



## Transportation Science

Publication details, including instructions for authors and subscription information:  
<http://pubsonline.informs.org>

### Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem

Luke Marshall, Natasha Boland, Martin Savelsbergh, Mike Hewitt

To cite this article:

Luke Marshall, Natasha Boland, Martin Savelsbergh, Mike Hewitt (2021) Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem. *Transportation Science* 55(1):29–51.  
<https://doi.org/10.1287/trsc.2020.0994>

Full terms and conditions of use: <https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions>

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact [permissions@informs.org](mailto:permissions@informs.org).

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2020, INFORMS

Please scroll down for article—it is on subsequent pages







With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes. For more information on INFORMS, its publications, membership, or meetings visit <http://www.informs.org>

# Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem

Luke Marshall,<sup>a</sup> Natasha Boland,<sup>b</sup> Martin Savelsbergh,<sup>b</sup> Mike Hewitt<sup>c</sup>

<sup>a</sup>Microsoft Research, Redmond, Washington 98052; <sup>b</sup>H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332; <sup>c</sup>Department of Information Systems and Supply Chain Management, Quinlan School of Business, Loyola University Chicago, Chicago, Illinois 60611

Contact: luke.jonathon.marshall@gmail.com,  <https://orcid.org/0000-0003-0633-4004> (LM); natashia.boland@gmail.com,  <https://orcid.org/0000-0002-6867-6125> (NB); martin.savelsbergh@isye.gatech.edu,  <https://orcid.org/0000-0001-7031-5516> (MS); mhewitt3@luc.edu,  <https://orcid.org/0000-0002-9786-677X> (MH)

Received: October 24, 2018

Revised: November 13, 2019

Accepted: March 11, 2020

Published Online in Articles in Advance:  
October 19, 2020

<https://doi.org/10.1287/trsc.2020.0994>

Copyright: © 2020 INFORMS

**Abstract.** We introduce an effective and efficient iterative algorithm for solving the continuous-time service network design problem. The algorithm achieves its efficiency by carefully and dynamically refining partially time-expanded network models so that only a small number of small integer programs, defined over these networks, need to be solved. An extensive computational study shows that the algorithm performs well in practice, often using time-expanded network models with size much less than 1% (in terms of number of variables and constraints) of a full time-expanded network model. The algorithm is inspired by and has many similarities to the dynamic discretization discovery algorithm introduced in Boland et al. [Boland N, Hewitt M, Marshall L, Savelsbergh M (2017) The continuous-time service network design problem. *Oper. Res.* 65(5):1303–1321.], but generates smaller partially time-expanded models, produces high-quality solutions more quickly, and converges more quickly.

**History:** Michel Gendreau served as guest editor-in-chief for this article.

**Funding:** This work was supported by the National Science Foundation, Division of Civil, Mechanical, and Manufacturing Innovation [Grants 1435625 and 1662848].

**Supplemental Material:** The online appendix is available at <https://doi.org/10.1287/trsc.2020.0994>.

**Keywords:** service network design • time-expanded network • iterative refinement • dynamic discretization discovery

## 1. Introduction

Optimization problems that are solved to decide when actions occur are often modeled using a time-expanded network. This involves discretizing the time horizon and mapping any parameters involving time to this discretization. Depending on the discretization scheme, the resulting model can yield a continuous-time optimal or an approximate solution. Typically, in practice, *fine* discretization schemes give good approximations at the expense of large, perhaps intractable, models, whereas *coarse* discretization schemes are more computationally tractable, but generally yield poor approximations. It is often difficult to choose a discretization scheme that balances computational tractability with solution quality (Boland et al. 2019).

Service network design is a fundamental activity in the freight transportation industry and focuses on tactical decisions that enhance profitability by increasing the utilization of transportation assets (Magnanti and Wong 1984, Crainic 2000, Wieberneit 2008). The variant we consider is of particular interest to less-than-truckload (LTL) freight carriers, who rely on consolidation to maintain their profitability, but who must meet service standards to remain competitive. Such carriers typically operate a set of terminals, distributed across a region, which serve as

points of origin or destination for the goods they transport, and/or as breakbulks, forming intermediate locations at which goods are transferred from one vehicle to another for the next leg of their journey. Breakbulk is a term generally used in freight operations; a breakbulk has a role similar to that of a sorting center in mail and package operations.

In LTL planning, it is critical to decide not only the number of vehicles to dispatch between each pair of terminals, but also the departure times of those vehicles. These times must be such that all goods to be consolidated on a vehicle are available at the vehicle's departure time. Furthermore, goods must be transported on a sequence of vehicle movements that reaches their destination within the due time dictated by their service standard. Thus, timing is central to service network design.

This context has motivated study of the continuous-time service network design problem (CTSNDP), in which terminals are modeled as nodes in a network. Arcs in the network represent pairs of terminals between which vehicles may operate. Vehicles may load goods at an arc's start terminal and unload them at its end terminal, after traveling from the arc's start to its end terminal. The time it takes a vehicle to travel from the start terminal of an arc to the end terminal of the arc is assumed known and fixed. Goods with a

common origin and destination terminal and a common available and due time are modeled as a commodity. For a given network with given arc transit times and a set of commodities, CTSNDP seeks to determine a (single) route for each commodity so as to minimize the total cost of transportation. A route for a commodity is a path through time and space, starting at its origin after the time at which it becomes available and ending at its destination before its due time, with an associated dispatch time at each intermediate location along the path, indicating the time at which the commodity departs the location. Whenever multiple commodities are dispatched on the same arc at the same time, they may be consolidated, thus reducing the cost of transportation. CTSNDP is naturally modeled and typically approximated using a time-expanded network.

This paper introduces an interval-based dynamic discretization discovery algorithm (DDDI) that iteratively refines a time-expanded network in order to solve CTSNDP to optimality or within a specified tolerance. Importantly, the time-expanded networks generated in the course of the algorithm are not based on a regular discretization of time, nor is the discretization of time uniform across nodes; the network is generated dynamically, refined only at nodes and during periods of time at which refinement is deemed necessary for the algorithm to make progress. Although CTSNDP has been solved using a similar approach (Boland et al. 2017), in the sense that both approaches iteratively and dynamically refine a time-expanded network and solve an integer program (IP) defined over that network at each iteration, the approach presented here is fundamentally different, with different motivations. Iteratively refining a time-expanded network means making its underlying time discretization finer, which results in a larger network; the new network has more nodes and arcs. This has been found to quite rapidly increase the difficulty of solving the integer program defined over that network. However, for the integer program to provide good quality solutions to the continuous time problem, the time-expanded network must be sufficiently fine. This presents a major challenge in the design of dynamic discretization discovery algorithms: the design of the refinement strategy. An aggressive refinement strategy ensures a sufficiently fine discretization is reached within a small number of iterations, meaning few integer programs must be solved. However, an aggressive refinement strategy also means that the size of the time-expanded network increases rapidly, making those same integer programs much harder to solve.

The driving motivation of our interval-based approach is to keep the time-expanded network as small as possible *while also* converging to an optimal solution quickly, needing few refinement iterations. The result is an algorithm that can solve larger

instances, with faster solution times. For example, the set of five instances derived from data from a large LTL carrier in the United States, which represented the frontier of what the method of Boland et al. (2017) could handle (for four out of the five instances their method reached the run time limit of two hours), can be solved by DDDI in an average time of *under five minutes*.

The remainder of the paper is organized as follows. We first give a brief overview of relevant literature. Section 2 gives a formal description of the CTSNDP and a high-level overview of our key contributions. Section 3 provides details on the construction of our time-expanded network and an investigation on the structure of solutions. How we exploit this structure to refine the discretization and to guarantee convergence is described in Section 4. Several variants of the DDDI algorithm are presented in Section 5. Section 6 shows the results of an extensive computational study. In Section 7, we compare DDDI to the method of Boland et al. (2017), both analytically and computationally. And, finally, in Section 8, we finish with conclusions and a discussion of possible future work.

### 1.1. Relevant Literature

The idea of iteratively refining a discretization is not new. For instance, Wang and Regan (2002, 2009) give a discretization scheme and a mixed integer program (MIP) model defined over a time-expanded network for the traveling salesman problem with time windows (TSPTW) and show that solutions to this MIP are guaranteed to converge to a continuous-time optimal solution, as the partition size in the discretization tends toward zero. However, unlike DDDI, their refinement strategy is based on a predetermined series of uniform widths to use as the partition size, rather than a dynamic nonuniform scheme. As a consequence, they were unable to perform more than a small number of refinement iterations before the MIPs became unmanageably difficult to solve and so could not solve larger instances to optimality.

Also in the context of the TSPTW, Dash et al. (2012) iteratively solve linear programming problems to determine a partition of the time windows (into what they call *buckets*), as a preprocessing step in a branch-and-cut framework. They observe that *how* a bucket is discretized has a significant effect on approximation quality and its balance with computational complexity. As highlighted by Boland et al. (2017), there are critical differences between the approach of Dash et al. (2012) for the TSPTW and dynamic discretization techniques for solving CTSNDP. In particular, Dash et al. (2012) (i) only discretize at the preprocessing stage of a MIP solution process; and (ii) exploit the property of the TSPTW that no unforced time spent waiting at a customer is needed. By contrast, waiting is often essential in continuous-time service networks for achieving consolidations.

For an alternative approach, Hosseiniinasab (2015) solves a variant of CTSNDP using a Benders decomposition method with a fixed fleet size (homogeneous capacity) by modeling the dispatch time of each vehicle using continuous variables. However, their focus is on vehicle cycles over periods, and although they allow freight to be split, it cannot be transferred from one vehicle to another at an intermediate location. In this way, the problem is closer to the pickup and delivery problem with time windows than it is to the CTSNDP. Their algorithm successfully solved smaller instances with 10 commodities over a network with 15 nodes.

Boland et al. (2017) introduce dynamic discretization discovery (DDD) and demonstrate its use in the context of the CTSNDP. The key to the approach is that it discovers exactly which times are needed to obtain an optimal, continuous-time solution, in an efficient way, by solving a sequence of (small) MIPs. The MIPs are constructed as a function of a subset of times, with variables indexed by times in the subset. They are carefully designed to be tractable in practice and to yield a lower bound (it is a cost minimization problem) on the optimal continuous-time value. Once the *right* (very small) subset of times is discovered, the resulting MIP model yields the continuous-time optimal value. The research presented here complements and extends the research presented in Boland et al. (2017).

## 2. An Overview of Key Concepts in the DDDI Algorithm

In this section, we first provide a formal definition of the CTSNDP, establishing our notation and terminology. We then give a brief overview of two of the key components of the DDDI approach. We first explain, in general terms, how interval-based time-expanded networks may be constructed and state the integer program that DDDI solves at each iteration to yield a lower bound on the value of the CTSNDP. Then, we give a brief summary of the central concept in our approach to time discretization refinement, which we call the *solution graph*, and explain its role in the overall algorithm. Next, we give an overview of the DDDI algorithm itself, showing how these key components fit together. Finally, we provide a summary of the key contributions of this paper.

### 2.1. The CTSNDP: Definition, Notation, and Terminology

As the name suggests, CTSNDP is a continuous-time variant of a service network design problem. It takes as input a set of commodities  $K$  and a network  $G = (N, A)$ , with nodes,  $N$ , and arcs,  $A \subseteq N \times N$ . To distinguish  $G$  from the time-expanded networks that we will construct later, we refer to it as the *flat-network*.

Each commodity  $k \in K$  has origin node  $o^k \in N$ , destination node  $d^k \in N$ , a time,  $e^k$ , at which it is first available at its origin for transport, a time,  $l^k$ , at which it must reach its destination, and a quantity  $q^k$ . We will refer to  $e^k$  as the *available time* or *early time* and to  $l^k$  as the *due time* or *late time* of commodity  $k$ . Each arc  $a \in A$  has an associated *transit time*,  $\tau_a$ , a vehicle *capacity*,  $u_a$ , which is an upper limit on the total quantity of commodities that can be carried on the arc per vehicle operating on it, a fixed cost,  $f_a$ , (cost per vehicle operating on the arc), and a variable cost,  $v_a$  (cost per unit of commodity flow on the arc).

The output of CTSNDP is a route for each commodity (with associated dispatch times) and consolidation decisions, so as to minimize the sum of fixed and variable transportation costs. A feasible solution to the problem specifies a (single) path in time through the network, for each commodity. We define a *flat-path* to be a simple path in the flat-network. Formally, we write the flat-path  $P$  as a set of arcs, so  $P \subseteq A$ , but we will abuse notation to write, for example,  $n \in P$ , to indicate that the node  $n$  is the head or tail node of some arc in  $P$ . The flat-path  $P$  is *k-feasible* for commodity  $k \in K$  if it is a flat-path from  $o^k$  to  $d^k$  that can be traversed so as to depart  $o^k$  no earlier than  $e^k$  and arrive at  $d^k$  no later than  $l^k$ , that is, for which  $e^k + \sum_{a \in P} \tau_a \leq l^k$ . The variable cost of transport over a set of flat-paths,  $\{P^k\}_{k \in K}$ , is given by

$$\sum_{k \in K} \sum_{a \in P^k} v_a q^k.$$

To evaluate the fixed cost of a set of  $k$ -feasible flat-paths,  $\{P^k\}_{k \in K}$ , the time,  $t_a^k$ , that each commodity  $k$  is dispatched along each arc,  $a \in P^k$ , in its path, is needed. Thus, a feasible solution must specify the set of times  $\mathbf{t}^k = \{t_a^k\}_{a \in P^k}$ , with  $e^k \leq t_{a_1}^k, t_{a_{i-1}}^k + \tau_{a_{i-1}} \leq t_{a_i}^k$  for each  $i = 2, \dots, m^k$ , and  $t_{m^k}^k \leq l^k$ , where  $P^k = \{a_1, a_2, \dots, a_{m^k}\}$  is a set of  $m^k$  arcs, indexed so that the head node of arc  $a_i$  is the tail node of arc  $a_{i+1}$  for each  $i$ . In this case, we refer to  $\mathbf{t}^k$  as  *$P^k$ -dispatch times*. A *feasible solution* to the CTSNDP is a set of  $k$ -feasible flat-paths,  $\{P^k\}_{k \in K}$ , together with  $P^k$ -dispatch times,  $\{\mathbf{t}^k\}_{k \in K}$ . Commodities dispatched on an arc at the same dispatch time may be consolidated, so, given such a solution, we define, for each arc  $a \in A$  and each dispatch time on that arc  $t \in \Theta(a) := \bigcup_{k \in K, a \in P^k} \{t_a^k\}$ , the consolidating set

$$\kappa(a, t) = \{k \in K : a \in P^k \text{ and } t_a^k = t\}.$$

Now the number of vehicles required to carry all commodities on arc  $a$  at time  $t$  is given by  $\lceil \sum_{k \in \kappa(a, t)} q^k / u_a \rceil$ , so the fixed cost of this solution is

$$\sum_{a \in P^k} \sum_{t \in \Theta(a)} f_a \left\lceil \frac{\sum_{k \in \kappa(a, t)} q^k}{u_a} \right\rceil.$$



The CTSNDP is the problem of finding a feasible solution that minimizes the sum of the variable and fixed costs, as defined above.

## 2.2. Constructing an Interval-Based Time-Expanded Network

A key element of the DDDI algorithm is the calculation of a lower bound on the value of an optimal solution to an instance of the CTSNDP by solving an integer program defined over a time-expanded network that is based on time intervals. Construction of the time-expanded network takes as input a given discretization,  $\mathcal{T}$ , defined as a finite set of node and time pairs:  $\mathcal{T} \subset N \times [0, H]$ , where  $H = \max_{k \in K} l^k + \epsilon$ , for some small  $\epsilon > 0$ , is a time “just after” the end of the time horizon. (The use of  $\epsilon$  is purely for notational convenience in what follows; it avoids having to make a special case for the last interval in the time horizon.) Note that we assume that  $\min_{k \in K} e^k = 0$ , since, if not, all early and late times can simply be shifted by a constant to make it so.

Here, we explain the concept of the interval-based time-expanded network in simple terms, as a monolithic structure. In practice, it is much more effective to create a structure for each commodity; we defer discussion of how that is done to Section 3.1.

The time-expanded network for given discretization  $\mathcal{T}$  is defined as  $\mathcal{G}_{\mathcal{T}} = (\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}})$ , where the elements of  $\mathcal{N}_{\mathcal{T}}$  are termed *node-intervals* and the elements of  $\mathcal{A}_{\mathcal{T}}$  are termed *timed-arcs*. The set of node-intervals,  $\mathcal{N}_{\mathcal{T}}$ , is defined as the set of triples  $(n, t, t') \in N \times [0, H]^2$  such that  $(n, t), (n, t') \in \mathcal{T}$  with  $t' > t$  are consecutive in the discretization, in the sense that there is no  $(n, t'') \in \mathcal{T}$  with  $t < t'' < t'$ . The node-interval  $(n, t, t')$  can be thought of as representing the interval of time  $[t, t')$  at the node  $n$ . The set of timed-arcs is divided into two sets: *dispatch-arcs*, denoted by  $\mathcal{D}_{\mathcal{T}}$ , and *holding-arcs*, denoted by  $\mathcal{H}_{\mathcal{T}}$ . The latter simply connect consecutive node-intervals at the same node: it is the set of all node-interval pairs of the form  $((n, t, t'), (n, t', t''))$  with  $(n, t, t'), (n, t', t'') \in \mathcal{N}_{\mathcal{T}}$ . The dispatch-arcs are defined “optimistically” to be the set of all node-interval pairs  $((n_1, t_1, t'_1), (n_2, t_2, t'_2))$  with  $(n_1, t_1, t'_1), (n_2, t_2, t'_2) \in \mathcal{N}_{\mathcal{T}}$  and  $(n_1, n_2) \in A$  such that it is possible to depart node  $n_1$  at some time in the interval  $[t_1, t'_1)$  and arrive at  $n_2$  at some time in the interval  $[t_2, t'_2)$ . In other words, there must exist  $t \in [t_1, t'_1)$  such that  $t + \tau_{(n_1, n_2)} \in [t_2, t'_2)$ . It is optimistic in the sense that, even when  $t_1 + \tau_{(n_1, n_2)} = t'_2 - \epsilon$ , we assume that any commodity dispatched from node  $n_1$  at some time during the interval  $[t_1, t'_1)$  can be next dispatched from node  $n_2$  at any time during the interval  $[t_2, t'_2)$ . It is the optimistic nature in which arcs are created that enables the solution to a model on this network to yield a lower bound on the objective function value of an optimal solution to the CTSNDP.

This time-expanded network is constructed with the intention that every feasible solution is represented in it, meaning that if  $P^k$  is a  $k$ -feasible flat-path for some commodity  $k$  and  $\mathbf{t}$  is a set of  $P^k$ -dispatch times, then for all  $a = (n_1, n_2) \in P^k$ , there exists  $((n_1, t_1, t'_1), (n_2, t_2, t'_2)) \in \mathcal{A}_{\mathcal{T}}$  with  $t_a \in [t_1, t'_1)$ . This can be guaranteed by requiring that the discretization  $\mathcal{T}$  contain node-time pairs  $(n, t)$  for sufficiently early and sufficiently late  $t$ . For example, we could require that  $(n, 0), (n, H) \in \mathcal{T}$  for all  $n \in N$ . Later in the paper, we give details of a more computationally effective approach. Here, we simply assume that such representation is guaranteed.

We extend CTSNDP parameters to the time-expanded network in a natural way: for each  $a = ((n_1, t_1, t'_1), (n_2, t_2, t'_2)) \in \mathcal{D}_{\mathcal{T}}$ , we define  $f_a := f_{(n_1, n_2)}$ ,  $v_a := v_{(n_1, n_2)}$ , and  $u_a := u_{(n_1, n_2)}$ . Finally, we define the net supply of commodity  $k$  required at each node-interval  $i = (n, t, t') \in \mathcal{N}_{\mathcal{T}}$  as

$$r_i^k := \begin{cases} 1 & \text{if } n = o^k \text{ and } t \leq e^k < t', \\ -1 & \text{if } n = d^k \text{ and } t \leq l^k < t', \\ 0 & \text{otherwise.} \end{cases}$$

Due to the “optimistic” construction of the time-expanded network, the following integer program yields a lower bound on the value of the CTSNDP. The integer program has decision variables

$$x_a^k = \begin{cases} 1 & \text{if commodity } k \in K \\ & \text{uses timed-arc } a, \\ 0 & \text{otherwise,} \end{cases} \quad \forall a \in \mathcal{A}_{\mathcal{T}}$$

$$z_a = \# \text{ of vehicles dispatched on timed-arc } a, a \in \mathcal{D}_{\mathcal{T}}.$$

The integer programming model, which we refer to as CTSNDP( $\mathcal{T}$ ), is given by

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

subject to

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

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

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

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

Here  $\delta_i^+$  and  $\delta_i^-$  denote the set of timed-arcs leaving and entering node-interval  $i$ , respectively. The objective (1a) minimizes the total cost (variable plus fixed costs). The flow constraints (1b) ensure that each commodity leaves its origin node during some interval that contains  $e^k$  and reaches its destination node during some interval that contains  $l^k$ . It is—again,

“optimistically”—assumed that all commodities traveling on the same timed-arc can be consolidated: the consolidation constraints (1c) require that enough vehicles are assigned to each dispatch-arc to carry all commodities transported on the arc at some time during the time interval of the dispatch-arc’s first node-interval. Lastly, (1d) and (1e) define the variable domains.

### 2.3. The Solution Graph

A feasible solution to CTSNDP( $\mathcal{T}$ ) does not necessarily, and certainly not immediately, provide a feasible solution to the CTSNDP. Since time is only approximated in the former, the path in the time-expanded network for a commodity  $k$  will consist of a sequence of timed-arcs, whose corresponding arc sequence (flat-path) in the flat-network may not be  $k$ -feasible. Furthermore, even if all such flat-paths are feasible, it may not be possible to coordinate their dispatch times to achieve the consolidations suggested by the solution to CTSNDP( $\mathcal{T}$ ).

Such situations, however, can be recognized efficiently. Define a path in the time-expanded network as a timed-path, and let  $\{\mathcal{P}^k\}_{k \in K}$  be a set of timed-paths, one for each commodity  $k$ , where  $\mathcal{P}^k$  is a path in the timed-network from  $(o^k, t_1, t'_1)$  with  $e^k \in [t_1, t'_1]$  to  $(d^k, t_2, t'_2)$  with  $l^k \in [t_2, t'_2]$ . We call such a collection of timed-paths a *timed-path-solution*. Given a timed-path-solution, which corresponds to a feasible solution to CTSNDP( $\mathcal{T}$ ), we define, for each dispatch-arc,  $a$ , the set of commodities whose path includes arc  $a$  as

$$\kappa_a := \{k \in K : a \in \mathcal{P}^k\}.$$

Under the current discretization,  $\mathcal{T}$ , the CTSNDP( $\mathcal{T}$ ) model evaluates the fixed cost associated with transport of the commodities in  $\kappa_{((n_1, t_1, t'_1), (n_2, t_2, t'_2))}$  on arc  $(n_1, n_2)$  as if they were consolidated; it assumes they can all be dispatched on arc  $(n_1, n_2)$  at a common time. A *consolidation* is defined as an arc in the flat-network together with a commodity set: it is an element of  $A \times \mathcal{P}(K)$ , where  $\mathcal{P}(K)$  denotes the power set of  $K$ . The timed-path-solution  $\{\mathcal{P}^k\}_{k \in K}$  can be used to derive the set of consolidations,

$$C = \left\{ ((n_1, n_2), \kappa_a) : a = ((n_1, t_1, t'_1), (n_2, t_2, t'_2)) \right. \\ \left. \in \bigcup_{k \in K} \mathcal{P}^k, n_1 \neq n_2 \right\},$$

and the set of flat-paths,  $\{\mathcal{P}^k\}_{k \in K}$ . We call the pair  $(\{\mathcal{P}^k\}_{k \in K}, C)$  a *flat-solution*.

A key component in DDDI is a procedure that, for a given discretization  $\mathcal{T}$  and an associated timed-path-solution (with its corresponding flat-solution), answers the following questions:

1. Can the consolidations be realized? That is, does there exist a set of  $P^k$ -dispatch times,  $t^k$ , (which implies that  $P^k$  is  $k$ -feasible), for every  $k \in K$ , such that, for every  $(a, \kappa) \in C$ ,  $t_a^k = t_a^{k'}$  for all  $k, k' \in \kappa$  with  $k \neq k'$ ?

2. If not, which (small set of) node and time point pairs should be added to discretization  $\mathcal{T}$  to give new discretization  $\mathcal{T}'$  so that no timed-path-solution to CTSNDP( $\mathcal{T}'$ ) yields this flat-solution?

At the heart of this procedure is the observation that the first question can be modeled as one of deciding whether a graph, the so-called solution graph, has a positive-length cycle or has a positive-length path from one of a set of origin nodes to one of a set of destination nodes. The special structure of the solution graph allows this to be done very efficiently with a labeling algorithm. Furthermore, and central to the success of DDDI, the labels generated in the course of the algorithm point to answers to the second question. By analysis of the structure of the solution graph and its labels, we are able to determine sets of node and time point pairs whose addition can be proven to cut off the current flat-solution. In special cases, only a *single* node and time point pair is needed, and in general, the sets of new pairs generated by this procedure are small. Details of the procedure, the theory underlying it, and the solution graph itself are given in Section 3.

#### Algorithm 1 (DDDI Algorithm)

**Require:** Flat-network  $G = (N, A)$ , commodities  $K$ , all instance parameters, and an optimality tolerance,  $\text{DDDI\_TOL}$ .

- 1:  $\mathcal{T} \leftarrow N \times \{0, H\}$
- 2:  $\text{model} \leftarrow \text{BUILD\_TIME-EXPANDED\_MODEL}(\mathcal{T})$
- 3:  $\text{lower\_bound} \leftarrow -\infty$ ,  $\text{upper\_bound} \leftarrow +\infty$
- 4: **while** True **do**
- 5:    $(\text{flat-solution}, \text{value}) \leftarrow \text{SOLVE\_LOWERBOUND\_IP}(\text{model})$
- 6:    $\text{lower\_bound} \leftarrow \max\{\text{lower\_bound}, \text{value}\}$
- 7:   **if** for every  $k \in K$  the path  $P^k$  in flat-solution is  $k$ -feasible **then**
- 8:      $(\text{solution}, \text{value}) \leftarrow \text{CONVERT\_SOLUTION}(\text{flat-solution})$
- 9:      $\text{upper\_bound} \leftarrow \min\{\text{upper\_bound}, \text{value}\}$
- 10:   **if**  $(\text{upper\_bound} - \text{lower\_bound}) < \text{upper\_bound} \times \text{DDDI\_TOL}$  **then**
- 11:     **Stop:** the solution corresponding to  $\text{upper\_bound}$  is within tolerance of the optimal
- 12:    $\mathcal{T} \leftarrow \text{REFINE\_DISCRETIZATION}(\text{flat-solution}, \mathcal{T})$
- 13:    $\text{model} \leftarrow \text{UPDATE\_TIME-EXPANDED\_MODEL}(\text{model}, \mathcal{T})$

### 2.4. An Overview of the DDDI Algorithm

The major steps of DDDI are shown in Algorithm 1. The function `BUILD_TIME-EXPANDED_MODEL` creates the initial interval-based time-expanded network and builds the integer programming model defined by (1).

The resulting model is then solved using an integer programming package in `SOLVE_LOWERBOUND_IP` to yield a lower bound value and timed-path-solution, from which a flat-solution is derived. If possible, the current best lower bound, *lower\_bound*, is updated.

DDDI then seeks to update the current best feasible solution and its corresponding value, *upper\_bound*, if possible. If the path  $P^k$  in the flat-solution is  $k$ -feasible for every  $k \in K$ , then there must exist a set of  $P^k$ -dispatch times,  $t^k$ , that yield a feasible solution to the CTSNDP, which can be evaluated to yield an upper bound on its value. Testing the feasibility condition is easy: for each commodity, check that its early time plus the sum of the travel times in its path does not exceed its late time. However, starting each path at its early time is unlikely to provide the consolidation needed for a low-cost solution, so DDDI seeks better dispatch times: it solves an optimization problem that determines  $t^k$ , for all  $k \in K$ , so as to minimize the fixed cost weighted sum of the total difference in dispatch times between all distinct commodities that the flat-solution suggests to consolidate: its objective is

$$\min \sum_{(a,k) \in C} \sum_{k' \in K, k' \neq k} f_a |t_a^k - t_a^{k'}|,$$

where  $C$  is the set of consolidations in the current flat-solution. This objective is a heuristic approximation of the true cost, chosen because it can be modeled as an LP, whereas the true cost requires integer variables to model. As this LP is similar to the one used in Boland et al. (2017), we omit it here. The procedure `CONVERT_SOLUTION` solves this LP and evaluates the cost of the resulting CTSNDP feasible solution, returning both.

If the current upper and lower bounds are not sufficiently close, then DDDI refines the discretization, using the procedure `REFINE_DISCRETIZATION`. This is where DDDI makes use of the solution graph to obtain a small set of new node-time pairs to add to the discretization, guaranteeing that the current flat-solution cannot recur. However, the theory given in Section 4 offers the possibility of many alternative choices of refinement; there may be many structures in the solution graph that yield a set of new node-time pairs satisfying the required conditions, any one of which would suffice. As we will see, this choice can have a huge impact on the solve time for an iteration and on the number of iterations required for convergence. Thus, it is important that `REFINE_DISCRETIZATION` embeds a clever *refinement strategy*. We present several alternative refinement strategies in Section 5 and demonstrate their effect on computational performance in Section 6.

As its final step, DDDI uses the procedure `UPDATE_MODEL` to modify the integer programming model in accord with the changes made in the

discretization. Important implementation details of this procedure are discussed in Section 5.6.

## 2.5. Key Contributions

A discretization of a planning period  $[0, H]$  can be viewed either as a set of time points in the period, that is,  $\{0, t_1, t_2, \dots, t_n, H\}$ , or as a set of intervals partitioning the period, that is,  $\{[0, t_1], [t_1, t_2], \dots, [t_n, H]\}$ . The chosen perspective impacts the interpretation of a (partially) time-expanded network constructed using a discretization.

The main contribution of this research is a demonstration of the advantages of using an interval-based view of discretization when developing a dynamic discretization discovery algorithm for the CTSNDP. This manifests itself primarily in the solution graph, which facilitates the analysis of a solution to the optimization model defined on a partially time-expanded network. By exploiting the special structure of the solution graph, it is possible to meticulously control the refinement of the discretization, which leads to more effective and efficient dynamic discretization discovery algorithms.

Other contributions include structuring the partially time-expanded network as a collection of smaller partially time-expanded networks, one for each commodity, which results in noticeably smaller optimization models, and showing how algorithm engineering, that is, carefully designing and implementing an algorithm, can result in significant performance improvements.

## 3. Time-Expanded Commodity Networks and the Solution Graph

In this section, we describe how we create the time-expanded network for each commodity, and establish our notation for these networks and for the solutions to the CTSNDP( $\mathcal{T}$ ) formulation that is based on them. We then define the solution graph and establish its basic properties. For ease of presentation, proofs of theorems and lemmas are given in the online appendix rather than in the text.

### 3.1. Time-Expanded Commodity Networks

In what follows, we describe how, for a given CTSNDP instance, we construct a time-expanded network  $\mathcal{G}_{\mathcal{T}}^k = (\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}^k)$ , for each commodity  $k$ . To keep these networks as small as possible, we make extensive use of shortest path calculations in the flat-network, where arc lengths are taken to be the arc transit times. Let  $\gamma_k(n_1, n_2)$  denote the time of the shortest flat-path (least transit time) from  $n_1$  to  $n_2$ , with  $n_1, n_2 \in N$ . The definition depends on  $k$ , since shortest paths for commodity  $k$  can be tightened by not allowing arcs that enter  $o^k$  or leave  $d^k$  to appear in the path.



The conditions that determine whether a timed-arc appears in  $\mathcal{A}_{\mathcal{T}}^k$ , for commodity  $k \in K$ , are summarized below, where we take  $\tau_{(n,n)} := 0$  for all  $n \in N$ . The set  $\mathcal{A}_{\mathcal{T}}^k$  is the set of all timed-node pairs,  $((n_1, t_1, t'_1), (n_2, t_2, t'_2))$ , for which all of the conditions below hold:

- [[1]]  $(n_1, n_2) \in A$  or  $n_1 = n_2$  and  $t'_1 = t_2$ .
- [[2]] There exists a path in  $A$  from  $o^k$  to  $d^k$  that uses  $(n_1, n_2)$ .
- [[3]]  $e^k + \gamma_k(o^k, n_1) < \min\{t'_1, t'_2 - \tau_{(n_1, n_2)}\}$ .
- [[4]]  $\max\{e^k + \gamma_k(o^k, n_1) + \tau_{(n_1, n_2)}, t_1 + \tau_{(n_1, n_2)}, t_2\} + \gamma_k(n_2, d^k) \leq l^k$ .
- [[5]]  $\tau_{(n_1, n_2)} < t'_2 - t_1$  and, if  $n_1 \neq n_2$ , then  $t_2 - t'_1 < \tau_{(n_1, n_2)}$ .

As before, the timed-arcs are partitioned into dispatch-arcs,  $\mathcal{D}_{\mathcal{T}}^k$ , and holding-arcs,  $\mathcal{H}_{\mathcal{T}}^k$ . The former set is defined as the set of all timed-arcs satisfying the first part of condition [[1]]; the latter is the set of all timed-arcs satisfying the second part of condition [[1]]. The utility for condition [[2]] is obvious. Condition [[3]] ensures that a path starting at  $o^k$  can reach node  $n_1$  before the end of the interval  $[t_1, t'_1]$  and use arc  $(n_1, n_2)$  to reach node  $n_2$  before the end of the interval  $[t_2, t'_2]$ . Condition [[4]] ensures that there exists a path using arc  $(n_1, n_2)$  that starts at  $o^k$  no earlier than the early time of commodity  $k$ , departs  $n_1$  no earlier than  $t_1$ , departs  $n_2$  no earlier than  $t_2$ , and reaches  $d^k$  before the late time for commodity  $k$ . Condition [[5]] ensures that there must exist  $t \in [t_1, t'_1]$  such that  $t + \tau_{(n_1, n_2)} \in [t_2, t'_2]$ .

Together, conditions [[2]]–[[5]] ensure that there exists a path for commodity  $k$  that uses arc  $(n_1, n_2)$  (or is held at  $n_1$  if  $n_2 = n_1$ ), departing from  $n_1$  in the interval  $[t_1, t'_1]$  and arriving at  $n_2$  in the interval  $[t_2, t'_2]$ , while starting at the origin of  $k$  and ending at its destination within its required early-late time window.

To use the time-expanded commodity networks in CTSNDP( $\mathcal{T}$ ), simply replace  $\mathcal{A}_{\mathcal{T}}$  by  $\mathcal{A}_{\mathcal{T}}^k$ ,  $\delta_i^+$  by  $\delta_{i,k}^+$ , and  $\delta_i^-$  by  $\delta_{i,k}^-$ , where  $\delta_{i,k}^+$  and  $\delta_{i,k}^-$  denote the set of timed-arcs in  $\mathcal{A}_{\mathcal{T}}^k$  entering and leaving node-interval  $i$ , respectively. Also, replace  $\mathcal{D}_{\mathcal{T}}$  by  $\mathcal{D}_{\mathcal{T}}^k := \cup_{k \in K} \mathcal{D}_{\mathcal{T}}^k$ .

### 3.2. Solutions

We now give notation and terminology that will be helpful in developing the theory underlying our discretization refinement approach.

First, we note that it is helpful to define  $\mathcal{T}^n := \{t \in [0, H] : (n, t) \in \mathcal{T}\}$ , so  $\mathcal{T}^n$  is the set of time points that appear at node  $n \in N$  in the discretization  $\mathcal{T}$ .

Let  $W^k$  be the set of all flat-paths, and let  $\mathcal{W}_{\mathcal{T}}^k$  be the set of all timed-paths, for commodity  $k \in K$ , so

$$W^k = \{P \in \mathcal{P}(A) : P \text{ is a flat-path from } o^k \text{ to } d^k\},$$

$$\mathcal{W}_{\mathcal{T}}^k = \{P \in \mathcal{P}(\mathcal{A}_{\mathcal{T}}^k) : P \text{ is a timed-path from } (o^k, t_o, t'_o) \text{ to } (d^k, t_d, t'_d)\},$$

where  $t_o \in \mathcal{T}^{o^k}$  is the unique time point such that  $t_o \leq e^k < t'_o$ , and  $t_d \in \mathcal{T}^{d^k}$  is the unique time point such that  $t_d \leq l^k < t'_d$ . A path-solution,  $Q^k \in \prod_{k \in K} W^k$ , is a set of flat-paths, one for each commodity. Let  $Q^k$  denote the flat-path in  $Q^k$  for commodity  $k \in K$ . So  $Q^k = \{Q^k\}_{k \in K}$ , where  $Q^k \in W^k$  for each  $k$ . A timed-path-solution,  $\mathcal{Q}_{\mathcal{T}}^k \in \prod_{k \in K} \mathcal{W}_{\mathcal{T}}^k$ , is a set of timed-paths, one for each commodity. Let  $\mathcal{Q}_{\mathcal{T}}^k$  denote the timed-path in  $\mathcal{Q}_{\mathcal{T}}^k$  for each commodity  $k \in K$ .

The set of commodities that use timed-arc  $a \in \mathcal{D}_{\mathcal{T}}^k$  in the timed-path-solution  $\mathcal{Q}_{\mathcal{T}}^k$  is denoted by

$$\kappa_a(\mathcal{Q}_{\mathcal{T}}^k) = \{k \in K : a \in \mathcal{Q}_{\mathcal{T}}^k\}.$$

Given a timed-path-solution,  $\mathcal{Q}_{\mathcal{T}}^k$ , we use the notation  $\mathcal{C}(\mathcal{Q}_{\mathcal{T}}^k)$  to denote the set of consolidations it suggests.

Solutions to CTSNDP( $\mathcal{T}$ ) are known as timed-solutions and, if feasible, have a corresponding timed-path-solution  $\mathcal{Q}_{\mathcal{T}}^k$ . A flat-solution,  $S$ , may now be defined as a pair of a path solution and a set of consolidations, that is,  $S = (Q^k, C)$ . Observe that there are only a finite number of flat-solutions, but that for each flat-solution there are possibly infinitely many corresponding timed-solutions.

A flat-solution  $S = (Q^k, C)$  is called *representable* in  $\mathcal{Q}_{\mathcal{T}}^k$  if there exists a feasible solution to CTSNDP( $\mathcal{T}$ ), that is,  $\mathcal{Q}_{\mathcal{T}}^k$ , that can be mapped to  $S$ . Similarly, a flat-solution  $S$  is called *implementable* if there exists a feasible solution to CTSNDP that can be mapped to  $S$ . Note that implementability implies representability, but not the converse.

Therefore, a flat-solution  $S$  is implementable if and only if there exist departure times  $t_n^k$  (or arrival time if  $n = d^k$ ), for every commodity  $k \in K$  and each  $n \in Q^k$ , such that (2), below, holds. In order to achieve consolidation  $((n_1, n_2), \kappa) \in C$ , we require  $t_{n_1}^{k_1} = t_{n_1}^{k_2}$  for all  $k_1, k_2 \in \kappa$ ; this can be modeled with the introduction of an auxiliary variable,  $s_{n_1, n_2}^{\kappa}$ , which represents the departure time of the consolidating commodities  $\kappa$  along arc  $(n_1, n_2)$ , as follows:

$$\begin{aligned} t_{o^k}^k &\geq e^k, & \forall k \in K, \\ t_{n_2}^k &\geq t_{n_1}^k + \tau_{(n_1, n_2)}, & \forall k \in K, (n_1, n_2) \in Q^k, \\ t_{d^k}^k &\leq l^k, & \forall k \in K, \text{ and} \\ t_{n_1}^k &= s_{n_1, n_2}^{\kappa}, & \forall ((n_1, n_2), \kappa) \in C, k \in \kappa. \end{aligned} \quad (2)$$

A set of times,  $(t_n^k)_{n \in N}^{k \in K}$ , or  $(t_n^k)$  for short, satisfying (2) is referred to as a *corresponding timing*.

### 3.3. Solution Graph

Visualizing a flat-solution as a graph is an intuitive way to see the interactions and flow of commodities in the form of consolidations. Furthermore, as will be shown later, it is also instrumental in the check for implementability and the choice for refining the discretization. Let  $GS(S) = (\mathcal{V}, \mathcal{E})$  be a directed graph constructed from the flat-solution  $S = (Q^k, C)$ , where the nodes  $\mathcal{V}$  are the union of commodity dispatch



nodes,  $\mathcal{V}^t = \{t_n^k : k \in K, n \in Q^k\}$ , and consolidation nodes,  $\mathcal{V}^s = \{s_{(n_1, n_2)}^\kappa : ((n_1, n_2), \kappa) \in C\}$ ; and edges  $\mathcal{E}$  are defined by

$$\begin{aligned} \mathcal{E} = & \left\{ \left( t_{n_1}^k, s_{(n_1, n_2)}^\kappa \right) : s_{(n_1, n_2)}^\kappa \in \mathcal{V}^s, k \in \kappa, \text{ with } |\kappa| > 1 \right\} \\ & \cup \left\{ \left( t_{n_1}^k, t_{n_2}^k \right) : s_{(n_1, n_2)}^\kappa \in \mathcal{V}^s, k \in \kappa, \text{ with } |\kappa| > 1 \right\} \\ & \cup \left\{ \left( t_{n_1}^k, t_{n_2}^k \right) : s_{(n_1, n_2)}^\kappa \in \mathcal{V}^s, k \in \kappa \text{ with } |\kappa| = 1 \right\}. \end{aligned}$$

The edge lengths, referred to as transit times, are defined by  $\rho_{(v_1, v_2)}$  for all  $(v_1, v_2) \in \mathcal{E}$ , where

$$\rho_{(v_1, v_2)} = \begin{cases} \tau_{(n_1, n_2)} & \text{if } (v_1, v_2) = (t_{n_1}^k, t_{n_2}^k) \text{ for some } \\ & k \in K, (n_1, n_2) \in Q^k, \\ \tau_{(n_1, n_2)} & \text{if } (v_1, v_2) = (s_{(n_1, n_2)}^\kappa, t_{n_2}^k) \text{ for some } \\ & ((n_1, n_2), \kappa) \in C, k \in \kappa, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that these nodes, edges, and edge lengths are a partial representation of the inequalities defining implementability in (2). To illustrate these definitions, consider the example shown in Figures 1, 2, and 3 and Table 1.

**3.3.1. Implementability.** To see how the solution graph can be used to check for implementability, first consider that the system of inequalities in (2) can be rearranged to give

$$\begin{aligned} -t_{n_1}^k &\leq -e^k, & \forall k \in K, \\ t_{n_1}^k - t_{n_2}^k &\leq -\tau_{(n_1, n_2)}, & \forall k \in K, (n_1, n_2) \in Q^k, \\ t_{d^k}^k &\leq l^k, & \forall k \in K, \\ t_{n_1}^k - s_{(n_1, n_2)}^\kappa &\leq 0, & \forall ((n_1, n_2), \kappa) \in C, k \in \kappa, \text{ and} \\ s_{(n_1, n_2)}^\kappa - t_{n_1}^k &\leq 0, & \forall ((n_1, n_2), \kappa) \in C, k \in \kappa. \end{aligned} \quad (3)$$

These inequalities can be written as a system of difference constraints by adding dummy variables  $D, D'$ :

$$\begin{aligned} D' - D &\leq 0, \\ D - t_{o^k}^k &\leq -e^k, & \forall k \in K, \\ t_{n_1}^k - t_{n_2}^k &\leq -\tau_{(n_1, n_2)}, & \forall k \in K, (n_1, n_2) \in Q^k, \\ t_{d^k}^k - D' &\leq l^k, & \forall k \in K, \\ t_{n_1}^k - s_{(n_1, n_2)}^\kappa &\leq 0, & \forall ((n_1, n_2), \kappa) \in C, k \in \kappa, \text{ and} \\ s_{(n_1, n_2)}^\kappa - t_{n_1}^k &\leq 0, & \forall ((n_1, n_2), \kappa) \in C, k \in \kappa. \end{aligned} \quad (4)$$

Figure 1. Network

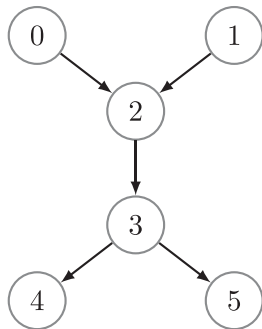
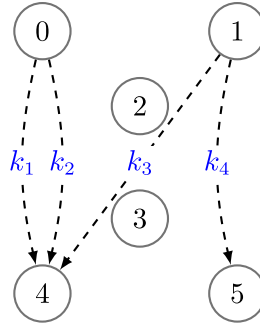


Figure 2. (Color online) Commodities



Thus, by extending the solution graph  $GS(S) = (\mathcal{V}, \mathcal{E})$  to include dummy nodes  $D, D'$  and arcs corresponding to (4), the constraint graph  $\widehat{GS}(S) = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$  is defined such that

$$\begin{aligned} \widehat{\mathcal{V}} &= \mathcal{V} \cup \{D, D'\}, \\ \widehat{\mathcal{E}} &= \mathcal{E} \cup \{(D, D')\} \\ &\quad \cup \{(D, t_{o^k}^k) : k \in K\} \\ &\quad \cup \{(t_{d^k}^k, D') : k \in K\} \\ &\quad \cup \{(s_{(n_1, n_2)}^\kappa, t_{n_1}^k) : (t_{n_1}^k, s_{(n_1, n_2)}^\kappa) \in \mathcal{E}\}, \end{aligned}$$

and edge length (as transit time), for all  $(v_1, v_2) \in \widehat{\mathcal{E}}$ ,

$$\widehat{\rho}_{(v_1, v_2)} = \begin{cases} e^k & \text{if } (v_1, v_2) = (D, t_{o^k}^k) \text{ for some } k \in K, \\ -l^k & \text{if } (v_1, v_2) = (t_{d^k}^k, D') \text{ for some } k \in K, \\ 0 & \text{if } (v_1, v_2) = (D, D'), \\ \rho_{(v_1, v_2)} & \text{otherwise.} \end{cases}$$

Observe that  $\widehat{GS}(S)$  is the graph representation of the inequalities in (4) and that  $GS(S)$  is a subgraph of  $\widehat{GS}(S)$ . To illustrate these extensions, Figure 4 shows the constraint graph corresponding to the example solution graph in Figure 3, with the additional nodes and edges highlighted for comparison.

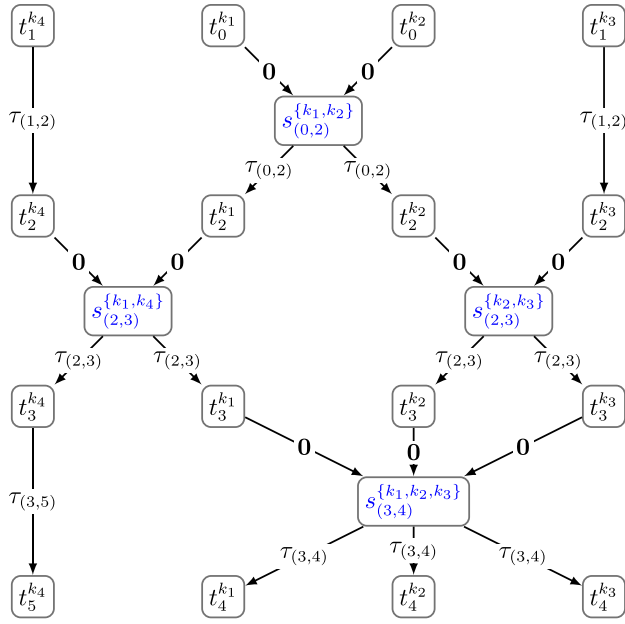
**Proposition 1.** *The flat-solution  $S = (Q^K, C)$  is implementable if and only if there does not exist a positive-length cycle in the constraint graph  $\widehat{GS}(S)$ .*

Since the solution and constraint graphs have a significant role in the DDDI algorithm, the following definitions are used to simplify the notation.

Table 1. Example  $S = (Q^K, C)$

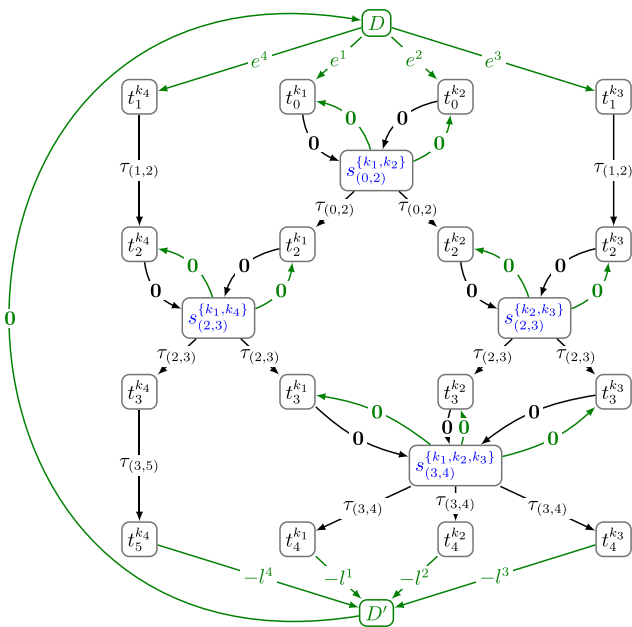
$k$	$Q^k$	$(n_1, n_2)$	$\kappa$
$k_1$	0,2,3,4	0,2	$\{k_1, k_2\}$
$k_2$	0,2,3,4	1,2	$\{k_3\}$
$k_3$	1,2,3,4	1,2	$\{k_4\}$
$k_4$	1,2,3,5	2,3	$\{k_1, k_4\}$
		2,3	$\{k_2, k_3\}$
		3,4	$\{k_1, k_2, k_3\}$
		3,5	$\{k_4\}$

**Figure 3.** (Color online) Solution Graph  $GS(S)$  Example



Let  $\tilde{P} = (\tilde{n}_1, \tilde{n}_2, \dots, \tilde{n}_r)$  be a simple path on the network  $\tilde{G} = (\tilde{N}, \tilde{A})$ . For node  $n \in \tilde{P}$  we write  $\tilde{P} \rightarrow^n$  to denote the subpath of  $\tilde{P}$  given by  $(\tilde{n}_1, \dots, \tilde{n}_i)$ , where  $i$  is the unique index,  $i \in \{1, \dots, r\}$ , such that  $\tilde{n}_i = n$ . Now, extend this notation to handle simple cycles: let the simple cycle  $\tilde{P}$  be specified as  $\tilde{P} = (n_1, n_2, \dots, n_r, n_1)$ ; we define  $\tilde{P} \rightarrow^{n_1}$  to be  $(n_1)$ , that is, the first occurrence of  $n_1$ . Since the remaining nodes  $n \in \tilde{P} \setminus \{n_1\}$  are uniquely indexed, the function is (already) well defined.

**Figure 4.** (Color online) Constraint Graph  $\widehat{GS}(S)$  Example



For any function of the form  $\psi: \tilde{A} \rightarrow \mathbb{R}$ , we write

$$\psi(\tilde{P}) = \sum_{a \in \tilde{P}} \psi(a),$$

or similarly, for any vector  $\psi \in \mathbb{R}^{|\tilde{A}|}$ , we write

$$\psi(\tilde{P}) = \sum_{a \in \tilde{P}} \psi_a.$$

As an example, given flat-solution  $S$ , let  $P$  be a simple path in the solution graph  $GS(S)$ . Then, the total transit time for that path is given by  $\rho(P)$ , and for node  $t_n^k \in P$ ,  $\rho(P \rightarrow t_n^k)$  gives the total transit time of the path up to and including node  $t_n^k$ .

### 3.4. Exploiting Structure to Evaluate Implementability

Based on Proposition 1, a check for implementability for the flat-solution  $S$  is reduced to a check for a positive-length cycle in the corresponding constraint graph, which can be done efficiently, in principle, by applying a label-correcting shortest path algorithm (with cycle detection) to the network. In practice, it is possible to exploit special structure in the network, allowing the positive-length cycles to be detected more efficiently via a two-step process, in which the second step is a shortest path algorithm in an acyclic graph. In addition, the information calculated in the second step can be used to determine how to refine the discretization. The special structure is as follows.

Observe that, from the construction of the constraint graph, it is clear that a cycle in the constraint graph must take one of three forms:

1. it consists of the pair of zero-length arcs corresponding to solution graph arcs of the form  $(t_n^k, s_{n,n'}^k)$ ,
2. it corresponds to a cycle in the solution graph, or
3. the cycle uses  $(D', D)$ .

The first form cannot have positive length. Any cycle in the constraint graph that corresponds to a cycle in the solution graph is of the second form and must have positive cost. This gives the first main result.

**Lemma 1.** *If there is a cycle in the solution graph  $GS(S)$ , then (2) has no solution, and hence,  $S$  is non-implementable.*

Finally then, consider the case when the solution graph is acyclic and any cycle in the constraint graph must have the form  $(D, P, D')$ , where  $P$  is a path in the constraint graph from node  $t_{o_k}^k$  to node  $t_{d_{k'}}^{k'}$  for  $k$  and  $k'$  two (possibly identical) commodities. The path  $P$  might possibly include zero-length loops around pairs of the form  $(t_n^k, s_{n,n'}^k)$ ; however, these can be simply ignored.

Since a positive-length cycle exists in a graph if and only if there exists a *simple* positive-length cycle, the flat-solution  $S$  is implementable if and only if there

is no simple positive-length cycle in the constraint graph, which (in this case) holds if and only if there is no positive-length cycle in the constraint graph of the form  $(D, P, D')$ , where  $P$  is a simple path in the constraint graph from  $t_{o^k}^k$  to  $t_{d^{k'}}^{k'}$  for some  $k, k'$ . This holds if and only if there is no path  $P'$  in the solution graph from  $t_{o^k}^k$  to  $t_{d^{k'}}^{k'}$  with  $e^k + \rho(P') - l^{k'} > 0$ . This gives the second main result.

**Lemma 2.** *If the solution graph is acyclic, then (2) has no solution, and hence  $S$  is non-implementable, if and only if there exist commodities  $k, k' \in K$  (possibly identical) and a simple path  $P$ , from  $t_{o^k}^k$  to  $t_{d^{k'}}^{k'}$ , in the solution graph with  $e^k + \rho(P) > l^{k'}$ .*

### 3.5. Earliest Departure Time

Given the solution  $S = (Q^K, C)$ , with acyclic solution graph  $GS(S) = (\mathcal{V}, \mathcal{E})$ , let the earliest departure time  $E_v$  be defined for a node  $v \in \mathcal{V}$  such that

$$E_v = \begin{cases} e^k & \text{if } v = t_{o^k}^k \text{ for some } k \in K, \\ \max_{k \in K} (E_{t_{n_1}^k}) & \text{if } v = s_{n_1, n_2}^k \text{ for some } ((n_1, n_2), \kappa) \in C, \\ E_{v_1} + \rho(v_1, v) & \text{if } v = t_{n^k}^k, n \neq o^k, \text{ and } (v_1, v) \in \mathcal{E} \text{ for some } k \in K. \end{cases}$$

Note that  $E_v$  is well defined due to the implicit structure of the solution graph, that is, if  $v \in \mathcal{V}$  has  $v = t_{n^k}^k, n \neq o^k$ , for some  $k$  and  $n$ , then there must exist a unique arc of the form  $(v_1, v) \in \mathcal{E}$ . Thus, we have the following result.

**Lemma 3.** *For the solution  $S = (Q^K, C)$  with acyclic solution graph, the length of the shortest path from  $D$  to  $v$  in the constraint graph is  $E_v$ .*

## 4. Refining the Discretization

Consider all the feasible solutions to CTSNDP( $\mathcal{T}$ ), with discretization  $\mathcal{T}$ . Recall that there are only a finite number of flat-solutions corresponding to all these feasible solutions, and at least one of these is implementable (assuming that a feasible continuous-time solution exists, that is,  $e^k + \gamma_k(o^k, d^k) \leq l^k$  for all  $k \in K$ ). A discretization is *refined* by splitting intervals, that is, by adding time points to the discretization, that is,  $\mathcal{T}'$  is a refinement of  $\mathcal{T}$  if  $\mathcal{T} \subset \mathcal{T}'$ . Suppose that the flat-solution  $S$ , corresponding to a feasible solution of CTSNDP( $\mathcal{T}$ ), is not implementable. Then, the goal of refining the discretization is to “remove”  $S$ , that is, to choose a refinement  $\mathcal{T}'$  such that  $S$  no longer corresponds to a feasible solution to CTSNDP( $\mathcal{T}'$ ). Intuitively, since there are only finitely many non-implementable flat-solutions, repeating this process finitely many times will guarantee that an optimal solution of CTSNDP( $\mathcal{T}'$ ) will be implementable and, hence, continuous-time optimal (since its value provides

a lower bound on the value of an optimal solution to CTSNDP).

Suitable time points can be determined by exploiting the structure of  $S$ . Specifically, they correspond to particular paths and cycles in the solution graph  $GS(S)$ . The following lemma and theorems cover these concepts in detail, but first it is important to establish the connection between the solution graph and the time-expanded network induced by the discretization  $\mathcal{T}$ .

**Observation 1.** *The solution graph is the network equivalent of a flat-solution, so, if the flat-solution  $S = (Q^K, C)$  is representable in  $\mathcal{G}_{\mathcal{T}}^K$ , then every path in  $GS(S)$  has a corresponding path in  $\mathcal{G}_{\mathcal{T}}^K$ . Furthermore, any cycle in  $GS(S)$  must also have a corresponding cycle in  $\mathcal{G}_{\mathcal{T}}^K$ —to see this, consider that cycles in the solution graph correspond to consolidations in the timed-solution that include at least one commodity that travels back in time (see Section 4.1.3).*

**Observation 2.** *Consider the discretization  $\mathcal{T}' \supset \mathcal{T}$ , that is,  $\mathcal{T}'$  is a refinement of  $\mathcal{T}$ , and assume the flat-solution  $S = (Q^K, C)$  is representable in  $\mathcal{G}_{\mathcal{T}}^K$ ; if there exists a path in  $GS(S)$  that does not have a corresponding path in  $\mathcal{G}_{\mathcal{T}'}^K$ , then  $S$  is not representable in  $\mathcal{G}_{\mathcal{T}'}^K$ —that is, no feasible solutions to CTSNDP( $\mathcal{T}'$ ) can correspond to  $S$ . This holds similarly for cycles.*

These two observations are essential to the theorems below. In addition to these, Lemma 4 gives a lower bound on the dispatch times of commodities along a simple path through the  $\mathcal{G}_{\mathcal{T}}^K$ , when  $\mathcal{T}$  contains time points of a particular form.

**Lemma 4.** *Let  $\widehat{P} = ((n_1, t_1, t'_1), (n_2, t_2, t'_2), \dots, (n_r, t_r, t'_r))$  be a simple path in  $(\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}^K)$  starting at  $n_1 = o^k$  with  $t_1 \geq e^k$ , for some  $k \in K$ , and suppose*

$$(n_i, e^k + \tau(\widehat{P} \rightarrow (n_i, t_i, t'_i))) \in \mathcal{T}, \quad \text{for all } i = 1, \dots, r,$$

then

$$t_i \geq e^k + \tau(\widehat{P} \rightarrow (n_i, t_i, t'_i)), \quad \text{for all } i = 1, \dots, r.$$

**Theorem 1.** *Let  $S$  be a non-implementable flat-solution that is representable in  $\mathcal{G}_{\mathcal{T}}^K$  and has an acyclic solution graph  $GS(S)$ . Furthermore, let  $P$  be a simple path in  $GS(S)$  from  $t_{o^{k_1}}^{k_1}$  to  $t_{d^{k_2}}^{k_2}$  for some  $k_1, k_2 \in K$  with  $e^{k_1} + \rho(P) > l^{k_2}$ , and  $e^{k_1} + \rho(P \rightarrow t_n^k) = E_{t_n^k}$ , where  $E_{t_n^k}$  is the earliest departure time from node  $t_n^k \in P$ . Then  $S$  is not representable in  $\mathcal{G}_{\mathcal{T}'}^K$ , with discretization refinement*

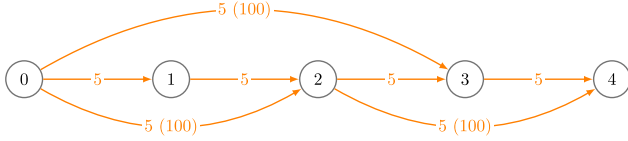
$$\mathcal{T}' = \mathcal{T} \cup \left\{ (n, E_{t_n^k}) \mid t_n^k \in P \cap \mathcal{V}^t \right\}.$$

Suppose that the solution to CTSNDP( $\mathcal{T}$ ) yields the non-implementable flat-solution  $S$ . Theorem 1 shows that there is a path in the solution graph  $GS(S)$  (which

can be easily found by a depth-first search on an acyclic graph) such that adding the time points along this path sufficiently refines the discretization to “remove”  $S$ .

Note that the time points used in Theorem 1 are not unique, nor minimal. To see this, consider the example in Figure 5 where a single commodity,  $k$ , travels from 0 to 4 with  $e^k = 0, l^k = 17$ . Travel times and costs (in brackets when different from travel time) are listed on the arcs. Let the initial discretization be  $\mathcal{T} = N \times \{0, 20\}$ .

**Figure 5.** (Color online) Example with Multiple Refinement Options



The path  $(0, 1, 2, 3, 4)$  is an optimal solution to  $\text{CTSNDP}(\mathcal{T})$ , but it is not  $k$ -feasible. By using Theorem 1,

$$\mathcal{T}'_1 = \mathcal{T} \cup \{(1, 5), (2, 10), (3, 15), (4, 20)\}.$$

However, it can be shown that

$$\mathcal{T}'_2 = \mathcal{T} \cup \{(2, t)\}, \text{ for any } t \in (7, 10],$$

is also a suitable refinement. To see that this single time point is sufficient, consider the time arcs  $((1, t_1, t'_1), (2, t, t'))$  and  $((2, t, t'), (3, t_3, t'_3))$ . Notice that they share the same node-interval  $(2, t, t')$ . From the conditions for time arcs, observe that  $t$  is valid for  $\llbracket 3 \rrbracket$  and  $\llbracket 4 \rrbracket$  on  $\mathcal{A}_{\mathcal{T}}^k$  when

$$\begin{aligned} e^k + \gamma_k(o^k, n_1) &< t'_2 - \tau_{(n_1, n_2)}, \\ 0 + \gamma_k(0, 2) &< t' - \tau_{(2, 3)}, \\ t' &> 10; \\ t_1 + \tau_{(n_1, n_2)} + \gamma_k(n_2, d^k) &\leq l^k, \\ t + \tau_{(2, 3)} + \gamma_k(3, 4) &\leq 17, \\ t &\leq 7. \end{aligned}$$

Adding the time point  $(2, t)$  creates the node-intervals  $(2, 0, t)$  and  $(2, t, 20)$ . If  $t$  is chosen to be within  $(7, 10]$ , this ensures that neither node-interval is valid for both inequalities above, and thus, at least one of the timed-arcs cannot exist. Hence, the timed-path cannot exist, and by Observation 2 the discretization has been sufficiently refined.

With this in mind, there can be many alternate discretization refinement approaches, and the choice can have a significant impact on the size of the discretization and speed of convergence. The next section considers a few approaches for special case scenarios.

#### 4.1. Special Cases

**4.1.1. Path.** Suppose that, for commodity  $k$ , the length of its path in the flat-solution is too long. Then, under certain circumstances (Theorem 2), it is possible to refine the discretization using a single unary time point. Unary

time points are almost always preferred, as they effectively refine the discretization without dramatically increasing the corresponding time-expanded network.

**Theorem 2.** If the non-implementable flat-solution  $S = (Q^K, C)$  is representable in  $\mathcal{G}_{\mathcal{T}}^K$  and there exist  $(n_1, n_2), (n_2, n_3) \in Q^k$  such that

$$e^k + \gamma_k(o^k, n_1) + \tau_{(n_1, n_2)} + \tau_{(n_2, n_3)} + \gamma_k(n_3, d^k) > l^k,$$

for some  $k \in K$ , then  $S$  is not representable in  $\mathcal{G}_{\mathcal{T}'}^K$ , where

$$\mathcal{T}' = \mathcal{T} \cup \{(n_2, e^k + \gamma_k(o^k, n_1) + \tau_{(n_1, n_2)})\}.$$

**4.1.2. Disjoint Time Window.** A time window for node  $n$  and commodity  $k$  defines the earliest and latest times that  $k$  could visit  $n$ . Suppose that in a flat-solution, at least two commodities  $k_1$  and  $k_2$  consolidate over  $(n_1, n_2)$  such that in reality the earliest time that  $k_1$  can reach  $n_2$  (traveling via  $n_1$ ) is after the latest time that  $k_2$  can leave and still reach its destination in time. In this scenario, commodities  $k_1$  and  $k_2$  have disjoint time windows, and under certain conditions (Theorem 3) it is possible to refine the discretization with a unary time point.

**Theorem 3.** If the non-implementable flat-solution  $S = (Q^K, C)$  is representable in  $\mathcal{G}_{\mathcal{T}}^K$  and there exists  $((n_1, n_2), \kappa) \in C$  such that

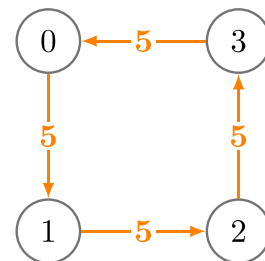
$$e^{k_1} + \gamma_{k_1}(o^{k_1}, n_1) + \tau_{(n_1, n_2)} + \gamma_{k_2}(n_2, d^{k_2}) > l^{k_2},$$

for some  $k_1, k_2 \in \kappa$ , then  $S$  is not representable in  $\mathcal{G}_{\mathcal{T}'}^K$ , where

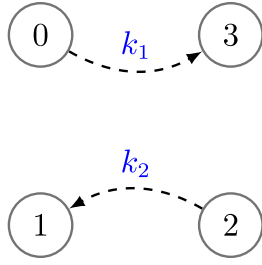
$$\mathcal{T}' = \mathcal{T} \cup \{(n_2, e^{k_1} + \gamma_{k_1}(o^{k_1}, n_1) + \tau_{(n_1, n_2)})\}.$$

**4.1.3. Cycle.** Theorem 1 assumes that the solution graph is acyclic; however, this is not always the case. If there exists a cycle, then Theorem 4 shows how to sufficiently refine the discretization. The structure of the time points chosen in Theorem 4 is similar to those chosen in Theorem 1; however, handling cycles typically requires significantly more time points. Observe that Theorems 2 and 3 do not make the acyclic assumption, and so even if a cycle is detected it may be possible (and preferable) to use those unary time point theorems. To see how a cycle could arise, consider the following example in Figures 6, 7, and 8 and Table 2.

**Figure 6.** (Color online) Network

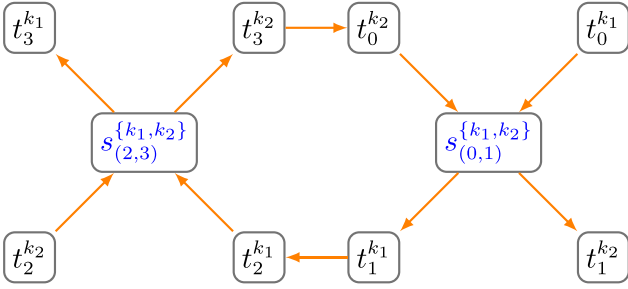




**Figure 7.** (Color online) Commodities**Table 2.** Cycle  $S = (Q^K, C)$ 

$k$	$Q^k$	$(n_1, n_2)$	$\kappa$
$k_1$	0,1,2,3	0,1	$\{k_1, k_2\}$
$k_2$	2,3,0,1	1,2	$\{k_1\}$
		2,3	$\{k_1, k_2\}$
		3,0	$\{k_2\}$

The solution graph for this example is given as follows.

**Figure 8.** (Color online) Example Cycle Instance

**Theorem 4.** Let  $S$  be a non-implementable flat-solution that is representable in  $\mathcal{G}_{\mathcal{T}}^K$ , with solution graph  $GS(S)$  containing the simple cycle  $\hat{P}$ . Furthermore, let  $P$  be a shortest path in  $GS(S)$  from  $t_n^{k_1}$  to  $\hat{P}$  for some  $k_1 \in K$ , let  $\{v\} = P \cap \hat{P}$  define the cycle such that  $\hat{P} = (v, \hat{P}_{(2)}, \dots, \hat{P}_{(|\hat{P}|)}, v)$ , and let  $\hat{m} = \min\{m : e^{k_1} + \rho(P) + m\rho(\hat{P}) > l^{k_1}\}$ . Then  $S$  is not representable in  $\mathcal{G}_{\mathcal{T}}^K$ , with refinement

$$\begin{aligned} \mathcal{T}' = \mathcal{T} \cup & \left\{ \left( n, e^{k_1} + \rho(P \rightarrow t_n^{k_1}) \right) \mid t_n^{k_1} \in P \cap \mathcal{V}^t \right\} \\ & \cup \left\{ \left( n, e^{k_1} + \rho(P) + m\rho(\hat{P}) + \rho(\hat{P} \rightarrow t_n^{k_1}) \right) \mid \right. \\ & \left. t_n^{k_1} \in \hat{P} \cap \mathcal{V}^t, m = 0, \dots, \hat{m} \right\}. \end{aligned}$$

## 5. Refinement Strategies and DDDI Implementation

As already previewed in Section 2.3, for the non-implementable flat-solution  $S = (Q^K, C)$ , there may be many possible choices of paths, cycles, or arcs to apply Theorems 1, 2, 3, and 4. Thus, an effective refinement strategy is needed when implementing these theorems. The following subsections present

three strategies, each of which addresses a different issue: default, reduced iterations, and fewer time points. This section concludes with some useful ideas for implementation of the lower bound IP, including additional constraints and a preprocessing technique to tighten the IP model, and an adaptive approach to setting the optimality tolerance used in the IP solver.

### 5.1. Default Refinement Strategy

The default implementation is a direct translation of the main theorems. Here, the solution graph is first checked for cycles (Theorem 4), and if acyclic, it is checked for violated paths (Theorem 1), in either case adding time points as prescribed. Only one violation is fixed per iteration, chosen in order of the commodity index.

#### Algorithm 2 (Default Refinement Strategy)

- 1: **procedure**  $\text{REFINE\_DISCRETIZATION}(GS(S), \mathcal{T})$
- 2:   **if** a cycle exists in  $GS(S)$  **then**
- 3:     **return**  $\mathcal{T}'$  as per Theorem 4
- 4:   **else if** a failed path exists in  $GS(S)$  **then**
- 5:     **return**  $\mathcal{T}'$  for a single path per Theorem 1
- 6: **return**  $\mathcal{T}$

### 5.2. Reduced Iterations

One way to view the algorithm is that non-implementable flat-solutions are induced by *issues* in the time-expanded network, which are *fixed* by refining the discretization. Since there are many reasons why a flat-solution is non-implementable, it stands to reason that there are many corresponding issues in the time-expanded network that can be fixed. A natural extension to the default algorithm fixes multiple issues in each iteration, which intuitively reduces the total number of iterations required for convergence.

If the solution graph is composed of disconnected subgraphs, then the commodities in different components do not interact, and so, adding time points to fix an issue in one component is unlikely to fix issues in other components. With this in mind, adding the set of fixes, one from each component (if applicable), should reduce the number of required iterations without unnecessarily increasing the associated time-expanded network.

To extend this idea further, it can also be beneficial to add several fixes from within a component, assuming the component is acyclic. We say that the consolidation  $(a_2, \kappa_2)$  is *downstream* from the consolidation  $(a_1, \kappa_1)$  if there exists a directed path in the solution graph connecting  $s_{a_1}^{k_1}$  to  $s_{a_2}^{k_2}$ . Similarly,  $(a_1, \kappa_1)$  is said to be *upstream* from  $(a_2, \kappa_2)$ . Furthermore, we say that the commodity  $k$  has *failed* if its solution path  $Q^k$  is not  $k$ -feasible; and we say that the consolidation  $(a, \kappa)$  has *failed* if the last commodity to arrive at the

consolidation is later than the earliest time another commodity must leave (to reach its destination in time). We refer to these failed commodities and failed consolidations as *failures*.

A failure, and its corresponding fix, is considered *independent* if there exist no other failures upstream. Thus, for an acyclic component, it is suitable to add all the independent fixes, without having too much concern for redundancy.

The last improvement focuses on the path used in Theorem 1. This path is chosen by traversing the solution graph from  $t_{d^{k_2}}^{k_2}$ , for some commodity,  $k_2$ , that arrives late in the given solution, and follows the  $\arg \max_{k \in K} E_{t_n^k}$  commodity at each consolidation. Observe that there may be multiple paths that match this definition, and although each of these paths is suitable, there may be one that is better than the others. A reasonable strategy is to simply use all paths. Although this adds some redundancy, empirical evidence suggests that the benefits typically outweigh any overheads.

#### Algorithm 3 (DDDI with Reduced Iterations)

```

1: procedure REFINE_DISCRETIZATION( $GS(S), \mathcal{T}$ )
2:    $\mathcal{T}' \leftarrow \mathcal{T}$ 
3:   for all  $CS \in \text{COMPONENT\_SUBGRAPHS}(GS(S))$  do
4:     if a cycle exists in  $CS$  then
5:       Add time points to  $\mathcal{T}'$  per Theorem 4
6:     else
7:       for all independent failures in  $CS$  do
8:         Add time points to  $\mathcal{T}'$  per Theorem 1
           (all suitable paths)
   return  $\mathcal{T}'$ 
    
```

### 5.3. Fewer Time Points

In DDDI, the majority of the computational effort is spent solving the lower bound IP. Since this solve time is strongly correlated with the size of the model, it is preferable to keep the discretization as small as possible. By a careful application of Theorems 2 and 3, it becomes possible to fix more issues in each iteration using far fewer time points—and hence directly reduce the total, and per iteration, solve times. Since both these theorems use a single time point to fix a failure, we will refer to these as *unary* time points and *unary* failures, respectively.

By including these unary time points, the number of options for fixing failures increases dramatically. Building upon the reduced iterations strategy, we again prefer fixes that are upstream. Intuitively,

any failures that occur earlier in the discretization (i.e., further upstream) have more significance, due to the flow-on effects for paths and consolidations downstream. Since Theorems 2 and 3 do not assume an acyclic solution graph, our algorithm prioritizes fixing unary failures, preferring those as upstream as possible, before checking for cycles or adding time points from Theorem 1. Note that this preference for unary fixes means that unary failures that are downstream from nonunary failures can still be chosen.

For the failed consolidation  $(a, \kappa)$ , observe that there are potentially  $\binom{|K|}{2}$  ways to apply Theorem 3. We attempt to choose the corresponding unary time point in a clever way. We choose the pair of commodities with the smallest time window gap. The *time window* for commodity  $k \in K$  at node  $n \in Q^k$  is defined as

$$[e_n^k, l_n^k] = [e^k + \tau((Q^k)^{-n}), l^k - \tau(Q^k) + \tau((Q^k)^{-n})],$$

that is,  $e_n^k$  is the earliest time to reach node  $n$  along path  $Q^k$ , and similarly  $l_n^k$  is the latest time to leave node  $n$  and still reach the destination in time. This interval is well defined when  $Q^k$  is a  $k$ -feasible path. Thus, the time window gap for a pair of commodities  $k_1, k_2 \in K$  on consolidation  $((n_1, n_2), \kappa) \in C$  is given by

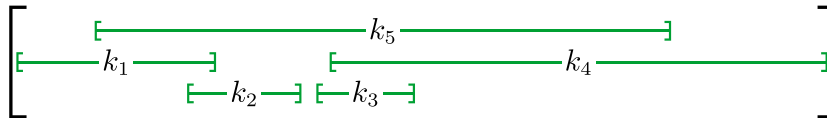
$$|e_{n_1}^{k_2} - l_{n_1}^{k_1}|.$$

Consider the example consolidation in Figure 9 where each commodity's time window is shown. There are up to four ways to apply Theorem 3, using commodity pairs  $(k_1, k_3)$ ,  $(k_1, k_4)$ ,  $(k_2, k_3)$ , or  $(k_2, k_4)$ . Our heuristic chooses  $(k_2, k_3)$  and, hence, partitions the consolidation evenly, preserving the clusters of overlapping commodities.

The intuition behind Theorem 3 is that the consolidation  $((n_1, n_2), \kappa)$  has disjoint time windows, that is,  $\cap_{k \in K} [e_{n_1}^k, l_{n_1}^k] = \emptyset$ . Moreover, it may be possible that  $\kappa$  has many clusters of commodities with nonoverlapping time windows; thus, there are potentially many commodity pairs,  $k_1, k_2 \in K$ , with disjoint time windows, that is,  $e_{n_1}^{k_2} > l_{n_1}^{k_1}$ . We choose the commodity pair that satisfies Theorem 3 with the smallest time window gap, as this is likely to preserve many clusters on either side of the partition.

Note that it may be possible to fix a cycle using unary time points from Theorem 3. This is preferred, since Theorem 4 requires potentially very many time points, due to the recursive nature of cycles. Although only a single unary time point may be sufficient to

Figure 9. (Color online) Disjoint Time Windows for a Consolidation



change a cycle, we have found that fixing every unary failure (around the cycle) is more effective at breaking it completely in one step, while still using far fewer time points than Theorem 4.

For cycles without unary failures, the number of time points can still be reduced by simply checking at each node whether any commodity (i.e., not only the one mentioned in Theorem 4) participating in the cycle has a shortest path to its destination that exceeds its late time; when this is found, no further time points are required. A similar idea can be applied to Theorem 1, that is, by using shortest paths it is possible to skip a number of nodes at the start and end of a violated path.

#### Algorithm 4 (DDDI with Fewer Time Points)

```

1: procedure REFINE_DISCRETIZATION( $GS(S), \mathcal{T}$ )
2:    $\mathcal{T}' \leftarrow \mathcal{T}$ 
3:   for all  $k \in K$  do
4:     Check Theorem 2 along path, add first time
       point (if exists) to  $\mathcal{T}'$ 
5:   for all unary failed consolidations do
6:     if consolidation is not downstream from
       another unary failure then
7:       Check Theorem 3 (add to  $\mathcal{T}'$  using
       prescribed heuristic)
8:   for all  $CS \in \text{COMPONENT\_SUBGRAPHS}(GS(S))$  do
9:     if cycle exists in  $CS$  that is not downstream a
       failure then
10:      if cycle can be broken using unary time
        points then
11:        Add time points to  $\mathcal{T}'$  per Theorem 3
12:      else
13:        Add time points to  $\mathcal{T}'$  per reduced
        Theorem 4
14:      else
15:        for all independent failures in  $CS$  not
        downstream from any fixes do
16:          Add time points to  $\mathcal{T}'$  per reduced
        Theorem 1 (all suitable paths)
    return  $\mathcal{T}'$ 

```

#### 5.4. Strengthening the IP Model

We include an additional valid inequality in the IP model that is used to tighten the formulation:

$$\left\lceil \frac{q^k}{u_a} \right\rceil x_a^k \leq z_a, \quad \forall k \in K, a \in \mathcal{D}_{\mathcal{T}}^k. \quad (5)$$

The benefits of including this inequality, computationally, are quite substantial (see Section 6).

We also observe that some timed-arcs can safely be removed, as there will always exist an alternative feasible solution with the same cost. We can then remove these, reducing the time-expanded network. Our reasoning is as follows.

The fixed and variable costs in our definition of CTSNDP are constant over time. Thus, the cost of an implementable solution does not depend on the time at which commodities are dispatched on an arc; all that matters is that commodities to be consolidated on an arc are dispatched on that arc at the same time. It suffices, then, to only consider dispatch times that occur as early as possible. We may, as a consequence, deduce that some arcs in the time-expanded network are redundant, in the sense that some optimal solution that does not use them must exist.

To be specific, we first define  $\mathcal{H}_a$  as the set of commodities that can use timed-arc  $a$ , so

$$\mathcal{H}_a := \{k \in K : a \in \mathcal{D}_{\mathcal{T}}^k\},$$

for each timed-arc  $a$ . Note that we only need this notation for  $a$  a dispatch-arc. Now, if, for any dispatch-arc  $a = ((n, s, s'), (n', t_1, t'_1)) \in \bigcup_{k \in K} \mathcal{D}_{\mathcal{T}}^k$ , there exists another, earlier, timed-arc  $a' = ((n, s, s'), (n', t_2, t'_2)) \in \bigcup_{k \in K} \mathcal{D}_{\mathcal{T}}^k$ , with  $t_2 < t_1$ , and with the property that  $\mathcal{H}_a \subseteq \mathcal{H}_{a'}$ , then  $a$  may safely be removed. It is important to note that here  $a$  and  $a'$  have the same node-interval as their tail and have a common node in the flat-network,  $n'$ , in their head node-interval.

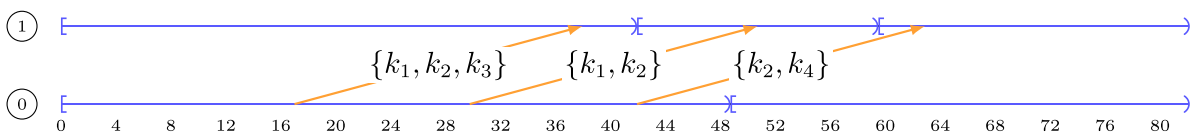
Figure 10 provides an illustration of this concept. It shows three dispatch-arcs from node-interval  $(0, 0, 48)$  to three distinct node-intervals at node 1 in the flat-network:  $(1, 0, 42)$ ,  $(1, 42, 60)$ , and  $(1, 60, 82)$ , respectively. Here

$$\mathcal{H}_{a_1} = \{k_1, k_2, k_3\}, \quad \mathcal{H}_{a_2} = \{k_1, k_2\}, \quad \text{and} \\ \mathcal{H}_{a_3} = \{k_2, k_4\},$$

where  $a_1 = ((0, 0, 48), (1, 0, 42))$ ,  $a_2 = ((0, 0, 48), (1, 42, 60))$ , and  $a_3 = ((0, 0, 48), (1, 60, 82))$ . Since  $\mathcal{H}_{a_2} \subseteq \mathcal{H}_{a_1}$  and  $a_2$  is a later arc, any solution using  $a_2$  for commodities in  $\mathcal{H}_{a_2}$  could instead use  $a_1$  and then use holding-arc  $((1, 0, 42), (1, 42, 60))$ , giving an equivalent solution with no greater cost. Thus,  $a_2$  may safely be removed. Note that  $a_3$  cannot be removed, as  $k_4 \in \mathcal{H}_{a_3} \setminus \mathcal{H}_{a_1}$ .

In general, if  $a$  meets the above condition for removal, then it is removed from  $\mathcal{D}_{\mathcal{T}}^k$  for all  $k \in \mathcal{H}_a$ . This means that the corresponding variables in the lower bound IP model— $x_a^k$  for all  $k \in \mathcal{H}_a$  and  $z_a$ —are never created.

Figure 10. (Color online) Redundant Consolidation Arc:  $(k_1, k_2)$



We call the procedure for removing arcs in this fashion `REMOVE_REDUNDANT_ARCS`, and we note that it can be implemented quite efficiently by exploiting the observation that if  $k \in \mathcal{K}_{a_1} \cap \mathcal{K}_{a_3}$  where  $a_1 = ((n, s, s'), (n', t_1, t'_1))$  and  $a_3 = ((n, s, s'), (n', t_3, t'_3))$  with  $t'_1 < t_3$ , then for any node-interval  $(n', t_2, t'_2)$  with  $t'_1 \leq t_2 < t'_2 \leq t_3$ , it must be that  $k \in \mathcal{K}_{a_2}$ , where  $a_2 = ((n, s, s'), (n', t_2, t'_2))$ . Thus, to decide whether a dispatch-arc can be removed, only the dispatch-arc from the same tail node-interval to the immediately preceding head node-interval need be inspected.

### 5.5. Adaptive IP Optimality Tolerance

In practice, we observe that, for the first several iterations of the DDDI algorithm, there is a large gap between its upper and lower bounds. Thus, there is very little benefit in using a small optimality tolerance in the IP solver used to find the next lower bound. Instead of having a fixed tolerance across all iterations, we instead adapt the tolerance as the DDDI algorithm progresses.

To avoid confusion, we define the following terms: `MIP_GAP`, `MIP_TOL`, `DDDI_GAP`, and `DDDI_TOL`. Every iteration of the DDDI algorithm solves an IP with a target `MIP_TOL`; we refer to the IP's optimality gap (after terminating the solve) as the `MIP_GAP`. Each iteration in Algorithm 1 has  $\text{DDDI\_GAP} = \frac{\text{upper\_bound} - \text{lower\_bound}}{\text{upper\_bound}}$ , which is the continuous-time optimality gap. The algorithm terminates when  $\text{DDDI\_GAP} < \text{DDDI\_TOL}$ .

Our implementation starts with  $\text{MIP\_TOL} = 0.04$ , and then for each iteration,

$$\text{MIP\_TOL} = \max\{\text{DDDI\_GAP} \times 0.25, \text{DDDI\_TOL} \times 0.98\}.$$

These values have been chosen based on empirical experimentation. In particular, note that `MIP_TOL` can be set to a value strictly less than `DDDI_TOL`. This is not required but has been found (empirically) to improve convergence in some cases.

### 5.6. Updating the IP Model

In our implementation, updates to the IP model (`UPDATE_MODEL`) are performed in place, not regenerated each iteration, and are made so that variables and constraints are reused as much as possible. That is, variables are relabeled instead of added or deleted. This reduces time spent generating the model and provides the opportunity for the IP solver to perform a warm start at the next iteration.

## 6. Computational Study

In our computational study, we solve the 558 CTSNDP instances used in Boland et al. (2019) to a 0.001% optimality gap with a computational time limit of one hour. Memory is limited to the available system memory

(256 GB), and processing is restricted to a single core. CPLEX 12.6 is used as the MIP solver. Note that the instances range in size, with the smallest having 20 nodes, 228 arcs, and 39 commodities, and the largest having 30 nodes, 685 arcs, and 400 commodities. Transit times as well as available and due times are expressed as integers, representing times in minutes, which implies that the full discretization (FD) has a time point at every integer in the time horizon.

Each instance was solved using the FD, as well as with four different variants of the DDDI algorithm. The default, reduced iterations, and fewer time points variants of DDDI use the discretization refinement strategies of the same name, given in Algorithms 2, 3, and 4, respectively. The adaptive variant uses the refinement strategy from fewer time points (Algorithm 4) on the reduced time-expanded network (Section 5.4), as well as incorporates the adaptive tolerance when solving the lower bound IP (Section 5.5). All variants include the additional constraints, (5), in the IP model and use the warm starting described in Section 5.6. For all variants, CPLEX was used with its default MIP settings, other than setting the parameter “Symmetry” to its most aggressive level, 5. This duplicates the settings used to produce the DDD results given in Boland et al. (2017) and which our experiments confirmed as being the most effective.

The results are grouped by the *flexibility* and the *cost ratio* of the instances. Boland et al. (2019) show that these metrics are reasonable indicators of the tractability of an instance, and so grouping by this scheme can highlight performance characteristics that might otherwise be lost by taking the average across all instances. In what follows, an instance is said to have low flexibility (LF) if

$$\min_{k \in K} \{t^k - (e^k + \gamma_k(o^k, d^k))\} < 227,$$

and low cost ratio (LC) if

$$\frac{1}{|A|} \sum_{a \in A} \frac{f_a}{v_a u_a} < 0.175.$$

The choice of the threshold values, 227 and 0.175, has been influenced by the results in Boland et al. (2019) as well as by the desire to spread instances across the groups equally. The resulting group sizes can be seen in Table 3.

**Table 3.** Group Allocation of the 558 Instances

	LF	HF
HC	183	177
LC	94	104



The summary of results, averaged for each of the groups, can be found in Table 4. Note that the time required to build the model was not counted toward the computation time limit; some instances require over 30 minutes to build the FD model, whereas DDDI barely takes seconds. Also note that due to the computation time limit, the FD IP (after building the model) did not produce a feasible solution for 28 instances (all in the HC/HF group). For these instances a gap of 100% was used to produce the summary of the results; this choice seems reasonable, since many similar instances from this group had a gap of over 100%, and so the results should not be skewed in favor of DDDI.

As can be seen from the results, all variants of DDDI clearly outperform solving the FD model. Also, there are significant differences between the refinement strategies, in terms of number of iterations and average gap %. As expected, instances with high cost and high flexibility are the most difficult to solve. Nevertheless, the DDDI variants all do a remarkable job finding high-quality solutions for these instances within the time limit.

It is interesting to note that, for HC/HF, the fewer time points strategy solves fewer instances to optimality than reduced iterations; however, the average gap of

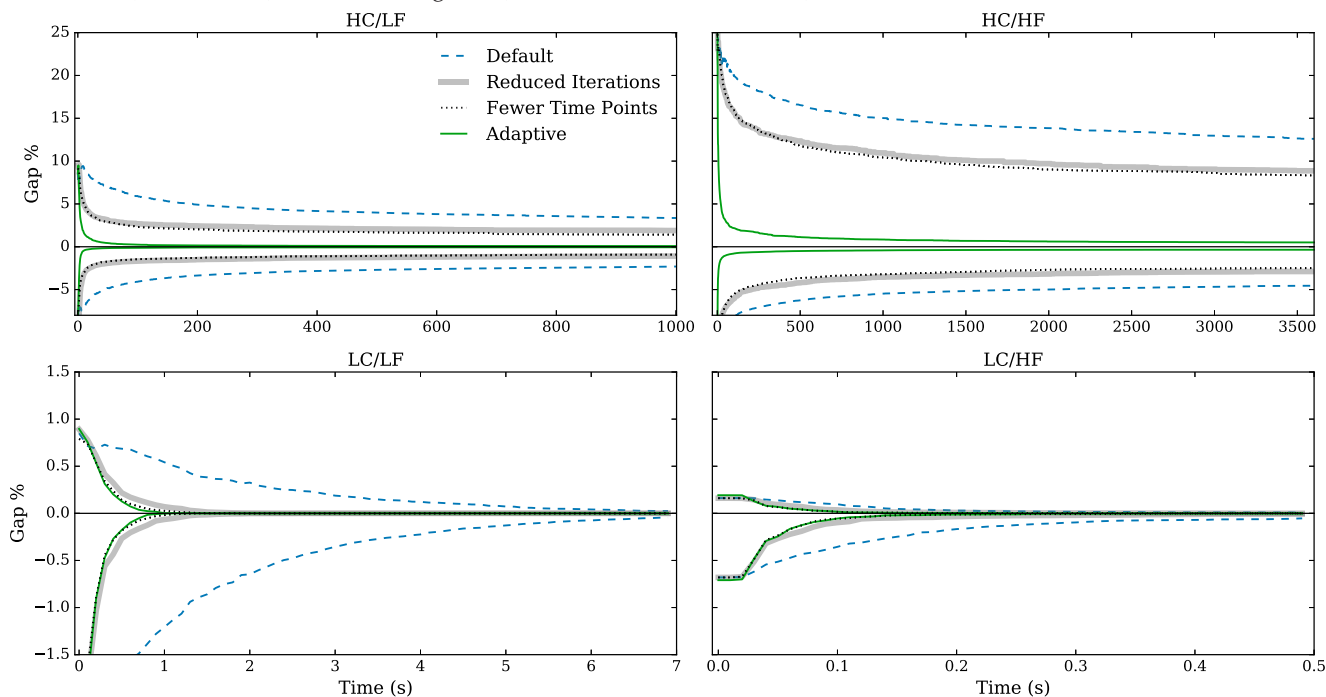
fewer time points is less. This highlights the delicate balance between convergence and model size and the difficulty of choosing a suitable refinement strategy. That is, in some instances the extra time points (of reduced iterations) help prove optimality in less time; however, on average, the smaller model size of fewer time points produces solutions with tighter optimality gaps.

To further compare the DDDI variants, Figures 11 and 12 show the convergence and model growth of the different approaches as a function of run time, where the latest time is either the time limit of one hour or the time when all approaches have reached an optimality gap of 0.001% or less (i.e.,  $\frac{\text{upper\_bound} - \text{lower\_bound}}{\text{lower\_bound}} \leq 0.00001$ ). The data for Figure 11 is produced by recording, for each instance, at each DDDI iteration, the time at which this iteration starts, the current lower bound, and the current upper bound. Each bound reported at each iteration of each instance is then recalculated as a gap relative to the optimal value<sup>1</sup> of the instance:  $\frac{\text{bound} - \text{optimal\_value}}{\text{optimal\_value}}$  is the relative gap, reported as a percentage. For *bound* taken to be the upper bound, we call this the *upper gap*; it will be nonnegative. Conversely, when *bound* is the lower bound, the result is the *lower gap*, which will be nonpositive. The plots in Figure 11 show, for each instance group and each DDDI

**Table 4.** (Color online) Computational Results for DDDI Variants vs. Full Discretization (One-Hour Solve Time Limit)

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

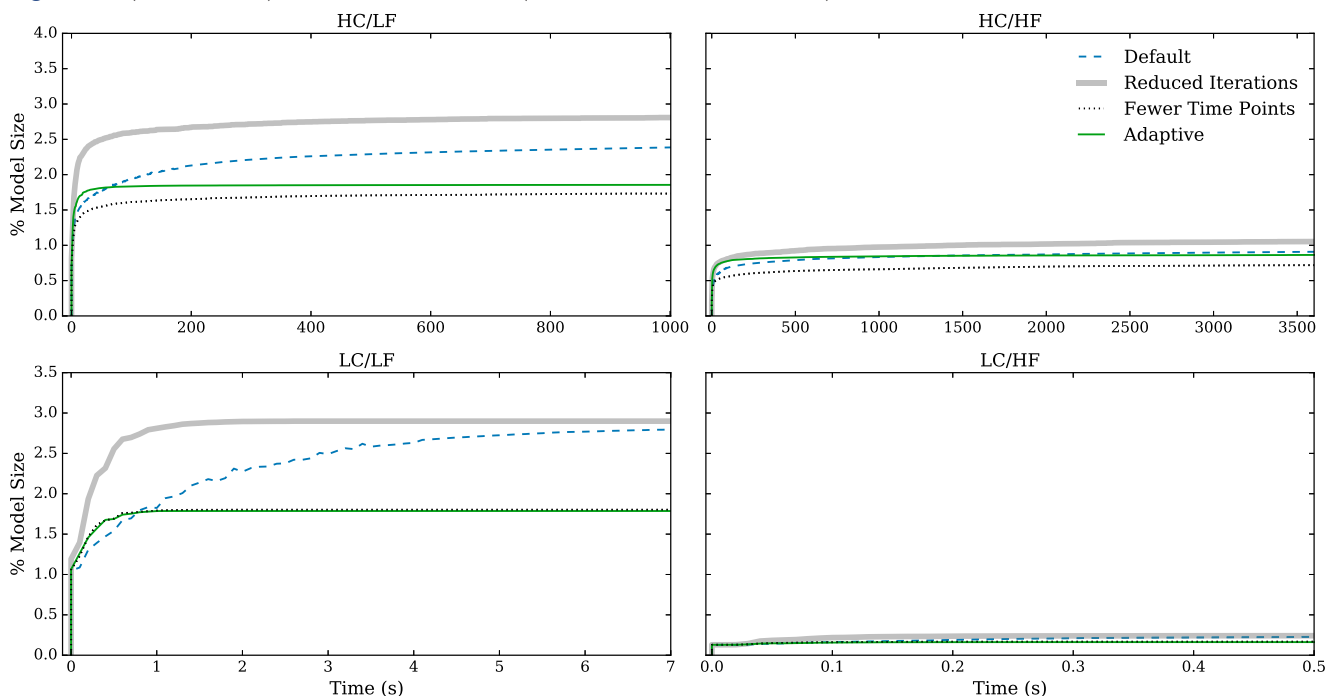
**Figure 11.** (Color online) DDDI Convergence (One-Hour Solve Time Limit)



variant, the average of these values over all instances in the group. The data for Figure 12 is produced by a similar process, recording, for each instance, at each DDDI iteration, the number of variables and the number of constraints in the IP model at this iteration. For every iteration of each DDDI variant running on each instance, the model size is calculated as the sum of

the number of variables and the number of constraints, taken as a percentage of that in the FD model for the instance. For all models, both from DDDI and from the FD, the numbers of variables and constraints are reported before presolve, that is, before CPLEX has performed its presolve routines, which identify and remove variables and constraints.

**Figure 12.** (Color online) DDDI Model Growth (One-Hour Solve Time Limit)



The adaptive variant has a remarkably quick initial convergence, even on the HC/HF instances. Most of the time is spent proving optimality. Since it typically produces feasible solutions at every iteration, it is obvious that DDDI is an excellent heuristic; it reliably finds good solutions quickly.

As expected, Figure 12 shows that, on average, the reduced iterations and fewer time points variants induce the largest and smallest model sizes, respectively. In all cases, most growth occurs near the start of the algorithm. Note that the average FD model size for each group is different, and so the groups cannot be directly compared with each other. However, even if these results are relatively weighted, we notice a similar pattern. In all DDDI variants, the IP models are very small relative to the FD. Note that the FD model has also been constructed using the conditions in Section 3.1; if the FD model was constructed naively, the relative size of the DDDI variant models would be even smaller.

Lastly, Figure 13 shows, for each instance group and each DDDI variant, the number of variables and the number of constraints in the IP model, each reported as a percentage of those in the FD model for the respective instance, at the final iteration of DDDI, both before and after the model is presolved, averaged over the instances in each group. Note that the values reported after presolve, presolve variables and presolve constraints, use the number of variables and constraints in the FD model after presolve, too, so that the comparison is fair. Thus, we can see that, in all

cases, DDDI produces models that are significantly smaller—typically over 100 times smaller—than those of the FD. This makes DDDI a more attractive approach for solving large-scale instances.

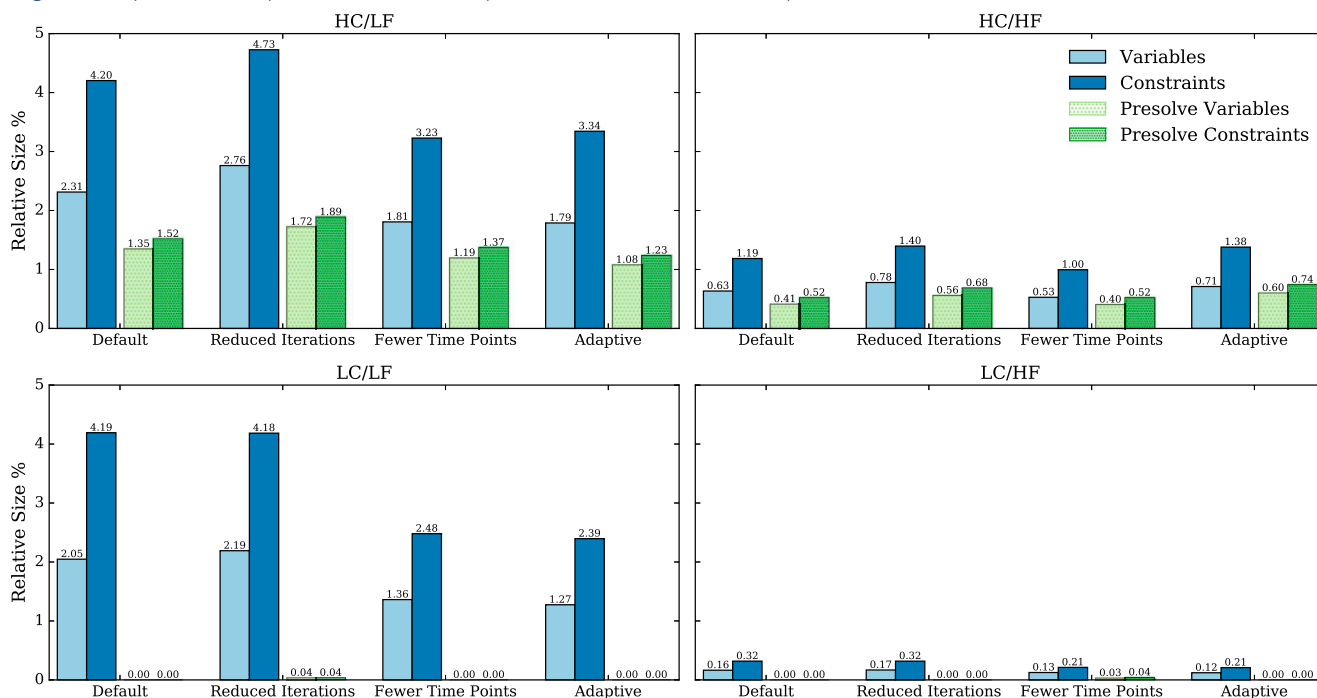
For the LC/LF and LC/HF groups, the (post)presolve variables and constraints in the models are practically at 0%. In many instances, CPLEX could remove all the variables and constraints for the final DDDI model, whereas the presolved FD model often remained large. (There were a few instances for which the FD model also had all variables and constraints removed in presolve; these instances were removed from the data used to produce the corresponding column in Figure 13, to avoid division by zero.) This shows that DDDI does exceptionally well for instances with low cost ratios.

To see the effect of the extra IP tightening constraints, (5), we performed the above calculations without them, using the adaptive algorithm variant. The comparative results can be seen in Table 5. Clearly, these constraints have a significant effect on convergence. When using these constraints, the average gap is nearly halved, and around 10% additional HC instances can be solved to optimality. Although the total number of constraints in the IP is roughly doubled by the addition of the constraints, this number relative to the FD is still very small.

## 7. Comparison with DDD

The approach of Boland et al. (2017), which we refer to as DDD, and that of DDDI have similarities. Both algorithms iteratively build and refine a time-expanded

Figure 13. (Color online) DDDI Model Size (One-Hour Solve Time Limit)



**Table 5.** (Color online) Results With/Without Cuts (One-Hour Solve Time Limit)

Algorithm	Gap %	Time (s)	Constraints %	# Iterations	% Optimal
<b>HC/LF</b>					
Adaptive (w/o cuts)	0.21	1,209.76	1.35	14.1	73.2
Adaptive	0.12	677.76	3.34	14.8	85.8
<b>HC/HF</b>					
Adaptive (w/o cuts)	1.61	2,005.20	0.50	15.7	46.9
Adaptive	0.84	1,693.76	1.38	17.5	56.5
<b>LC/LF</b>					
Adaptive (w/o cuts)	0.00	0.84	1.15	6.8	100.0
Adaptive	0.00	0.59	2.39	6.5	100.0
<b>LC/HF</b>					
Adaptive (w/o cuts)	0.00	0.14	0.11	3.3	100.0
Adaptive	0.00	0.13	0.21	3.2	100.0

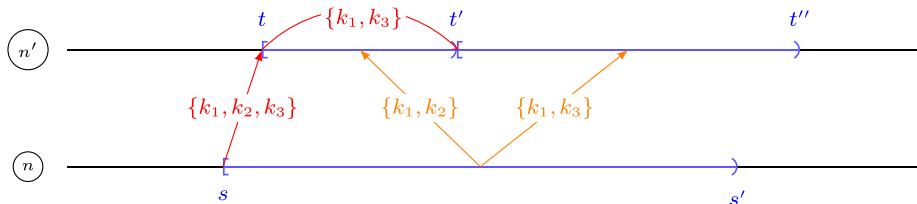
network, and both solve an associated integer program to obtain a lower bound. Both relax transit time in order to guarantee the lower bound. However, each algorithm recovers true transit times in quite different ways, via markedly different discretization refinement processes. For the following discussion, unless otherwise specified, we adopt the notation from DDDI. When referring to properties or propositions of Boland et al. (2017), we follow the reference by “(DDD).” The major differences between DDD and DDDI are summarized below:

- During preprocessing, DDD only removes (does not create) variable  $x_a^k$  if it is impossible for commodity  $k$  to be dispatched on arc  $(n, n')$  at any time without violating its early or late times, whereas DDDI removes it (does not create it) if  $k$  cannot travel on  $(n, n')$  so as to be dispatched from  $n$  at some time in the interval  $[s, s')$  and be dispatched from  $n'$  at some time in the interval  $[t, t')$ , where  $a = ((n, s, s'), (n', t, t'))$ .
- For each node-time pair in the discretization  $(n, t)$ , each arc  $(n, n')$  in the flat-network, and each commodity  $k$ , DDD will create at most one commodity flow variable. By contrast, DDDI may create multiple such variables, to model arrival within different time intervals. For example, DDD creates  $x_{((n, s), (n', t))}^k$  if there is some time at which  $k$  can depart  $n$  in the interval  $[s, s')$  and arrive at  $n'$  in the interval  $[t, t')$ . It may also be possible that there is some time at

which  $k$  can depart  $n$  in the interval  $[s, s')$  and arrive at  $n'$  in the interval  $[t', t'')$ , but this is captured by the holding variable  $x_{((n', t), (n', t'))}^k$ . DDDI, on the other hand, will create both  $x_{((n, s, s'), (n', t, t'))}^k$  and  $x_{((n, s, s'), (n', t', t''))}^k$ .

- Because of the previous point, DDDI may create more commodity flow and vehicle variables than DDD, even when taking into account its stronger preprocessing (first point) and the removal of redundant arcs (as described in Section 5.4). However, the additional arcs (variables) can serve to produce a tighter lower bound. To illustrate this, consider three commodities  $k_1$ ,  $k_2$ , and  $k_3$ , and consider timed points  $(n, s)$ ,  $(n, s')$ ,  $(n', t)$ ,  $(n', t')$  and  $(n', t'')$ , with  $(n, n')$  an arc in the flat-network that may be used by any of these three commodities and with  $t \leq s + \tau_{(n, n')} < t'$  and  $s \leq t'' - \tau_{(n, n')} < s'$ . Then DDD will have an IP model with variables  $x_{((n, s), (n', t))}^k$  for each of  $k = k_1, k_2, k_3$  and, if all of these are set to one, will allow all three commodities to consolidate on the timed-arc when calculating the fixed cost. However, in the case in which timed-arc  $((n, s, s'), (n', t, t'))$  cannot be used by  $k_3$ , because the earliest this commodity can arrive at  $n'$  via  $(n, n')$  is at or after  $t'$ , and the timed-arc  $((n, s, s'), (n', t', t''))$  cannot be used by  $k_2$ , because the latest this commodity can depart  $n'$  is before  $t'$  (and both timed-arcs can be used by  $k_1$ ), then the DDDI model forces a choice between consolidating  $k_1$  with  $k_2$  for travel on the earlier timed-arc and consolidating  $k_1$  with  $k_3$  for

**Figure 14.** (Color online) Modeling Difference Between DDD and DDDI





**Table 6.** (Color online) Computational Results DDD vs. DDDI (One-Hour Solve Time Limit)

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

travel on the later timed-arc; all three cannot be consolidated on  $(n, n')$  if dispatched in the interval  $[s, s')$  (see Figure 14).

- DDDI does not add transit-time knapsack constraints  $\sum_a \tau_a x_a^k \leq l^k - e^k$  to the IP model. Although these constraints ensure that all commodity paths used in the lower bound IP solution are feasible with respect to transit time (and are necessary for the refinement step in DDD), they were found, in our experience, to lead to noticeably slower IP solution times. DDDI, instead, adds strengthening constraints  $\lceil q^k / u^a \rceil x_a^k \leq z_a$  to the IP model (which are not used in DDD).

- The convergence of DDD relies on the fact that there are only finitely many arc-length extensions, whereas the convergence DDDI relies on the fact that there are only finitely many flat-solutions. Consequently, it is possible for DDD to return the same flat-solution for multiple iterations, unlike DDDI.

- DDD and DDDI use the same upper bound LP, except that DDDI weights terms in the objective by the fixed cost of their associated arc, whereas DDD does not. These weights favor preservation of consolidations on arcs with higher fixed cost, which often yields a better upper bound.

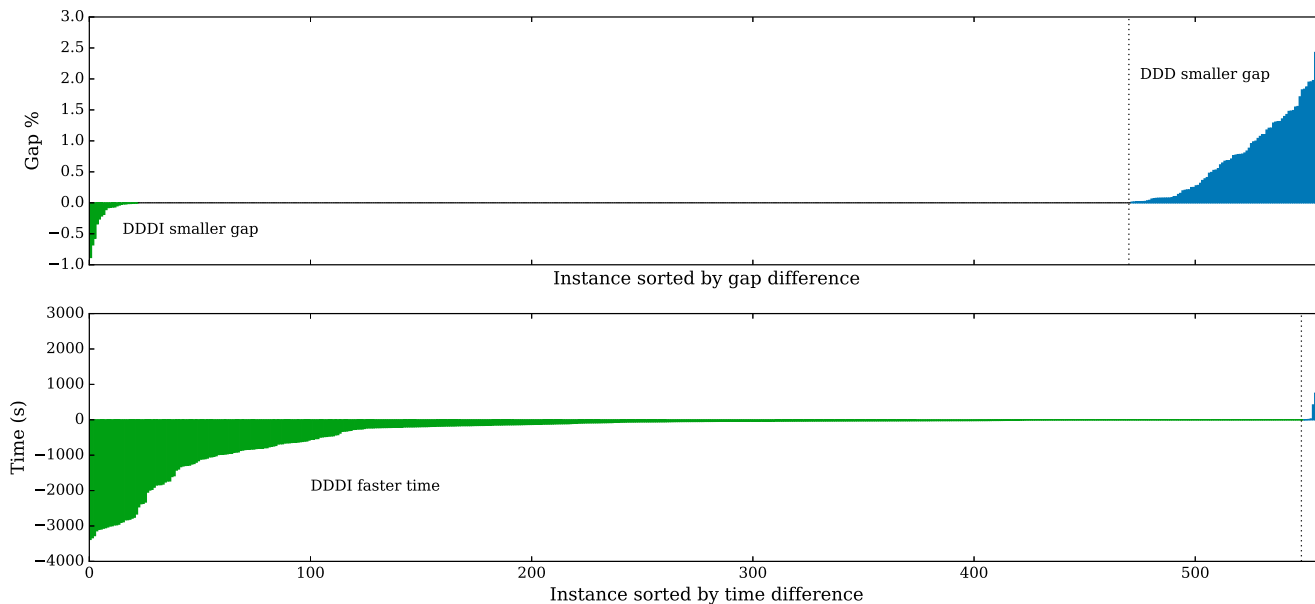
- DDD solves a MIP to suggest how to refine the discretization: it keeps the flat-paths used in the lower bound solution and decides dispatch times for each commodity on each arc in its flat-path so that the resulting timed-path is feasible for the commodity, forces all consolidating commodities from the lower bound solution to be dispatched at the same time, but permits some dispatch-arcs to have less than their true travel time. The number of such dispatch-arcs is minimized by the MIP; any remaining in its optimal solution suggest when and where the discretization can be refined. DDDI exploits the structure of the solution graph to efficiently identify refinements (involving mostly shortest path calculations on an acyclic graph).

## 7.1. Empirical Results

Each instance was solved by DDD, using the same computing platform and limits, except for the optimality tolerance, which was set to 1% for these experiments. A summary of the results is shown alongside those from the adaptive variant of DDDI (also solved to within 1%) in Table 6. Again, the average over the instance groups defined by the flexibility and cost ratio are reported. The model size information (variables, constraints) reported is that of the lower bound IP at the final iteration. As expected, the HC instances are more difficult to solve, and the HC/HF group is the most difficult. DDDI solves more instances to optimality, with a much smaller model size: the final model of DDD has, on average, more than 12 times the number of variables and over five times the number of constraints compared to that of DDDI. That said, on average, DDD has a lower final gap, meaning that on the instances that did *not* solve to optimality, DDD produced a notably smaller gap. To visualize this aspect of the results, in Figure 15, the difference between DDD and DDDI in both gap percent and time to solve (limited to one hour) are plotted for all of the 558 instances. Negative values on the left are instances where DDDI outperformed DDD. For example, a time of  $-2,000$  is interpreted as DDDI solving 2,000 seconds quicker than DDD. Positive values on the right indicate instances where DDD outperforms DDDI. This figure shows clearly that, for instances not solved to optimality, DDD tends to yield smaller gaps within the time limit than DDDI does, with the DDD gap percent up to three percentage points lower than that of DDDI. However, DDDI tends to solve more instances in much less computational time.

To investigate the source of smaller DDD gaps on instances not solved to optimality, we obtained continuous-time optimal values by allowing the DDDI algorithm to run for longer on the instances not solved

**Figure 15.** (Color online) DDD vs. DDDI Gap and Solve Time (One-Hour Solve Time)

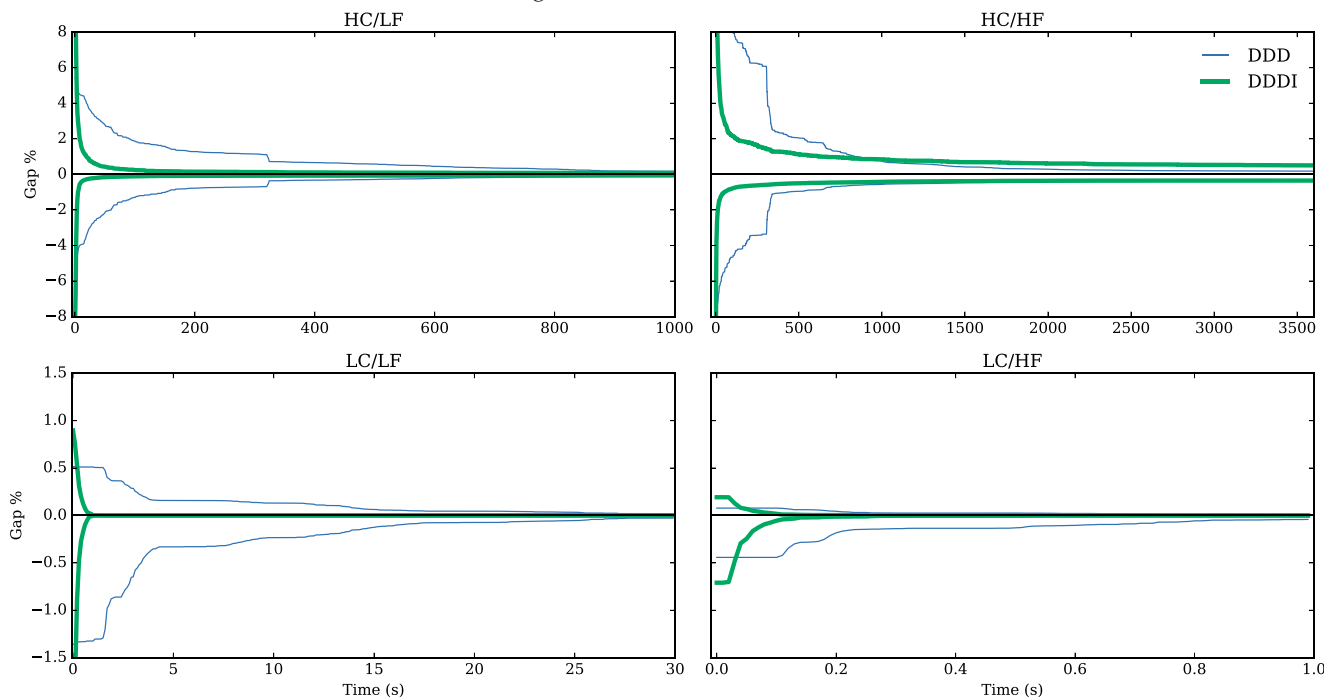


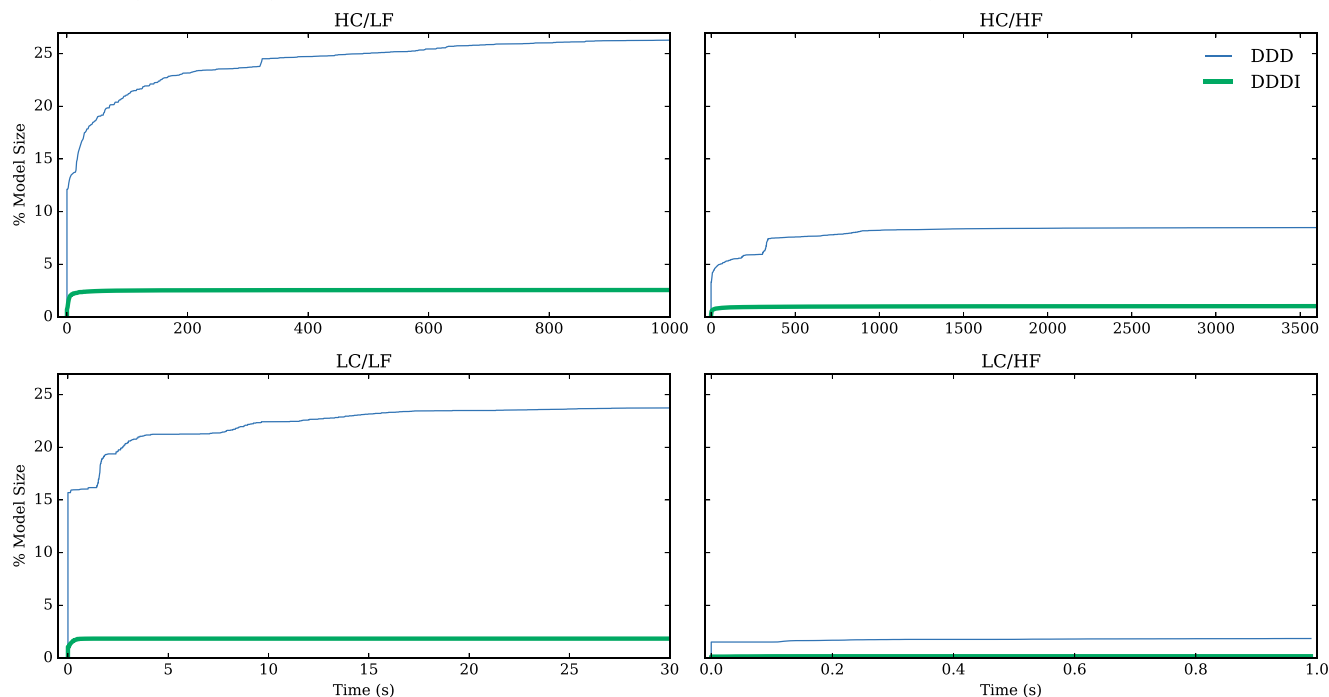
within the one-hour run time limit. Comparing the upper and lower bounds produced by DDDI and DDD to optimal values, we found that DDD usually produced better upper bounds, whereas DDDI produced better lower bounds, but these were not better to a degree sufficient to compensate for DDD's better upper bounds.

The difference in the gaps on both the lower and upper bound sides are shown in Figure 16, which gives convergence profiles over time. DDDI converges rapidly for all groups, but in the HC/HF

instances it struggles to improve the solution after a certain point, whereas DDD (on average) continues to find better solutions. In Figure 17 the growth, over time, of the lower bound IP model size (the number of variables plus the number of constraints) is shown as a percentage of the size of the FD model. It is clear that DDDI produces significantly smaller model sizes than DDD does, and the size of the model in DDDI appears to stabilize relatively quickly. Both DDDI and DDD generate noticeably smaller models than the FD does on

**Figure 16.** (Color online) DDD vs. DDDI Convergence (One-Hour Solve Time Limit)



**Figure 17.** (Color online) DDD vs. DDDI Model Growth (One-Hour Solve Time Limit)

the HF group of instances compared with the LF group of instances. The effect is particularly marked in the case of DDD. In the case of the HC/HF group of instances, it appears that the careful discretization refinement approach of DDDI is paying off in terms of model size, but not in terms of finding upper bounds to close the final gap; the refinement strategy of DDD is more effective in this respect for this group of instances.

Table 7 shows results for DDD and DDDI on five instances that were created using data from a large LTL carrier in the United States, restricted to the states listed. These were the most challenging instances presented in Boland et al. (2017). Note that computation was limited to two hours. The results clearly show that DDDI outperforms DDD and the FD for these instances. Not only does DDDI find high-quality solutions quickly, it is also able to prove that these solutions are of high quality quickly. Even though DDD is also able to find high-quality solutions, it is unable to prove, in two hours of computing time, that these

solutions are within 1% of the optimal ones for four of the five instances.

## 8. Conclusions and Future Work

We have presented an effective and efficient dynamic discretization discovery algorithm for solving the continuous-time service network design problem. The algorithm can handle larger instances and can generate high-quality solutions more quickly than the current state-of-the-art approach of Boland et al. (2017).

The algorithm can be easily modified to support extensions to CTSNDP, for example, allowing freight splitting and in-tree loading. Freight splitting allows for increased utilization by breaking commodities into smaller pieces (it can also be used as a modeling technique to reduce the problem size—by aggregating commodities with common origin/destination). In-tree loading simplifies operational processes at a terminal, by requiring freight with a common ultimate destination cross-docked at a terminal to travel

**Table 7.** Performance on Instances Derived from U.S. LTL Carrier

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

along the same path; in this way terminal operators need only look at the ultimate destination in order to decide which next destination a shipment is loaded to.

Other extensions appear to be more challenging, in particular, how to accommodate holding costs, that is, when it is no longer free to hold freight at a terminal to consolidate it on a later dispatch, and terminal capacities, that is, when the number of dispatches that can occur per hour is limited. These extensions are of interest for future research.

### Endnote

<sup>1</sup>The optimal value of each instance is obtained by running DDDI for a very long time. For the few instances (40 of the HC/HF instances and 12 of the HC/LF instances) that could not be solved to optimality, the best upper bound found was used in place of *optimal\_value*. Of the 62 instances for which this was done, the best upper bound was never more than 1.00% away from optimal and was, on average, 0.66% away; the average final DDDI gap for these long run times was 0.06% for HC/LF instances and 0.17% for HC/HF instances.

### References

- Boland N, Hewitt M, Marshall L, Savelsbergh M (2017) The continuous-time service network design problem. *Oper. Res.* 65(5):1303–1321.
- Boland N, Hewitt M, Marshall L, Savelsbergh M (2019) The price of discretizing time: A study in service network design. *Eur. J. Transportation Logist.* 8(2):195–216.
- Crainic T (2000) Service network design in freight transportation. *Eur. J. Oper. Res.* 122(2):272–288.
- Dash S, Günlük O, Lodi A, Tramontani A (2012) A time bucket formulation for the traveling salesman problem with time windows. *INFORMS J. Comput.* 24(1):132–147.
- Hosseiniinasab A (2015) The continuous time service network design problem. Unpublished doctoral dissertation, University of Waterloo, Waterloo, ON, Canada.
- Magnanti TL, Wong RT (1984) Network design and transportation planning: Models and algorithms. *Transportation Sci.* 18(1):1–55.
- Wang X, Regan AC (2002) Local truckload pickup and delivery with hard time window constraints. *Transportation Res. Part B: Methodological* 36(2):97–112.
- Wang X, Regan AC (2009) On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Comput. Indust. Engrg.* 56(1):161–164.
- Wieberneit N (2008) Service network design for freight transportation: A review. *OR Spectrum* 30(1):77–112.