

OPTIMAL REFINEMENT IN DYNAMIC DISCRETISATION DISCOVERY FOR CONTINUOUS-TIME SERVICE NETWORK DESIGN

Helber, Alexander , Bachtler, Oliver 

Chair of Operations Research, RWTH Aachen University, Germany

helber@or.rwth-aachen.de | bachtler@or.rwth-aachen.de

Wüllner, Tom 

RWTH Aachen University, Germany

tom.wuellner@rwth-aachen.de

Abstract. Dynamic Discretisation Discovery (DDD) is a framework for solving a problem by iteratively solving and refining a relaxed version of the problem. DDD is the basis for state-of-the-art algorithms for various problems, including the continuous-time service network design problem (CTSNDP), which we study here. In the refinement step, such algorithms must identify *conflicts* in the relaxation solution that make it infeasible for the original problem and then apply *fixes* to the relaxation to prevent them from recurring. We study the *optimal refinement problem* of selecting from a set of fixes a subset that fixes all conflicts and yields a relaxation of minimum size, with the aim of making it easier to solve.

We demonstrate that while the number of conflicts may be exponential in the instance parameters, we can represent them more compactly, using only pseudo-polynomial space and time. Based on our more compact representation, we develop exact methods for solving the refinement problem. The best method allows us to find small relaxations in negligible amounts of time, more than 100 times faster than a natural integer programming formulation for some instance classes. Using this refinement approach in a DDD algorithm for the CTSNDP, we observe reductions in model sizes and running times of about 10 % to 20 % compared to existing approaches.

1. INTRODUCTION

Service network design problems arise in the planning of transport systems, especially for consolidation-based carriers [CGG21]. These carriers transport shipments that are usually much smaller than the capacity of the vehicles used, so economical operations rely on consolidating shipments. Therefore, the routing of commodities and the dispatches of vehicles transporting them must be determined jointly. We study the continuous-time service network design problem [Bol+17], in which dispatch times must also be determined, while ensuring that all commodities arrive by their respective deadlines.

Current state-of-the-art approaches [Hel26, Mar+21, SXB25] for this problem are all based on the dynamic discretisation discovery (DDD) framework first introduced by Boland

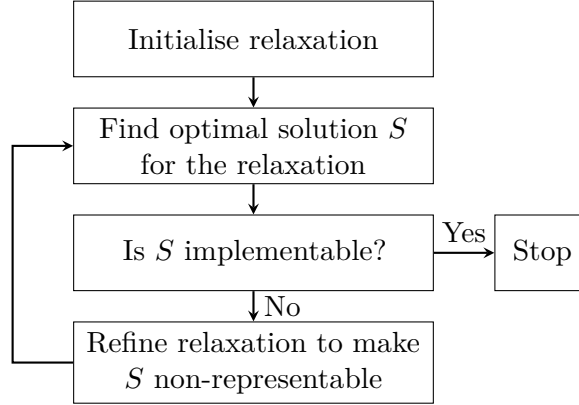


Figure 1: Overview of the dynamic discretisation discovery approach.

et al. [Bol+17]. So far, DDD has been mostly applied to problems that can be modelled on time-expanded networks. We note that the DDD concept is more general and can be applied to other expanded graphs. For example, Riedler, Ruthmair, and Raidl [RRR19] describe its application to the rooted distance-constrained minimum spanning tree problem. In the remainder of the introduction, we will focus on time-based problems, but the general ideas are relevant for any problem amenable to DDD.

The DDD framework is based upon the idea of constructing a relaxed version of the problem to be solved, also called a *relaxation*. This relaxed problem is formulated on a time-expanded network based on a given discretisation of time (for the nodes/arcs of the network that is being expanded). If the discretisation is sufficiently sparse, the relaxed problem may be solved much faster than the original problem, yet its optimal solutions provide good dual bounds on the value of the original problem.

We say a solution that is feasible for the relaxed problem is *representable* with respect to the chosen discretisation [Mar+21] and call feasible solutions to the original problem *implementable*. In the framework, an optimal solution to the relaxation is computed and, if it is implementable (or can be converted into an implementable solution of the same value), we can terminate with a provably optimal solution. Otherwise, we refine the discretisation (and thus the relaxation) in such a way that the non-implementable solution also becomes non-representable and cannot recur, guaranteeing eventual convergence. This process is also outlined in Figure 1.

In this work, we mostly concern ourselves with the refinement step, thus it is interesting to reflect on existing approaches. When designing a DDD algorithm, we require some concept of *conflicts* that prevent a representable solution from being implementable [BS19, Mar+21]. Then, we would like to have one or more *fixes* available for each conflict; if a fix is applied to the discretisation, the solution becomes non-representable. A fix usually consists of adding one or more time points to the discretisation. It may be possible to describe multiple types of conflicts, and for each conflict type multiple types of fixes.

While it suffices to fix a single conflict in each iteration to guarantee convergence, it has

been observed that fixing many conflicts at once can decrease the number of iterations required. But, a fix may increase the size of the relaxed problem, making it harder to solve. So to design an efficient DDD algorithm, one has to decide which (types of) conflicts to resolve and with which fixes, balancing relaxation difficulty against the number of iterations. This trade-off has been observed in many studies where different refinement strategies are compared [BS19, He+22a, He+22b, Mar+21, RRR19, Sch+20]. It is also possible to fix some conflicts by adding valid inequalities to the relaxed problem instead of changing the discretisation [BS19, Hew19], but this leads to essentially the same trade-off.

For various problem classes, Marshall et al. [Mar+21] and Riedler, Ruthmair, and Raidl [RRR19] find that it generally pays off to fix (almost all) observed problems, but using fixes that increase discretisation size as little as possible. To this end, Marshall et al. [Mar+21] also identify conflicts that lie ‘upstream’ of other conflicts and prefer to fix these first, in the hope that this also fixes the downstream conflicts.

The hope that an observed conflict may not recur if other conflicts are fixed is also reflected in the idea of removing time points from the discretisation [Hew19, RRR19, Sch+20]. We also observed in preliminary experiments that, once DDD converges, quite a substantial number (often more than 30 %) of time points can be removed from the discretisation before the bound quality changes. Yet it is unclear which time points are safe to remove; the removal of a necessary time point may cause another iteration of the algorithm. In fact, Van Dyk [Van24] proves that a naive strategy that simply removes time points that are not in the support of the solution can lead to cycling, preventing the algorithm from converging.

To summarise, we have observed the following: there is substantial freedom in which conflicts to fix (and how), the chosen fixes influence the difficulty of solving the relaxed problem and the number of DDD iterations, many added time points may not be required in later iterations, but removing time points naively can prevent algorithm convergence. These observations motivate us to ask the following question:

Can we find a minimum discretisation that fixes all observed conflicts?

Finding such a discretisation in each iteration would directly address the last concern: the new discretisation would not necessarily contain the time points of the previous one, effectively allowing us to remove time points. But since all previously observed conflicts remain fixed, no cycling can occur. Since the discretisation is small (with respect to some so far undefined size measure), we may hope for relaxations that are easier to solve.

This approach would also make full use of situations where many different fixes are available for each conflict, selecting a convenient subset of them. Note that for the scope of this study, we set aside the question of which conflicts to fix and commit to fixing all of them. We leave the selection of a good subset of conflicts to fix (and keep fixed) to future work.

To the best of our knowledge, the literature that explicitly optimises the set of fixes to be applied is limited to the following works.

Lieshout and Schaft [LS26] solve a vehicle scheduling problem with trip shifting using DDD. In their approach, a (representable) solution consists of a flow over a time-expanded network that is then decomposed into duties for each vehicle. If a duty cannot be implemented, the relaxation is refined to ensure that it cannot occur again. Since they know how many time points each duty would add to the discretisation, they pose an auxiliary problem that selects a duty decomposition leading to the fewest added time points. We note that the additional time points they add to prohibit a single non-implementable duty are sufficient, but not necessary to achieve this goal. In other words, there may be more fixes available than the ones they utilise. They also do not account for double-counting of time points that might be added by multiple duties. In their computational experiments, they observe little change in running time from finding such smaller discretisations. The approach is outperformed by a method that simply refines all non-implementable duties possible in the relaxation solution, leading to marginally larger networks but fewer iterations and shorter running times.

For the service network design problem studied in this work, Shu, Xu, and Baldacci [SXB25] proposed a conflict concept that subsumes the concepts introduced by Marshall et al. [Mar+21]. Notably, they introduce a sufficient characterisation of a discretisation that prevents a conflict from recurring. This characterisation implies that one conflict may permit multiple fixes. Based on this observation, the authors introduce a greedy heuristic to construct a discretisation that fixes all observed conflicts. In this work, we introduce a closely related but simpler sufficient criterion for such fixes, and use it to develop various exact methods for solving the *optimal refinement problem* of finding a minimum discretisation that selects one of these fixes for each conflict. In our short paper Wüllner and Helber [WH26] (which the current work extends) we already demonstrated that this problem is NP-hard and that being able to solve it efficiently can be beneficial for the overall DDD algorithm.

We note that shortly before finishing this work, we became aware of the new version [SXB26] of the previously mentioned preprint [SXB25]. In the updated version, the authors also explicitly pose the optimal refinement problem, using a slightly stronger characterisation than ours. They further propose a matheuristic to solve the problem. We discuss the similarities and differences between the approaches where appropriate throughout the paper.

Our contributions in this paper are:

- We provide a focused exposition of the main concepts for refinement introduced by Marshall et al. [Mar+21] and Shu, Xu, and Baldacci [SXB25, SXB26], with small extensions to theoretical results.
- We describe situations in which the approach of Shu, Xu, and Baldacci [SXB26] leads to a conflict set whose size is exponential in some instance parameters.
- We develop a more compact graph-based representation of the conflict set, whose

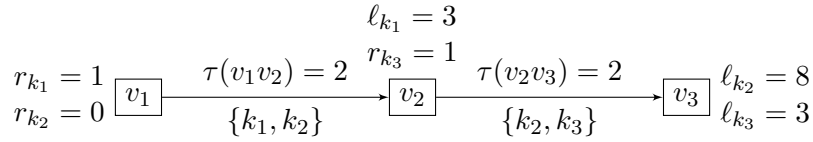


Figure 2: Example of an SND instance with three commodities $\mathcal{K} = \{k_1, k_2, k_3\}$.

size is pseudo-polynomial in the instance parameters and much smaller in practice.

- This representation allows us to restate the optimal refinement problem and develop a mixed-integer programming formulation as well as a DDD approach to solve it.
- We conduct computational experiments to demonstrate the performance of our methods and investigate the impact this refinement approach has on the overall DDD algorithm.

2. THE CONTINUOUS-TIME SERVICE NETWORK DESIGN PROBLEM

Before we introduce the continuous-time service network design problem, we want to briefly collect typical notation used throughout the paper. Here, all our intervals contain only integer values. Thus, when we write $[m, n]$, for $m, n \in \mathbb{N}$ with $m \leq n$, we refer to the set $\{m, \dots, n\}$. When $m = 0$, we also just write $[n]$ instead of $[0, n]$ and we also use this notation for open and half-open intervals such as $(m, n] = \{m + 1, \dots, n\}$ and $(n) = \{1, \dots, n - 1\}$.

We write VD and AD for the node and arc sets of a *directed graph* D . For a node v , we denote its predecessors and successors $N^{\text{in}} v, N^{\text{out}} v$ and its in- and outgoing arcs $A^{\text{in}} v, A^{\text{out}} v$. A *walk* W in D is a sequence $v_0 a_1 v_1 \dots a_n v_n$ of nodes and arcs such that $a_i = v_{i-1} v_i$. Depending on the context, we just denote W by $a_1 \dots a_n$ or $v_0 \dots v_n$. We also write $v_i W v_j$ for the sub-walk $v_i \dots v_j$ of W , where $v_i W = v_i W v_n$ and $W v_j = v_0 W v_j$. If W repeats no nodes, we call it a *path*. We also use the typical notation $\tau(W) = \sum_{i=1}^n \tau(a_i)$ for an arc length $\tau: AD \rightarrow \mathbb{Z}_{>0}$. Additionally, we write $d_\tau(v, v')$ for the length of a shortest path from v to v' with respect to τ .

Now let us define the continuous-time service network design problem (SND). We are given a network D (a directed graph) with nodes VD and arcs AD , through which a set of commodities $\mathcal{K}$ should be transported. Each commodity k is associated with a release time $r_k \in \mathbb{N}$ and a deadline $\ell_k \in \mathbb{N}$, and each arc a with a travel time $\tau(a) \in \mathbb{N}_{>0}$. For each commodity, we need a path P_k in D from its origin node o_k to its destination node d_k . A small example can be seen in Figure 2, where we specify the release time r_k at the origin o_k and deadline ℓ_k at the destination node d_k of commodity k . Note that the paths P_k are unique in this example.

Furthermore, we want to decide for every arc of P_k with which other commodities k is consolidated on that arc, that is, which commodities traverse the arc together. If we want to say that the commodities $\kappa \subseteq \mathcal{K}$ are consolidated on arc a , we combine them in a tuple $c = (a, \kappa)$ and call c a *consolidation*. In the example, we have tried to consolidate

everything that could potentially be consolidated, leading to the two consolidations $c_1 = (v_1v_2, \{k_1, k_2\})$ and $c_2 = (v_2v_3, \{k_2, k_3\})$.

Hence, we are effectively looking for a set of consolidations $\mathcal{C} \subseteq AD \times 2^{\mathcal{K}}$ such that each commodity appears in exactly one consolidation for each arc in its path, and does not appear in any other consolidations. A *solution* $S = (\mathcal{P}, \mathcal{C})$ now consists of paths $\mathcal{P} = (P_k)_{k \in \mathcal{K}}$ for each commodity and corresponding consolidations $\mathcal{C}$.

Next, we specify which solutions are feasible. We now require consistent dispatch times for each consolidation, such that all commodities are transported within their respective time windows. Denote the unique consolidation of the arc $a \in P_k$ containing k by $c(a, k)$. Moreover, for a consolidation $c = (a, \kappa)$ we write $a(c) = a$ and $\kappa(c) = \kappa$. We say the solution S is *feasible* for SND or *implementable* if dispatch times $t: \mathcal{C} \rightarrow \mathbb{N}$ exist such that, for all $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$ and $c_i = c(a_i, k)$ for $i \in [n]$,

- (a) $t(c_1) \geq r_k$,
- (b) $t(c_{i+1}) \geq t(c_i) + \tau(a_i)$ for all $i \in [n]$, and
- (c) $t(c_n) + \tau(a_n) \leq \ell_k$.

The first condition ensures that a commodity is not dispatched before it is released, the second that it cannot be dispatched from an intermediate node before arriving there, and the third that it arrives by its deadline. Here we see that our specified solution is not implementable, since consolidating k_2 and k_3 on the second arc requires $t(c_2) \geq r_{k_2} + \tau(v_1v_2) = 2$ but $t(c_2) + \tau(v_2v_3) \leq \ell_{k_3} = 3$, which is impossible. By replacing c_2 by the singleton ‘consolidations’ $(v_2v_3, \{k_2\})$ and $(v_2v_3, \{k_3\})$, we can obtain a feasible solution: dispatch c_1 at time 1, then k_2 on the second arc at time 3 and k_3 at time 1.

The objective is to minimise the total transportation cost, which consists of two parts. First, a commodity k incurs a flow cost of $c_k(a)$ for traversing arc a . Second, transporting the commodities in a consolidation (a, κ) requires a certain number of vehicles. Each commodity k requires a quantity q_k to be transported, and for each arc a we are given the capacity $u(a)$ of available vehicles and cost $f(a)$ for dispatching one vehicle on this arc. Overall, solving the continuous-time service network design problem (SND) consists of finding a feasible solution S minimising

$$c(S) = \sum_{k \in \mathcal{K}} \sum_{a \in P_k} c_k(a) + \sum_{(a, \kappa) \in \mathcal{C}} f(a) \left\lceil \frac{\sum_{k \in \kappa} q_k}{u(a)} \right\rceil.$$

For the remainder of this paper, we assume that the input data for the problem are given and refrain from stating it in all of our statements. The following derived instance parameters will also come in handy later:

2.1 Definition (Earliest and latest dispatch times). For $k \in \mathcal{K}$ and $v \in VD$, the *earliest dispatch time* of commodity k at node v is $r_k + d_\tau(o_k, v)$ and is denoted by $\underline{t}_k(v)$. The *latest dispatch time* of k at v , $\bar{t}_k(v)$, is $\ell_k - d_\tau(v, d_k)$.

We extend the notation to arcs $a = vv'$ by setting $\underline{t}_k(a) = \underline{t}_k(v)$ and $\bar{t}_k(a) = \bar{t}_k(v') - \tau(a)$.

Thus, the interval in which a commodity k can be dispatched on an arc a is specified by $I_k(a) = [\underline{t}_k(a), \bar{t}_k(a)]$. $\triangleleft$

We assume that the network is sufficiently connected for these values to be defined. Additionally, we assume without loss of generality that $\min_{k \in \mathcal{K}} r_k = 0$ and denote the time horizon $\max_{k \in \mathcal{K}} \ell_k + 1$ by H .

2.2 Observation. Let $S = (\mathcal{P}, \mathcal{C})$ be an implementable solution with dispatch times $t: \mathcal{C} \rightarrow \mathbb{N}$. Then, for $c = (a, \kappa) \in \mathcal{C}$ and $k \in \kappa$, $t(c) \in I_k(a)$. $\triangleleft$

Proof. Let $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$ and $c_i = c(a_i, k)$. Then $t(c_1) \geq r_k = \underline{t}_k(a_1)$ and, inductively, $t(c_{i+1}) \geq t(c_i) + \tau(a_i) \geq \underline{t}_k(a_i) + \tau(a_i) \geq \underline{t}_k(a_{i+1})$ for $i \in (n)$.

Similarly, $t(c_n) \leq \ell_k - \tau(a_n) = \bar{t}_k(a_n)$ and $t(c_i) \leq t(c_{i+1}) - \tau(a_i) \leq \bar{t}_k(a_{i+1}) - \tau(a_i) \leq \bar{t}_k(a_i)$ for $i \in (n)$ and the claim follows since for each $c = (a, \kappa) \in \mathcal{C}$ and $k \in \kappa$, $a \in P_k$. $\square$

Returning to our example, we see that $\underline{t}_{k_2}(v_1) = 0$, $\underline{t}_{k_2}(v_2) = 2$, and $\underline{t}_{k_2}(v_3) = 4$. Note that, if the network had shorter paths between these nodes, then these values would be lower, despite the solution using the longer paths. In future illustrations, we only specify the bounds of interest and assume any omitted values are sufficiently loose, as may result from parts of the network that are not depicted.

3. A DYNAMIC DISCRETISATION DISCOVERY ALGORITHM

SND can be modelled as a mixed-integer program on a time-expanded network [Bol+17], though this quickly becomes intractable for instances with finer time resolutions [Bol+19]. Two alternative formulations that do not rely on a time expansion have been proposed [HL23, SXQ23]. Both of them perform well on instances where commodities have only a small set of paths in the network to choose from. But in general, the state-of-the-art approach for solving SND is dynamic discretisation discovery with various later enhancements [e.g., Bol+17, Hel26, Hew19, Mar+21, SXB26]. In this section, we develop the algorithm outlined in Figure 1 for SND, which will form the basis for the rest of this work.

3.1. The Relaxed Problem

The centrepiece of a DDD algorithm is a relaxation of the problem, which should be significantly easier to solve. We utilise a simplified version of the arc-based relaxation proposed by Helber [Hel26], which is based on a discretisation of time for each arc in the network.

3.1 Definition. A *time discretisation* $\mathcal{T}$ consists of a set $\{0, H\} \subseteq \mathcal{T}_a \subseteq [H]$ of time points for each arc $a \in AD$. If $\mathcal{T}_a = \{t_0, \dots, t_n\}$ with $0 = t_0 < t_1 < \dots < t_n = H$,

then we define $\mathcal{T}_a^I = \{[t_0, t_1), [t_1, t_2), \dots, [t_{n-1}, t_n)\}$ as the intervals corresponding to the discretisation. $\triangleleft$

The relaxed problem R-SND is identical to SND, with the exception that it takes a time discretisation $\mathcal{T}$ as an additional input and the notion of feasibility changes. Specifically, instead of specifying the dispatch time for each consolidation, we only want to determine the time interval in which it is dispatched. Formally, a solution $S = (\mathcal{P}, \mathcal{C})$ is *feasible* for R-SND or $\mathcal{T}$ -representable if dispatch intervals $h: \mathcal{C} \rightarrow 2^{\mathbb{N}}$ exist with $h(c) \in \mathcal{T}_{a(c)}^I$ such that, for all $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$ and $c_i = c(a_i, k)$ for $i \in (n]$,

- (a) $h(c_i) \cap I_k(a_i) \neq \emptyset$ for all $i \in (n]$, and
- (b) $\min h(c_i) + \tau(a_i) \leq \max h(c_{i+1})$ for all $i \in (n)$.

The first condition ensures that a commodity can only be part of a consolidation c dispatched on arc a in an interval $h(c)$ that at least overlaps with the interval $I_k(a)$ in which the commodity itself can traverse a . The second condition ensures that consecutive consolidations of a commodity select consistent intervals.

If we recall our example in Figure 2, then we could use the time discretisation $\mathcal{T}_{v_1 v_2} = \{0, 1, H\}$ and $\mathcal{T}_{v_2 v_3} = \{0, H\}$. With this discretisation, the solution is representable since we can assign the intervals $[1, H)$ and $[0, H)$ to the two consolidations, which satisfies both conditions. This is an example of a discretisation that is too coarse to detect that the solution is non-implementable. If we insert the time point 3 into $\mathcal{T}_{v_2 v_3}$, then we must now select the interval $[0, 3)$ for the second consolidation in order to satisfy the first condition since $I_{k_3}(v_2 v_3) = [1, 1]$. The first consolidation must still use $[1, H)$ since $I_{k_1}(v_1 v_2) = [1, 1]$, so the second condition would require $1 + 2 \leq 2$, which is false. Thus, adding this time point is enough to ensure that the solution also does not appear when solving the relaxation.

For the following discussion, it does not matter how exactly R-SND is solved. But for completeness we give an integer program in Appendix A.

We do want to briefly note that the problem described here is, indeed, a relaxation.

3.2 Lemma. R-SND is a relaxation of SND. For a full discretisation, it is tight. $\triangleleft$

Proof. Let $S = (\mathcal{P}, \mathcal{C})$ be a solution and $\mathcal{T}$ be a time discretisation. If S is implementable, there exist dispatch times $t: \mathcal{C} \rightarrow \mathbb{N}$. For $c \in \mathcal{C}$, we let $h(c)$ be the interval in $\mathcal{T}_{a(c)}^I$ containing $t(c)$.

Let $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$ and $c_i = c(a_i, k)$ for $i \in (n]$. The first property holds since $t(c_i) \in h(c_i) \cap I_k(a_i)$ by Observation 2.2. Finally, since $t(c_{i+1}) \geq t(c_i) + \tau(a_i)$, we get $\min h(c_i) + \tau(a_i) \leq \max h(c_{i+1})$ as well. Thus, S is $\mathcal{T}$ -representable.

If $\mathcal{T}$ is the full discretisation and S is $\mathcal{T}$ -representable, then we get single element dispatch intervals $h(c)$ for $c = (a, \kappa) \in \mathcal{C}$ and we set $t(c)$ to the single element of $h(c)$. Since $h(c, k) := h(c) \cap I_k(a) \neq \emptyset$ for all $k \in \kappa$, $t(c)$ is actually equal to the element in $h(c, k)$ as well.

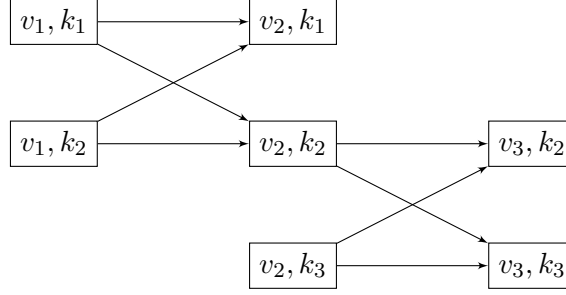


Figure 3: The dispatch-node graph for the example from Figure 2.

Again, let $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$ and $c_i = c(a_i, k)$ for $i \in [n]$. Then $t(c_1) = \min h(c_1, k) \geq \underline{t}_k(a_1) = \underline{t}_k(o_k) = r_k$, $t(c_{i+1}) = \max h(c_{i+1}) \geq \min h(c_i) + \tau(a_i) = t(c_i) + \tau(a_i)$, and $t(c_n) + \tau(a_n) = \max h(c_n, k) + \tau(a_n) \leq \bar{t}_k(a_n) + \tau(a_n) = \bar{t}_k(d_k) = \ell_k$. $\square$

3.2. Implementable and Representable Solutions

As we saw in the example, a solution S to R-SND for some given discretisation $\mathcal{T}$ need not be implementable. Recall that in this case we want to identify *conflicts*, which are some structures of the solution that stand in the way of implementability, and *fixes*, which are changes to the relaxation that prevent these conflicts from recurring. The conflicts we study are the so-called *too-long paths* in the *dispatch-node graph* introduced by Shu, Xu, and Baldacci [SXB26, Definition 1]. This graph contains nodes (v, k) for every node v in each commodity's path P_k , and arcs between two nodes (v, k) and (v', k') exactly when commodities k and k' are consolidated on arc vv' .

3.3 Definition (Dispatch-node graph). Given a solution $S = (\mathcal{P}, \mathcal{C})$, its *dispatch-node graph* D_S is given by

$$\begin{aligned} VD_S &= \{(v, k) : k \in \mathcal{K}, v \in P_k\}, \\ AD_S &= \{(v, k)(v', k') : (vv', \kappa) \in \mathcal{C}, k, k' \in \kappa\}. \end{aligned}$$

For $b = (v, k)(v', k') \in AD_S$, we write $a(b) = vv'$ for its underlying arc in D , and set $\tau(b) = \tau(a(b))$ and $\mathcal{T}_b = \mathcal{T}_{a(b)}$. For $u = (v, k) \in VD_S$, we set $\underline{t}(u) = \underline{t}_k(v)$ and $\bar{t}(u) = \bar{t}_k(v)$. $\triangleleft$

The dispatch-node graph for our running example from Figure 2 is shown in Figure 3.

Looking at the example, we observe the following simple but useful property of the neighbourhood of a node in D_S .

3.4 Observation. Let $u = (v, k) \in VD_S$. Then $v \in P_k$. Note that any arc $b = (v', k')(v, k) \in A^{\text{in}} u$ originates from a consolidation $c = (v'v, \kappa)$ with $k \in \kappa$. By the definition of a solution, $v'v \in P_k$. In particular, if $v = o_k$, then $N^{\text{in}} u = \emptyset$. Otherwise, let v' be the predecessor of v in P_k . Then $c = c(v'v, k)$ and $N^{\text{in}} u = \{(v', k') : k' \in \kappa\}$.

Similarly, $N^{\text{out}} u = \emptyset$ if $v = d_k$ and otherwise $N^{\text{out}} u = \{(v', k') : k' \in \kappa\}$, where v' is the successor of v in P_k and $c(vv', k) = (vv', \kappa)$. $\triangleleft$

As described by Shu, Xu, and Baldacci [SXB26, Section 7.2], a walk W from $u = (v, k)$ to $u' = (v', k')$ in D_S gives a lower bound $\underline{t}(u) + \tau(W)$ on the arrival time of k' at v' under the given sequence of consolidations. If this is later than the latest time $\bar{t}(u')$ at which k' can be dispatched from v' , this evidently contradicts implementability of S . In Figure 3, this happens for the path $(v_2, k_2)(v_3, k_3) = u_1 u_2$, which has $\underline{t}(u_1) = 2$ and $\bar{t}(u_2) = 3$, so its real arrival time is $2 + 2 = 4 > 3$. This motivates the following definition:

3.5 Definition (Real arrival time and too-long paths). For a walk $W = u_0 \dots u_n$ in D_S and $i \in [n]$, we define the real arrival time $t_W(i) = \underline{t}(u_0) + \tau(Wu_i)$ and call W a too-long path if $t_W(n) > \bar{t}(u_n)$. $\triangleleft$

3.6 Remark. This definition differs slightly from that of Shu, Xu, and Baldacci [SXB26, Definition 2]; they require that a too-long path starts at a commodity's origin, i.e., $u = (o_k, k)$ for some $k \in \mathcal{K}$. In most respects, both definitions yield essentially the same results, but the subsequent discussion is easier to follow with our definition, since it allows us to disregard various edge cases. We note that to stay consistent with their terminology, we say too-long path even though walks are also permitted. $\triangleleft$

Despite our slightly different definition of a too-long path, the following result transfers directly:

3.7 Theorem ([SXB26, Theorem 2]). A solution S is implementable if and only if the corresponding dispatch-node graph D_S contains no too-long path. $\triangleleft$

Proof. Let $S = (\mathcal{P}, \mathcal{C})$ be implementable with dispatch times $t: \mathcal{C} \rightarrow \mathbb{N}$. Moreover, let $W = u_0 \dots u_n$ be a walk in D_S with $u_i = (v_i, k_i)$, for $i \in [n]$ and $a_i = v_{i-1}v_i$, and $c_i = c(a_i, k_i)$ for $i \in [n]$.

Since, for $i \in (n)$, c_i and c_{i+1} are consolidations on consecutive arcs of P_{k_i} , we get $t(c_{i+1}) \geq t(c_i) + \tau(a_i)$ and, thus, $t(c_n) + \tau(a_n) \geq t(c_1) + \tau(W)$. By Observation 2.2, $t(c_1) \geq \underline{t}_{k_0}(a_1) = \underline{t}_{k_0}(v_0) = \underline{t}(u_0)$ and $t(c_n) \leq \bar{t}_{k_n}(a_n) = \bar{t}_{k_n}(v_n) - \tau(a_n) = \bar{t}(u_n) - \tau(a_n)$, so W is not too-long.

Conversely, assume that D_S has no too-long path. Then, in particular, D_S is acyclic. By Observation 3.4, the nodes without incoming arcs are exactly the nodes (o_k, k) for $k \in \mathcal{K}$, for which we define $t((o_k, k)) = r_k$. For the remaining nodes u , we set $t(u) = \max \{t(u') + \tau(u'u) : u' \in N^{\text{in}} u\}$, which is well-defined as D_S is acyclic. Finally, for $c = (vv', \kappa) \in \mathcal{C}$, we define $t(c) = \max \{t((v, k)) : k \in \kappa\}$.

We first note that, by Observation 3.4, $N^{\text{in}}((v, k)) = \{(v', k') : k' \in \kappa\}$, where v' is the predecessor of v in P_k and $c = c(v'v, k) = (v'v, \kappa)$. In particular, $t(u) = t(c) + \tau(v'v)$.

We now simply verify that t satisfies the desired properties. Let $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$, $c_i = c(a_i, k) = (a_i, \kappa_i)$, and $a_i = v_{i-1}v_i$ for $i \in [n]$. Then $t(c_1) \geq t((o_k, k)) = r_k$ and, for $i \in (n)$, $t(c_{i+1}) \geq t((v_i, k)) = t(c_i) + \tau(a_i)$.

Finally, we show that for each $u \in D_S$ there exists a $u'u$ -path W such that $t(u) \leq \underline{t}(u') + \tau(W)$. This implies the final property, since applied to $u = (v_n, k) = (d_k, k)$ we obtain $t(c_n) + \tau(a_n) = t(u) \leq \underline{t}(u') + \tau(W) \leq \bar{t}(u) = \ell_k$.

To see that such paths exist, we use the topological ordering we get from acyclicity. For the induction start, we know that $u = (o_k, k)$ for some $k \in \mathcal{K}$, for which $t(u) = r_k = \underline{t}(u)$, so we can use the trivial path $W = u$. In the induction step, we get that $t(u) = \underline{t}(u'') + \tau(u''u)$ for some $u'' \in N^{\text{in}} u$. By the induction hypothesis, we get a $u'u''$ -path W' satisfying $t(u'') \leq \underline{t}(u') + \tau(W')$. We extend this path by the arc $u''u$ to get a $u'u$ -path W satisfying $t(u) = t(u'') + \tau(u''u) \leq \underline{t}(u') + \tau(W)$ as required. $\square$

Note that the above construction directly gives a feasible timing if S is implementable. Otherwise, we have identified a too-long path in the dispatch-node graph, and now need to find a fix for it, that is, an adjustment of the discretisation that gets rid of it. More formally:

3.8 Definition. Let S be a solution, W be a walk in D_S , and $\mathcal{T}$ be a time discretisation. We say W is *eliminated* by $\mathcal{T}$ if there exists no $\mathcal{T}$ -representable solution S' whose dispatch-node graph $D_{S'}$ contains W . $\triangleleft$

In fact, since any too-long path makes a solution non-implementable, we want to eliminate *all* too-long paths in D_S . Luckily, we get many of them for free, thanks to the following observation [SXB26, Section 7.2].

3.9 Observation. If a walk W is eliminated by $\mathcal{T}$, then all extensions W' of W are also eliminated by $\mathcal{T}$. $\triangleleft$

Proof. If W' is not eliminated, there exists a solution S' that is $\mathcal{T}$ -representable such that W' appears in $D_{S'}$. But then W also appears in $D_{S'}$. $\square$

This motivates the following definition, similar to Definition 3 of [SXB26], but again not requiring too-long paths to start at a commodity's origin:

3.10 Definition (Minimal too-long path). A too-long path is *minimal* if none of its proper sub-walks are too-long. $\triangleleft$

3.11 Corollary. A solution S is implementable if and only if the corresponding dispatch-node graph D_S contains no minimal too-long path. $\triangleleft$

Hence, it is, in fact, sufficient to eliminate all minimal too-long paths. In our example from Figures 2 and 3, we already saw that $W = (v_2, k_2)(v_3, k_3)$ is too-long; it is the unique minimal too-long path. The walk $W' = (v_1, k_1)(v_2, k_2)(v_3, k_3)$ is also too-long and its real arrival time is $1 + 2 + 2 = 5 > 3$, but since it contains W as a subpath, it suffices to eliminate W .

To actually eliminate the minimal too-long paths, we characterise representability in terms of too-long paths as well and use this to determine a sufficient criterion for a too-long path W to be eliminated. Essentially, the idea is to check if the consolidations implied by W cause the last commodity to arrive too late even in the relaxation. Since the relaxation specifies intervals instead of time points and the consistency requirement just asks consecutive intervals to contain time points that would allow the transition, we always select the earliest possible interval.

Returning to our example in Figures 2 and 3 and using the same time discretisation $\mathcal{T}_{v_1v_2} = \{0, 1, H\}$ and $\mathcal{T}_{v_2v_3} = \{0, 3, H\}$ as before, the walk $W = (v_2, k_2)(v_3, k_3) = u_1u_2$ is unproblematic for the relaxation. We are available at time $\underline{t}(u_1) = 2$ at u_1 , so the interval $[0, 3)$ would be an option. Since the relaxation does not distinguish between the points in this interval, we can take the earliest point in the interval, namely its lower bound, letting us (fictitiously) arrive at u_2 at time 2, which is early enough to not conflict with $\bar{t}(u_2) = 3$. Indeed, W is not eliminated: splitting c_1 into singleton consolidations allows us to assign $[1, H)$ to k_1 and $[0, 1)$ to k_2 on the first arc, and $[0, 3)$ to c_2 , giving a representable solution that still contains W .

However, if we extend this path backwards by $u_0 = (v_1, k_1)$, then starting at $\underline{t}(u_0) = 1$ and using $[1, H)$ on v_1v_2 gives a relaxed arrival time of 3 at u_1 . This requires us to use the interval $[3, H)$ on v_2v_3 , so, even in the relaxed problem, we cannot arrive before time 5 at u_2 , which is too late since $\bar{t}(u_2) = 3$. The following definition formalises this computation of the relaxed earliest arrival time.

3.12 Definition (Relaxed arrival times and relaxed too-long paths). Let $W = u_0b_1u_1 \dots b_nu_n$ be a walk in D_S and $\mathcal{T}$ be a time discretisation. We define the *relaxed arrival time* $\pi_W(i)$ at the i th node of W by

$$\pi_W(i) = \begin{cases} \underline{t}(u_0) & \text{if } i = 0, \\ \lfloor \pi_W(i-1) \rfloor_{\mathcal{T}_{b_i}} + \tau(b_i) & \text{otherwise} \end{cases}$$

where $\lfloor t \rfloor_{\mathcal{T}_a} = \max \{t' \in \mathcal{T}_a : t' \leq t\}$. If $\pi_W(n) > \bar{t}(u_n)$, we say W is $\mathcal{T}$ -relaxed too-long. $\triangleleft$

3.13 Remark. In the short paper [WH26] we say a too-long path is eliminated if and only if it fulfils this criterion, which is not consistent with the terminology in this paper. The idea behind this criterion was already used by Helber [Hel26, Theorem 3] to check if too-long paths are already eliminated. A stronger variant of the criterion is also introduced in the newer version of the preprint Shu, Xu, and Baldacci [SXB26, Lemma 2]. We discuss how our contributions can be adapted to their criterion in Section 5, but do not use it for our exposition since it increases notational burden and edge cases without yielding more interesting results. $\triangleleft$

Conveniently, relaxed too-long paths are obstacles to representability, just as too-long paths are to implementability.

3.14 Theorem. Let $\mathcal{T}$ be a time discretisation. A solution S is $\mathcal{T}$ -representable if and only if D_S contains no $\mathcal{T}$ -relaxed too-long path. $\triangleleft$

Proof. Let $S = (\mathcal{P}, \mathcal{C})$ be $\mathcal{T}$ -representable with dispatch intervals $h: \mathcal{C} \rightarrow 2^{\mathbb{N}}$. Moreover, let $W = u_0 \dots u_n$ be a walk in D_S with $u_i = (v_i, k_i)$ for $i \in [n]$ and $a_i = v_{i-1}v_i$, $c_i = c(a_i, k_i) = c(a_i, k_{i-1})$ for $i \in [n]$.

We show that $\pi_W(i-1) \leq \max h(c_i)$, for all $i \in [n]$. For $i = 1$, since $h(c_1) \cap I_{k_0}(a_1) \neq \emptyset$, $\max h(c_1) \geq \underline{t}_{k_0}(a_1) = \underline{t}(u_0) = \pi_W(0)$. For $i \in (n)$, using the induction hypothesis and the fact that S is $\mathcal{T}$ -representable, $\pi_W(i) = \lfloor \pi_W(i-1) \rfloor_{\mathcal{T}_{a_i}} + \tau(a_i) \leq \lfloor \max h(c_i) \rfloor_{\mathcal{T}_{a_i}} + \tau(a_i) = \min h(c_i) + \tau(a_i) \leq \max h(c_{i+1})$.

Since $h(c_n) \cap I_{k_n}(a_n) \neq \emptyset$, $\min h(c_n) \leq \bar{t}_{k_n}(a_n) = \bar{t}(u_n) - \tau(a_n)$ and, consequently, $\pi_W(n) = \lfloor \pi_W(n-1) \rfloor_{\mathcal{T}_{a_n}} + \tau(a_n) \leq \lfloor \max h(c_n) \rfloor_{\mathcal{T}_{a_n}} + \tau(a_n) = \min h(c_n) + \tau(a_n) \leq \bar{t}(u_n)$. Thus, W is not relaxed too-long.

For the converse, we assume that D_S contains no relaxed too-long path. This lets us define a dispatch time at every node u in D_S by setting

$$t(u) = \max \{ \pi_W(n) : W \text{ is a walk in } D_S \text{ ending at } u \}.$$

Note that we explicitly allow walks of length 0 here, meaning $t(u) \geq \underline{t}(u)$. Also, since no relaxed too-long paths exist, this value is finite and satisfies $t(u) \leq \bar{t}(u)$. For $c = (vv', \kappa)$, we let $t(c) = \max \{ t((v, k)) : k \in \kappa \}$ and $h(c)$ be the interval in $\mathcal{T}_{vv'}^I$ containing $t(c)$. We use this to show that S is $\mathcal{T}$ -representable.

To this end, let $k \in \mathcal{K}$ with $P_k = a_1 \dots a_n$, $c_i = c(a_i, k) = (a_i, \kappa_i)$, and $a_i = v_{i-1}v_i$ for $i \in [n]$. Furthermore, let $u_i = (v_i, k)$ for $i \in [n]$. By definition, for $i \in [n]$, $\max h(c_i) \geq t(c_i) \geq t(u_{i-1}) \geq \underline{t}(u_{i-1}) = \underline{t}_k(a_i)$.

Again, let $i \in [n]$, and let $u' = (v_{i-1}, k')$ with $k' \in \kappa_i$ such that $t(c_i) = t(u')$. Moreover, let W be a walk of length $m \geq 0$ in D_S ending at u' with $\pi_W(m) = t(u') = t(c_i)$. By extending W to u_i , we get a walk W' ending at u_i and, thus, $t(u_i) \geq \pi_{W'}(m+1) = \lfloor t(c_i) \rfloor_{\mathcal{T}_{a_i}} + \tau(a_i)$. Since W' is not relaxed too-long, $\pi_{W'}(m+1) \leq \bar{t}(u_i) = \bar{t}_k(a_i) + \tau(a_i)$. Hence, $\min h(c_i) = \lfloor t(c_i) \rfloor_{\mathcal{T}_{a_i}} \leq \bar{t}_k(a_i)$ and $h(c_i) \cap I_k(a_i) \neq \emptyset$.

For the final property, let $i \in (n)$. Using the above computations, we see that $\min h(c_i) + \tau(a_i) = \lfloor t(c_i) \rfloor_{\mathcal{T}_{a_i}} + \tau(a_i) \leq t(u_i) \leq t(c_{i+1}) \leq \max h(c_{i+1})$ and the proof is complete. $\square$

We point out that we cannot restrict ourselves to minimal too-long paths in Theorem 3.14, since our example showed that a solution can be non-representable because a non-minimal too-long path is relaxed too-long, but none of the minimal ones are.

Evidently, if we want to make a non-implementable solution non-representable, we need to select a discretisation that makes at least one walk W in D_S relaxed too-long. Since, by definition, $\pi_W(i) \leq t_W(i)$ for all i , such a walk must be too-long. Furthermore, relaxed too-long paths are guaranteed not to reappear in representable solutions, leading to the following natural criterion for elimination.

3.15 Observation. A too-long path W is eliminated by a discretisation $\mathcal{T}$ if it is $\mathcal{T}$ -relaxed too-long. $\triangleleft$

3.16 Example. We demonstrate that the converse of Observation 3.15 is not true; a too-long path may be eliminated by $\mathcal{T}$ even if it is not $\mathcal{T}$ -relaxed too-long. For non-minimal paths this follows directly from Observation 3.9, but it is not at all obvious that the same is true for minimal paths. Consider the example in Figure 4, especially the minimal too-long path W . We select a discretisation $\mathcal{T}$ that is full (i.e., $\mathcal{T}_a = [H]$) except that $5 \notin \mathcal{T}_{v_2v_3}$ and $6 \notin \mathcal{T}_{v_3v_4}$. Then W is not relaxed too-long, since $\pi_W(0) = 2, \pi_W(1) = 3, \pi_W(2) = 5, \pi_W(3) = 6, \pi_W(4) = 7$, and $\pi_W(5) = 9 \not\geq 9 = \ell_{k_1}$. For this path to exist in a solution's dispatch-node graph, commodity k_1 must traverse v_0v_1 together with k_3 , as well as v_1v_2 and v_4v_5 . Taking v_2v_5 would prevent it from traversing the required arc v_4v_5 , so it must take v_2v_3 . This implies that W' also exists in the dispatch-node graph, and this path is relaxed too-long, since $\pi_{W'}(0) = 2, \pi_{W'}(1) = 3, \pi_{W'}(2) = 5, \pi_{W'}(3) = 6 > 5 = \bar{t}_{k_1}(v_3)$. Since existence of W implies W' and this path is relaxed too-long, both of them are eliminated.

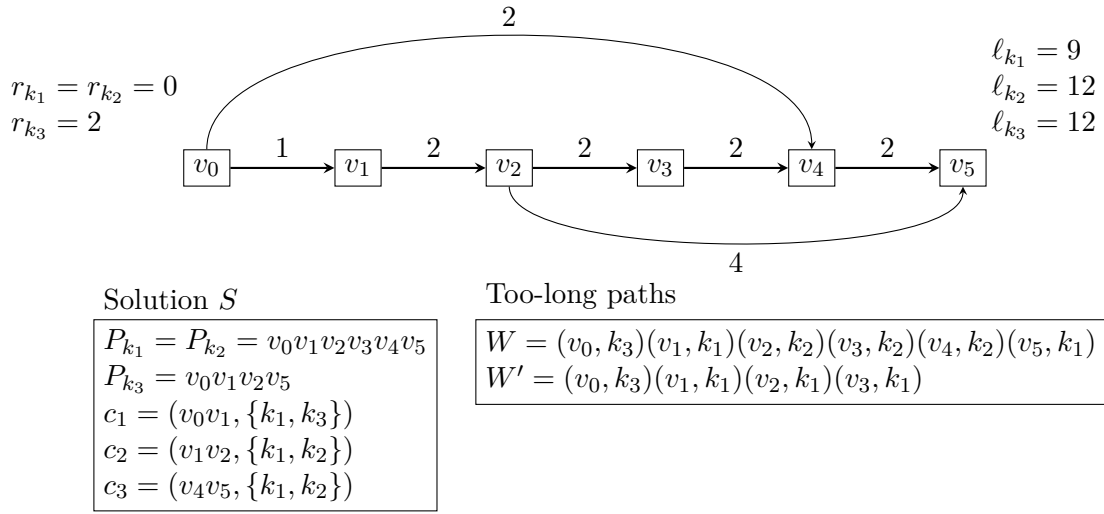


Figure 4: Example of a minimal too-long path that is eliminated by $\mathcal{T}$ but not $\mathcal{T}$ -relaxed too-long. Arcs are labelled with travel time. Commodities' origins and destinations are labelled with their release time and deadline. All unlisted consolidations are singletons.

So we see that selecting $\mathcal{T}$ so that all minimal too-long paths are $\mathcal{T}$ -relaxed too-long may be too conservative; a smaller discretisation may already eliminate them due to additional routing considerations. Finding a practical characterisation of an eliminated too-long path is an interesting open question. But we suspect that in practice, the increase in discretisation size from using our sufficient characterisation will usually be negligible.

4. THE OPTIMAL REFINEMENT PROBLEM

Now that we have identified conflicts (minimal too-long paths) and potential fixes (making them relaxed too-long), we can now define the problem studied in the rest of this work: Given a solution S to SND, the *optimal refinement problem* (ORP) is to find a discretisation $\mathcal{T}$ of minimum size $\sum_{a \in A_D} |\mathcal{T}_a|$ such that all minimal too-long paths in D_S are $\mathcal{T}$ -relaxed too-long.

The objective of minimising the number of time points is motivated by the wish to reduce the size of the integer programming models used to solve R-SND.

4.1 Remark. In the short paper [WH26], we show that under this objective function, the problem of making an arbitrary set of too-long paths also relaxed too-long is NP-hard. We note that it is possible to construct instances of SND such that the hardness result transfers to ORP as stated here, but omit the construction, which provides little additional insight. $\triangleleft$

This section is devoted to solving the ORP. First, we look at properties of minimal too-long paths in Section 4.1 before showing how the ORP can be solved if one has explicitly enumerated all minimal too-long paths in Section 4.2. An algorithm that enumerates all these paths can be found in Section 7.4 of [SXB26] (and can be easily adapted to our definition of a too-long path). We then demonstrate that this approach can be unattractive since the number of such paths may be exponential in instance parameters. This motivates us to develop a more compact representation of all minimal too-long paths in Section 4.3 and corresponding solution approaches in Sections 4.4 to 4.6.

For the remainder of the section, let $\mathcal{T}$ denote a time discretisation and $\mathcal{W}$ the set of all minimal too-long paths in D_S .

4.1. Properties of Minimal Too-Long Paths

For everything that happens later in this section, from formulating an integer program for the ORP to developing a sufficient criterion for eliminating all minimal too-long paths, it is helpful to know certain properties that these minimal too-long paths have. We use this subsection to collect several such properties.

First, we note that if node v is visited by commodity k , then this commodity needs to arrive at v at some point in the interval $[\underline{t}_k(v), \bar{t}_k(v)]$. Thus, this interval describes a certain amount of slack that is available at the node. If we deviate from the lower bound $\underline{t}_k(v)$ we have used up some of the available slack, and if we exceed the upper bound $\bar{t}_k(v)$, then we are late, which happens at the end of too-long paths. Let us formally define these values.

4.2 Definition (Slack consumption and lateness). Let $u \in V_{D_S}$, $t \in \mathbb{N}$, and $w =$

(u, t) . We define the *slack consumption* and *lateness* at w by

$$\Delta_S(w) = t - \underline{t}(u), \quad \Delta_L(w) = t - \bar{t}(u). \quad \triangleleft$$

Minimal too-long paths are only late at their last node. Since starting such a path at a later node lets it finish earlier by exactly its slack consumption, this value must be at least equal to the lateness at all but the first node (where it is zero), otherwise the path would not be minimal. Formally, we obtain the following observation.

4.3 Observation. If $W = u_0 \dots u_n$ is a minimal too-long path and $w_i = (u_i, t_W(i))$, then the following properties hold:

- $\Delta_L(w_n) > 0$ and $\Delta_L(w_i) \leq 0$ for all $i \in [n]$.
- $\Delta_S(w_0) = 0$ and $\Delta_S(w_i) \geq \Delta_L(w_n) > 0$ for all $i \in (n]$. $\triangleleft$

Proof. For the lateness, $\Delta_L(w_n) > 0$ since W is too-long and, for $i \in [n]$, $\Delta_L(w_i) \leq 0$ since Wu_i is not too-long.

For the slack consumption, $\Delta_S(w_0) = 0$ since $t_W(0) = \underline{t}(u_0)$, and, for $i \in (n]$, u_iW is not too-long, so $\underline{t}(u_i) + \tau(u_iW) \leq \bar{t}(u_n) = t_W(n) - \Delta_L(w_n) = \underline{t}(u_0) + \tau(W) - \Delta_L(w_n) = t_W(i) + \tau(u_iW) - \Delta_L(w_n)$. Thus, $t_W(i) \geq \underline{t}(u_i) + \Delta_L(w_n)$ and $\Delta_S(w_i) \geq \Delta_L(w_n)$. $\square$

In fact, these properties are actually equivalent.

4.4 Lemma. If $W = u_0 \dots u_n$ is a walk in D_S and $w_i = (u_i, t_W(i))$, then W is minimal too-long if and only if

- $\Delta_L(w_i) \leq 0$ for all $i \in [n]$ and
- $\Delta_S(w_i) \geq \Delta_L(w_n) > 0$ for all $i \in (n]$. $\triangleleft$

Proof. The forward implication follows from Observation 4.3. For the converse, let $W = u_0 \dots u_n$ be a walk satisfying the given properties and $w_i = (u_i, t_W(i))$. Since $\Delta_L(w_n) > 0$, W is a too-long path. Let u_iWu_j be a sub-walk of W that is too-long. If $j < n$, then $\underline{t}(u_i) + \tau(u_iWu_j) \leq \underline{t}(u_0) + \tau(u_0Wu_j) = t_W(j) = \Delta_L(w_j) + \bar{t}(u_j) \leq \bar{t}(u_j)$. Thus, $j = n$.

If $i > 0$, then $\underline{t}(u_i) + \tau(u_iW) = t_W(i) - \Delta_S(w_i) + \tau(u_iW) = \underline{t}(u_0) + \tau(W) - \Delta_S(w_i) \leq t_W(n) - \Delta_L(w_n) = \bar{t}(u_n)$. Hence, $i = 0$ and W is minimal. $\square$

Another useful observation is that any minimal too-long path that is also relaxed too-long must have relaxed arrival times that exceed the earliest arrival times at all but the first node.

4.5 Observation. Let $\mathcal{T}$ be a time discretisation and $W = u_0 \dots u_n$ be a minimal too-long path that is $\mathcal{T}$ -relaxed too-long. Then $\pi_W(i) > \underline{t}(u_i)$ for all $i > 0$. $\triangleleft$

Proof. Let $W = u_0 \dots u_n$ be a minimal too-long path that is $\mathcal{T}$ -relaxed too-long and $i > 0$. Since u_iW is not too-long, $\underline{t}(u_i) + \tau(u_iW) \leq \bar{t}(u_n) < \pi_W(n) \leq \pi_W(i) + \tau(u_iW)$, so $\underline{t}(u_i) < \pi_W(i)$. $\square$

4.2. Path-Based Approaches

We want to formulate the ORP as a binary integer program, which can already be found in our short paper [WH26, Section 4]. For the upcoming IP-formulation it is helpful to have an alternative characterisation for when a too-long path is actually relaxed too-long.

4.6 Lemma. A too-long path $W = u_0 b_1 u_1 \dots b_n u_n$ is $\mathcal{T}$ -relaxed too-long if and only if for every $i \in (n]$ there is a time point $t_i \in \mathcal{T}_{b_i}$ such that

- (a) $t_1 \leq \underline{t}(u_0)$,
- (b) $t_i + \tau(b_i) \geq t_{i+1}$ for all $i \in (n)$,
- (c) and $t_n + \tau(b_n) > \bar{t}(u_n)$.

Additionally, $t_i \in I_W(i) = [t_W(i-1) - \Delta_L(u_n, t_W(n)) + 1, t_W(i-1)]$ for all $i \in (n]$. $\triangleleft$

Proof. Evidently, $\pi_W(i) - \tau(b_i) \in \mathcal{T}_{b_i}$ for $i > 0$. Assume W is relaxed too-long. Then we can set $t_i = \pi_W(i) - \tau(b_i)$. Then, by definition of $\pi_W(i)$, all three conditions hold.

Conversely, assume that t_i fulfilling the conditions exist. Inductively, $\pi_W(i-1) \geq t_i$: the base case follows from Property (a), and since $t_i \in \mathcal{T}_{b_i}$, we obtain $\pi_W(i) = \lfloor \pi_W(i-1) \rfloor_{\mathcal{T}_{b_i}} + \tau(b_i) \geq t_i + \tau(b_i)$. For $i < n$, this is at least t_{i+1} by Property (b); for $i = n$, it is greater than $\bar{t}(u_n)$ by Property (c). Thus, W is relaxed too-long.

For the additional statement, observe that ((a)) and ((b)) together give

$$t_i \leq t_1 + \sum_{j=1}^{i-1} \tau(b_j) \leq \underline{t}(u_0) + \sum_{j=1}^{i-1} \tau(b_j) = t_W(i-1).$$

Similarly, Properties (b) and (c) give

$$t_i + \sum_{j=i}^n \tau(b_j) \geq t_n + \tau(b_n) > \bar{t}(u_n)$$

and therefore

$$t_i > \bar{t}(u_n) - \sum_{j=i}^n \tau(b_j) = t_W(i-1) - \Delta_L(u_n, t_W(n)). \quad \square$$

Based on this reformulation, we obtain the following binary program (1). Recall that we denote the set of all minimal too-long paths $\mathcal{W}$ and for a dispatch-node arc $b = (v, k)(v', k')$, we write $a(b) = vv'$ for its underlying arc in D . For a walk W we write n_W for its number of arcs. The binary variable y_{at} indicates whether t is included in $\mathcal{T}_a$, and x_{Wit} indicates whether t is chosen as the time for the i th arc of W . By restricting t to $I_W(i)$, we guarantee that both Properties (a) and (c) hold, so these conditions are not explicitly imposed. For an arc $a \in AD$, let I_a denote the union of all intervals $I_W(i)$ where

$$a(b_i) = a.$$

$$\min \sum_{a \in AD} \sum_{t \in I_a} y_{at} \quad (1a)$$

$$\text{s.t.} \quad \sum_{t \in I_W(i)} x_{Wit} = 1 \quad \forall W \in \mathcal{W}, i \in (n_W] \quad (1b)$$

$$x_{Wit} \leq y_{a(b_i)t} \quad \forall W \in \mathcal{W}, i \in (n_W], t \in I_W(i) \quad (1c)$$

$$\sum_{t \in I_W(i)} tx_{Wit} + \tau(b_i) \geq \sum_{t \in I_W(i+1)} tx_{W,i+1,t} \quad \forall W \in \mathcal{W}, i \in (n_W) \quad (1d)$$

$$x_{Wit} \in \{0, 1\} \quad \forall W \in \mathcal{W}, i \in (n_W], t \in I_W(i) \quad (1e)$$

$$y_{at} \in \{0, 1\} \quad \forall a \in AD, t \in I_a \quad (1f)$$

4.7 Remark. Shu, Xu, and Baldacci [SXB26] also proposed a mixed-integer program for this problem (using their slightly stronger elimination criterion). Due to their differing definition of too-long paths that may start only at origin nodes (o_k, k) , the MIP is significantly more complex and not reproduced here. $\triangleleft$

Alternatively, a very simple heuristic solution for ORP can be obtained by simply adding time points for each too-long path according to its real arrival time:

4.8 Theorem ([SXB26, Theorem 3]). Let $W = b_1 \dots b_n$ denote a too-long path in D_S . For a discretisation $\mathcal{T}$ with $t_W(i-1) \in \mathcal{T}_{b_i}$ for $i \in (n]$, W is $\mathcal{T}$ -relaxed too-long (and thus eliminated). $\triangleleft$

Shu, Xu, and Baldacci [SXB26, Section 7.4] also propose a matheuristic for ORP. They keep all time points from the previous discretisation and then process the too-long paths in batches. For each batch, they solve their MIP to determine the fewest time points to add to make all paths in the current batch relaxed too-long. They further restrict the MIP to selecting time points from among those implied by Theorem 4.8.

One drawback of these approaches is that they require a full enumeration of all minimal too-long paths, which may be impractical. Helber [Hel26, Section 4.4] gives an example where a cycle in the dispatch-node graph leads to an exponential number of too-long paths. Since such cycles are easy to detect and may be handled separately [see e.g., Mar+21, Section 4.1.3.], we give another motivating example that does not rely on such cycles.

4.9 Example. Consider an instance on the network depicted in Figure 5, which consists of the path $P = v_0 \dots v_n$ together with the nodes v' and v'' as well as arcs v_0v' , $v'v_1$, $v_{n-1}v''$, and $v''v_n$. Arcs v_0v_1 and $v_{n-1}v_n$ have length 3 and all others have length 1. Commodities k_1 and k_2 must both be transported from v_0 to v_n . Note that the nodes v' and v'' exist simply to make $\underline{t}_k(v)$ and $\bar{t}_k(v)$ irrelevant for nodes v_1 to v_{n-1} .

The path P has length $n + 4$, so each commodity could traverse this path by itself and arrive on time, but when consolidated, k_1 arrives too late. If we consider the dispatch-node

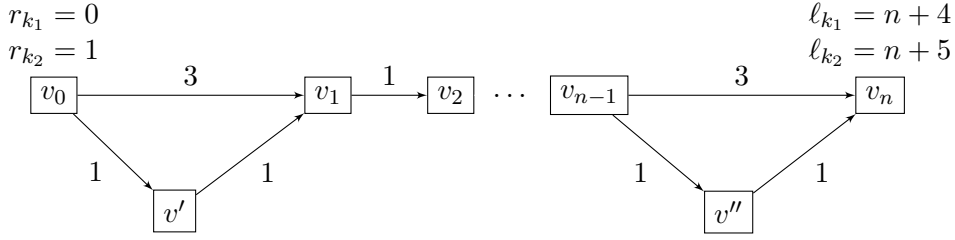


Figure 5: Example of an instance with exponentially many too-long paths. Arc weights represent travel time τ .

graph for the solution that consolidates them on all arcs of P , we see that the too-long paths must start at (v_0, k_2) and end at (v_n, k_1) , since this is the only way to achieve an arrival time of $n + 5$ and the only way to reach the only deadline this violates. However, the nodes in between are irrelevant, so we can choose either (v_i, k_1) or (v_i, k_2) for all intermediate $i \in (n)$. Thus, we obtain 2^{n-1} minimal too-long paths in this example.

Despite this abundance of minimal too-long paths, it suffices to add the time points 1 on arc v_0v_1 and $i + 2$ on the arcs $v_{i-1}v_i$ for $i \in [2, n]$ to eliminate them all (by making them relaxed too-long). $\triangleleft$

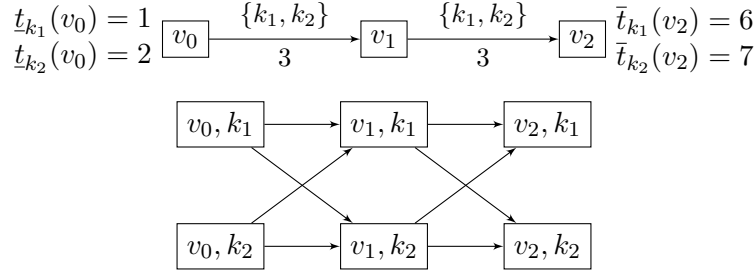
We also see in this example that a necessary ingredient of this blow-up in the number of too-long paths is that on most arcs we can choose between the two commodities. One can imagine that for instances where good solutions require consolidations of many commodities, the number of too-long paths may grow drastically. Such instances have already been observed to be harder to solve [see e.g., the HC classification of [Mar+21](#)]. This motivates us to find a more compact representation of the conflicts encoded by too-long paths.

4.3. A Graph-Based Criterion

As we just saw, there may be exponentially many too-long paths that can be eliminated in the same way. To deal with such cases, we want to find a more compact representation that lets us make all minimal too-long paths $\mathcal{T}$ -relaxed too-long, without checking them all individually.

To motivate our upcoming construction, we use the partial network shown in Figure 6 and its dispatch-node graph. The time bounds not shown in the figure are chosen so that no single-arc path is too-long; their exact values are irrelevant. For ease of notation, we write u_{ij} for (v_i, k_j) . Here, the minimal too-long paths are the paths that start at u_{01} or u_{02} and end at u_{21} or u_{22} , except the two paths $u_{01}u_{11}u_{22}$ and $u_{01}u_{12}u_{22}$. We use the full time discretisation $\mathcal{T}$, which clearly ensures that all too-long paths are $\mathcal{T}$ -relaxed too-long.

First, we observe that certain too-long paths are ‘more critical’ than others. For example, consider the two too-long paths $W = u_{01}u_{11}u_{21}$ and $W' = u_{02}u_{11}u_{21}$. Then $\pi_W(2) = 7$

Figure 6: Illustrative example to motivate the construction of $D_{\mathcal{W}}$.

and $\pi_{W'}(2) = 8$. Thus, it suffices to make W relaxed too-long, since W' is then relaxed too-long as well. As a result, we are interested in the ‘most critical arrival times’ at the nodes where minimal too-long paths end.

In order to compute them, the following propagation approach would be natural: since the most critical arrival time at u is obtained by some minimal too-long path $W = u_0 \dots u_{n-1}u_n$ with $u_n = u$, its most critical arrival time $\pi_W(n)$ is equal to $\lfloor \pi_W(n-1) \rfloor \tau_{u_{n-1}u_n} + \tau(u_{n-1}u_n)$. Thus, we could simply compute these values for the predecessors of u (on minimal too-long paths) and take the smallest one.

This propagation does not work, however. In our small example, we would obtain that u_{22} has 7 as its most critical arrival time, which does not exceed $\bar{t}(u_{22})$ despite the fact that all minimal too-long paths are eliminated. This occurs since there is ‘cross-contamination’: the minimal too-long path $W = u_{01}u_{12}u_{21}$ propagates its relaxed arrival time of 4 to u_{12} , which is then moved to u_{22} to obtain the 7 using the walk $W' = u_{02}u_{12}u_{22}$. But the walk $u_{01}u_{12}u_{22}$ that actually leads to the 7 is not even too-long.

Thus, we need to ensure that this cross-contamination does not occur. In the example above, this happened since $\pi_W(1) = 4$ but $\pi_{W'}(1) = 5$, so the walk obtained by retracing the minimal one was not long enough. We fix this by means of a time-expansion: we augment the nodes u_{ij} by their real arrival times in the minimal too-long paths they appear in. This ensures that all prefixes that meet at a node have equal real arrival time and can be used interchangeably to obtain too-long paths. Thus, only the worst (earliest) one is of interest and this motivates the following definition.

4.10 Definition (Time-expanded too-long path graph). The *time-expanded too-long path graph* $D_{\mathcal{W}}$ is given by

$$\begin{aligned} VD_{\mathcal{W}} &= \{(u_i, t_W(i)) : W = u_0 \dots u_n \in \mathcal{W}, i \in [n]\}, \\ AD_{\mathcal{W}} &= \{(u_i, t_W(i))(u_{i+1}, t_W(i+1)) : W = u_0 \dots u_n \in \mathcal{W}, i \in [n]\}. \end{aligned}$$

For notational purposes, we identify a node $w = (u, t)$ with its first component u , letting us write $\underline{t}(w)$ or $a(w)$.

We also define $\mathcal{S}_{\mathcal{W}}$ and $\mathcal{B}_{\mathcal{W}}$ to be the sets of nodes in $D_{\mathcal{W}}$ originating from the initial and final nodes of walks in $\mathcal{W}$. ◁

This graph expands the minimal too-long paths by their real arrival times. Since we frequently move between a minimal too-long path and its time-expanded version, we introduce some helpful notation.

4.11 Notation.

- For $W = u_0 \dots u_n \in \mathcal{W}$, we write P_W for the path $(u_0, t_W(0)) \dots (u_n, t_W(n))$ in $D_{\mathcal{W}}$.
- For a path $P = w_0 \dots w_n$ in $D_{\mathcal{W}}$ with $w_i = (u_i, t_i)$ for $i \in [n]$, we write W_P for the walk $u_0 \dots u_n$ in D_S . $\triangleleft$

Note that, using this notation, $D_{\mathcal{W}} = \bigcup_{W \in \mathcal{W}} P_W$.

The rest of this subsection is essentially dedicated to showing that, on this graph, the propagation described above actually works. To this end, we first note a few properties.

4.12 Observation.

- Let $w = (u, t) \in VD_{\mathcal{W}}$.
- (a) $N^{\text{in}} w = \emptyset$ if and only if $w \in \mathcal{S}_{\mathcal{W}}$ if and only if $\Delta_S(w) = 0$.
 - (b) $N^{\text{out}} w = \emptyset$ if and only if $w \in \mathcal{B}_{\mathcal{W}}$ if and only if $\Delta_L(w) > 0$. $\triangleleft$

Proof. By Observation 4.3, the nodes $w \in D_{\mathcal{W}}$ that have $\Delta_S(w) = 0$ are exactly the initial nodes of minimal too-long paths. Similarly, the nodes with $\Delta_L(w) > 0$ are exactly the final nodes of minimal too-long paths. $\square$

4.13 Corollary.

- (a) If P is an $\mathcal{S}_{\mathcal{W}}\mathcal{B}_{\mathcal{W}}$ -path in $D_{\mathcal{W}}$, then W_P is too-long.
- (b) If P is a path in $D_{\mathcal{W}}$ and W_P is too-long, then P ends at a node in $\mathcal{B}_{\mathcal{W}}$. $\triangleleft$

Proof. We notice that each arc $(u, t)(u', t')$ in $D_{\mathcal{W}}$ satisfies $t' = t + \tau(uu')$, giving us that any path P from $w = (u, t)$ to $w' = (u', t')$ satisfies $t' = t + \tau(W_P)$. Now let $P = w_0 \dots w_n$ be a path in $D_{\mathcal{W}}$ with $w_i = (u_i, t_i)$ for all $i \in [n]$.

For the first part, let P be an $\mathcal{S}_{\mathcal{W}}\mathcal{B}_{\mathcal{W}}$ -path. Then $\underline{t}(u_0) + \tau(W_P) = t_0 + \tau(W_P) = t_n > \bar{t}(u_n)$, by Observation 4.12. Thus, W_P is too-long.

For the second part, $\bar{t}(w_n) < \underline{t}(w_0) + \tau(W_P)$ since this path is too-long. Using Observation 4.12, $\underline{t}(w_0) \leq t_0$, giving us $\bar{t}(w_n) < t_0 + \tau(W_P) = t_n$, so $w_n \in \mathcal{B}_{\mathcal{W}}$. $\square$

Next, let us define the propagation we already motivated at the start of the subsection.

4.14 Definition (Relaxed arrival times in $D_{\mathcal{W}}$). For $w \in D_{\mathcal{W}}$, we define the *relaxed arrival time* π_w at w by

$$\pi_w = \begin{cases} \underline{t}(w) & \text{if } N^{\text{in}} w = \emptyset, \\ \min \left\{ \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'w}} + \tau(w'w) : w' \in N^{\text{in}} w \right\} & \text{otherwise.} \end{cases} \quad \triangleleft$$

Note that this is well defined since $D_{\mathcal{W}}$ is acyclic and $N^{\text{in}} w = \emptyset$ if and only if $t = \underline{t}(w)$.

We now establish the desired characterisation, namely that making the critical prefixes computed by the propagation relaxed too-long is necessary and sufficient.

4.15 Theorem. Let $\mathcal{T}$ be a time discretisation. Then all minimal too-long paths in D_S are $\mathcal{T}$ -relaxed too-long if and only if $\pi_w > \bar{t}(w)$ for all $w \in \mathcal{B}_{\mathcal{W}}$. $\triangleleft$

Proof. We start with the backward implication for which we need the following claim.

Claim 1. For all $w = (u, t) \in VD_{\mathcal{W}}$ and for all $W \in \mathcal{W}$ for which u is the i th node and $t = t_W(i)$, we get $\pi_w \leq \pi_W(i)$. $\triangleleft$

Proof. We proceed by induction on a topological ordering. For nodes $w = (u, t) \in \mathcal{S}_{\mathcal{W}}$, if u occurs in $W \in \mathcal{W}$, then u is the initial node and $\pi_w = \underline{t}(u) = \pi_W(0)$.

For the induction step, we let $w = (u, t) \in VD_{\mathcal{W}} \setminus \mathcal{S}_{\mathcal{W}}$, $W \in \mathcal{W}$ contain u as its i th node, and $t = t_W(i)$. Let u' be the predecessor of u on W and $t' = t_W(i-1)$. Then $w' = (u', t')$ is a predecessor of w in $D_{\mathcal{W}}$. Thus, using the induction hypothesis, we conclude that $\pi_w \leq \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'u}} + \tau(w'u) \leq \lfloor \pi_W(i-1) \rfloor_{\mathcal{T}_{u'u}} + \tau(u'u) = \pi_W(i)$. $\diamond$

Now let $W = u_0 \dots u_n \in \mathcal{W}$. We know that $w = (u_n, t_W(n)) \in \mathcal{B}_{\mathcal{W}}$ and $\bar{t}(w) < \pi_w \leq \pi_W(n)$, so W is $\mathcal{T}$ -relaxed too-long.

For the forward implication, we need to associate the nodes of $D_{\mathcal{W}}$ with walks in D_S that realise their relaxed arrival times.

Claim 2. For each $w \in VD_{\mathcal{W}}$ there exists a path $P = w_0 \dots w_n$ in $D_{\mathcal{W}}$ such that $w_n = w$, $w_0 \in \mathcal{S}_{\mathcal{W}}$, and, for all $i \in [n]$,

- (a) $t_{W_P}(i) = t_i$ and
- (b) $\pi_{W_P}(i) = \pi_{w_i}$. $\triangleleft$

Proof. We prove the claim by induction on some topological ordering, starting with $w_0 = (u_0, t_0) \in \mathcal{S}_{\mathcal{W}}$ once more. In this case, we can take the trivial path $P = w_0$ and get $t_{W_P}(0) = \underline{t}(u_0) = t_0$ and $\pi_{w_0} = \underline{t}(u_0) = \pi_{W_P}(0)$.

For the induction step, let $w = (u, t) \in VD_{\mathcal{W}} \setminus \mathcal{S}_{\mathcal{W}}$. Then $\pi_w = \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'u}} + \tau(w'u)$ for some $w' = (u', t') \in N^{\text{in}} w$. By definition of the arcs, $t' = t - \tau(w'u)$.

Then, by induction, we get a path $P' = w_0 \dots w_{n'}$ in $D_{\mathcal{W}}$ with $w_0 \in \mathcal{S}_{\mathcal{W}}$, $w_{n'} = w'$, and, for all $i \in [n']$, $w_i = (u_i, t_i)$, $t_{W_{P'}}(i) = t_i$ and $\pi_{w_i} = \pi_{W_{P'}}(i)$. Let $P = w_0 \dots w_{n'}w$ and $n = n' + 1$. Then $w_0 \in \mathcal{S}_{\mathcal{W}}$, $w_n = w$ and the remaining two properties continue to hold for $i \in [n']$. For n we get $t_{W_P}(n) = t_{W_{P'}}(n') + \tau(w'u) = t = t_n$ and $\pi_w = \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'u}} + \tau(w'u) = \lfloor \pi_{W_{P'}}(n') \rfloor_{\mathcal{T}_{u'u}} + \tau(u'u) = \pi_{W_P}(n)$, as desired. $\diamond$

We can also show a lower bound on the values π_w .

Claim 3. For all $w \in VD_{\mathcal{W}}$, $\pi_w \geq \underline{t}(w)$. $\triangleleft$

Proof. We prove the result inductively, where it clearly holds for nodes in $\mathcal{S}_{\mathcal{W}}$. So, let $w = (u, t) \in VD_{\mathcal{W}} \setminus \mathcal{S}_{\mathcal{W}}$. By definition of $D_{\mathcal{W}}$, w is on $P = P_W$ for some minimal too-long path $W \in \mathcal{W}$. By Claim 2 we obtain a w_0w -path P' in $D_{\mathcal{W}}$ and corresponding walk $W' = W_{P'}$ with $w_0 \in \mathcal{S}_{\mathcal{W}}$ that realises the relaxed arrival times.

We now regard the splicings $P'' = P'wP = w_0 \dots w_n$ and $W'' = W'uW = u_0 \dots u_n$. Since P'' starts in $\mathcal{S}_{\mathcal{W}}$ and ends in $\mathcal{B}_{\mathcal{W}}$, W'' is too-long by Corollary 4.13. It thus contains a minimal too-long path $W^* = u_i W'' u_j$, meaning $P^* = w_i P'' w_j$ ends in $\mathcal{B}_{\mathcal{W}}$ by Corollary 4.13. Hence, $j = n$.

Since W is a minimal too-long path and $w \notin \mathcal{S}_{\mathcal{W}}$, uW is not too-long, and, thus, $u = u_s$ is in W^* . Consequently, $i < s$, $u_i \in W'$, and $\pi_{w_i} = \pi_{W'}(i)$. By induction, we know that $\pi_{W'}(i) = \pi_{w_i} \geq \underline{t}(w_i) = \pi_{W^*}(0)$. Therefore, $\pi_{W'}(i + \ell) \geq \pi_{W^*}(\ell)$ for all $\ell \leq s - i$, in particular, $\pi_w = \pi_{W'}(s) \geq \pi_{W^*}(s - i)$ holds at w . By Observation 4.5, $\pi_{W^*}(s - i) > \underline{t}(w)$, so $\pi_w > \underline{t}(w)$ and the claim follows. $\diamond$

Now let $w \in \mathcal{B}_{\mathcal{W}}$. Claim 2 yields a path $P = w_0 \dots w_n$ in $D_{\mathcal{W}}$ with $w_i = (u_i, t_i)$ for all $i \in [n]$ and $w_0 \in \mathcal{S}_{\mathcal{W}}$ and $w_n = w$. Again, we obtain that W_P is a too-long path and contains a minimal too-long path $W^* = u_i W_P u_n$, by Corollary 4.13.

Since W^* is $\mathcal{T}$ -relaxed too-long, $\pi_{W^*}(n - i) > \bar{t}(u_n) = \bar{t}(w)$. Moreover, by Claim 3, $\pi_{W_P}(i) = \pi_{w_i} \geq \underline{t}(w_i) = \pi_{W^*}(0)$ and, inductively, $\pi_{W_P}(i + \ell) \geq \pi_{W^*}(\ell)$. Thus, $\pi_w = \pi_{W_P}(n) \geq \pi_{W^*}(n - i) > \bar{t}(w)$ as desired. $\square$

4.16 Remark. We note that we only needed to use the set $\mathcal{W}$ of all minimal too-long paths in the proof above to get the forward implication. For the converse, any set $\mathcal{W}'$ of minimal too-long paths would have worked, so, indeed, if $\pi_w > \bar{t}(w)$ for all $w \in \mathcal{B}_{\mathcal{W}'}$, then all minimal too-long paths in $\mathcal{W}'$ are $\mathcal{T}$ -relaxed too-long. $\triangleleft$

4.4. Constructing the Time-Expanded Too-Long Path Graph

In order to use the criterion obtained in Theorem 4.15, we need to be able to construct the time-expanded too-long path graph $D_{\mathcal{W}}$. Evidently, it can be obtained by first enumerating all minimal too-long paths in $\mathcal{W}$. However, we want to avoid this explicit enumeration and this section describes an algorithm that constructs $D_{\mathcal{W}}$ in $\mathcal{O}(|A D_S| \cdot H)$ time. While this is still pseudo-polynomial in the instance parameters, it avoids the enumeration of potentially exponentially many minimal too-long paths.

To do so, we first construct a *candidate graph* $D_{\mathcal{W}}^\dagger$ that contains $D_{\mathcal{W}}$ as a subgraph and then remove the unnecessary nodes and arcs. The algorithm to compute the candidate graph is given in Algorithm 4.1. It starts by adding timed copies of all nodes at their earliest time. It then expands each timed copy along outgoing arcs, so long as the expansion still has positive slack consumption (otherwise, this timed arc cannot be induced by a minimal too-long path). The nodes reached this way are also considered for expansion if they have non-positive lateness (otherwise, they are already too late). We now establish the running-time bound and prove that $D_{\mathcal{W}}$ is indeed a subgraph.

Algorithm 4.1: Building the candidate graph

```

1 def CONSTRUCTCANDIDATEGRAPH( $D_S$ ):
2    $D_{\mathcal{W}}^{\uparrow} \leftarrow (\{(u, \underline{t}(u)) : u \in VD_S\}, \emptyset)$ 
3    $L \leftarrow V(D_{\mathcal{W}}^{\uparrow})$ 
4   while  $L \neq \emptyset$  do
5     pop  $w = (u, t)$  from  $L$  with minimal  $t$ 
6     for  $u' \in N^{\text{out}}_u$  do
7        $t' \leftarrow t + \tau(uu')$ ,  $w' \leftarrow (u', t')$ 
8       if  $\Delta_S(w') > 0$  then
9         insert  $w'$  into  $V(D_{\mathcal{W}}^{\uparrow})$ 
10        insert  $ww'$  into  $A(D_{\mathcal{W}}^{\uparrow})$ 
11        if  $\Delta_L(w') \leq 0$  then
12          insert  $w'$  into  $L$ 
13  return  $D_{\mathcal{W}}^{\uparrow}$ 

```

4.17 Lemma (Candidate graph). Algorithm 4.1 runs in $\mathcal{O}(|A D_S| \cdot H)$ time and the computed *candidate graph* $D_{\mathcal{W}}^{\uparrow}$ satisfies $D_{\mathcal{W}} \subseteq D_{\mathcal{W}}^{\uparrow}$. $\triangleleft$

Proof. For the time complexity, note that L can only contain nodes (u, t) with $0 \leq \underline{t}(u) \leq t \leq \bar{t}(u) < H$. Since we process the nodes with increasing t , no node is processed twice and the iterations of the inner for loop are thus bounded by $|A D_S| \cdot H$. Extracting the node (u, t) from L can be done in constant time by using a separate list for each of the H lengths in question. All other operations are easy to do in constant time, yielding the desired running time.

To see that $D_{\mathcal{W}} \subseteq D_{\mathcal{W}}^{\uparrow}$, let $W = u_0 \dots u_n \in \mathcal{W}$ and let $P_W = w_0 \dots w_n$. We show by induction that $P_W \subseteq D_{\mathcal{W}}^{\uparrow}$, which also shows $D_{\mathcal{W}} \subseteq D_{\mathcal{W}}^{\uparrow}$ since $W \in \mathcal{W}$ was arbitrary.

By Line 2, $w_0 \in D_{\mathcal{W}}^{\uparrow}$. Now let $w_i \in D_{\mathcal{W}}^{\uparrow}$ for $i < n$. Then $\Delta_L(w_i) \leq 0$ by Observation 4.3. Thus, w_i was added to L in some iteration of Algorithm 4.1 and, when it is removed, $u_{i+1} \in N^{\text{out}}_{w_i}$. Since $\Delta_S(w_{i+1}) > 0$ by Observation 4.3, the next node w_{i+1} as well as the arc $w_i w_{i+1}$ are in $D_{\mathcal{W}}^{\uparrow}$, too. $\square$

Note that, by construction, the nodes without incoming arcs are exactly the nodes in $\mathcal{S}' = \{w \in V(D_{\mathcal{W}}^{\uparrow}) : \Delta_S(w) = 0\}$. Similarly, a superset of $\mathcal{B}_{\mathcal{W}}$ is given by $\mathcal{B}' = \{w \in V(D_{\mathcal{W}}^{\uparrow}) : \Delta_L(w) > 0\}$. Note that in contrast to $D_{\mathcal{W}}$, there may be nodes $w \in D_{\mathcal{W}}^{\uparrow}$ with $\Delta_L(w) \leq 0$ that have no outgoing arcs.

We still need to determine which nodes and arcs are actually needed. We focus on the arcs, since if we can correctly determine these, then we can discard all isolated nodes afterwards (assuming that $\underline{t}(u) \leq \bar{t}(u)$ for all $u \in D_S$).

Note that an arc $ww' \in D_{\mathcal{W}}^{\uparrow}$ is part of $D_{\mathcal{W}}$ if and only if ww' is on a path P_W for some $W \in \mathcal{W}$. This path is also part of $D_{\mathcal{W}}^{\uparrow}$ and satisfies the properties listed in Lemma 4.4.

In particular, it starts in $\mathcal{S}'$ and ends in $\mathcal{B}'$. Since the lemma is a characterisation, we can also get the converse:

4.18 Corollary. An arc $ww' \in D_{\mathcal{W}}^\uparrow$ is in $D_{\mathcal{W}}$ if and only if there exists a $\mathcal{S}'\mathcal{B}'$ -path $P = w_0 \dots w_n$ in $D_{\mathcal{W}}^\uparrow$ that contains ww' and for which $\Delta_S(w_i) \geq \Delta_L(w_n)$ holds for all $i \in (n]$. $\triangleleft$

Proof. The forward implication follows from the preceding argument. For the converse, let $P = w_0 \dots w_n$ be a path in $D_{\mathcal{W}}^\uparrow$ that contains ww' and for which $\Delta_S(w_i) \geq \Delta_L(w_n)$ holds for all $i \in (n]$. Then W_P is a walk in D_S for which the properties of Lemma 4.4 are satisfied: the first is guaranteed by our construction already since we do not add outgoing arcs to nodes with positive lateness and $\Delta_L(w_n) > 0$ holds since $w_n \in \mathcal{B}'$. $\square$

We therefore need an efficient way of verifying whether an arc $ww' \in D_{\mathcal{W}}^\uparrow$ is on such a path. This means we need an $\mathcal{S}'w$ -path P_ℓ and a $w'\mathcal{B}'$ -path P_r that, when combined, satisfy the condition of the corollary. But this means that, for the prefix P_ℓ , a larger slack consumption is always better. Thus, we say that w is *s-reachable* if there exists an $\mathcal{S}'w$ -path in $D_{\mathcal{W}}^\uparrow$ whose nodes, except the first, all have slack consumption at least s . Thus, we want to compute the best such s , which we call

$$\text{cap}(w) = \sup \{s : w \text{ is } s\text{-reachable}\}.$$

In particular, $\text{cap}(w) = \infty$ for all $w \in \mathcal{S}'$ since they are s -reachable for any value of s .

For the suffix P_r , our goal is to be as unrestrictive as possible. Thus, lower values of $\Delta_L(w_n)$ for the end node $w_n \in \mathcal{B}'$ are desirable, as long as we can reach them by a path whose minimum slack consumption is at least this value. Similarly to the above, we call w' *ℓ -extendable* if there exists a $w'\mathcal{B}'$ -path in $D_{\mathcal{W}}^\uparrow$ ending at $w_n \in \mathcal{B}'$ such that the slack consumption of all its nodes is at least $\Delta_L(w_n) = \ell$. We define

$$\text{req}(w) = \min \{\ell : w \text{ is } \ell\text{-extendable}\}.$$

Here and below, we use the convention $\min \emptyset = +\infty$. In particular, $\text{req}(w) = \infty$ if $w \in \mathcal{S}'$.

These two properties let us characterise which arcs of $D_{\mathcal{W}}^\uparrow$ are actually part of $D_{\mathcal{W}}$.

4.19 Theorem. An arc $ww' \in A(D_{\mathcal{W}}^\uparrow)$ is in $D_{\mathcal{W}}$ if and only if

$$\text{req}(w') < \infty \quad \text{and} \quad \text{cap}(w) \geq \text{req}(w'). \quad (2)$$

$\triangleleft$

Proof. Similarly, if $ww' \in A(D_{\mathcal{W}}^\uparrow)$ is in $D_{\mathcal{W}}$, then Corollary 4.18 yields a path $P = w_0 \dots w_n$ in $D_{\mathcal{W}}^\uparrow$ that contains ww' and for which $\Delta_S(w_i) \geq \Delta_L(w_n)$ holds for all $i \in (n]$. In particular, w is $\Delta_L(w_n)$ -reachable and w' is $\Delta_L(w_n)$ -extendable, yielding $\text{cap}(w) \geq \text{req}(w')$ and $\text{req}(w') < \infty$.

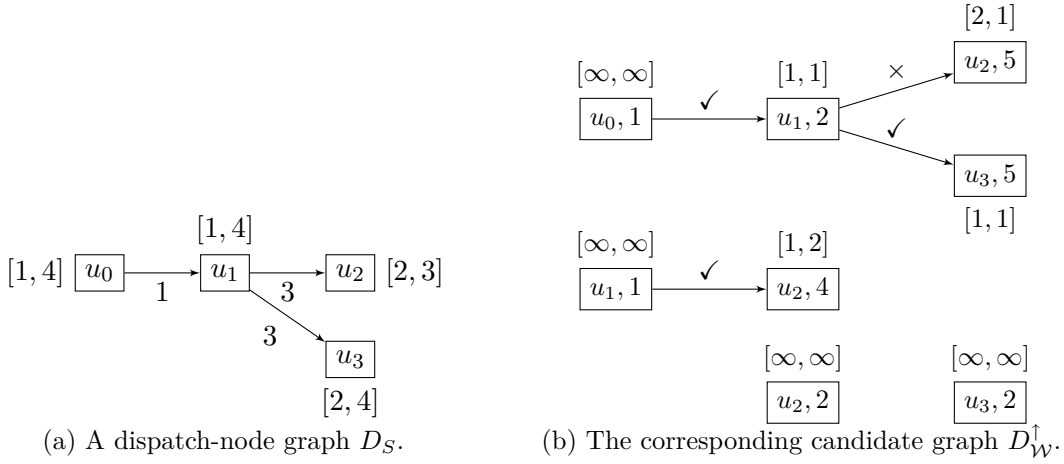


Figure 7: Illustration of the marking rule in Theorem 4.19. The nodes u in the dispatch-node graph are labelled with $[\underline{t}(u), \bar{t}(u)]$. In the candidate graph, the nodes w are labelled by $[\text{req}(w), \text{cap}(w)]$.

Conversely, assume Equation (2) holds for the arc ww' . Let $P_\ell = w_0 \dots w_k$, with $w_k = w$, be a path realising $\text{cap}(w)$ and $P_r = w_{k+1} \dots w_n$, with $w_{k+1} = w'$, be a path realising $\text{req}(w')$, which exists since the value is finite by assumption. Then $\Delta_S(w_i) \geq \Delta_L(w_n) = \text{req}(w')$ for all $i \geq k+1$ and $\Delta_S(w_i) \geq \text{cap}(w) \geq \text{req}(w') = \Delta_L(w_n)$ for $0 < i \leq k$. Hence, the path $P_\ell P_r$ is a $\mathcal{S}'\mathcal{B}'$ -path that contains ww' and satisfies the condition of Corollary 4.18, so $ww' \in D_W$. $\square$

Let us illustrate this on a small example.

4.20 Example. Consider the dispatch-node graph D_S in Figure 7a and the corresponding candidate graph $D_W^\uparrow$ in Figure 7b. The minimal too-long paths are $u_1 u_2$ and $u_0 u_1 u_3$: the path $u_1 u_3$ arrives at time 4 and is feasible, whereas $u_0 u_1 u_3$ arrives at time $5 > \bar{t}(u_3)$. The check of Theorem 4.19 keeps their time-expanded arcs and rejects the arc from $(u_1, 2)$ to $(u_2, 5)$. This also lets us remove the now isolated nodes $(u_2, 5)$, $(u_3, 2)$, and $(u_2, 2)$.

To complete the algorithm, we want to efficiently determine $\text{cap}(w)$ and $\text{req}(w)$.

4.21 Observation. The following recurrences determine $\text{cap}(w)$ and $\text{req}(w)$.

$$\text{cap}(w) = \begin{cases} \infty & w \in \mathcal{S}' \\ \min \{ \Delta_S(w), \max_{w' \in N^{\text{in}} w} \text{cap}(w') \} & w \notin \mathcal{S}' \end{cases} \quad (3)$$

$$\text{req}(w) = \begin{cases} \Delta_L(w) & w \in \mathcal{B}' \\ \min \{ \text{req}(w') : w' \in N^{\text{out}} w, \Delta_S(w) \geq \text{req}(w') \} & w \notin \mathcal{B}' \end{cases} \quad (4)$$

Using these, we can compute the values of $\text{cap}(w)$ and $\text{req}(w)$ for all $w \in V(D_W^\uparrow)$ in $\mathcal{O}(|V(D_W^\uparrow)| + |A(D_W^\uparrow)|)$ time. $\triangleleft$

Algorithm 4.2: Computing D_W .

```

1 def CONSTRUCTTIMEEXPANDEDGRAPH( $D_S$ ):
2    $D_W^\uparrow \leftarrow \text{CONSTRUCTCANDIDATEGRAPH}(D_S)$ 
3   Compute  $\text{cap}(w)$  and  $\text{req}(w)$  for all  $w \in D_W^\uparrow$  using (3) and (4)
4    $D_W \leftarrow (\emptyset, \emptyset)$ 
5   for  $ww' \in A(D_W^\uparrow)$  do
6     if  $\text{req}(w') < \infty$  and  $\text{cap}(w) \geq \text{req}(w')$  then
7       Add  $w$ ,  $w'$ , and  $ww'$  to  $D_W$ 
8   return  $D_W$ 

```

Proof. We start with the recurrence for $\text{cap}(w)$ and inductively obtain that the formula is correct, where this is true for all nodes in $\mathcal{S}'$. For the remaining nodes, $\text{cap}(w)$ is determined by a path to w whose nodes have a minimum slack consumption that is as large as possible. Thus, it uses a predecessor w' of maximum $\text{cap}(w')$, without loss of generality, and $\text{cap}(w) = \min \{\text{cap}(w'), \Delta_S(w)\}$.

For $\text{req}(w)$, we again see that the value for $w \in \mathcal{B}'$ is set correctly since $\Delta_S(w) \geq \Delta_L(w)$. For the remaining nodes, the path determining $\text{req}(w)$ (if it exists) uses a successor w' that minimises $\text{req}(w')$ subject to $\Delta_S(w) \geq \text{req}(w')$, and its requirement is then identical. If $\text{req}(w) = \infty$, then any path to a node in $\mathcal{B}'$ has insufficient slack consumption; in particular, $\Delta_S(w) < \text{req}(w')$ for all successors w' .

Since for either recurrence each node and arc is processed exactly once, the procedure completes in $\mathcal{O}(|V(D_W^\uparrow)| + |A(D_W^\uparrow)|)$ time. $\square$

To summarise this section, we obtain Algorithm 4.2 that constructs D_W . Evidently, computing $\text{cap}(w)$ and $\text{req}(w)$ requires no more operations than determining $D_W^\uparrow$ in the first place, so with Lemma 4.17, Theorem 4.19, and Observation 4.21, Algorithm 4.2 computes D_W in $\mathcal{O}(|A D_S| \cdot H)$ time.

We further improve the practical running time of Algorithm 4.2 by preprocessing D_S to remove nodes and arcs that cannot lie on too-long paths. The idea is to compute for each $u \in V D_S$ the latest real arrival time of any walk in D_S , i.e.,

$$t_{\max}(u) = \max \{t_W(n) : \exists W = u_0 \dots u_n, u_n = u\},$$

and similarly the earliest real arrival time a walk must have at u to be extendable into a too-long path, i.e.,

$$t_{\min}(u) = \min \{\bar{t}(u_n) - \tau(W) : \exists W = u_0 \dots u_n, u_0 = u\}.$$

Evidently, if $t_{\max}(u) \leq t_{\min}(u)$, it cannot lie on a too-long path and can be removed. If D_S is acyclic, these values are easy to compute by forward and backward propagation. Otherwise, we can first determine the condensation graph of D_S by merging every strongly connected component into a single node, and then determine these values on the resulting graph. Every node in a strongly connected component containing a cycle is on a too-long path in any case and thus we set $t_{\min}(u) = -\infty$ and $t_{\max}(u) = \infty$ for them.

4.5. Solving the ORP using $D_{\mathcal{W}}$

Similarly to Lemma 4.6, we now prove another alternative characterisation that lends itself better to the formulation of an IP.

4.22 Lemma. All minimal too-long paths are $\mathcal{T}$ -relaxed too-long if and only if there exist time points $t(w'w) \in \mathcal{T}_{w'w}$ for all $w'w \in A(D_{\mathcal{W}})$ and $\pi(w)$ for all $w \in V(D_{\mathcal{W}})$ such that

- (a) $\pi(w) \leq \underline{t}(w)$ for all $w \in \mathcal{S}_{\mathcal{W}}$,
- (b) $t(w'w) \leq \pi(w')$ for all $w'w \in A(D_{\mathcal{W}})$,
- (c) $t(w'w) + \tau(w'w) \geq \pi(w)$ for all $w'w \in A(D_{\mathcal{W}})$, and
- (d) $\pi(w) > \bar{t}(w)$ for all $w \in \mathcal{B}_{\mathcal{W}}$.

Additionally, for $w'w \in A(D_{\mathcal{W}})$ with $w' = (u', t')$, $t(w'w) \in I_{w'w} = (t' - \text{req}(w), t']$. $\triangleleft$

Proof. Note that all minimal too-long paths are $\mathcal{T}$ -relaxed too-long if and only if the relaxed arrival times π_w at the boundary nodes $w \in \mathcal{B}_{\mathcal{W}}$ satisfy $\pi_w > \bar{t}(w)$, by Theorem 4.15.

For the forward implication, we use $\pi(w) = \pi_w$ and $t(w'w) = \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'w}}$. Then, for $w \in \mathcal{S}_{\mathcal{W}}$, $\pi(w) = \pi_w = \underline{t}(w)$ and, for $w \in \mathcal{B}_{\mathcal{W}}$, $\pi(w) = \pi_w > \bar{t}(w)$. For $w'w \in A(D_{\mathcal{W}})$, $t(w'w) = \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'w}} \leq \pi_{w'} = \pi(w')$ and $t(w'w) + \tau(w'w) = \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'w}} + \tau(w'w) \geq \pi_w = \pi(w)$.

For the converse, let t and π satisfy the four specified properties. Then, inductively, $\pi(w) \leq \pi_w$ for all $w \in V(D_{\mathcal{W}})$, which implies that $\pi_w \geq \pi(w) > \bar{t}(w)$ for all $w \in \mathcal{B}_{\mathcal{W}}$. For $w \in \mathcal{S}_{\mathcal{W}}$ this is clear and for the other nodes w we note that Property (b) implies that $t(w'w) \leq \lfloor \pi(w') \rfloor_{\mathcal{T}_{w'w}}$ since $t(w'w) \in \mathcal{T}_{w'w}$. Using this, we obtain $\pi(w) \leq t(w'w) + \tau(w'w) \leq \lfloor \pi(w') \rfloor_{\mathcal{T}_{w'w}} + \tau(w'w) \leq \lfloor \pi_{w'} \rfloor_{\mathcal{T}_{w'w}} + \tau(w'w)$. Since this holds for all predecessors w' of w , $\pi(w) \leq \pi_w$.

For the final part, let $w'w \in A(D_{\mathcal{W}})$ with $w' = (u', t')$. Choose $W \in \mathcal{W}$ of minimum lateness ℓ among those whose time-expanded path $P_W = w_0 \dots w_n$ contains $w'w$. Inductively, $t(w'w) \leq \pi(w') \leq \pi(w_0) + \tau(P_W w') \leq \underline{t}(w_0) + \tau(P_W w') = t'$.

For the other direction, $t(w'w) \geq \pi(w_n) - \tau(w'P_W) > \bar{t}(w_n) - \tau(w'P_W) = t' - \Delta_L(w_n) = t' - \ell$. By Observation 4.3, the suffix wP_W shows that w is ℓ -extendable. It cannot be ℓ' -extendable for some $\ell' < \ell$ since, otherwise, we can extend the prefix $P_W w$ by that suffix, preserving $w'w$, to obtain a minimal too-long path of lower lateness by Lemma 4.4. Thus, $\text{req}(w) = \ell$ and $t(w'w) \in (t' - \text{req}(w), t']$. $\square$

Based on this reformulation, we can solve ORP with the mixed-integer program (5). It contains continuous variables π_w to decide $\pi(w)$ and binary variables $x_{w'wt}$ which represent the desired time points $t(w'w)$. Let I_a denote the union of all intervals $I_{w'w}$

where the underlying arc $a(w'w)$ is a .

$$\min \quad \sum_{a \in A} \sum_{t \in I_a} y_{at} \quad (5a)$$

$$\text{s.t.} \quad \sum_{t \in I_{w'w}} x_{w'wt} = 1 \quad \forall w'w \in A(D_{\mathcal{W}}) \quad (5b)$$

$$x_{w'wt} \leq y_{a(w'w)t} \quad \forall w'w \in A(D_{\mathcal{W}}), t \in I_{w'w} \quad (5c)$$

$$\sum_{t \in I_{w'w}} tx_{w'wt} \leq \pi_{w'} \quad \forall w'w \in A(D_{\mathcal{W}}) \quad (5d)$$

$$\sum_{t \in I_{w'w}} tx_{w'wt} + \tau(a(w'w)) \geq \pi_w \quad \forall w'w \in A(D_{\mathcal{W}}) \quad (5e)$$

$$\pi_w = t \quad \forall w = (u, t) \in \mathcal{S} \quad (5f)$$

$$\pi_w \geq \bar{t}(u) + 1 \quad \forall w = (u, t) \in \mathcal{B} \quad (5g)$$

$$y_{at} \in \{0, 1\} \quad \forall a \in A, t \in I_a \quad (5h)$$

$$x_{w'wt} \in \{0, 1\} \quad \forall w'w \in A(D_{\mathcal{W}}) \quad (5i)$$

$$\pi_{(u,t)} \geq 0 \quad \forall (u, t) \in V(D_{\mathcal{W}}) \quad (5j)$$

While this model can be significantly more compact than (1), its size is still strongly dependent on the scaling of the time parameters. So it seems promising to use dynamic discretisation discovery to solve ORP as well, which is the topic of the next subsection.

4.6. A Dynamic Discretisation Discovery Algorithm for the ORP

Like before, we want to use DDD to replace the time points by intervals in order to obtain a smaller model. Thus, we again use a time discretisation on the arcs of the original network, which we denote by $\mathcal{D}$ here to distinguish it from our previous ones. The relaxation will, once again, use the intervals obtained by the discretisation and allow behaviours that are realisable for some point in the interval.

We shall be fairly brief in this section since none of the ideas used here are really different from what we did for the service network design problem. When differences appear, we point them out.

To get started, let us state the problem and the relaxation we are trying to solve by using the reformulation Lemma 4.22 (or the IP (5)). The objective of the ORP is to add as few time points as possible, so our goal is to ‘consolidate’ as many arcs $w'w$ that correspond to the same arc $a(w'w)$ as possible and ‘solve’ them by adding a single time point. Thus, a solution is a partition Φ of the arcs $A(D_{\mathcal{W}})$ such that all arcs $w'w$ in a part $\phi \in \Phi$ correspond to the same arc $a(w'w)$. We write $\phi(w'w)$ for the part $\phi \in \Phi$ containing $w'w$.

For feasibility, we need time points $t: A(D_{\mathcal{W}}) \rightarrow \mathbb{N}$ and $\pi: V(D_{\mathcal{W}}) \rightarrow \mathbb{N}$ that satisfy the conditions of Lemma 4.22. Since we want all arcs in the same part of Φ to have the same t -value, it suffices to specify time points for each part of the partition.

Hence, Φ is feasible if time points $t: \Phi \rightarrow \mathbb{N}$ and $\pi: V(D_{\mathcal{W}}) \rightarrow \mathbb{N}$ exist such that

- (a) $\pi(w) \leq \underline{t}(w)$ for all $w \in \mathcal{S}_{\mathcal{W}}$,
- (b) $t(\phi(w'w)) \leq \pi(w')$ for all $w'w \in A(D_{\mathcal{W}})$,
- (c) $t(\phi(w'w)) + \tau(w'w) \geq \pi(w)$ for all $w'w \in A(D_{\mathcal{W}})$, and
- (d) $\pi(w) > \bar{t}(w)$ for $w \in \mathcal{B}_{\mathcal{W}}$.

Recall that these conditions imply that $t(\phi(w'w)) \in I_{w'w}$.

In the relaxation R-ORP, we use the time discretisation $\mathcal{D}$ and assign each part ϕ of the partition Φ an interval $h(\phi) \in \mathcal{D}_{a(\phi)}^I$, where $a(\phi)$ is the arc in D corresponding to ϕ . We now require that $h(\phi(w'w)) \cap I_{w'w} \neq \emptyset$ for all $w'w \in A(D_{\mathcal{W}})$ together with

- (a) $\pi(w) \leq \underline{t}(w)$ for all $w \in \mathcal{S}_{\mathcal{W}}$,
- (b) $\min h(\phi(w'w)) \leq \pi(w')$ for all $w'w \in A(D_{\mathcal{W}})$,
- (c) $\max h(\phi(w'w)) + \tau(w'w) \geq \pi(w)$ for all $w'w \in A(D_{\mathcal{W}})$, and
- (d) $\pi(w) > \bar{t}(w)$ for $w \in \mathcal{B}_{\mathcal{W}}$.

Clearly, this is a relaxation (simply assign the interval containing a time point) and it is tight for the full discretisation (then the interval minimum and maximum are just the same single point). To solve it, we can use a mixed-integer program similar to (5), selecting intervals instead of time points and replacing the coefficient t in the linking constraints by the upper and lower bounds of the intervals.

Now we need to find our ‘problems’, so an equivalent to the too-long paths in the dispatch-node graph, and how to refine the discretisation to eliminate them, so an equivalent of making them $\mathcal{T}$ -relaxed too-long. We note that an analogous construction to the one we presented before would be plausible, though we deviate slightly from it. Before we describe how we proceed in this setting, we want to describe the approach we take now in terms of the SND.

In the SND, we could have defined a different graph to verify implementability than the dispatch-node graph: it would have also been possible to construct a *consolidation graph* that has a node for every consolidation and connects consolidations that appear consecutively on a path. Then analogous definitions of ($\mathcal{T}$ -relaxed) too-long paths can be derived as well as the corresponding characterisations. The main difference between these two options is that the consolidation graph has a smaller size, but also ‘solves’ fewer problems, resulting in potentially more iterations. For the SND, the dispatch-node graph was computationally advantageous, but for the ORP, the graph $D_{\mathcal{W}}$ is significantly larger, so we opt for the more compact representation given by the consolidation graph.

Since the consolidations correspond to parts of the partition here, the graph defined below has these as its nodes.

4.23 Definition (Partition-linking graph). Given a solution Φ , its *partition-linking graph* D_{Φ} is given by

$$VD_{\Phi} = \Phi$$

$$A D_{\Phi} = \{\phi(w_1 w_2) \phi(w_2 w_3) \mid w_1 w_2, w_2 w_3 \in A(D_{\mathcal{W}})\}.$$

For arcs $\phi' \phi$ we set $\tau(\phi' \phi) = \tau(a(\phi'))$ and for nodes ϕ we set

$$\underline{t}(\phi) = \max \{\min I_{w'w} : w'w \in \phi\}, \quad \bar{t}(\phi) = \min \{\max I_{w'w} : w'w \in \phi\}. \quad \triangleleft$$

Note that in a feasible solution of ORP with time points t and π , $t(\phi) \in I_{w'w}$ for all $w'w \in \phi$, so $t(\phi) \in [\underline{t}(\phi), \bar{t}(\phi)]$. Similarly, in a feasible solution of R-ORP with time points h and π , $h(\phi) \cap I_{w'w} \neq \emptyset$ for all $w'w \in \phi$, so $\max h(\phi) \geq \underline{t}(\phi)$ and $\min h(\phi) \leq \bar{t}(\phi)$.

Since we now want to be sufficiently late at the boundary nodes, the obstacle to being a feasible solution is that paths in the partition-linking graph might be too short.

4.24 Definition. A $\phi' \phi$ -path P in D_{Φ} is *too-short* if $\bar{t}(\phi') + \tau(P) < \underline{t}(\phi)$. $\triangleleft$

These too-short paths are the obstruction to implementability.

4.25 Theorem. A solution Φ is feasible for the ORP if and only if D_{Φ} contains no too-short path. $\triangleleft$

Proof. Let Φ be feasible with time points t and π . Moreover, let $P = \phi_1 \dots \phi_n$ be a path in D_{Φ} . For any arc $\phi_i \phi_{i+1}$ we get nodes $w_1, w_2, w_3 \in V(D_{\mathcal{W}})$ such that $\phi_i = \phi(w_1 w_2)$ and $\phi_{i+1} = \phi(w_2 w_3)$. By feasibility, $t(\phi_i) = t(\phi(w_1 w_2)) \geq \pi(w_2) - \tau(w_1 w_2) \geq t(\phi(w_2 w_3)) - \tau(a(\phi_i)) = t(\phi_{i+1}) - \tau(\phi_i \phi_{i+1})$. Thus, $\bar{t}(\phi_1) + \tau(P) \geq t(\phi_1) + \tau(P) \geq t(\phi_n) \geq \underline{t}(\phi_n)$, so P is not too-short.

Conversely, let D_{Φ} contain no too-short path. In this case, we can define $t(\phi) = \min \{\bar{t}(\phi') + d_{\tau}(\phi', \phi) : \phi' \in \Phi\} \geq \underline{t}(\phi)$ for $\phi \in \Phi$. We also set $\pi(w) = \underline{t}(w)$ for $w \in \mathcal{S}_{\mathcal{W}}$, and $\pi(w) = \min \{t(\phi(w'w)) + \tau(w'w) : w' \in N^{\text{in}} w\}$. The triangle inequality yields the following helpful inequality: for $\phi' \phi \in A D_{\Phi}$

$$t(\phi) \leq t(\phi') + \tau(\phi' \phi). \quad (6)$$

Using these values, $\pi(w) = \underline{t}(w)$ for $w \in \mathcal{S}_{\mathcal{W}}$. Next, let $w'w \in A D_{\mathcal{W}}$ and $\phi = \phi(w'w)$. If $w' \in \mathcal{S}_{\mathcal{W}}$, we get $t(\phi) \leq \bar{t}(\phi) \leq \max I_{w'w} = \underline{t}(w') = \pi(w')$. Otherwise, $\pi(w') = t(\phi(w''w')) + \tau(w''w')$ for some $w'' \in N^{\text{in}} w'$. Let $\phi(w''w') = \phi'$. Then $\phi' \phi \in A D_{\Phi}$, giving us $t(\phi) \leq t(\phi') + \tau(\phi' \phi) = \pi(w')$, by (6). Also, $\pi(w) \leq t(\phi) + \tau(w'w)$ by definition.

Finally, for $w = (u, t) \in \mathcal{B}_{\mathcal{W}}$, we get a $w' = (u', t') \in N^{\text{in}} w$ with $\pi(w) = t(\phi(w'w)) + \tau(w'w) \geq \underline{t}(\phi(w'w)) + \tau(w'w) \geq \min I_{w'w} + \tau(w'w) > t' + \tau(w'w) - \text{req}(w) = t - \Delta_{\text{L}}(w) = \bar{t}(w)$. $\square$

Just as for refining the discretisation $\mathcal{T}$ for R-SND, we have many options available for refining the discretisation $\mathcal{D}$ for R-ORP. In fact, we could transfer the concepts we developed to solve ORP to finding small discretisations $\mathcal{D}$ for R-ORP. But in preliminary experiments we found that empirically, a very simple initial discretisation was always

sufficient that the R-ORP solution was also feasible for ORP. So we simply describe this initialisation and a sufficient refinement approach (which was never necessary in our experiments).

We initialise $\mathcal{D}_a = \bigcup_{w'w \in A D_{\mathcal{W}}: a(w'w)=a} \{\min I_{w'w}, \max I_{w'w} + 1\}$, i.e., by splitting at the interval endpoints for all $w'w$ that need to select a time point on arc a . If the solution of R-ORP is nonetheless infeasible for ORP, we could find (e.g., by enumeration) one or more too-short paths and then add time points according to the following statement, which is conceptually equivalent to Theorem 4.8:

4.26 Lemma. Let Φ be a solution to ORP, $P = \phi_1 \dots \phi_n$ be a too-short path in D_Φ , and $t_i = \bar{t}(\phi_1) + \tau(P\phi_i)$. If $\mathcal{D}$ is a discretisation with $t_i + 1 \in \mathcal{D}_{a(\phi_i)}$ for $i \in (n]$, then Φ is not feasible for R-ORP. $\triangleleft$

Proof. Suppose Φ was feasible for R-ORP, then we obtain intervals $h(\varphi) \in \mathcal{D}_{a(\phi)}^I$ that satisfy all five conditions in the definition of the relaxation. For any arc $\phi_i\phi_{i+1}$ we get nodes $w_1, w_2, w_3 \in V(D_{\mathcal{W}})$ such that $\phi_i = \phi(w_1w_2)$ and $\phi_{i+1} = \phi(w_2w_3)$. By feasibility, $\max h(\phi_i) = \max h(\phi(w_1w_2)) \geq \pi(w_2) - \tau(w_1w_2) \geq \min h(\phi(w_2w_3)) - \tau(a(\phi_i)) = \min h(\phi_{i+1}) - \tau(\phi_i\phi_{i+1})$.

Note that $\min h(\phi_1) \leq \bar{t}(\phi_1) = t_1$ and $t_1 + 1 \in \mathcal{D}_{a(\phi_1)}$, so $\max h(\phi_1) \leq t_1$ as well. Now, inductively, $\min h(\phi_{i+1}) \leq \max h(\phi_i) + \tau(\phi_i\phi_{i+1}) \leq \bar{t}(\phi_1) + \tau(P\phi_{i+1}) = t_{i+1}$. Again, since $t_{i+1} + 1$ is in the discretisation of $a(\phi_{i+1})$, we get $\max h(\phi_{i+1}) \leq t_{i+1}$. In particular, $\max h(\phi_n) \leq t_n < \underline{t}(\phi_n) \leq \max h(\phi_n)$, which is a contradiction. $\square$

5. DISCUSSION OF SOME MODIFICATIONS

In this section, we discuss several modifications to the proposed framework and how our results carry over. The first modification is a slightly different definition of (relaxed) too-long paths, the second a stronger relaxed problem, and the third an initialisation strategy for the discretisation.

5.1. Alternative ($\mathcal{T}$ -Relaxed) Too-Long Criteria

The following definition is similar to the elimination criterion in [SXB26, Lemma 2], though slightly different since we again do not restrict too-long paths to start at commodity origins.

5.1 Definition (($\mathcal{T}$ -Relaxed) too-long* paths). Let uu' be an arc in D_S with $u = (v, k)$, and $u' = (v', k')$. We define

$$\underline{t}(uu') = \max \{\underline{t}_k(v), \underline{t}_{k'}(v')\}, \quad \bar{t}(uu') = \min \{\bar{t}_k(v), \bar{t}_{k'}(v')\}.$$

We also say that a walk $W = u_0 \dots u_n$ in D_S is a *too-long** path if $\underline{t}(u_0u_1) + \tau(W) > \bar{t}(u_{n-1}u_n)$.

Similarly, we change the initialisation of the relaxed arrival times and set $\pi'_W(0) = \underline{t}(u_0u_1)$ and say W is $\mathcal{T}$ -relaxed too-long* if $\pi'_W(n) > \bar{t}(u_{n-1}u_n)$. $\triangleleft$

Note that we continue to assume that single-node too-long paths do not exist, so this new definition makes sense. Essentially, this definition makes use of the observation that if two commodities are consolidated on an arc, they must also both traverse that arc, and we can disregard the less restrictive commodity.

5.2 Example. Consider the partial solution in Figure 8. Assume other paths exist between the nodes such that the unspecified values of $\underline{t}_k(v)$, $\bar{t}_k(v)$ are irrelevant. There exist two (minimal) too-long paths: $(v_1, k_1)(v_2, k_1)(v_3, k_1)$ and $(v_1, k_1)(v_2, k_1)(v_3, k_2)$. Assume the first arc is fully discretised in $\mathcal{T}$; then under our definition, we require one time point in $[2, 6]$ on the second arc to make the first path $\mathcal{T}$ -relaxed too-long and one time point in $[4, 6]$ to make the second path $\mathcal{T}$ -relaxed too-long. In contrast, we require only a time point in $[2, 6]$ to make both paths $\mathcal{T}$ -relaxed too-long*. Evidently, this is less restrictive and, depending on the other too-long paths, a smaller discretisation suffices under this definition.

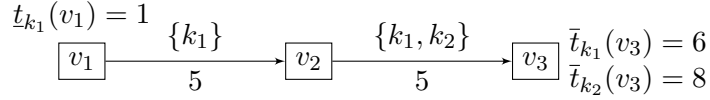


Figure 8: Example (partial) solution to demonstrate that fewer time points might suffice.

Any ($\mathcal{T}$ -relaxed) too-long path is also a ($\mathcal{T}$ -relaxed) too-long* path. The statements regarding implementable and representable solutions (Theorems 3.7 and 3.14) carry over, so we can also transfer the path-based refinement strategies. It is also possible to construct a graph similar to D_W and show a statement corresponding to Theorem 4.15. But because the modified elimination criterion considers the minimum over the last two commodities, the graph construction is more complicated. Preliminary experiments indicated that using this stronger criterion has little impact on the resulting discretisations and overall running times, so for clarity of exposition we use the simpler criterion.

5.2. Stronger Relaxation

Next, we discuss a possible modification to the relaxed problem R-SND. For a given arc a and interval I we define the part of the interval in which a commodity k can actually be dispatched by $J_k(a, I) = I \cap I_k(a)$. Then, we can replace Property (b) for a representable solution by the more restrictive condition

$$\min J_k(a_i, h(c_i)) + \tau(a_i) \leq \max J_k(a_{i+1}, h(c_{i+1})) \quad \forall i \in (n).$$

This essentially ensures that commodity k cannot be dispatched on a in I and immediately after on a' in I' if such a transfer is not possible while respecting the commodity's dispatch

windows, even if a dispatch in either interval by itself is possible. We also describe how this modified problem can be modelled as an integer program in Appendix A. The commodity-specific subnetworks used for the relaxations in recent works [Hel26, Mar+21, SXB26] reflect this rule for allowed arc sequences. Evidently, this modified relaxation is more restrictive and every solution that is not representable for R-SND is also not representable under the modification. So all of our refinement strategies are directly applicable when using this relaxation as well.

Similarly, when discussing DDD for the optimal refinement problem in Section 4.6, we could have also strengthened the relaxation by not only requiring $J_{w'w} = h(\phi(w'w)) \cap I_{w'w} \neq \emptyset$, but also using this more restrictive interval in Properties (b) and (c). These would then become

- (b) $\min J_{w'w} \leq \pi(w')$ for all $w'w \in A(D_{\mathcal{W}})$ and
- (c) $\max J_{w'w} + \tau(w'w) \geq \pi(w)$ for all $w'w \in A(D_{\mathcal{W}})$.

In our computational experiments, we use these stronger relaxations when solving both SND and ORP with DDD.

5.3. Initial Discretisation

Lastly, we discuss how the significant time points of Shu, Xu, and Baldacci [SXB26] can be integrated into our refinement approaches. For an arc $a \in A D$ we define the set of commodities that can traverse this arc in a feasible solution by $\mathcal{K}_a = \{k \in \mathcal{K} : \underline{t}_k(a) \leq \bar{t}_k(a)\}$ and then observe that some infeasible consolidations are easy to avoid:

5.3 Theorem (Significant time points, [SXB26, Theorem 1]).

Given an arc $uv \in A D$ and two different commodities $k_1, k_2 \in \mathcal{K}_{uv}$ with $\underline{t}_{k_2}(u) + \tau(uv) > \bar{t}_{k_1}(v)$, if there exists a time point $t \in \mathcal{T}_{uv}$ with $t \in (\bar{t}_{k_1}(v) - \tau(uv), \underline{t}_{k_2}(u)]$, then k_1 and k_2 cannot be consolidated in any $\mathcal{T}$ -representable solution. $\triangleleft$

By adding such time points to the discretisation, we essentially directly eliminate all single-arc minimal too-long paths. Shu, Xu, and Baldacci [SXB26] propose solving a minimum hitting set problem to identify the smallest set of time points that eliminate all these paths and use this set to initialise the discretisation.

Specifically, for each arc uv we want a minimum-size set of time points that hits all intervals in $\mathcal{I}_{uv} = \{(\bar{t}_{k_1}(v) - \tau(uv), \underline{t}_{k_2}(u)) : k_1, k_2 \in \mathcal{K}_{uv}, \underline{t}_{k_2}(u) + \tau(uv) > \bar{t}_{k_1}(v)\}$. We could require these time points to remain in the discretisation during refinement, or instead add all corresponding one-arc paths to the set $\mathcal{W}$ for the path-based approaches or into the graph $D_{\mathcal{W}}$. Since these paths can be quite numerous, we instead propose the following approach (for each arc). First, we solve the hitting set problem, obtaining a set of time points T_{uv} . We then assign each interval I to one of the selected points it contains, which we call $t(I)$. Evidently, we could replace a $t \in T_{uv}$ with any $t' \in I(t) = \bigcap_{I \in \mathcal{I}_{uv} : t(I)=t} I$ and still hit every interval. So in the initial discretisation, we simply set $\mathcal{T}_{uv} = \{0, H\} \cup T_{uv}$ and during refinement, we add a constraint to our integer programs that the discretisation

Table 1: Characteristics of the SND instance sets used.

Source	Class	Instances	Commodities	Nodes	Arcs
[Bol+17]	HC/HF	177	100–400	20–30	228–683
	HC/LF	183	100–400	20–30	229–686
[VK24]	Critical times	192	150–200	20	230–300
	Designated paths	192	150–300	20–25	230–480
	Hub-and-spoke (easy)	192	100	20	45–95
[Hel26]	Grid	100	50	16	48
	Hub-and-spoke (hard)	100	75	30	120–204
	Star	100	100	21	40
	Wheel	100	100	12	44

needs to include at least one $t' \in I(t)$ for each $t \in T_{uv}$. Essentially, we allow the significant time points to be repositioned to help eliminate newly found conflicts.

6. COMPUTATIONAL STUDY

We present the results of two groups of experiments. First, we demonstrate the efficiency of our exact methods for solving instances of the optimal refinement problem and of our algorithm for constructing $D_{\mathcal{W}}$ in Section 6.1. Then, we investigate the benefit of optimal refinement for the overarching DDD approach when solving SND in Section 6.2.

We implemented all algorithms in Python 3.11.4 and solved all mixed-integer programs with Gurobi 12.0.0 [Gur26]. The implementation and data will be made publicly available upon acceptance of this manuscript. All experiments were conducted on machines with a Xeon L5630 Quad Core 2.13 GHz processor.

We utilise the instance sets for the SND proposed by Boland et al. [Bol+17] (and classified in [Mar+21]), Van Dyk and Koenemann [VK24], and Helber [Hel26]. Table 1 gives an overview of the size of these instances. For more details we refer to the corresponding papers. Note that we omit the LC/LF and LC/HF instances of Boland et al. [Bol+17], since almost all are solved in a single iteration with state-of-the-art DDD solvers, so no refinement takes place.

6.1. Comparing Solvers for ORP

To generate instances of ORP, we ran the DDD solver for one iteration on each instance with a time limit of one hour and a relative optimality-gap limit of 1 % and stored the dispatch-node graph D_S of the resulting relaxation solution S . The initial discretisation for this first iteration contained the significant time points as described by Shu, Xu, and Baldacci [SXB26]. We made this choice to obtain ORP instances that are representative

Table 2: Characteristics of derived ORP instance sets.

Source	Class	Instances	Nodes	Arcs	$ \mathcal{W} \geq 10^5$
[Bol+17]	HC/HF	177	298–1261	317–4041	0
	HC/LF	149	273–1242	260–2982	0
[VK24]	Critical times	188	455–732	1143–4001	0
	Designated paths	192	490–1296	852–8196	51
	Hub-and-spoke (easy)	158	293–476	1091–2756	7
[Hel26]	Grid	87	151–204	259–524	0
	Hub-and-spoke (hard)	84	250–311	440–729	0
	Star	100	284–299	482–653	0
	Wheel	100	281–316	520–712	0

of those encountered in state-of-the-art DDD approaches.

Table 2 reports the characteristics of the resulting ORP instances. About 90 % of graphs contained fewer than 10 000 minimal too-long paths, about 4 % contained between 10 000 and 50 000, about 1 % contained between 50 000 and 100 000, but about 5 % contained more than 100 000. For graphs with more than 100 000 minimal too-long paths, enumerating them almost always exceeded the memory limit (16 GB), likely due to exponential blow-up. We therefore capped enumeration at 100 000 minimal too-long paths for approaches where this is relevant; the last column reports the number of instances that hit this cap in each class.

Note that the instance counts are lower for some classes than the number of available SND instances. We excluded 30 of the (easy) hub-and-spoke instances because they had arcs with $\tau(a) = 0$, which may lead to infinitely many minimal too-long paths. For the omitted grid and hard hub-and-spoke instances, the solver ran into the one-hour time limit while solving the first relaxation. The remaining instances were omitted because the dispatch-node graph obtained after the first iteration contained no too-long paths and was thus not relevant for the ORP.

We first compare the efficiency of Algorithm 4.2 for deriving $D_{\mathcal{W}}$ from D_S to the naive approach of first enumerating all minimal too-long paths and then merging them. The box plot in Figure 9 summarises the distribution of running times we observed for instances in each class. We observe that for most classes, the mean and the quartiles are similar between the two construction methods. Recall that most of our dispatch-node graphs contain (significantly) fewer than 10 000 minimal too-long paths, so it is not surprising that enumerating them is not a large burden. But for some classes, we observe that path-based construction may lead to outliers with significantly longer running times. Importantly, due to our enumeration limit of 100 000 paths, the path-based approach did not create the full $D_{\mathcal{W}}$ on the 58 instances in which it ran into the limit, while our graph-based algorithm was able to create the full graph every time. So our time-expanded too-long path graph allows us to represent all conflicts even when the path enumeration

balloons past the memory limit.

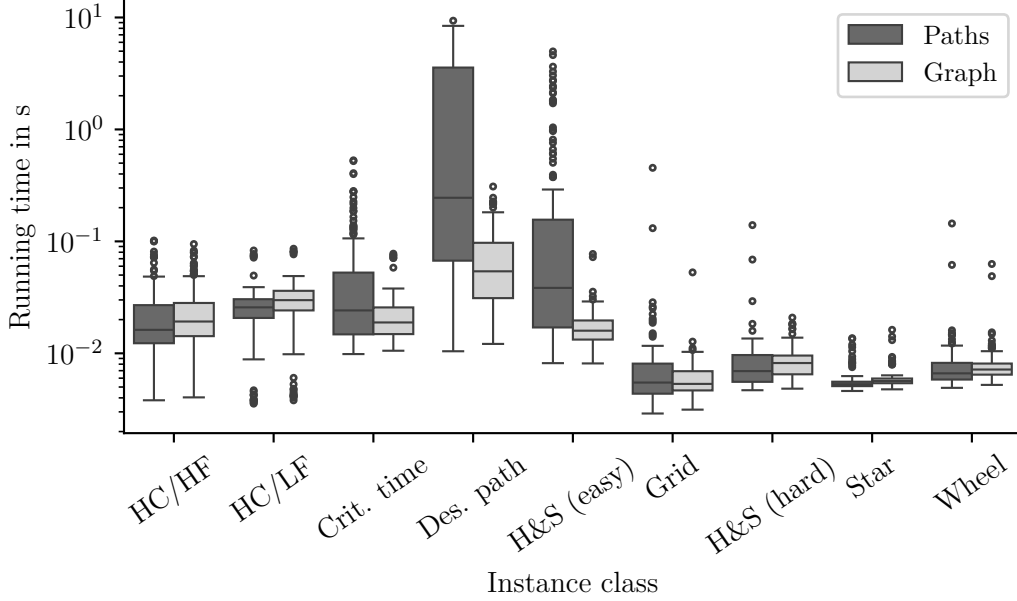


Figure 9: Time to create $D_{\mathcal{W}}$ by enumerating and merging all minimal too-long paths (Paths) or by our Algorithm 4.2 (Graph).

Next, we compare the efficacy of various approaches for solving the ORP. As a baseline (BL) heuristic, we add for each enumerated minimal too-long path time points according to its real arrival time, which by Theorem 4.8 will eliminate it. We also use the matheuristic (MH) of Shu, Xu, and Baldacci [SXB26] adapted to the arc-based discretisation. Recall that this heuristic uses their stronger elimination criterion and so in principle could find smaller discretisations than our approaches. For exact optimisation, we use the path-based MIP Equation (1) (Path), the graph-based MIP Equation (5) (Graph), and our graph-based DDD approach (DDD). Note that Path is only exact for instances where all minimal too-long paths could be enumerated.

Table 3 reports the mean running time and solution quality per class. The running time also includes the time for enumerating too-long paths or constructing $D_{\mathcal{W}}$. Recall that for some instances, not all minimal too-long paths were enumerated; in principle, the path-based approaches (BL, MH, Path) may produce solutions for these instances that do not eliminate all paths (and thus are infeasible). This affects 78 designated path, 29 (easy) hub-and-spoke, and 9 critical time instances. The numbers here are higher than in the previous experiments because MH uses the too-long path definition of Shu, Xu, and Baldacci [SXB26], which yields additional too-long paths (that would be non-minimal under our definition). We retain these instances in Table 3a, since we still obtain useful information about the time it takes to compute a discretisation that eliminates many

too-long paths. But for Table 3b, we exclude them, since the number of time points cannot sensibly be compared with the number for feasible solutions.

We observe that BL adds many more time points than are strictly necessary, with discretisation sizes between 20 % and 68 % above the optimum, while the matheuristic significantly reduces this gap, even reaching near-optimal solutions for the star instances. But the graph-based MIP and DDD approach achieve the optimal solution values significantly faster on average than MH and Path. For some classes, DDD is even faster than the simple BL heuristic, since it never requires enumeration of all minimal too-long paths. We also depict the performance profile for the three exact algorithms in Figure 10. DDD is almost always the fastest, with Graph and Path taking at least 10 times as long as DDD on about 20 % and 50 % of instances, respectively. Due to the initial time points, DDD always finished in one iteration. In summary, for the ORP instances resulting from SND instances from the literature, our DDD approach allows us to find optimal refinements with mean running times below 1.3 s in every instance class.

Table 3: Comparison of ORP solution methods.

(a) Mean running time in s.						(b) Mean percentage excess in discretisation size relative to the ORP optimum.		
Class	BL	MH	Path	Graph	DDD			
HC/HF	0.01	10.06	27.51	7.34	0.19	Class	BL	MH
HC/LF	0.01	3.38	2.98	1.75	0.16	HC/HF	67.98	18.62
Crit. time	0.05	4.43	2.02	0.11	0.08	HC/LF	56.94	14.48
Des. path	1.36	44.72	56.96	1.36	1.27	Crit. time	25.05	10.76
H&S (easy)	0.39	6.96	16.57	0.17	0.12	Des. path	60.24	28.79
Grid	0.01	6.34	1.12	0.10	0.05	H&S (easy)	20.43	8.57
H&S (hard)	0.01	3.40	0.66	0.19	0.05	Grid	40.64	11.54
Star	0.00	0.06	0.09	0.08	0.01	H&S (hard)	51.38	6.41
Wheel	0.00	1.74	0.36	0.18	0.03	Star	36.66	0.04
						Wheel	36.58	2.41

6.2. Benefit of Optimal Refinement for Solving SND

Next, we investigate the impact of different refinement strategies on the performance of the DDD algorithm for the SND. For these experiments, we disregard the instances of Van Dyk and Koenemann [VK24], since they are trivial to solve with existing methods [Hel26]. We compare various refinement strategies in a shared DDD algorithm. This algorithm is based on the stronger relaxed problem discussed in Section 5 and uses various features of state-of-the-art algorithms. We use the repair heuristic of Marshall et al. [Mar+21] to obtain feasible solutions to SND in each iteration and use their adaptive gap to only solve the relaxations to certain solution quality. To ensure that the heuristic always finds a feasible solution, we also add a constraint to enforce $\tau(P_k) \leq \ell_k - r_k$ for each

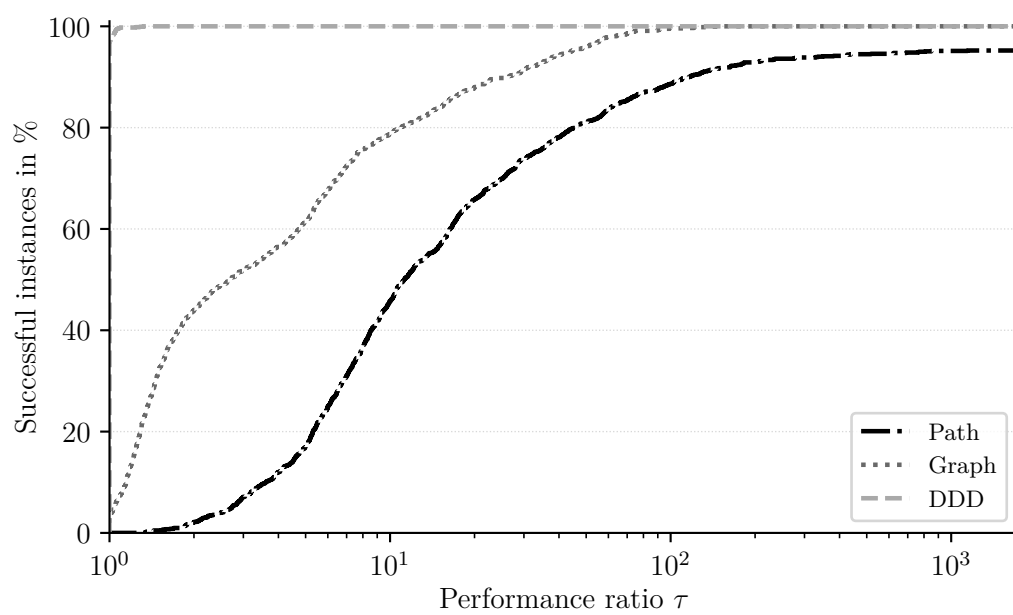


Figure 10: Performance profile for exact ORP solvers: percentage of instances solved to optimality within τ times the running time of the fastest solver on each instance.

commodity k [Bol+17]. The algorithm also uses every solution found by the relaxation solver for refinement and repair, as suggested by Shu, Xu, and Baldacci [SXB26], and initialises the relaxation with their significant time points. Each instance is solved with a gap target of 1 % and time limit of one hour.

Again, we use the baseline (BL) strategy of simply adding the time points of Theorem 4.8 for all minimal too-long paths, as well as the matheuristic (MH) of Shu, Xu, and Baldacci [SXB26]. Both approaches simply add time points to the existing discretisation. Similarly, we introduce an approach that uses the DDD solver for the ORP to find the smallest set of time points to add to the existing discretisation to eliminate all newly observed too-long paths (Iterative). As mentioned in the literature overview, we also investigate strategies that can remove time points from the discretisation. We propose two methods using this idea. The method Parallel finds a smallest discretisation $\mathcal{T}$ that makes all too-long paths in the dispatch-node graphs of all relaxation solutions obtained so far $\mathcal{T}$ -relaxed too-long. This makes the refinement problem substantially larger and potentially harder to solve, even with the DDD approach. This motivates the method Merged, which takes a similar approach but merges the dispatch-node graphs, reducing the refinement problem’s size at the possible cost of additional time points. For Parallel and Merged, we also require them to include significant time points as discussed in Section 5.3.

Table 4 reports statistics for each approach. Note that we did not include the instances that were solved in one iteration or for which the first relaxation could not be solved within the time limit, since refinement does not impact their performance (i.e., instance counts per class are as in Table 2). Note, also, that the model size refers to the number of commodity transport variables relative to the number in a fully time-expanded model, as in [Hel26]. The average number of iterations for instances in each class only varied by about 0.2 between the approaches and is not reported in detail. The differences in running time primarily reflect differences in the difficulty of solving the relaxations rather than differences in iteration counts due to weaker or stronger relaxations. We see that approaches using optimal refinement (Parallel, Merged, Iterative) achieve the lowest running times for each instance class and solve slightly more instances than the heuristic approaches. But overall, the gains in mean running time are modest. For example, Iterative uses 12 % and 23 % less time on average than MH and BL, respectively, on the HC/HF instances. We observe that Parallel usually achieves the smallest model sizes at termination, almost 20 % smaller than the naive BL approach on the HC/HF class. Merged achieves essentially the same size while requiring less time for refinement. But again, the decreases in model size are modest, with the largest decrease of 18 % from 0.76 % of a fully time-expanded model to 0.62 % for Parallel versus BL on HC/HF.

We note that we also conducted experiments in which we solved the ORP with a weighted objective function to prefer arc time points that lead to timed copies for fewer commodities. This resulted in negligible differences in model size and running time; therefore we omit the detailed results here.

Table 4: Mean results per class for each refinement strategy.

Class	Method	Solved (%)	Model size (%)	Ref. (s)	Time (s)
HC/HF	BL	97.7	0.76	0.53	252.83
	MH	99.4	0.67	19.78	221.47
	Parallel	98.9	0.62	12.19	203.85
	Merged	100.0	0.62	5.87	205.15
	Iterative	100.0	0.65	2.33	194.12
HC/LF	BL	100.0	2.06	0.28	30.74
	MH	100.0	1.89	8.13	35.96
	Parallel	100.0	1.80	5.02	32.01
	Merged	100.0	1.80	3.51	26.85
	Iterative	100.0	1.84	1.37	26.71
Grid	BL	73.6	20.62	0.19	1425.75
	MH	74.7	19.34	24.98	1412.90
	Parallel	82.8	18.40	2.59	1280.40
	Merged	82.8	18.41	1.32	1244.41
	Iterative	78.2	18.90	0.65	1306.17
H&S (hard)	BL	70.2	9.49	0.21	1565.40
	MH	75.0	9.22	19.70	1444.95
	Parallel	77.4	9.09	6.36	1431.72
	Merged	76.2	9.10	4.15	1456.04
	Iterative	77.4	9.16	1.10	1446.40
Star	BL	100.0	15.28	0.04	20.89
	MH	100.0	14.77	7.22	24.20
	Parallel	100.0	14.57	1.01	18.93
	Merged	100.0	14.58	0.96	18.79
	Iterative	100.0	14.60	0.21	16.42
Wheel	BL	85.0	5.14	0.16	1473.16
	MH	89.0	4.76	5.43	1269.75
	Parallel	93.0	4.59	2.26	1112.51
	Merged	92.0	4.58	1.17	1076.32
	Iterative	91.0	4.71	0.66	1199.93

7. CONCLUSION

We introduced a more compact representation of conflicts in a DDD approach, which we used to design efficient methods for finding small discretisations that eliminate all conflicts. Solving our refinement problem optimally leads to smaller relaxation problems that can, on average, be solved faster without a significant change in iteration count. But, as reported by [Hel26], the growth in relaxation size from the first to the final iteration for state-of-the-art DDD solvers is already not very large, leaving limited scope for further reductions. We note that for instances requiring more refinement than the existing benchmarks, the benefit of our approaches may be significantly larger. For example, instances with wider time windows $\ell_k - r_k$ or larger capacities u_a relative to commodity quantities q_k may lead to more consolidations, more conflicts and thus more refinement.

One promising application of our efficient refinement methods is the development of better heuristics for the relaxation, such that relevant conflicts can be discovered quickly while keeping the model size small. Then, one might get lucky and only have to solve the relaxation once to obtain an optimal solution. So far, there is also no satisfactory treatment of cases where dispatch-node graphs contain cycles; especially if commodity time windows are long, eliminating conflicts involving cycles may require many time points, even under optimal refinement. On the theoretical side, a practical characterisation of an eliminated too-long path would be interesting, though we do not expect this to lead to significantly smaller discretisations. Similarly, existing approaches are based on preventing conflicts from recurring in *any feasible* solution, but it would be sufficient to prevent them from recurring in *optimal* solutions. It would also be interesting to extend our methodology to other problems or service network design variants that require different refinement strategies to accommodate extensions such as hub-capacity constraints [He+22a] or holding costs [SXB24].

REFERENCES

- [Bol+17] N. Boland et al. ‘The continuous-time service network design problem’. *Operations Research* 65.5 (2017), pages 1303–1321. DOI: [10.1287/opre.2017.1624](https://doi.org/10.1287/opre.2017.1624).
- [Bol+19] N. Boland et al. ‘The price of discretizing time: a study in service network design’. *EURO Journal on Transportation and Logistics* 8.2 (2019), pages 195–216. DOI: [10.1007/s13676-018-0119-x](https://doi.org/10.1007/s13676-018-0119-x).
- [BS19] N. L. Boland and M. W. P. Savelsbergh. ‘Perspectives on integer programming for time-dependent models’. *TOP* 27.2 (2019), pages 147–173. DOI: [10.1007/s11750-019-00514-4](https://doi.org/10.1007/s11750-019-00514-4).

- [SXB25] S. Shu, Z. Xu, and R. Baldacci. *New dynamic discretization discovery strategies for continuous-time service network design*. Preprint, Optimization Online. 2025. January 2025 version; superseded by the August 2026 revision.
- [SXB26] S. Shu, Z. Xu, and R. Baldacci. *New dynamic discretization discovery strategies for continuous-time service network design*. Preprint, Optimization Online. 2026. URL: <https://optimization-online.org/?p=29090>. August 2026 revision.
- [SXQ23] S. Shu, Z. Xu, and J. Qi. *Robust service network design under travel time uncertainty: Formulations and exact solutions*. Preprint, Optimization Online. 2023. URL: <https://optimization-online.org/?p=24646>.
- [Van24] M. Van Dyk. ‘Routing, scheduling, and sorting in consolidated networks’. PhD thesis. University of Waterloo, 2024. URL: <https://hdl.handle.net/10012/20501>.
- [VK24] M. Van Dyk and J. Koenemann. ‘Sparse dynamic discretization discovery via arc-dependent time discretizations’. *Computers & Operations Research* 169 (2024), 106715. DOI: [10.1016/j.cor.2024.106715](https://doi.org/10.1016/j.cor.2024.106715).
- [WH26] T. Wüllner and A. Helber. ‘Finding minimal discretizations in dynamic discretization discovery for continuous-time service network design’. *Proceedings of the 12th International Network Optimization Conference (INOC), Liège, Belgium, April 22–24, 2026*. OpenProceedings.org, 2026, pages 57–62. DOI: [10.48786/inoc.2026.11](https://doi.org/10.48786/inoc.2026.11).

Appendix

A. INTEGER PROGRAM FOR THE RELAXED PROBLEM

We construct a time-expanded network $D_{\mathcal{T}}$ to formulate the integer program for R-SND with discretisation $\mathcal{T}$. Each arc $a \in AD$ receives a timed copy for each interval in $\mathcal{T}_a^I$, so we obtain the transport arcs

$$A_T = \{(a, I) : a \in AD, I \in \mathcal{T}_a^I\}.$$

We also identify arcs this way and explicitly specify their head and tail nodes later. Each node receives a timed copy for each arrival time obtained by dispatching at the start of an interval on an incoming arc. We also add a timed copy at the beginning of the time horizon. Formally, we obtain the derived node time discretisation $\mathcal{T}_v = \{t + \tau(a) : a \in A^{\text{in}} v, [t, t'] \in \mathcal{T}_a^I\} \cup \{0\}$ and the time nodes

$$VD_{\mathcal{T}} = \{(v, t) : v \in VD, t \in \mathcal{T}_v\}.$$

Then, for an arc $b = (a, I) = (uv, [t, t']) \in AD_{\mathcal{T}}$ we say that $\text{head}(b) = (v, t + \tau(a))$ and $\text{tail}(b) = (u, \max\{t'' \in \mathcal{T}_u : t'' \leq t' - 1\})$. We also specify holding arcs A_H connecting successive timed copies of each node. The complete arc set is then $AD_{\mathcal{T}} = A_T \cup A_H$. For each commodity we specify the subnetwork $D_{\mathcal{T},k}$ available to that commodity by removing arcs (a, I) with $I \cap I_k(a) = \emptyset$. Finally, we extend the arc-specific parameters to the time-expansion, i.e., for each $b = (a, I) \in A_T$ we get $c_k(b) = c_k(a)$, $f(b) = f(a)$, and $u(b) = u(a)$. Holding arcs incur no cost. We define each commodity's timed origin and destination as the nodes $\hat{o}_k = (o_k, 0)$ and $\hat{d}_k = (d_k, \max \mathcal{T}_{d_k})$.

We can now state the following integer program (IP) for R-SND:

$$\min \sum_{k \in \mathcal{K}} \sum_{b \in A(D_{\mathcal{T},k})} c_k(b) x_{k,b} + \sum_{b \in A_T} f(b) y_b \quad (7a)$$

$$\text{s.t.} \quad \sum_{b \in A^{\text{out}} w} x_{k,b} - \sum_{b \in A^{\text{in}} w} x_{k,b} = \begin{cases} 1 & \text{if } w = \hat{o}_k, \\ -1 & \text{if } w = \hat{d}_k, \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in \mathcal{K}, w \in V(D_{\mathcal{T},k}) \quad (7b)$$

$$\sum_{k \in \mathcal{K} : b \in A(D_{\mathcal{T},k})} q_k x_{k,b} \leq u(b) y_b \quad \forall b \in A_T \quad (7c)$$

$$x_{k,b} \in \{0, 1\} \quad \forall k \in \mathcal{K}, b \in A(D_{\mathcal{T},k}) \quad (7d)$$

$$y_b \in \mathbb{N} \quad \forall b \in A_T \quad (7e)$$

For each arc $b = (a, I) \in A(D_{\mathcal{T},k}) \cap A_T$, the variable $x_{k,b}$ takes the value 1 if and only if $a \in P_k$ and $h(c(a, k)) = I$. We assume (without loss of optimality) that all commodities that traverse arc a in the same interval I are consolidated together, and $y_{(a,I)}$ then denotes the number of vehicles necessary for this consolidation.

A.1 Remark. Our definitions of SND and R-SND require each commodity to follow a path (i.e., with no node repetitions). For the relaxation and the theoretical results, we could in fact also allow walks; the results remain valid, but the notation becomes cumbersome. We could add a constraint

$$\sum_{(a,I) \in A(D_{\mathcal{T},k}) : a \in A^{\text{out}} v} x_{k,(a,I)} \leq 1 \quad \forall k \in \mathcal{K}, v \in VD$$

to only obtain paths in the relaxation, but as previously noted [Hel26], this tends to make the relaxation harder to solve without substantially improving the lower bound. Therefore, we omit this constraint when solving R-SND. $\triangleleft$

The IP evidently correctly computes the cost of a solution; it remains to show that each of its optimal solutions can be mapped to a solution of R-SND with the same value and vice versa. First, we note that R-SND allows multiple consolidations using the same interval, while the IP does not. But in this case, we could merge the consolidations, preserving feasibility and the objective value (otherwise the solution would not have been optimal). After merging any such consolidations, the feasible paths and consolidations are exactly the same. By construction of $D_{\mathcal{T},k}$, a commodity k may select only arc-interval combinations (a, I) fulfilling Property (a). Two successive timed transport arcs (a, I) and (a', I') can be selected for a commodity in the IP if and only if $\min I + \tau(a) \leq \max I'$ (fulfilling Property (b)). To see this, consider that (a, I) arrives at the intermediate node exactly at time $\min I + \tau(a)$ and (a', I') leaves from $t = \max \{t' \in \mathcal{T}_v : t' \leq \max I'\}$. These two timed nodes are identical (or linked by holding arcs) if and only if the condition holds.

Next we specify how to solve the relaxed problem when we replace Property (b) by the stronger condition where we set $J_k(a, I) = I \cap I_k(a)$ and demand

$$\min J_k(a_i, h(c_i)) + \tau(a_i) \leq \max J_k(a_{i+1}, h(c_{i+1})) \quad \forall i \in (n).$$

In fact, it suffices to change the commodity-specific time-expanded networks. We can then use exactly the same IP. First, we specify for each commodity and each node the time discretisation

$$\mathcal{T}_{k,v} = \{t_k(v)\} \cup \{\min J_k(a, I) + \tau(a) : a \in A^{\text{in}} v, I \in \mathcal{T}_a^I, J_k(a, I) \neq \emptyset\}$$

which adds time points for the earliest arrivals for all intervals on incoming arcs, while also considering the commodity's dispatch window $I_k(a)$. For each commodity, we then construct the following timed node copies:

$$V(D_{\mathcal{T},k}) = \{(v, t) : v \in VD, t_k(v) \leq \bar{t}_k(v), t \in \mathcal{T}_{k,v}\}.$$

The timed transport and holding arcs are as before, with a slight modification to the nodes connected by each transport arc. For $b = (a, I)$ with $a = uv$ we set $\text{head}_k(b) = (v, \min J_k(a, I) + \tau(a))$ and $\text{tail}_k(b) = (u, \max \{t \in \mathcal{T}_{k,u} : t \leq \max J_k(a, I)\})$. Finally, we define $\hat{o}_k = (o_k, r_k)$ and $\hat{d}_k = (d_k, \max \mathcal{T}_{k,d_k})$. By essentially the same argument as before, the IP on these time-expanded networks solves the stronger R-SND variant.