solutions, compared to solving more restricted instances to a smaller gap. Again we observe that the arc-based discretization allows us to obtain the highest quality solutions.

6 Conclusions and Future Work

We presented a novel arc-based relaxation for the continuous-time service network design problem, adapted various algorithmic enhancements, and proposed a new one. Our experiments showed that under compara-

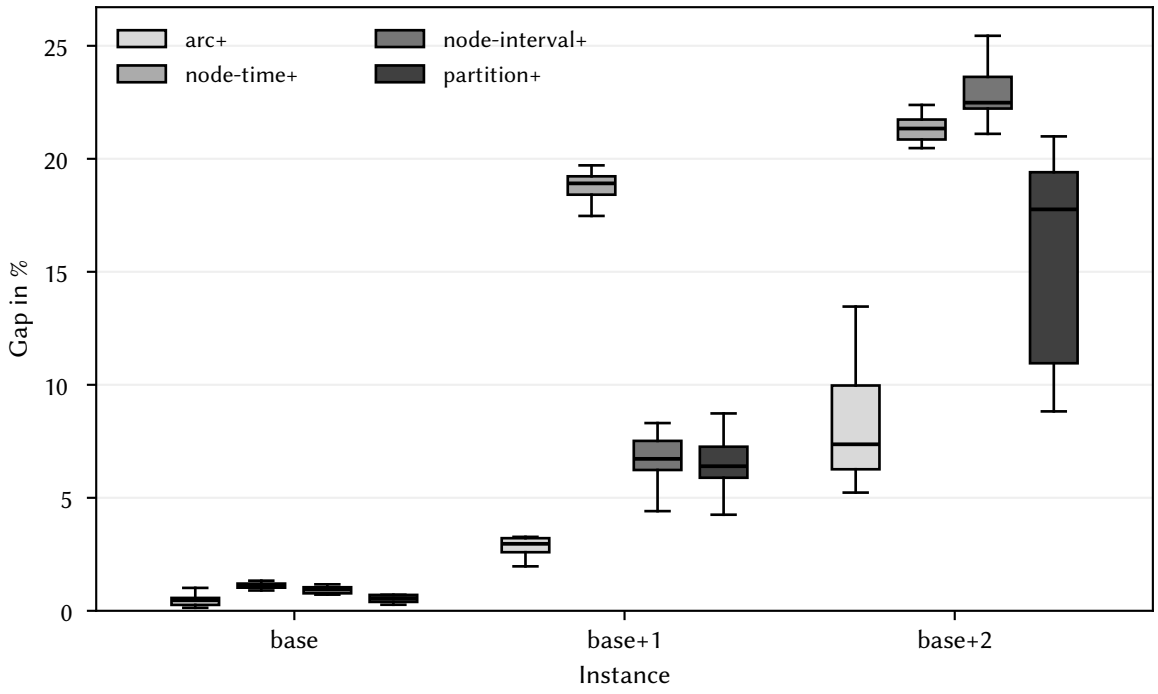


Figure 11: Boxplot of gap values at termination on the sbbc-large instances, 0.1 % gap limit, 1 h time limit

Algorithm	base	base+1	base+2
`arc+`	**100.00**	**94.52**	**93.53**
`node-time+`	**100.00**	107.67	95.42
`node-interval+`	100.05	95.77	107.54
`partition+`	**100.00**	94.86	96.26

Table 8: Best primal bound found over 10 seeds for each sbbc-large instance, normalized to the best value obtained by any solver on the base instance

ble refinement strategies the arc-based relaxation can yield comparable lower bounds with significantly smaller time-expanded networks. While more refinement iterations can be necessary, our approach matches existing methods on benchmarks and solves some challenging instance sets with hub-like nodes faster. On the SBB Cargo instances, our approach solved small instances about three times faster and achieved smaller gaps on larger instances. We also demonstrated that increasing routing flexibility can yield lower cost solutions, and our arc-based approach scales better to such cases.

In principle, we can interpret node-based approaches as arc-based approaches that 'accidentally' refine time on other arcs, preventing illegal consolidations in future iterations. Our anticipatory refinement is a first heuristic step towards predicting such consolidations, but future theoretical insights may allow more systematic refinement. Another question is how to determine a smallest discretization that prevents all too-long paths seen in previous iterations. This would allow us to remove time points while ensuring no backsliding with respect to nonimplementable solutions.

On the application side, it is promising to apply arc-based discretization to problem variants with per-arc constraints that may benefit from fewer timed arc copies, such as time-dependent travel times,

arc availabilities, or limits to the number of dispatches on an arc per time interval. Further algorithmic developments would also benefit from publicly available challenging data sets from real-life service network design applications.

Code and Data for our implementation and the used instance sets (except the SBB Cargo instances) will be published on GitHub if accepted.

Acknowledgements We thank Carsten Moldenhauer and SBB Cargo for providing instances and insightful discussions on this and related network design problems.

A Adapted Algorithmic Components

A.1 Significant Time Points

The coarsest initial arc discretization is $\mathcal{T}_a = \{[\underline{t}_a, \bar{t}_a + 1)\}$ for all $a \in \mathcal{A}$, with commodity-node discretizations derived by Theorem 1. We can strengthen the initial discretization by adding additional time points as proposed by Shu et al. (2025). They note that some commodities can never be consolidated on an arc (u, v) because the earliest time at which some commodity can arrive at u plus the travel time $\tau_{(u,v)}$ might already be later than the latest time at which another commodity has to be dispatched at v to still arrive on time. They show that these consolidations can be made impossible in the relaxation by inserting what they call *significant time points*. We adapt their statement to our relaxation and notation:

Theorem 5 (Significant time points (adapted from Shu et al. 2025))**.** *Given two different commodities $k_1, k_2 \in \mathcal{K}$ and an arc $a = (u, v) \in \mathcal{A}^{k_1} \cap \mathcal{A}^{k_2}$ with $\underline{t}_{k_2 u} + \tau_{(u,v)} > \bar{t}_{k_1 v}$, if there exists an interval $[t, t') \in \mathcal{T}_a$ with $t \in [\bar{t}_{k_1 v} - \tau_{(u,v)} + 1, \underline{t}_{k_2 u}]$, then no interval in $\mathcal{T}_a$ can be feasible for both k_1 and k_2.*

Proof. Consider that the interval $[t, t')$ cannot be feasible for k_1, since the earliest possible arrival time $t + \tau_{(u,v)} \geq \bar{t}_{k_1 v} + 1$ is already after the latest possible dispatch time $\bar{t}_{k_1 v}$, compare Definition 3. Evidently, all later intervals are also not feasible for k_1. Similarly, no interval before $[t, t')$ can be feasible for k_2, since the end of any such interval would need to be before the earliest dispatch time $\underline{t}_{k_2 u}$. Therefore, there exists no interval that is feasible for both commodities, and they can never be consolidated on this arc. □

For an arc $(u, v) \in \mathcal{A}$, we can collect the set of intervals of significant time points for all impossible consolidations as

$$I_{(u,v)} = \{[\bar{t}_{k_1 v} - \tau_{(u,v)} + 1, \underline{t}_{k_2 u}] \mid k_1, k_2 \in \mathcal{K} \colon (u, v) \in \mathcal{A}^{k_1} \cap \mathcal{A}^{k_2}, \underline{t}_{k_2 u} + \tau_{(u,v)} > \bar{t}_{k_1 v}\}.$$

Shu et al. (2025) propose solving a minimum hitting set problem to determine the smallest set of time points to cover all intervals. Since they can only add time points for a node and thereby to all outgoing arcs of that node, they determine the minimum hitting set for the intervals of all outgoing arcs of a node u, i.e., for $I_u = \bigcup_{v \colon (u,v) \in \mathcal{A}} I_{(u,v)}$. Our arc-based approach allows us to solve the minimum hitting set problem for each arc and add the resulting time points only to the discretization for this arc. This can potentially lead to fewer intervals for each arc in the relaxed problem compared to adding the same time points for all outgoing arcs. Let the ordered set $T_{(u,v)} = \{t_1, \ldots, t_m\}$ denote a minimum hitting set for $I_{(u,v)}$. We initialize the set of intervals for arc (u, v) as $\mathcal{T}_{(u,v)} = \{[\underline{t}_a, t_1), [t_1, t_2), \ldots, [t_m, \bar{t}_a + 1)\}$.

A.2 Primal Heuristic

To obtain primal feasible solutions before we find an implementable flat-solution, we utilize the heuristic first proposed by Boland et al. (2017) with the modified objective function of Marshall et al. (2021). As an input, it receives a relaxation solution $W = (P, \mathcal{I})$ and solves a linear program to determine consistent dispatch times for the flat-paths P while minimizing the deviation of dispatch times between commodities that are consolidated in the relaxed solution. We denote by n^k the number of arcs visited by the path p^k and by a_i^k the i-th arc in that path. In our notation, the linear program is as follows:

$$\min \quad \sum_{a \in \mathcal{A}} \sum_{I \in \mathcal{T}_a} \sum_{\{k_1,k_2\} \subseteq \bar{\kappa}(a,I)} f_a \delta_a^{\{k_1,k_2\}} \tag{4}$$

$$\text{s.t.} \quad \gamma^k_{a_i^k} + \tau_{a_i^k} \leq \gamma^k_{a_{i+1}^k} \qquad \forall k \in \mathcal{K}, \forall i \in [n^k) \tag{5}$$

$$\gamma^k_{a_1^k} \geq r^k \qquad \forall k \in \mathcal{K} \tag{6}$$

$$\gamma^k_{a_{n^k}^k} + \tau_{a_{n^k}^k} \leq \ell^k \qquad \forall k \in \mathcal{K} \tag{7}$$

$$\delta_a^{\{k_1,k_2\}} \geq \gamma_a^{k_1} - \gamma_a^{k_2} \; \forall a \in \mathcal{A}, \forall I \in \mathcal{T}_a, \forall \{k_1,k_2\} \subseteq \bar{\kappa}(a,I) \tag{8}$$

$$\delta_a^{\{k_1,k_2\}} \geq \gamma_a^{k_2} - \gamma_a^{k_1} \; \forall a \in \mathcal{A}, \forall I \in \mathcal{T}_a, \forall \{k_1,k_2\} \subseteq \bar{\kappa}(a,I) \tag{9}$$

$$\gamma_a^k \geq 0 \qquad \forall k \in \mathcal{K}, \forall a \in p^k \tag{10}$$

$$\delta_a^{\{k_1,k_2\}} \geq 0 \qquad \forall a \in \mathcal{A}, \forall I \in \mathcal{T}_a, \forall \{k_1,k_2\} \subseteq \bar{\kappa}(a,I) \tag{11}$$

The variables γ_a^k represent the dispatch time t_a^k in the resulting solution $S = (P, T)$ to SND-RR. The variables $\delta_a^{\{k_1,k_2\}}$ represent the difference in dispatch times between commodities that are consolidated in the relaxed solution. Constraints (5)–(7) ensure that the dispatch times are consistent and the remaining constraints enforce the described interpretation of the δ-variables. Since the flat-paths in the relaxation solution are k-feasible, this linear program is always feasible. Note that if the flat-solution corresponding to W is implementable, this primal heuristic will find dispatch times T so that the objective value of the linear program is zero. If W was also optimal for R-SND-RR($\mathcal{T}$), then $S = (P, T)$ will also be optimal for SND-RR.

B Reproducing Existing Relaxations

We compare our relaxation R-SND-RR($\mathcal{T}$) to existing relaxation approaches from the literature. Specifically, we demonstrate that for both a node-interval-based and an arc-dependent discretization, we can specify an arc-based discretization that results in an identical mixed-integer program. For these reproductions, we specify both arc and commodity-node discretizations explicitly to preserve the original time-expanded networks. The node-time relaxation of Boland et al. (2017) is less flexible than the node-interval-based one and also includes less pre-processing of unnecessary timed nodes and arcs. We therefore omit a dedicated analysis of their approach.

B.1 Reproducing the Node-Interval-Based Relaxation

Marshall et al. (2021) introduce a relaxation based on an interval-based discretization of time for each node. They use this discretization to construct relaxed time-expanded networks for each commodity. Based on these networks, they construct a MIP to solve their relaxed problem. Given specific networks, their MIP is equivalent to ours. So we will demonstrate that for any interval-based discretization we can construct an arc-based discretization that results in equivalent networks and thus MIPs.

We adapt their notation to our restricted-route variant of the CTSNDP. Without loss of generality, assume that $\min_{k\in\mathcal{K}} r^k = 1$ and let $H = \max_{k\in\mathcal{K}} \ell^k$ denote the end of the time horizon, i.e., all transportation occurs inside the time horizon $[H] = \{1, \dots, H\}$. They define a time discretization $\bar{\mathcal{T}} = (\bar{\mathcal{T}}_v)_{v\in\mathcal{V}}$ that contains for each node a partition of the time horizon $[H]$ into intervals $\bar{\mathcal{T}}_v = \{[t_1^v, t_2^v), [t_2^v, t_3^v), \dots, [t_{n_v}^v, t_{n_v+1}^v)\}$ with $t_{n_v+1}^v = H + 1$. Based on this discretization, they construct relaxed time-expanded networks $\bar{G}_{\bar{\mathcal{T}}}^k = (\bar{V}_{\bar{\mathcal{T}}}^k, \bar{A}_{\bar{\mathcal{T}}}^k)$ for each commodity k. Each node receives timed copies for each interval in its discretization, i.e., $\bar{V}_{\bar{\mathcal{T}}}^k = \{(v, I) \mid v \in \mathcal{V}^k, I \in \bar{\mathcal{T}}_v\}$. For ease of this comparison, we preprocess this set of nodes and remove nodes $(v, [t, t'))$ for which either $t' \le \underline{t}_{kv}$ or $t > \bar{t}_{kv}$. These nodes represent time intervals not reachable by commodity k. Note that this reduction is strictly in their favor: these nodes would never be part of a feasible solution and corresponding variables would trivially be removed by the presolver. Then, they introduce timed arcs $\bar{A}_{\bar{\mathcal{T}}}^k = \bar{A}_{\bar{\mathcal{T}}\mathrm{t}}^k \cup \bar{A}_{\bar{\mathcal{T}}\mathrm{h}}^k$ consisting of a set of timed transport arcs and holding arcs. The holding arcs $\bar{A}_{\bar{\mathcal{T}}\mathrm{h}}^k$ connect timed copies representing the same nodes at successive intervals. The timed arcs are given by

$$\bar{A}_{\bar{\mathcal{T}}\mathrm{t}}^k = \{((v, I), (v', I')) \mid (v, v') \in \mathcal{A}^k, I \in \bar{\mathcal{T}}_v, I' \in \bar{\mathcal{T}}_{v'}, \\ \exists t \in I \cap [\underline{t}_{kv}, \bar{t}_{kv'} - \tau_{(v,v')}] \colon t + \tau_{(v,v')} \in I'\}.$$

So for each arc (v, v'), we connect an interval I at the tail node v with an interval I' at the head node v' if and only if there is a legal dispatch time in I such that we still arrive inside interval I'.

Furthermore, Marshall et al. (2021) do one more preprocessing step: if two timed copies $((v, I), (v', I'))$ and $((v, I), (v', I''))$ of the same arc leave from the same interval I, with $I' < I''$, and the earlier copy can transport every commodity eligible for the later copy, they remove the copy leading to the later interval.

We now demonstrate how to construct an arc-based discretization that leads to exactly the same relaxed time-expanded network (up to re-labeling nodes and arcs). For the node discretizations we copy (up to clamping) the intervals from $\bar{\mathcal{T}}_v$, i.e., $\mathcal{T}_{kv} = \{I \cap [\underline{t}_{kv}, \bar{t}_{kv}] \mid I \in \bar{\mathcal{T}}_v\} \setminus \{\emptyset\}$. Evidently, each node $(v, I) \in \bar{V}^k$ then corresponds to exactly one node $(v, I \cap [\underline{t}_{kv}, \bar{t}_{kv}]) \in V_{\mathcal{T}}^k$ in our time-expanded network. Therefore, we also have the same set of holding arcs in both networks. Next, we want to choose $\mathcal{T}_a$ so that the transport arcs correspond one-to-one. For each commodity $k \in \mathcal{K}$ and each arc $((v, I), (v', I')) \in \bar{A}_{\bar{\mathcal{T}}\mathrm{t}}^k$ with $I = [t_1, t_2)$, $I' = [t_1', t_2')$, and $a = (v, v')$, we require that there is an interval $I_a \in \mathcal{T}_a$ starting at $\beta = \max\{\underline{t}_a, t_1, t_1' - \tau_a\}$.

We use exactly these starts and the final endpoint $\bar{t}_a + 1$ to partition the arc time horizon.

Observe that no start of an interval in $\mathcal{T}_a$ then lies strictly between β and $\min\{t_2, t_2' - \tau_a, \bar{t}_a + 1\}$: every internal boundary of $\mathcal{T}_a$ is the start of an interval of $\bar{\mathcal{T}}_v$ (when t_1 attains the maximum) or the start of an interval of $\bar{\mathcal{T}}_{v'}$ shifted back by τ_a (when $t_1' - \tau_a$ does). A boundary γ with $\beta < \gamma < \min\{t_2, t_2' - \tau_a, \bar{t}_a + 1\}$ would therefore place the start of an interval of $\bar{\mathcal{T}}_v$ strictly inside I (if $\gamma = s_1$, then $t_1 \le \beta < s_1 < t_2$) or the start of an interval of $\bar{\mathcal{T}}_{v'}$ strictly inside I' (if $\gamma = s_1' - \tau_a$, then $t_1' < s_1' < t_2'$), contradicting that $\bar{\mathcal{T}}_v$ and $\bar{\mathcal{T}}_{v'}$ are partitions.

We now show that, for each commodity k, the transport arcs of $G_{\mathcal{T}}^k$ and $\bar{G}_{\bar{\mathcal{T}}}^k$ are in one-to-one correspondence. First, each arc $((v, I), (v', I')) \in \bar{A}_{\bar{\mathcal{T}}\mathrm{t}}^k$ induces a feasible interval $I_a \in \mathcal{T}_a$ for k: there exists a dispatch time $t \in I \cap [\underline{t}_{kv}, \bar{t}_{kv'} - \tau_a]$ with $t + \tau_a \in I'$ that satisfies $\beta \le t < \min\{t_2, t_2' - \tau_a, \bar{t}_a + 1\}$. So by the previous observation it lies in I_a, and being a legal dispatch time it makes I_a feasible. Conversely, let $I_a \in \mathcal{T}_a$ be feasible for k with start β, let t^* be the earliest legal dispatch time of k in I_a, and let $I \in \bar{\mathcal{T}}_v$, $I' \in \bar{\mathcal{T}}_{v'}$ contain t^* and $t^* + \tau_a$. Then t^* is a time point that would fulfill the check to create $((v, I), (v', I')) \in \bar{A}_{\bar{\mathcal{T}}\mathrm{t}}^k$, which induces the boundary $\beta^* = \max\{\underline{t}_a, t_1, t_1' - \tau_a\} \le t^*$. By the previous observation, t^* lies in the $\mathcal{T}_a$-interval starting at β^*, and since $t^* \in I_a$ and I_a starts at β we get $\beta^* = \beta$, so this arc induces exactly I_a.

Lastly, we need to demonstrate that the timed arc copy (a, I_a) connects the two nodes $(v, I \cap [\underline{t}_{kv}, \bar{t}_{kv}])$ and $(v', I' \cap [\underline{t}_{kv'}, \bar{t}_{kv'}])$. First, for the tail node v, consider the relaxed dispatch interval $I^-(k, a, I_a)$. By construction, $I_v = I \cap [\underline{t}_{kv}, \bar{t}_{kv}] = [t, t')$ overlaps I_a. If there is a next-higher interval $I_v' = [t', t'')$, then

one of two cases holds. In the first case, we can still dispatch on this arc in I'_v. Then, by construction of the discretization, t' is a boundary of $\bar{\mathcal{T}}_v$ and must also be the start of the next interval on arc a. In the second case, the next node interval is too late, i.e., $t' > \bar{t}_{kv'} - \tau_a$. In either case, I_v is the relaxed dispatch interval for arc interval I_a (by Definition 4). Applying the same argument and Definition 5, $I_{v'} = I' \cap [\underline{t}_{kv'}, \bar{t}_{kv'}]$ is the relaxed arrival interval.

Overall, we find that $G^k_{\mathcal{T}}$ is equivalent (up to re-labeling) to $\bar{G}^k_{\bar{\mathcal{T}}}$. So it is always possible to reproduce the node-interval-based relaxation with the arc-based relaxation.

B.2 Reproducing the Arc-Dependent Relaxation

The original construction of Van Dyk and Koenemann (2024) built upon the node-time-based approach. We present their construction instead applied to the more general node-interval-based approach. This allows us to make a stronger claim about the generality of the proposed approach, since it generalizes not only the interval-based and arc-dependent approaches individually, but also if they are combined.

Their idea is as follows: for each node $v \in \mathcal{V}$, determine a partition P_v of the set $\delta^+(v)$ such that for all $k \in \mathcal{K}$ it holds that $\delta^+_k(v) \subseteq p$ for one $p \in P_v$. They also provide an algorithm to determine the unique finest possible partition. Additionally, they insert $p_0 = \emptyset$ into P_v for all v, representing commodities that terminate at v and thus have no outgoing arcs.

Then, a discretization $\bar{\mathcal{T}}$ is specified by providing for each node $v \in \mathcal{V}$ and each part of its partition $p \in P_v$ a partition of the time horizon $[H]$ into intervals $\bar{\mathcal{T}}_{vp} = \{[t_1^{vp}, t_2^{vp}), \dots\}$. For each commodity k and node $v \in \mathcal{V}^k$, let $p(k,v) \in P_v$ denote the unique part of the arc-partition for which $\delta^+_k(v) \subseteq p(k,v)$ (and $p(k,v) = \emptyset$ if $\delta^+_k(v) = \emptyset$).

Then, the relaxed time-expanded networks $\bar{G}^k_{\bar{\mathcal{T}}} = (\bar{V}^k_{\bar{\mathcal{T}}}, \bar{A}^k_{\bar{\mathcal{T}}})$ are constructed very similarly to the previous section. The sets of nodes $\bar{V}^k_{\bar{\mathcal{T}}}$ and holding arcs $\bar{A}^k_{\bar{\mathcal{T}}\,\mathrm{h}}$ are constructed exactly as before, except that we use $\bar{\mathcal{T}}_{vp(k,v)}$ in place of $\mathcal{T}_v$. The set of transport arcs is constructed a bit differently as

$$\bar{A}^k_{\bar{\mathcal{T}}\mathrm{t}} = \{((v,v'), I) \mid (v,v') \in \mathcal{A}^k, I \in \bar{\mathcal{T}}_{vp(k,v)} \colon I \text{ overlaps } [\underline{t}_{kv}, \bar{t}_{kv'} - \tau_{(v,v')}]\}$$

by specifying just the flat-arc and interval at the tail node. For an arc $\hat{a} = ((v,v'), I)$ in commodity k's network, we specify $\mathrm{tail}(\hat{a}) = (v, I)$ and $\mathrm{head}(\hat{a}) = (v', I')$ with $I' = \min\{I'' \in \bar{\mathcal{T}}_{v'p(k,v')} \mid \exists t \in I \cap [\underline{t}_{kv}, \bar{t}_{kv}] \colon t + \tau_{(v,v')} \in I''\}$. This especially means that there is only (at most) one timed copy of an arc for each interval at the tail node. Contrast this with the node-interval-based approach, where there can be multiple timed copies of the same arc emanating from the same interval. This tail- and head-node dependent creation of timed arc copies is not compatible with the arc-dependent approach, because two commodities k, k' dispatching from v to v' in the interval I may arrive in two different intervals at v', if $p(k,v') \neq p(k',v')$. Then it would be unclear on which timed arcs which commodities should be consolidated.

As before, we can now construct an arc-based discretization $\mathcal{T}$ that reproduces (up to re-labeling) the networks $\bar{G}^k_{\bar{\mathcal{T}}}$. We select for all $k \in \mathcal{K}$ and $v \in \mathcal{V}^k$ the discretization $\mathcal{T}_{kv} = \{I \cap [\underline{t}_{kv}, \bar{t}_{kv}] \mid I \in \bar{\mathcal{T}}_{vp(k,v)}\} \setminus \{\emptyset\}$ by clamping the intervals of each node as before. Then, the mapping between nodes and holding arcs works as for the node-interval case before. For the arc intervals, we simply set $\mathcal{T}_a = \{I \cap [\underline{t}_a, \bar{t}_a] \mid I \in \bar{\mathcal{T}}_{vp}\} \setminus \{\emptyset\}$ with $p \ni a$ for each $a = (v,v') \in \mathcal{A}$, i.e., we use the discretization of the corresponding partition, clamped to the arc time horizon. Evidently, we again have a one-to-one correspondence between arcs in $\bar{G}^k_{\bar{\mathcal{T}}}$ and in $G^k_{\mathcal{T}}$.

C Details on Newly Generated Instances

For all instances we set the vehicle capacity on all arcs to $u_a = 1$. All commodities have randomly drawn source and sink nodes and their quantity q^k is drawn uniformly from the real interval $[0.01, 0.5]$ and

rounded to 2 decimal places. This ensures that the vehicle capacities are relatively large compared to most commodities' quantities, making consolidation attractive. The time windows of each commodity were determined as described in Boland et al. (2017), choosing parameter values that lead to more flexible time windows. For each commodity k, we determine the length of the shortest path with respect to the travel time (described below for each class) from its origin to its destination $\tau^k_{o^k d^k}$. Let $L = \max\{\tau^k_{o^k d^k} \mid k \in \mathcal{K}\}$ be the longest minimal travel time. Then, we draw the release time r^k for each commodity k from a normal distribution with mean L and standard deviation $\frac{L}{3}$. Then we draw its flexibility F^k from a normal distribution with mean $\frac{L}{2}$ and standard deviation $\frac{L}{12}$. Its deadline is set to $\ell^k = r^k + \tau^k_{o^k d^k} + F^k$. Both draws are resampled if below zero and rounded down to the next integer. The fixed costs f_a for dispatching on each arc are determined based on the travel time on that arc, also described below. For the variable costs, we set $c^k_a = \lfloor f_a \cdot q^k \rfloor$.

Each commodity's subnetwork is restricted to arcs and nodes that can be part of a time-feasible path for that commodity. Further restrictions that remove arcs that can trivially not occur in optimal solutions are discussed below where relevant.

For each class, the network size and number of commodities were chosen based on the order of magnitude of other instances in the literature and adjusted until not all instances in the class were either trivial to solve or unsolvable within the one hour time limit.

C.1 Grid

We generated networks with a four by four grid of nodes and 48 arcs. Each arc has travel time and fixed cost $\tau_a = f_a = 10$. All instances in this class have 50 commodities.

C.2 Hub-and-Spoke (hard)

We generated networks with 5 hubs and 25 satellites. For each node we uniformly generated coordinates in the real square $[0, 100]^2 \subset \mathbb{R}^2$ and assigned each node to its closest hub by Euclidean distance, forming this hub's region. Each node has arcs to and from its hub, and we further generated arcs randomly between nodes of the same hub until $50\,\%$ of the possible arcs in this region were created. The hubs are fully interconnected. The travel time of each arc is the Euclidean distance between the positions of the two nodes rounded up plus one. The fixed cost of each arc is its length plus 10 to reflect some constant amount of effort for each dispatch. All instances in this class have 75 commodities. Each commodity's subnetwork consists only of the arcs in its origin region, destination region, and the arcs between hubs. We further remove arcs from a non-origin hub to the origin-hub and from the destination hub to any other hub. If origin and destination regions differ, we also remove arcs from the origin hub to a regional node, and from a regional node to the destination hub.

C.3 Star

We generated networks with one hub and 20 spoke nodes, each of which is connected with the hub. The two arcs between the hub and each spoke have the same travel time, uniformly drawn from the integer interval $[10, 100]$. Again the fixed cost of each arc is its length plus 10. All instances in this class have 200 commodities. Each commodity's subnetwork consists only of the one or two arcs on the direct path from its origin to its destination.

C.4 Wheel

We generated networks with one hub and 11 spokes that are distributed evenly around the hub. Each arc to or from the hub has a travel time of $\tau_a = 100$, while the arcs between the spokes have a travel time of

$\tau_a = \left\lfloor \frac{2\pi 100}{11} \right\rfloor = 57$, i.e., the time for traveling along a circular path around the hub from one spoke to the next, rounded down. Again the fixed cost of each arc is its length plus 10. All instances in this class have 100 commodities.

D Performance Comparison with Available Implementations

To demonstrate that our implementation of existing discretizations is of high quality, we compare it on identical hardware and with identical software (Python, Gurobi) to available implementations. We are very grateful to the authors for making their code and instances accessible, and will follow their example. As before, all instances were solved to a 1 % gap limit and 1 h time limit.

Marshall et al. (2021) provide an open-source implementation of their algorithm at https://github.com/mathgeekcoder/DDDI. We compare our base implementation `node-interval` against their "fewer-time-points" algorithm, which we name `MBSH-OG`. They both use the interval-based relaxation, but use different refinement strategies, and `MBSH-OG` does not add the travel time constraint $\sum_{\hat{a} \in A^k_{\mathcal{T}_t}} \tau_{\hat{a}} x^k_{\hat{a}} \leq \ell^k - r^k$. To enable a fair comparison, we also introduce our implementation `MBSH`, which uses our re-implementation of their refinement strategy and does not include the travel time constraint. Note that `MBSH-OG` has one additional advantage, in that during refinement it modifies the existing relaxation MIP in place, while our implementation rebuilds the model from scratch. Our experiments show that for reasonably hard instances, this overhead is negligible.

Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
`MBSH-OG`	552.38	15.0	0.54	1.66	90.8
`MBSH`	450.88	13.7	0.54	1.32	94.2
`node-interval`	260.78	5.4	0.54	1.61	97.5

Table 9: Results for node-interval-based algorithms on HC instances from Boland et al. (2017)

The computational results are summarized in Table 9. We observe that our reimplementation `MBSH` achieves very similar performance to their implementation `MBSH-OG`, but terminates somewhat faster and with fewer iterations. This may be due to minor implementation differences, since their paper did not specify their refinement scheme in full detail. In any case, our implementation of the interval-based discretization is evidently competitive. We also observe that `node-interval`, using the refinement method of Shu et al. (2025), is about twice as fast as `MBSH`.

Van Dyk and Koenemann (2024) also provide an open-source implementation of their algorithm at https://github.com/madisonvandyk/SND-RR, which we call `VDK-OG`. It uses the same relaxed problem as `partition`, but a different refinement algorithm, and does not add the strong linking inequalities. To enable a fair comparison, we also introduce our implementation `VDK`, which uses our re-implementation of their refinement strategy and does not add the strong linking inequalities. The computational results are summarized in Table 10. Here we observe differences between the reference `VDK-OG` and our implementation `VDK`. A major difference between the implementations is the amount of preprocessing on the time-expanded networks. By construction, our implementation does not add timed nodes and arcs that are outside the time window in which a commodity could traverse them in feasible solutions. But `VDK-OG` adds them, and commodities may actually traverse these in relaxation solutions. This explains why they require significantly more iterations and solving time, and why their network sizes at termination are actually *larger* than the size of a full discretization as we define it. In any case, our implementation of their relaxation problem (with additional preprocessing) is evidently quite competitive. Again we see that

Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
`partition`	17.95	5.3	13.82	30.33	100.0
`VDK`	140.34	13.9	13.82	32.09	99.5
`VDK-OG`	698.34	24.7	23.70	195.01	93.6

Table 10: Results for arc-dependent algorithms on instances from Van Dyk and Koenemann (2024)

switching from their refinement, which adds very few time points in each iteration, to that of Shu et al. (2025) in `partition`, significantly reduces iteration counts and solving times.

Overall, we observe that our implementations of the alternative relaxations are quite competitive and further improved by moving all approaches to the same refinement scheme. This demonstrates that `node-interval` and `partition` are fair baselines to compare our approach `arc` to.

E Results of Base Algorithms on Topology Instances

Table 11 and Table 12 contain the results when running the base algorithms on the instances with specific topologies. Similar to the results on the instances of Boland et al. (2017), the node-based approaches outperform the arc-based approach almost everywhere, which generally requires more iterations.

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
Des. path	`arc`	20.71	5.8	7.42	28.26	100.0
	`node-time`	23.51	4.4	16.95	32.22	100.0
	`node-interval`	18.28	4.4	7.42	31.40	100.0
	`partition`	18.65	5.6	9.98	28.90	100.0
Crit. time	`arc`	58.78	7.4	18.05	32.93	99.5
	`node-time`	18.91	4.6	23.28	38.12	100.0
	`node-interval`	11.83	3.9	18.05	38.94	100.0
	`partition`	18.74	5.0	18.05	37.12	100.0
H&S (easy)	`arc`	26.06	7.3	13.42	22.32	100.0
	`node-time`	15.83	4.5	18.59	27.08	100.0
	`node-interval`	11.77	4.3	13.42	25.57	100.0
	`partition`	16.45	5.3	13.42	24.98	100.0

Table 11: Results on instances from Van Dyk and Koenemann (2024), 1 % gap limit, 1 h time limit

Class	Algorithm	Time (s)	Iter.	Net. first (%)	Net. last (%)	Solved (%)
H&S (hard)	arc	3450.21	4.6	2.37	4.37	10.0
	node-time	2630.50	4.3	3.56	6.93	41.0
	node-interval	2454.43	4.1	2.37	7.50	46.0
	partition	2819.13	4.8	2.89	6.62	35.0
Grid	arc	2858.39	6.3	6.78	16.69	35.0
	node-time	2503.79	4.1	10.79	20.89	44.0
	node-interval	2426.03	4.2	6.78	22.47	48.0
	partition	2510.58	4.5	8.68	20.98	45.0
Wheel	arc	3395.64	6.2	1.87	4.70	17.0
	node-time	3092.11	4.4	4.01	6.10	29.0
	node-interval	3004.90	4.5	1.87	6.73	32.0
	partition	3236.62	4.0	3.06	6.58	22.0
Star	arc	42.96	7.1	1.07	3.03	100.0
	node-time	1412.23	4.6	3.79	4.85	80.0
	node-interval	673.87	3.5	1.07	4.50	90.0
	partition	46.79	7.5	1.78	3.08	100.0

Table 12: Results on new instances, 1 % gap limit, 1 h time limit

F Relaxation Size Example

We construct step by step an instance family with $\mathcal{O}(n)$ commodities for which the proposed arc-based relaxation needs only $\mathcal{O}(n)$ commodity-specific timed transport arcs to achieve the optimal value, while the node-interval relaxation requires $\Omega(n^2)$ such arcs. Thus, the ratio of the numbers of commodity-specific transport variables is unbounded at the same lower bound.

Then, we shall introduce additional commodities so that the size advantage is also possible versus the arc-dependent relaxation and previously proposed preprocessing rules.

Consider the network in Figure 12. Unless stated otherwise, all commodities have quantity $q^k = 1$; all arcs have capacity $u_a = 2$, fixed cost $f_a = 1$, flow cost $c_a^k = 0$, and travel time $\tau_a = 1$. Let D denote some sufficiently large integer. We denote commodities with letters to improve readability once multiple families of commodities are introduced.

The basic construction can be explained with the four commodities in Table 13. In an implementable solution commodity a can be consolidated with either b on arc (v_1, v_2) or with c on (v_2, v_3), while x cannot be consolidated with anyone. The optimal value is therefore four: three vehicles for a, b, c and one for x. To achieve a tight lower bound, any relaxation needs to prevent a from consolidating with both b and c. Evidently, an arc-based approach can do so without creating additional timed copies of the arc (v_2, v_1) for commodity x, while node-based approaches may be forced to do so. For example, in the arc-based approach it suffices to insert the arc time points 2 on (v_1, v_2) and 3 on (v_2, v_3). The resulting strict node discretization excludes the transfer for commodity a from arrival time 3 to departure time 2 at v_2. The node-interval-based approach must also separate times 2 and 3 at node v_2: otherwise, a can use the vehicle carrying b at time 2 and the vehicle carrying c at time 2, giving a relaxed value of three. This node time point also creates an additional transport arc copy for x. Note that this copy could be removed by pre-processing approaches from the literature, which we will prevent with upcoming modifications to the example.

We will build on this idea by introducing families of commodities a, b, and c that require some time points in the discretization and 'unrelated' commodities x that suffer the consequences of additional timed

Commodity k	$\mathcal{A}^k$	r^k	ℓ^k
a	$(v_1, v_2), (v_2, v_3)$	1	4
b	(v_1, v_2)	2	3
c	(v_2, v_3)	2	3
x	(v_2, v_1)	1	D

Table 13: Commodities illustrating the basic mechanism of relaxation size increase.

copies. We first replace the four commodities by the first four families in Table 14, where $n \geq 1$, $i \in [n]$, $T = n + 2$, $t_i = T + 4(i - 1)$, and $D = T + 4n + 4$. For each i, we have three commodities a_i, b_i, and c_i, forcing us to add time points to prevent a_i from consolidating with both of the others in the same solution, and again yielding additional transport arc copies for the x_i commodities.

Specifically, adding the arc time points $t_i, t_i + 1$ to (v_1, v_2) and $t_i + 1, t_i + 2$ to (v_2, v_3) for $i \in [n]$ suffices for a tight bound from the arc-based approach, with the commodity-node discretizations derived as in Section 4.2. The lower time point on each arc is added to prevent b_i from consolidating with b_{i+1} and so on, i.e., to separate the families of commodities, while the higher is necessary to prevent a_i from consolidating with both b_i and c_i. In the node-based approach, a discretization yielding a tight bound must include $t_i + 2$ at v_2 for each $i \in [n]$. Otherwise, the arrival of b_i at time $t_i + 2$ and the departure of c_i at time $t_i + 1$ happen in the same node interval, allowing a_i to consolidate with both. Since all these time points lie inside every x_j's departure window, they yield at least n transport arc copies for each of the n commodities x_j. This is the core argument to show that the node-based approach can require an arbitrarily larger time-expansion, since its size grows quadratically compared to the linear growth for arc-based.

Significant-point initialization as discussed in Section A.1 preserves this size comparison. No arc-specific significant points are needed on (v_2, v_1), since all x_i departure windows overlap, while a_i can individually be consolidated with b_i and c_i. The introduced arc time points already separate the incompatible departure windows of different i-periods.

Next, we extend the example to preserve the argument even if the preprocessing of Marshall et al. (2021) is applied. Let $\mathcal{K}_{\hat{a}}$ denote the set of commodities that can traverse timed arc $\hat{a}$. Recall, if two timed transport arcs $\hat{a}_1 = ((v, I), (v', I'))$ and $\hat{a}_2 = ((v, I), (v', I''))$ leave from the same node interval but with I' earlier than I'' and $\mathcal{K}_{\hat{a}_2} \subseteq \mathcal{K}_{\hat{a}_1}$, we may remove arc $\hat{a}_2$. It is also possible to invert the rule, i.e., of the two arcs $\hat{a}_1 = ((v, I'), (v', I))$ and $\hat{a}_2 = ((v, I''), (v', I))$ with I' earlier than I'' and $\mathcal{K}_{\hat{a}_1} \subseteq \mathcal{K}_{\hat{a}_2}$, the earlier arc $\hat{a}_1$ can be removed. This is evidently the case here. But we can simply introduce the three families of commodities a'_i, b'_i, and c'_i from Table 14. They do essentially the same as a_i, b_i, c_i, but can never be consolidated with any of them. For the arc-based approach, they again require a linear number of additional time points to obtain a tight bound. But for the node-based approach, they require n time points $t_i + 3$ at node v_1. Together with the time points at v_2, this partitions the time horizon into $n - 1$ bands such that timed-arc copies in different bands cannot share a starting or ending interval, independently of any other time points at these nodes. So at least $n - 1$ timed arc copies remain for each of the x_i even under these preprocessing rules.

The same example can be extended to the approach of Van Dyk and Koenemann (2024) by adding two additional commodities. For odd n, we extend each x_i to (v_2, v_1, v_3) and increase its deadline by one. We also add commodities Q, R with quantity $1/2$, release time 1, and deadline D. Commodity Q can use (v_2, v_3) or (v_2, v_1, v_3), and R can use (v_1, v_3) or (v_1, v_2, v_3). These alternatives force the outgoing arcs at both v_1 and v_2 into a single valid part of the partition. Since the x_i now pass through v_1, separate copies of the destination cannot avoid the boundaries imposed there. The added commodities fit into the spare capacity when the x_i travel together, so the argument forcing the boundaries remains unchanged.

These changes add only $\mathcal{O}(n)$ transport-arc copies to the arc-based relaxation, preserving the asymptotic argument.

Finally, none of the preprocessing rules removing time-infeasible arcs from commodities' subnetworks (from Marshall et al. 2021; Shu et al. 2025) are relevant here, since all of the x_i commodities can traverse (v_2, v_1) at any of the time points that this construction forces.

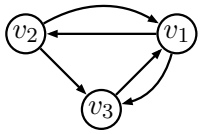


Figure 12: Network for relaxation size example.

Commodity k	$\mathcal{A}^k$	r^k	ℓ^k
a_i	$(v_1, v_2), (v_2, v_3)$	t_i	$t_i + 3$
b_i	(v_1, v_2)	$t_i + 1$	$t_i + 2$
c_i	(v_2, v_3)	$t_i + 1$	$t_i + 2$
x_i	(v_2, v_1)	i	$D + i$
a'_i	$(v_3, v_1), (v_1, v_2)$	$t_i + 1$	$t_i + 4$
b'_i	(v_3, v_1)	$t_i + 2$	$t_i + 3$
c'_i	(v_1, v_2)	$t_i + 2$	$t_i + 3$

Table 14: Commodity families for the shared-node comparison, with $i \in [n]$.

References

Boland, Natashia, Mike Hewitt, Luke Marshall, and Martin Savelsbergh (2017). "The Continuous-Time Service Network Design Problem". In: *Operations Research* 65.5, pp. 1303–1321.

— (2019). "The price of discretizing time: a study in service network design". In: *EURO Journal on Transportation and Logistics*. Special Issue: Advances in vehicle routing and logistics optimization: exact methods 8.2, pp. 195–216.

Crainic, Teodor Gabriel and Mike Hewitt (2021). "Service Network Design". In: *Network Design with Applications to Transportation and Logistics*. Ed. by Teodor Gabriel Crainic, Michel Gendreau, and Bernard Gendron. Cham: Springer International Publishing, pp. 347–382.

Gurobi Optimization, LLC (2025). *Gurobi Optimizer Reference Manual*.

Hewitt, Mike (2019). "Enhanced Dynamic Discretization Discovery for the Continuous Time Load Plan Design Problem". In: *Transportation Science* 53.6, pp. 1731–1750.

Lodi, Andrea and Andrea Tramontani (2014). "Performance Variability in Mixed-Integer Programming". In: *INFORMS TutORials in Operations Research*, pp. 1–12.

Marshall, Luke, Natashia Boland, Martin Savelsbergh, and Mike Hewitt (2021). "Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem". In: *Transportation Science* 55.1, pp. 29–51.

Moldenhauer, Carsten, Philipp Germann, Cedric Heimhofer, Caroline Spieckermann, and Andreas Andresen (2025). "Real-time Optimization of Transport Chains for Single Wagon Load Railway Transport". In: *Proceedings of the 11th International Conference on Railway Operations Modelling and Analysis (RailDresden 2025)*.

Shu, Shengnan, Zhou Xu, and Roberto Baldacci (2025). “New Dynamic Discretization Discovery Strategies for Continuous-Time Service Network Design”. In: *Optimization Online*, Preprint, https://optimization–online.org/?p=29090.

Van Dyk, Madison and Jochen Koenemann (2024). “Sparse dynamic discretization discovery via arc-dependent time discretizations”. In: *Computers & Operations Research* 169, p. 106715.