Finding Minimal Discretizations in Dynamic Discretization Discovery for Continuous-Time Service Network Design

Tom Wüllner
RWTH Aachen
Aachen, Germany
tom.wuellner@rwth-aachen.de

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

ABSTRACT

The dynamic discretization discovery framework is a powerful tool for solving network design problems with a temporal component by iteratively refining a time-discretized model. Existing approaches refine the time discretization in ways that guarantee eventual termination. However, refinement choices are not unique, and better choices can yield smaller and easier-to-solve time-discretized models. We pose the *optimal refinement problem* (ORP) of finding a discretization that minimizes the number of timed arc copies, and we show that ORP is NP-hard. We further develop a mixed-integer programming model for ORP. Experiments show that, within a dynamic discretization discovery algorithm, solving ORP can reduce model size and improve solution times on benchmark instances.

1 INTRODUCTION

The *continuous-time network design problem* (CTSNDP) is an important problem in logistics in which commodities need to be routed through a network at minimum cost and the timing of dispatches is relevant [1]. Specifically, commodities dispatched on the same arc at the same time can be consolidated into one vehicle, leading to lower costs than if they were dispatched at different times and needed to travel in different vehicles.

The state-of-the-art approaches for solving CTSNDP are based on the dynamic discretization discovery (DDD) framework introduced by Boland et al. [1]. In this framework, a relaxed problem is solved over a time-expanded network based on a time discretization. Due to space restrictions, we do not explain how these relaxations work in detail but give an example to motivate this work. Consider the network depicted in Figure 1 through which two commodities need to be transported. Commodity 1 is released at node v_1 at time 1 and needs to reach node v_3, but its due time is irrelevant. Commodity 2 is released at node v_2 at time 2 and is due at node v_3 at time 5. Commodity 1 arrives at node v_2 at time 4 at the earliest. If we consolidate it with commodity 2 on the arc (v_2, v_3), then commodity 2 would arrive too late at its destination, making the solution infeasible. But in a relaxation we allow a commodity to be dispatched from a node before it has arrived there. So in our example, we might allow dispatching commodities 1 and 2 together from node v_2 at time 2, even though commodity 1 cannot arrive there so early.

If such an infeasible solution is obtained, the relaxation must be restricted such that the solution cannot recur. This restriction

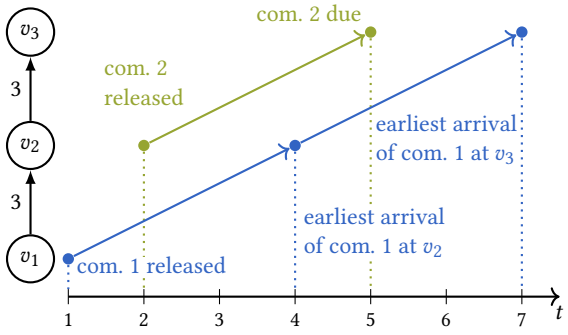


Figure 1: Example of two commodities that cannot be consolidated. The values on the arcs indicate the travel time.

is achieved by refining the time discretization on which the relaxed problem is built. Then the modified relaxation is solved again, and this approach is iterated until a provably optimal solution is obtained. Previous works differ in how they construct the relaxation and how they refine the time discretization. The original work of Boland et al. [1] proposed a linear program that determines where commodities are being dispatched too early (infeasible in the original problem, feasible in the relaxation) and adjusts the relaxation such that commodities at these times cannot 'cheat' with regard to their travel time, i.e., dispatch before they arrive. While this approach is sufficient to guarantee termination, it may lead to illegal consolidations recurring in later iterations, just at different times. Marshall et al. [6] present an alternative approach that constructs a so-called *flat-solution* from solutions to the relaxation that contains only the paths for each commodity and the planned consolidations, but no dispatch times. They propose a graph algorithm that identifies whether such a flat-solution can be turned into a feasible solution for CTSNDP, in which case they call the flat-solution *implementable*. If the flat-solution is not implementable, their algorithm determines how to refine the discretization such that the flat-solution cannot recur. Shu et al. [7] proposed a similar approach to prevent infeasible flat-solutions from recurring by identifying so-called *too-long paths* in another graph constructed based on flat-solutions. Importantly, they show that if a flat-solution is not implementable, there is always a too-long path and the discretization can be refined such that the same too-long path cannot recur. This approach was then adapted to a more general relaxation by Helber [4].

In this paper, we will work with the concept of too-long paths as utilized in [4]. Due to space constraints, we introduce the relevant concepts in a simplified notation, but they are nonetheless equivalent to those in [4]. In general, a too-long path for a given release and due time is a path with travel time longer than the difference between due and release time. It represents a chain of arcs along which consolidation decisions can lead a commodity to arrive too late at its destination. To return to our example, assume we obtain a relaxation solution in which commodities 1 and 2 are consolidated on arc (v_2, v_3). In this case, we would say that $p = ((v_1, v_2), (v_2, v_3))$ is a too-long path with $r_p = 1$ and $d_p = 5$. Specifically, the path tells us that if a commodity k_1 (here $k_1 = 1$) is released at time 1 at v_1, travels with some (identical or different) commodity k_2 (here $k_2 = k_1 = 1$) to node v_2, and then k_2 travels with another commodity k_3 (here $k_3 = 2$) to v_3, then k_3 would be delayed, because it is due at time 5.

Next, we sketch how a relaxation may be adjusted to prevent the illegal consolidation represented by a too-long path from recurring. The relaxations in [4] are based on a time discretization $\mathcal{T}_a \subset \mathbb{N}$ for each arc a in the network. The basic idea is that while a commodity can be dispatched from a node before it arrives there, how much earlier it can be dispatched is limited by the discretization. Assume a commodity arrives at a node at time t, then the earliest time at which it can be dispatched on an arc a leaving that node is $\max\{t' \in \mathcal{T}_a \mid t' \leq t\}$. So the denser a time discretization is, the less 'cheating' is possible in the relaxation, but also the larger and harder to solve the relaxation problem becomes. If a time discretization is sufficiently fine, illegal consolidations become impossible even in the relaxed problem. For our example, let us assume that $\mathcal{T}_{(v_2,v_3)} = \{2, 3\}$. Then, commodity 1 can still be dispatched earlier (at time 3) from node v_2 than it arrives (at time 4), but it cannot be consolidated anymore with commodity 2, since that commodity would then arrive at time 6, which exceeds its deadline. We say that a too-long path is *eliminated* with respect to a given time discretization if consolidations along this path would not be feasible in the relaxed problem based on the time discretization. The goal of our paper is to find small time discretizations that eliminate a given set of too-long paths.

In the remainder of the paper, we will refer to some sets of integers as follows: for $n, m \in \mathbb{N}$ with $n < m$ we define

$$[n] \coloneqq \{1, 2, \ldots, n\} \text{ and } [n, m] \coloneqq \{n, n+1, \ldots, m\}.$$

2 FORMAL PROBLEM DESCRIPTION

Motivated by the example but without formal justification, we use the following definitions. For a directed network $G = (V, A)$ with travel times $\tau_a \in \mathbb{N}$ for $a \in A$, we call a path $p = (a_1, a_2, \ldots, a_{n_p})$ in G *too-long* for a given release time $r_p \in \mathbb{N}$ and due time $d_p \in \mathbb{N}$ if and only if $\sum_{i=1}^{n_p} \tau_{a_i} > d_p - r_p$. A time discretization $\mathcal{T} = (\mathcal{T}_a)_{a \in A}$ contains for each arc a a set of time points $\mathcal{T}_a \subset \mathbb{N}$.

Definition 2.1 (Eliminated too-long path). A too-long path $p = (a_1, a_2, \ldots, a_{n_p})$ is eliminated with regard to a discretization $\mathcal{T}$ if and only if for every arc a_i there is a time point $t_i \in \mathcal{T}_{a_i}$ such that

(1) $t_1 \leq r_p$,
(2) $t_i + \tau_{a_i} \geq t_{i+1}$ for all $i \in [n_p - 1]$,
(3) and $t_{n_p} + \tau_{a_{n_p}} > d_p$.

The *optimal refinement problem* (ORP) is to find a discretization of minimum size $\sum_{a \in A} |\mathcal{T}_a|$ such that every too-long path in a given set $\mathcal{P}$ is eliminated. From now on, we will simply refer to too-long paths as paths, since no other paths will be considered.

3 ORP IS NP-COMPLETE

First, observe that a discretization that eliminates all paths never needs to contain more than $\sum_{p \in \mathcal{P}} n_p$ time points and checking if such a discretization eliminates all paths can be done in polynomial time. Therefore, ORP is in NP. The rest of this section is dedicated to showing that ORP is also NP-hard by a reduction from MAX-2SAT, which is NP-hard [2].

Definition 3.1 (MAX-2SAT). Let $V = \{v_1, v_2, \ldots, v_n\}$ denote a set of Boolean variables. A literal is either a variable v_i or its negation $\neg v_i$. For a Boolean formula $C = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in conjunctive normal form where each clause C_j consists of at most two literals over V, the MAX-2SAT problem is to find a truth assignment that maximizes the number of satisfied clauses.

We start by introducing another representation of ORP that is easier to handle. Based on this representation, we show how to construct an instance $I' = (G, \tau, \mathcal{P}, r, d)$ of ORP from a given instance $I = (V, C)$ of MAX-2SAT such that an optimal solution to the ORP instance corresponds to an optimal solution to the MAX-2SAT instance.

3.1 Alternative Representation of ORP

We start with an observation that only certain time points can be selected to eliminate a given too-long path:

Observation 3.2. *If a discretization $\mathcal{T}$ eliminates a path $p = (a_1, \ldots, a_{n_p})$, for every arc $a_i \in p$ it must contain a time point $t_i \in \mathcal{T}_{a_i}$ with $t_i \in [d_p - \sum_{j=i}^{n_p} \tau_{a_j} + 1, r_p + \sum_{j=1}^{i-1} \tau_{a_j}]$. The length of this interval is the same for all arcs and reflects by how much the path is too long.*

Proof. For the first arc a_1 Condition (1) of Definition 2.1 requires $t_1 \leq r_p$, giving us the upper limit of the interval for t_1. For the second arc, Condition (2) tells us that $t_2 \leq t_1 + \tau_{a_1} \leq r_p + \tau_{a_1}$, again giving us the upper limit of the interval for t_2. We can repeat this process for the remaining arcs to obtain all upper limits. We obtain the lower limits in the same way by starting from Condition (3). □

In the following, we will restrict ourselves to paths such that the corresponding intervals each have a length 1. We represent a path as a series of intervals representing the possible choices for the time point on each arc with an arrow in between for the travel time. See for example Figure 2, which also demonstrates possible assignments of time points to arcs. The first assignment does not eliminate the path because the distance between the time points is too large, i.e., $t_1 + \tau_{(v_1,v_2)} = 1 + 3 \not\geq t_2 = 5$, violating Condition (2).

In the ORP, each arc a has the same travel time τ_a in each path, but the following presentation can be substantially shortened if we allow different travel times τ_{ap} for the same arc in different paths p. We denote this variant as ORP′ and sketch conceptually how ORP′ reduces to ORP as long as the arcs with varying travel times take at least three units of time. Assume we have two paths p_1, p_2 that share an arc $a = (i, j)$ with $\tau_{ap_1} \neq \tau_{ap_2}$. We can then

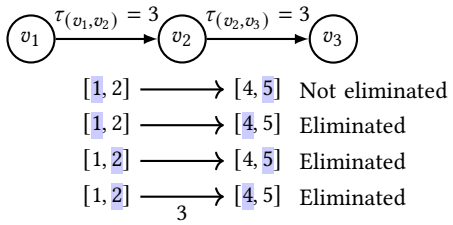


Figure 2: A too-long path with $r_p = 2$, $d_p = 6$ and the possible assignments of time points to eliminate it.

introduce two auxiliary nodes j'_1, j'_2 and replace this arc with the arcs $(i, j), (j, j'_1), (j'_1, j)$ in p_1 and with the arcs $(i, j), (j, j'_2), (j'_2, j)$ in p_2 where $\tau_{(i,j)} = \tau_{(j,j'_1)} = \tau_{(j,j'_2)} = 1$, $\tau_{(j'_1,j)} = \tau_{a,p_1} - 2$, and $\tau_{(j'_2,j)} = \tau_{a,p_2} - 2$. By adding the auxiliary nodes and arcs that occur in no other paths, we increase the objective value of an optimal solution by 4, since each of the new arcs needs exactly one time point. But restricting ourselves to the original set of arcs, the optimal solutions of ORP′ are identical to those of ORP. So for the rest of this section, we treat arcs as having different travel times τ_{ap} in different paths and do not explicitly consider the additional auxiliary nodes and arcs required for the corresponding ORP instance.

3.2 Variable Gadget

Each Boolean variable in the MAX-2SAT instance is represented by two paths in the ORP instance. These paths share all arcs, but one arc will have different lengths for each path. They are constructed such that there are only two ways of eliminating them with a small number of time points, representing the variable being true or false. We say a selection of time points that eliminates the paths in a variable gadget *fulfills* the gadget. We denote the two paths in the variable gadget for variable $v_i \in V$ as p^i_1 and p^i_2. Let m denote the number of clauses. Then we introduce $2(m+1)$ arcs that are shared by the two paths, i.e., $p^i_1 = p^i_2 = (a^i_1, a^i_2, \ldots, a^i_{2m+2})$. The travel times are $\tau_{a^i_k p^i_1} = \tau_{a^i_k p^i_2} = 3$ for all $k \in [2m+2] \setminus \{m+1\}$, $\tau_{a^i_{m+1} p^i_1} = 7$, and $\tau_{a^i_{m+1} p^i_2} = 5$. The first path has early time $r_{p^i_1} = 2$ and late time $d_{p^i_1} = 2 + 3(2m+1) + 7 - 2 = 6m + 10$. The second path has early time $r_{p^i_2} = 3$ and late time $d_{p^i_2} = 3 + 3(2m+1) + 5 - 2 = 6m + 9$. Figure 3 depicts the intervals for the possible time points for the two paths for a variable gadget in an instance with $m = 1$ clause. Intervals for the same arc are marked with the same color and a subscript number.

For the first $m+1$ arcs $a^i_k, k \le m+1$, the intervals are $I^i_{1,k} = [3k-2, 3k-1]$ in the first path and $I^i_{2,k} = [3k-1, 3k]$ in the second path. For the last $m+1$ arcs $a^i_k, k \ge m+2$, the intervals are $I^i_{1,k} = [3k+2, 3k+3]$ for the first path and $I^i_{2,k} = [3k+1, 3k+2]$ for the second path. Observe that these intervals all have width 1.

$[1,2]_1 \to [4,5]_2 \longrightarrow [11,12]_3 \to [14,15]_4$
$[2,3]_1 \to [5,6]_2 \to [10,11]_3 \to [13,14]_4$

Figure 3: Representation of a variable gadget for $m = 1$.

Observation 3.3. *If we want to select the same time point on an arc a^i_k for both paths, that time point needs to be $3k-1$ if $k \le m+1$ and $3k+2$ if $k \ge m+2$, the only points where the intervals intersect.*

Observation 3.4. *If we select the shared time point for one of the first $m+1$ arcs, then we also need to select the earlier time point for all following intervals of the second path by Condition (2) of Definition 2.1. Since the earlier time points for arcs in the second half in the path are not available in the corresponding intervals for the first path, we need to select two time points for those arcs. More formally, if there exists an arc a^i_k with $k \le m+1$ for which both paths select time point $3k-1$, then for all arcs a^i_k with $k \ge m+2$ two different time points have to be selected.*

Lemma 3.5 (Lower bound). *At least $3(m+1)$ time points are needed to fulfill a gadget.*

Proof. To fulfill the gadget with $3(m+1)$ or less timepoints, there evidently needs to be at least one arc for which only a single time point is selected. Assume this arc is one of the first $m+1$ arcs: then by Observation 3.3 the time point needs to be $3k-1$ and consequently we need to use two time points for the last $m+1$ arcs. Since we need at least one time point for every arc, we need at least $3(m+1)$ time points in total. We obtain the same bound analogously if we assume that the arc is one of the last $m+1$ arcs. □

Lemma 3.6 (Classification of lower bound solutions). *We can fulfill the gadget with $3(m+1)$ time points only if we either select one time point for each of the first $m+1$ arcs and two for the last $m+1$ arcs or the other way around.*

If only one time point is selected for each of the first $m+1$ shared arcs, we consider the corresponding variable to be true. If only one time point is selected for each of the last $m+1$ shared arcs, we consider the corresponding variable to be false. For an instance with $m = 1$, Figure 4 shows a corresponding selection of time points for each arc. Note that there is some flexibility in which time points are selected for the arcs where two time points are necessary. But importantly, for a true variable we can always use the later time points for the second half of the first path and have to use the earlier time points for the second path. Similarly, for a false variable we have to use the later time points the second path and have to use the earlier time points for the second half of the first path. So any solution using the minimum number of time points for a variable gadget can either select the later time points for the second half of the first path or for the first half of the second path, but not both, resulting in a single truth value for the variable. Note that the many shared arcs exist solely to enforce that a time point selection that would assign both true and false to the variable is too expensive to occur in optimal solutions, as we will demonstrate.

$[1,2]_1 \to [4,5]_2 \longrightarrow [11,12]_3 \to [14,15]_4$
$[2,3]_1 \to [5,6]_2 \to [10,11]_3 \to [13,14]_4$ True

$[1,2]_1 \to [4,5]_2 \longrightarrow [11,12]_3 \to [14,15]_4$
$[2,3]_1 \to [5,6]_2 \to [10,11]_3 \to [13,14]_4$ False

Figure 4: A solution corresponding to a true/false variable.

3.3 Clause Gadget

The clause gadget is used to model a single clause involving two literals. It also consists of two paths that share some arcs with each other and some with the paths in the variable gadgets for the variables belonging to the literals in the clause. Specifically, the clause gadget consists of an anchor block and one logic block. The anchor block is used to transfer the state of the variables to the logic block and the logic block ensures that a truth assignment that satisfies the clause uses less time points than one that does not. We first give and explain conceptually an example of a clause gadget for an instance with only a single clause in Figure 5, then we give a formal definition.

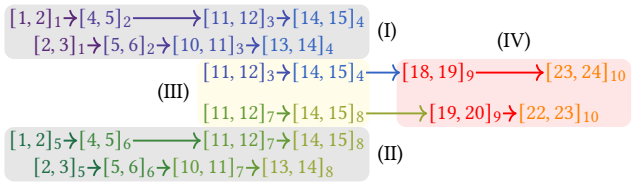


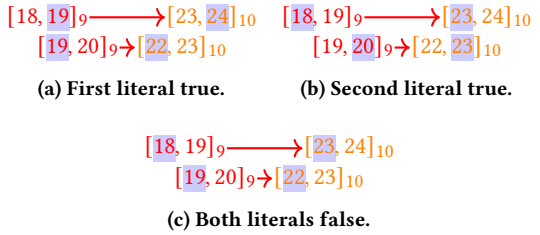
Figure 5: Clause gadget for clause $(v_1 \vee v_2)$ with variable blocks marked in gray (I, II), the anchor block marked in yellow (III), and the logic block in red (IV).

3.3.1 Anchor Block. The anchor block uses the fact that some time points for each of the arcs need to be selected anyway to fulfill the variable gadget, so the anchor block can also be fulfilled with the same time points at no extra cost. Let us assume a variable is false, i.e., in the first path of the variable gadget the first time point is selected for the $m+1$ last arcs. By Definition 2.1, we need to also select the first time point for all arcs in the anchored clause gadget path if we do not want to select additional time points for the arcs in the anchor block. To negate a variable we can switch the anchor, i.e., change it from corresponding to the last $(m+1)$ arcs of the variable to the first $(m+1)$ arcs, also selecting the release and due time such that the intervals line up with those of the second path in the variable gadget. Note that if a literal is false and the anchor block selects the same time points as the variable gadget, we are forced to select the first time point in the following intervals in the logic block.

3.3.2 Logic Block. The logic block ensures that one time point more is needed if the clause is violated than if it is fulfilled, so that we prefer solutions that violate as few clauses as possible. Figure 6 depicts possible solutions for the logic block depending on the truth value of the literals.

Observation 3.7. *If both literals are false, we need to select the first time points in each of the intervals in the logic block, requiring four time points. If at least one of the literals is true, we can select the second time point for one of the paths and only need three time points in the logic block. If both literals are true, either of the configurations in Figure 6(a) or Figure 6(b) is possible.*

3.3.3 Formal Definition. For a clause $C_j = (x_1 \vee x_2)$ over the two literals x_1, x_2 we add two paths $\bar{p}_1^j$ and $\bar{p}_2^j$. If the literal x_1



(a) First literal true. **(b) Second literal true.**

(c) Both literals false.

Figure 6: Possible solutions for logic block.

consists of the non-negated variable v_i, the first path is $\bar{p}_1^j = (a^i_{m+2}, \ldots, a^i_{2m+2}, a_1^{C_j}, a_2^{C_j})$. If the literal consists of the negated variable $\neg v_i$, the first path is $\bar{p}_1^j = (a^i_1, \ldots, a^i_{m+1}, a_1^{C_j}, a_2^{C_j})$. The path for the second literal is constructed equivalently.

For the first path, the late time is always $d_{\bar{p}_1^j} = 6m + 19$, the first m arcs have length 3 and the last two arcs have length 5 and 3. If the variable is not negated, the early time is $r_{\bar{p}_1^j} = 3m + 9$ and the third-to-last arc has length 4. If the variable is negated, the early time is $r_{\bar{p}_1^j} = 2$ and the third-to-last arc has length $3m + 11$. The parameters for the second path are constructed the same way, except that the third-to-last arc is one unit longer, the second-to-last arc is two units shorter and the late time is one unit earlier.

Lemma 3.8. *It is always cheaper to select one extra time point for an unsatisfied clause than to incur an additional cost of $m+1$ time points by deviating from the lower bound of a variable gadget to obtain a state where a variable is both true and false.*

Based on this, we see that minimizing the number of time points needed also maximizes the number of fulfilled clauses.

3.4 Mapping ORP to MAX-2SAT Solutions

Let $I = (V, C)$ denote a MAX-2SAT instance with n variables and m clauses and $I' = (G, \tau, \mathcal{P}, r, d)$ the corresponding ORP instance constructed as described. Let $\mathcal{T}$ denote a solution to I' with $z(\mathcal{T}) = \sum_{a \in A} |\mathcal{T}_a|$ time points. We can construct a solution $X \in \{0,1\}^n$ to I by reading the truth values from the time points in the variable gadgets. Let $g(X)$ denote the number of fulfilled clauses. Every clause which has 4 time points selected in the logic block is violated by this solution and every clause with 3 time points is fulfilled.

Lemma 3.9. *Given an optimal solution $\mathcal{T}$ to the ORP instance I', we can construct from it a solution X to the MAX-2SAT instance I with $g(X) = m - z(\mathcal{T}) + (3m + 3n(m+1))$ fulfilled clauses.*

The other way around, given a solution to MAX-2SAT X, we can construct a corresponding solution for I' by selecting time points as described for the variable gadgets and then selecting three time points for the logic block in gadgets corresponding to fulfilled clauses and four time points for violated clauses.

Lemma 3.10. *If we have an optimal solution X to the MAX-2SAT instance I, then we can construct from it a solution $\mathcal{T}$ to the ORP instance I' with $z(\mathcal{T}) = m - g(X) + (3m + 3n(m+1))$.*

So maximizing the number of fulfilled clauses is equivalent to minimizing the number of used time points, concluding the proof.

4 BINARY PROGRAMMING MODEL FOR ORP

Due to the above hardness result, we might consider developing problem-specific heuristics, approximation methods or exact methods for solving ORP. As a first evaluation of the potential for such methods, we propose a binary programming (BP) model for ORP. Let $H = \max_p d_p$ denote the time horizon – certainly no arc will require a time point larger than H. Let $I_{pi} = [d_p - \sum_{j=i}^{n_p} \tau_{a_j}, r_p + \sum_{j=1}^{i-1} \tau_{a_j}]$ denote the possible time points for the i-th arc in a path p, as per Observation 3.2. Note that so far we have considered each time point in the discretization to be equally bad. But not every commodity can traverse every arc at every time, so not every time point results in the same number of additional variables in the DDD relaxation. We therefore slightly generalize ORP and use a weighted objective function where c_{at} is the number of commodities that can traverse a at time t in feasible solutions to CTSNDP (if we set $c_{at} = 1$ for all a, t, we obtain the ORP as introduced previously). Then, we state the following program ORP_{BP}:

$$\min \quad \sum_{a \in A} \sum_{t \in [H]} c_{at} y_{at} \tag{1}$$

$$\text{s.t.} \quad \sum_{t \in I_{pi}} x_{pit} = 1 \qquad \forall p \in \mathcal{P}, i \in [n_p] \tag{2}$$

$$x_{pit} \leq y_{at} \qquad \forall p \in \mathcal{P}, i \in [n_p], t \in I_{pi} \tag{3}$$

$$\tau_{a_i} + \sum_{t \in I_{pi}} t x_{pit} \geq \sum_{t \in I_{p(i+1)}} t x_{p(i+1)t} \quad \forall p \in \mathcal{P}, i \in [n_p - 1] \tag{4}$$

$$y_{at} \in \{0, 1\} \qquad \forall a \in A, t \in [H] \tag{5}$$

$$x_{pit} \in \{0, 1\} \qquad \forall p \in \mathcal{P}, i \in [n_p], t \in I_{pi} \tag{6}$$

The variable y_{at} takes value one if and only if $t \in \mathcal{T}_a$ and variable x_{pit} takes value one if and only if time point t is selected for arc a_i to eliminate the path p as per Definition 2.1. The objective function is to minimize the number of time points in the discretization. Constraint (2) ensures that for every arc in every path a time point inside the interval is selected. Constraint (3) ensures that the selected time points are also part of the discretization. It remains to ensure that the paths are actually eliminated by the discretization. By the definition of I_{pi}, Conditions (1) and (3) of Definition 2.1 are fulfilled. Constraint (4) ensures that Condition (2) is also fulfilled and thus the too-long paths are eliminated by any discretization corresponding to a feasible solution of this program.

5 COMPUTATIONAL EXPERIMENT

We use ORP_{BP} as an oracle to obtain small time discretizations in a DDD approach for solving the CTSNDP. This gives us an indication for how much smaller the relaxation models can get compared to existing approaches and how much faster they can be solved. Our experimental setup is based on the implementation of the arc-based DDD approach in [4]. We compare three algorithms. The first is `default`, which corresponds to the most performant method `ALL-DISPATCH` of [4] and constructs relaxations in a manner consistent with our assumptions in this paper. For eliminating too-long paths, it uses the three-stage method of [7], which can be seen as a heuristic for the ORP, since it iteratively considers too-long paths and does not add time points if the existing discretization already eliminates the path. The second algorithm `refineall` is like `default` but simply adds all time points for paths without considering if they are already eliminated. The third algorithm, `optimal`, solves ORP_{BP} in each iteration to obtain an entirely new discretization that eliminates all too-long paths seen in the previous iterations. Note that the first two approaches only ever add time points into the discretization in each iteration, while `optimal` may use totally different discretizations in each iteration, possibly sharing no time points at all.

Note that all three approaches also add the *significant time points* proposed by [7]. They observed that for a given arc, there may be two commodities that can never be consolidated, and these consolidations can be prevented in the relaxation by adding a single time point inside a specific interval to the discretization. Each pairwise infeasible consolidation on an arc may result in different intervals. They propose to solve a minimum hitting set problem to determine the smallest number of time points that 'hits' all intervals. We use the same approach for the initial relaxations. In `optimal`, we utilize the flexibility that any time point in the interval is acceptable and during refinement simply demand that one time point from each interval is selected. Formally, given a set $I'_a = \{I'_{a1}, \ldots, I'_{an_a}\}$ of intervals with significant time points for an arc a, we add

$$\sum_{t \in I'_{ai}} y_{at} \geq 1 \quad \forall a \in A, I'_{ai} \in I'_a.$$

We evaluate all three methods on the instances proposed in [1] that have also been used to evaluate all DDD methods for CTSNDP since then. Due to space restrictions, we refer to [1] for a description of the instances. We only consider instances in the two harder classes HC/LF and HC/HF, as instances in the other classes solved on average in 1.1 iterations, so there is no need for better refinement.

All experiments are executed on machines with a Xeon L5630 Quad Core 2.13 GHz processor, using Python 3.12.2 for the DDD algorithm and Gurobi 12.0.1 [3] to solve the relaxations and the binary program for finding the discretization. All instances were solved with a one-hour time limit and a 1 % target gap, using the adaptive gap procedure for solving the relaxations proposed in [6]. We only report results for instances that were solved to the target gap within the time limit for all methods, which were 181 of 183 in the class HC/LF and 143 of 177 in the class HC/HF.

We report aggregate statistics for each class and method in Table 1. The first column shows the average number of iterations to termination, the second column the average time to termination, the third column the average time to termination minus the time spent on the refinement step and the last column the average

Table 1: Results on HC/LF and HC/HF instances.

Class	Algorithm	Iter.	Time (s)	w/o ref. (s)	Variables
HC/HF	`default`	4.6	93.12	92.39	9,735
	`optimal`	4.9	625.88	76.34	8,061
	`refineall`	4.5	117.78	116.85	10,517
HC/LF	`default`	3.3	23.39	23.02	12,655
	`optimal`	3.5	128.48	20.32	11,282
	`refineall`	3.3	22.27	21.82	13,481

number of variables in the relaxation model in the last iteration. Note that solving ORP_{BP} is not competitive, increasing the overall running time by about a factor of six. But the resulting relaxation models are indeed smaller, by about 10 % for the HC/LF class and by about 17 % for the HC/HF class. If we assume that we could obtain an optimal discretization for free, i.e., if we disregard the time spent on the refinement procedure, we can actually solve instances faster: compare the average time of 76.34 s for `optimal` to the average time of 92.39 s for `default` on the HC/HF class. This is despite the fact that `optimal` actually takes more iterations on average, probably because the larger discretizations of the other approaches 'accidentally' prevent previously unobserved problems from occurring. Evidently, the heuristic approach for solving ORP that `default` employs already is beneficial compared to naively eliminating each path separately as done by `refineall`, but it leaves some potential on the table.

6 DISCUSSION

We have demonstrated that finding smaller discretizations by solving ORP can be computationally beneficial for DDD-based methods, but doing so is NP-hard. Using a black-box commercial solver with the proposed formulation ORP_{BP} to solve ORP is evidently not computationally beneficial, since the time needed to solve ORP_{BP} to optimality is significantly larger than the time savings due to smaller relaxations. We note that for the correctness of the overarching DDD framework, it is only necessary to find feasible solutions to ORP, not optimal ones. So we propose to develop heuristic methods to quickly obtain near-optimal solutions. Since in each iteration we can already start from the previous discretization to obtain a near-feasible solution, methods based on local search seem promising. As for exact approaches, it seems natural to use the dynamic discretization discovery paradigm on the ORP as well instead of using the fully time-discretized formulation we proposed in this paper. In fact, ORP itself is quite similar to CTSNDP with fixed routes for each commodity, as selecting a time point that can be used to eliminate multiple too-long paths is similar to dispatching a vehicle to transport multiple commodities. But the linking of the times on each arc along a path is different between the two problems. In the CTSNDP, the dispatch time on an arc always has to be no earlier than the arrival time, commodities cannot be dispatched before their release time and need to arrive before their deadline. In the ORP, the selected time always has to be no later than the 'arrival time', the time on the first arc needs to be no later than the release time and in the end the path still has to be too-long. We want to explore further in this direction and explore this 'duality' between the two problems.

Some thought has also been put into deriving relevant time points for a discretization a priori, for example by adding significant time points [7] or by solving the LP relaxation of the relaxation problem [5]. Perhaps machine-learning methods could also be used to predict time intervals on arcs where adding time points might be beneficial. Then, selecting some time points from these intervals could be integrated into ORP as we did for the significant time points in our computational experiments. This could reduce the number of iterations needed to solve challenging instances.

On the more theoretical side, we note that it is unclear if *too-long paths* are indeed the best concept to identify and fix problems with relaxation solutions. While they provide a very elementary classification of such problems, they are also often quite numerous. Indeed, it is easy to construct examples in which a number of too-long paths exponential in the instance size are found. So perhaps another concept of problematic substructures in a solution could be found that is easier to handle while still leading to improvements in the relaxation that are as strong as those from fixing too-long paths.

Last but not least, we note that the discretizations found by ORP are minimal in the sense that no smaller discretization exists for which all observed too-long paths are eliminated in feasible solutions to the relaxation. It could be that smaller discretizations exist for which the too-long paths are *not* eliminated, but do not occur in *optimal* solutions. Characterizing such discretizations remains an intriguing open question and requires insights not just into the structure of feasible, but also optimal solutions.

ACKNOWLEDGMENTS

We thank Oliver Bachtler and Marco Lübbecke for proofreading and suggestions on the presentation.

STATEMENT ON AI USAGE

ChatGPT was asked to suggest candidate NP-hard problems for the reduction proof, but provided no usable suggestions. GitHub Copilot was used for code autocompletion during the implementation of ORP_{BP}. All text was written manually without AI assistance. Copilot was asked to review the draft and provided feedback on unclear points, typos and inconsistent notation which we incorporated.

REFERENCES

[1] Natashia Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. 2017. The Continuous-Time Service Network Design Problem. *Operations Research* 65, 5 (Oct. 2017), 1303–1321. https://doi.org/10.1287/opre.2017.1624

[2] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA.

[3] Gurobi Optimization, LLC. 2025. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[4] Alexander Helber. 2025. Arc-Based Dynamic Discretization Discovery for Continuous-Time Service Network Design. *Optimiziation Online* (April 2025), Preprint. https://optimization-online.org/?p=30155

[5] Mike Hewitt. 2019. Enhanced Dynamic Discretization Discovery for the Continuous Time Load Plan Design Problem. *Transportation Science* 53, 6 (Nov. 2019), 1731–1750. https://doi.org/10.1287/trsc.2019.0890 Publisher: INFORMS.

[6] Luke Marshall, Natashia Boland, Martin Savelsbergh, and Mike Hewitt. 2021. Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem. *Transportation Science* 55, 1 (Jan. 2021), 29–51. https://doi.org/10.1287/trsc.2020.0994 Publisher: INFORMS.

[7] Shengnan Shu, Zhou Xu, and Roberto Baldacci. 2025. New Dynamic Discretization Discovery Strategies for Continuous-Time Service Network Design. *Optimiziation Online* (Jan. 2025), Preprint. https://optimization-online.org/?p=29090