European Journal of Operational Research 302 (2022) 831–846

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solving the time dependent minimum tour duration and delivery man problems with dynamic discretization discovery

Duc Minh Vu[a,*], Mike Hewitt[b], Duc D. Vu[c]

[a] ORLab, Faculty of Computer Science, PHENIKAA University, Yen Nghia, Ha Dong, Hanoi 12116, Viet Nam
[b] Quinlan School of Business, Loyola University Chicago, United States
[c] The University of Texas at Dallas, United States

ARTICLE INFO

Article history:
Received 10 March 2021
Accepted 17 January 2022
Available online 16 February 2022

Keywords:
Traveling salesman problem
Time dependent travel times
Duration
Delivery man
Dynamic discretization discovery

ABSTRACT

In this paper, we present exact methods for solving the Time Dependent Minimum Tour Duration Problem (TD-MTDP) and the Time Dependent Delivery Man Problem (TD-DMP). Both methods are based on a *Dynamic Discretization Discovery* (DDD) approach for solving the Time Dependent Traveling Salesman Problem with Time Windows (TD-TSPTW). Unlike the TD-TSPTW, these problems involve objective functions that depend in part on the time at which the vehicle departs the depot. As such, optimizing these problems adds a scheduling dimension to the problem. We present multiple enhancements to the DDD method, including enabling it to dynamically determine which waiting opportunities at the depot to model. With an extensive computational study we demonstrate that the resulting methods outperform all known methods for both the TD-MTDP and TD-DMP on instances taken from the literature.

1. Introduction

Building and effectively solving optimization models of routing problems that explicitly recognize time dependent travel times presents multiple challenges. One challenge is to mathematically represent the dependance of travel times on departure times in a way that is amenable to solution techniques. One widely-accepted representation is as a piecewise-linear function (Ichoua, Gendreau, & Potvin, 2003), with pieces corresponding to periods of time during which the travel time is constant. With this approach, the number of pieces in such a function depends on variability in travel times. When travel times are very variable, embedding the resulting representation in a mixed integer program often leaves an optimization problem that is computationally difficult to solve (Montero, Méndez-Dıaz, & Miranda-Bront, 2017).

Vu, Hewitt, Boland, & Savelsbergh (2020) proposes a different mechanism for representing such dependencies. Namely, to model vehicle movements on a time-expanded network and thus embed the relationship between travel times and departure times in the definition of the network itself. They study this idea in the context of solving the Time Dependent Traveling Salesman Problem with Time Windows (TD-TSPTW) under a *Makespan* objective. With this objective, the time at which the vehicle returns to the depot is minimized. Vu et al. (2020) proposes an algorithm based on *Dynamic Discretization Discovery* (DDD) (Boland, Hewitt, Marshall, & Savelsbergh, 2017a) and computational results indicate that algorithm is currently the most effective method for solving that problem.

Another challenge associated with time dependent travel times is that they can introduce a scheduling dimension to a routing problem. While routing problems with time windows involve scheduling when a vehicle departs a location, it is usually done from a feasibility perspective. As observed in Sun, Veelenturf, Hewitt, & Van Woensel (2018), when travel times are time dependent, there are objective functions wherein optimizing when the vehicle departs its initial location, often called a *depot*, can improve the value of the objective function. In this paper, we focus on this challenge in the context of a TD-TSPTW problem, but with objective functions that require the departure time from the depot to be optimized. Specifically, we seek to solve the Time Dependent Minimum Tour Duration Problem and the Time Dependent Delivery Man Problem.

At the heart of the Time Dependent Minimum Tour Duration Problem (TD-MTDP) is the Traveling Salesman Problem (TSP). In the TSP, a single vehicle departs from a depot, visits each location in a given set exactly one time, and then returns to the depot. Such a sequence of moves is often referred to as a *tour*. In most applications of the TSP there is a cost associated with transportation between each pair of locations and the optimization problem seeks to find the tour with the minimum total transportation cost.

[*] Corresponding author.
E-mail address: minh.vuduc@phenikaa-uni.edu.vn (D.M. Vu).

https://doi.org/10.1016/j.ejor.2022.01.029
0377-2217/© 2022 Elsevier B.V. All rights reserved.

Closer to the problem we study is the Traveling Salesman Problem with Time Windows (TSPTW). In this problem, not only must each location be visited exactly one time but there is also a time window during which the vehicle must arrive at that location. If the vehicle arrives at a location before the time window begins, the vehicle must wait. In this problem, both a travel cost and travel time are associated with transportation between each pair of locations, although they are sometimes the same. However, in the TSPTW, these travel times are assumed to be constant. Specifically, they do not depend on departure times.

The Time Dependent Traveling Salesman Problem with Time Windows (TD-TSPTW) relaxes this assumption. Specifically, the travel time between two locations does depend on the time at which travel begins. However, it is customary to assume, particularly in transportation settings, that travel times satisfy the First-In-First-Out (FIFO) property. Namely, that for any pair of locations and any two different times, departing from the first location at the later time can not lead to an earlier arrival at the second location than departing at the earlier time. We make this assumption as well. For the Makespan objective, the FIFO property implies there is an optimal solution wherein the vehicle does not wait at any location unless it is for a time window to open.

While closely related, the same statement can not be made when minimizing the duration of time the vehicle is away from the depot (the *Duration* objective). One of the optimization problems we study in this paper, the Time Dependent Minimum Tour Duration Problem (TD-MTDP), minimizes this objective. A weaker statement can be made regarding the TD-MTDP. Namely, that all optimal solutions may require the vehicle to wait at the depot, but that at least one optimal solution does not involve waiting at any other location unless it is for a time window to open. The same characterization of optimal solutions holds regarding the other problem we consider, the Time Dependant Delivery Man Problem (TD-DMP). This problem considers the amount of time a customer waits between when the vehicle departs the depot and arrives at their location and seeks to design a TSP tour that minimizes the sum of these waiting times. Thus, both problems we consider in this paper require optimizing the time the vehicle departs from the depot. We note that while both problems we study consider time windows at locations, we do not indicate as such in their names to be consistent with the literature.

Both problems are particularly relevant to urban distribution systems, wherein travel times are known to fluctuate due to changing traffic conditions. For a distribution company, minimizing the duration of each vehicle route increases the capacity of its fleet. Of course, distribution companies often have multiple vehicles. However, as noted in Archetti & Bertazzi (2021), services such as Amazon Flex (https://flex.amazon.com) enable the *uberization* of delivery operations, wherein individuals make their own time and vehicle available for delivering goods. From the individual's perspective, minimizing the duration of each tour enables them to deliver more goods and earn more revenues. While the duration objective is appropriate when optimizing from the transportation carrier's perspective, the Delivery Man problem considers an objective that focuses on customer perception of delivery service quality. As its name suggests, this problem is particularly relevant to the burgeoning restaurant delivery industry consisting of companies such as DoorDash (www.doordash.com) or UberEats (https://www.ubereats.com). While relevant to current and (likely) future trends in delivery services, both problems considered in this paper have received little attention in the academic literature.

Vu et al. (2020) propose the only known algorithm to date for the TD-MTDP, an adaptation of an algorithm they propose for solving the TD-TSPTW. Both algorithms employ a strategy wherein each iteration is executed by solving an instance of a TD-MTDP (or TD-TSPTW) formulated on a *partially time-expanded network*, a time-expanded network wherein not every location is represented at every point in time. The computational results reported (Vu et al., 2020) indicate that the proposed algorithm for the TD-TSPTW is one of the best-performing methods to date, computationally-speaking. Because all optimal solutions to the TD-MTDP may require the vehicle to wait at the depot before departure, the adaptation of the algorithm for the TD-TSPTW to that problem involves starting the algorithm with a partially time-expanded network that contains a node for the depot at each potential departure time. Their results show that the performance of the DDD-based algorithm degrades when solving the TD-MTDP in comparison to when solving the TD-TSPTW, and that this degradation can be partially attributed to the larger partially time-expanded networks that result from this enumeration of potential departure times. We are unaware of any proposed methods for solving the TD-DMP.

To avoid the observed performance degradation when solving instances of the TD-MTDP or TD-DMP, we present multiple enhancements to the DDD-based algorithm of Vu et al. (2020) that enable it to dynamically determine what departure times at the depot to represent in the partially time-expanded network used at an iteration. These enhancements include a modified relaxation that is solved at each iteration and guarantees that the DDD-based algorithm will converge to an optimal solution to the original problem, even though not all departure times at the depot may be represented. In addition, we prove that the modified relaxation we propose is stronger than the relaxation used by Vu et al. (2020). The enhancements also include a new method for refining the partially time expanded network at an iteration that recognizes what missing departure times at the depot should be modeled. Finally, we propose new integer programming-based heuristic techniques for producing high-quality primal solutions. One of these heuristics focuses on optimizing the departure time of a given tour from the depot. The other focuses on repairing solutions to the relaxation from which a solution to the original problem can not be directly derived. The enhancements we propose can be applied, mostly unchanged, when solving either the TD-MTDP or the TD-DMP, and thus yield algorithms for both problems. The respective algorithms for the TD-MTDP and TD-DMP can also be executed, mostly unchanged, for the variants of these problems wherein travel times are static.

We believe this paper makes the following contributions. First, it proposes the first exact algorithms designed for the TD-MTDP and the TD-DMP, two problems relevant to modern urban distribution tactics and trends. Computational results suggest these algorithms are the best-performing methods to date for these two problems. Second, the DDD-based framework underlying these algorithms is extensible to other TSPTW-type problems as it can be executed, *nearly unchanged*, on variants of these problems with static travel times. From a software development perspective, that the DDD-based framework proposed in this paper yields an effective algorithm for four TSPTW-type problems is meaningful. Third, the techniques it proposes for dynamically adding nodes that represent departure times at the depot can be used with objective functions other than the ones considered in this paper. More specifically, those wherein all optimal solutions require the vehicle to wait at one or more locations other than the depot after the time window has opened. The total transportation cost associated with a tour is an example of such an objective. Note with this objective there is no penalty associated with the vehicle waiting at a location. Thus, if the transportation cost between two locations decreases over a fixed period of time, total transportation costs may be reduced by allowing the vehicle to wait at the first location before traveling to the second. Fourth, the proposed repair heuristic enables the overall algorithm to solve some of the largest instances considered in the literature for either problem and can be used (nearly) unchanged by DDD-based algorithms for other TSPTW-type problems, including ones with time-dependent and ones with static travel times.

The rest of this paper is organized as follows. Section 2 reviews the literature relevant to the research we present in this paper. Section 3 presents formal mathematical models of the optimization problems we seek to solve, the TD-MTDP and the TD-DMP. Then, Section 4 reviews at a high level the algorithmic framework we enhance in order to solve those problems. Section 5 then presents the enhancements we propose while Section 6 analyzes their impact through an extensive computational study. Finally, Section 7 provides concluding remarks and proposes avenues for future research.

2. Literature review

There is a vast literature on TSP-type problems. As we propose in this paper exact methods for solving variants of the TD-TSPTW, the TD-MTDP and the TD-DMP, we focus our literature review on papers that present exact methods for TSPTW, MTDP, or DMP-type problems, and in particular those that recognize time-dependent travel times. However, we begin with exact methods for the most-studied variants of these problems wherein travel times are constant.

Dynamic programming (DP) is an algorithmic strategy at the heart of many solution approaches for the TSPTW. Dumas, Desrosiers, Gélinas, & Solomon (1995) presents a DP-based solution approach that is augmented with label-elimination tests that enable it to perform well on instances with tight time windows. Similarly, Christofides, Mingozzi, & Toth (1981) presents a DP-based solution approach to the TSPTW that is based on state-space relaxation. The technique of state-space relaxation is also used in the solution approaches presented in Mingozzi, Bianco, & Ricciardelli (1997) and Baldacci, Mingozzi, & Roberti (2012). Balas & Simonetti (2001) presents a dynamic programming-based approach for a TSPTW with precedence constraints. That approach is based on an extension of the solution method preveiously proposed in Balas (1999). At the time of this writing, the performance of the method proposed in Baldacci et al. (2012) on benchmark instances suggest it is the state-of-the-art method for solving TSPTWs.

DP has also formed the basis of effective solution approaches for the MTDP, which to the best of our knowledge was first studied in Savelsbergh (1992). Goel (2012) proposes a DP-based solution approach for a variant of the MTDP that is inspired by truck driver scheduling. Specifically, they embed in their model representations of many of the regulations that drivers must follow when executing a route. Tilk & Irnich (2017) focus on the classical MTDP (i.e. without driver regulations) and propose a DP-based solution that generalizes the approach of Baldacci et al. (2012). Relatedly, Dabia, Ropke, Van Woensel, & De Kok (2013) propose a dynamic programming-based algorithm for a time-dependent elementary shortest path problem with resource constraints that has an objective that necessitates optimizing the departure time from the depot. They propose solving this problem as a pricing problem in the context of a branch-and-price scheme for a time dependent vehicle routing problem with time windows wherein the total route duration of vehicles is minimized.

Another algorithmic strategy used to solve the TSPTW is integer programming. Ascheuer, Fischetti, & Grötschel (2000, 2001) present some of the first branch-and-cut-based algorithms for this class of problem. Dash, Günlük, Lodi, & Tramontani (2012) also propose a branch-and-cut-based approach, albeit based on a different formulation. Specifically, they present a formulation that is based on partitioning time windows into smaller, sub-time-windows, which the authors refer to as time buckets. These time buckets are produced, and iteratively refined, by solving a series of linear relaxations of the problem. This is done in the context of a pre-processing scheme, after which a “final” formulation is input to the branch-and-cut algorithm. Boland, Hewitt, Vu, & Savelsbergh (2017b) also propose an integer programming-based approach for the TSPTW. However, their approach is based on formulating the problem on a time-expanded network and then solving that formulation with the *Dynamic Discretization Discovery* (DDD) procedure proposed in Boland et al. (2017a) for the *Service Network Design* problem. DDD can be viewed as an algorithmic strategy that iteratively solves an aggregated formulation of an optimization problem that is constructed in such a way as to produce a relaxation of the original problem. When solutions to that aggregation are not feasible for the original problem, a disaggregation step is performed, and the process repeats until a provably optimal solution is found. DDD has mostly been applied to variants of the *Scheduled Service Network Design* problem (Hewitt, 2019; Hewitt, Boland, Savelsbergh, & Hewitt, 2021; Scherr, Hewitt, Saavedra, & Mattfeld, 2020). However, it has been applied to problems that have a routing component (Boland, Hewitt, Vu, & Savelsbergh, 2017c; Medina, Hewitt, Lehud, & Pton, 2019; Vu et al., 2020). Clautiaux, Hanafi, Macedo, Voge, & Alves (2017) also present an algorithmic framework for network-based optimization problems that relies on iterative aggregation and disaggregation.

Returning to MTDPs, Kara & Derya (2015) present different integer programming formulations of the problem. A problem related to the Delivery Man Problem is the Minimum Latency Problem, which minimizes the sum of the arrival times of the vehicle at customer locations. Heilporn, Cordeau, & Laporte (2010) propose mixed integer programming formulations for the DMP with time windows. Both Salehipour, Sörensen, Goos, & Bräysy (2011) and Silva, Subramanian, Vidal, & Ochi (2012) propose metaheuristics for the latency objective when time windows are not present. Roberti & Mingozzi (2014) also consider a latency objective when time windows are not present. However, they present an exact, dynamic programming-based algorithm for the problem.

The approaches discussed so far have focused on TSP-type problems wherein travel times are constant. However, time-dependent variants of the TSP have begun to receive attention. One of the earliest papers that considers a time-dependent TSP is Picard & Queyranne (1978), which presents an enumeration-based scheme for a problem motivated by single-machine scheduling. Lucena (1990) presents a scheme for deriving tight lower bounds for a time-dependent TSP as well as adapt that scheme to the time-dependent traveling deliveryman problem. Various exact methods, mostly integer programming-based, have also been proposed (Abeledo, Fukasawa, Pessoa, & Uchoa, 2013; Albiach, Sanchis, & Soler, 2008; Bront, Méndez-Díaz, & Zabala, 2014; Gouveia & Voz, 1995; Melgarejo, Laborie, & Solnon, 2015; Méndez-Díaz, Bront, Toth, & Zabala, 2011; Stecco, Cordeau, & Moretti, 2008). One weakness of the early research on time-dependent TSPs is that the mathematical models solved were based on travel time functions that did not observe the FIFO property.

To remedy this issue, Ichoua et al. (2003) presented a piecewise linear model of time-dependent travel times that ensures the FIFO property. In this model, travel time is assumed to be constant within an interval, but can change at interval boundaries. Due in part to its ability to maintain the FIFO property, this model has been included in all subsequent papers that consider time-dependent TSP-type problems. Examples include Cordeau, Ghiani, & Guerriero (2014), which solves a TD-TSP with a *Makespan* objective, and Ghiani & Guerriero (2014) which further analyzes the model of Ichoua et al. (2003). Tas, Gendreau, Jabali, & Laporte (2016) consider a related problem, the TSP with time-dependent service times, and present and evaluate multiple mathematical programming-based solution approaches to the problem.

Turning to TD-TSPTWs, Montero et al. (2017) present an integer programming-based algorithm for the problem based on the Makespan objective. Arigliano, Ghiani, Grieco, Guerriero, & Plana (2018) focus on a TD-TSPTW with asymmetric travel costs. They establish properties wherein solving a time-independent variant of the problem will in fact yield the optimal solution to the time-dependent variant. They also show that when those properties do not hold, solving the time-independent variant can be used as a lower bounding procedure in the context of a branch-and-bound scheme. Vu et al. (2020) adapt the DDD-based approach presented in Boland et al. (2017b) to a TD-TSPTW wherein the *Makespan* objective is optimized. Computational experiments in Vu et al. (2020) on benchmark instances suggest it is the state-of-the-art method for solving the TD-TSPTW under a Makespan objective. Vu et al. (2020) also present an adaptation of the proposed algorithm for the TD-TSPTW to what we believe is the first algorithm for the TD-MTDP. A more general discussion of integer programming models for optimization problems that recognize time dependency can be found in Boland & Savelsbergh (2019).

3. Time-expanded network formulations

In this section, we present optimization models of the two problems we consider, the TD-MTDP and TD-DMP, that are formulated on a time-expanded network. The use of a time-expanded network necessitates a discretization of time. However, the solution method we propose can accommodate arbitrarily precise discretizations, and thus continuous time. While both can be presented in many contexts, our description will be grounded in a transportation and logistics setting. We present a detailed mathematical formulation of the TD-MTDP and then discuss how it can be modified to a formulation of the TD-DMP.

3.1. Formulation of the TD-MTDP

Mathematically, we define the TD-MTDP as follows. We let the node set $N = \{0, 1, 2, \dots, n\}$ denote the physical locations that must be visited, with node 0 representing the depot from which the vehicle departs each day and returns to at the end of its tour. Associated with location $i = 1, \dots, n$ is a time window $[e_i, l_i]$ during which the location must be visited. Similarly, there is a time window $[e_0, l_0]$, associated with the depot which indicates that the vehicle can not depart the depot before e_0 and must return no later than l_0.

The arc set $A \subseteq N \times N$ contains arcs that represent physical travel between locations. We note that TSPs are generally formulated on complete graphs ($A = N \times N$). This is the case in the instances we use in our computational study, but is not a requirement. Recall that we allow travel times to depend on departure times. Thus, associated with arc $(i, j) \in A$ and time, t, at which travel can begin on arc (i, j), is a travel time, $\tau_{ij}(t)$. The FIFO property that we assume holds implies that for each arc $(i, j) \in A$ and times t, t' wherein $t \le t'$, we must have $t + \tau_{ij}(t) \le t' + \tau_{ij}(t')$. A precise definition of a tour is that the vehicle must depart node 0 at time $t_0 \ge e_0$, visit every node $i \in N$ at a time t_i, wherein $e_i \le t_i \le l_i$, and then return to node 0 before time l_0.

One can formulate the TD-MTDP as an integer program defined on a time-expanded network, $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, with node set $\mathcal{N}$ and arc set $\mathcal{A}$. Regarding nodes in this network, for each node $i \in N, t \in [e_i, l_i]$, $\mathcal{N}$ contains the node (i, t). Regarding arcs, the set $\mathcal{A}$ contains arcs that represent the vehicle traveling between locations. These arcs are of the form $((i, t), (j, t'))$ wherein $i \ne j$, $(i, j) \in A$, $t \ge e_i, t' = \max\{e_j, t + \tau_{ij}(t)\}$ (the vehicle can not visit early), and $t' \le l_j$ (the vehicle can not arrive late).

To formulate the integer program, for each arc $a = ((i, t), (j, t')) \in \mathcal{A}$ we define the binary variable x_a to represent whether the vehicle takes that arc. Regarding the objective, as we seek to minimize the time the vehicle is away from the depot, we define the arc costs $c_a, a = ((i, t), (j, t')) \in \mathcal{A}$ as follows:

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

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

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

subject to

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

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

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

Constraints (3) ensure that the vehicle departs each location exactly once during its time window. Constraints (4) then ensure that the vehicle departs every node representing a location other than the depot at which it arrives. Finally, constraints (5) define the decision variables and their domains.

We note that our use of a time-expanded network to formulate this problem presumes that time may be discretized into a finite set of time points, which is theoretically true in some cases, such as when travel times are integer and time windows are bounded and defined by integer values, and practically true in nearly all transportation and logistics applications.

3.2. Formulation of the TD-DMP

Recall that the Delivery Man Problem seeks to design a TSP tour that minimizes the total amount of time each customer has to wait between when the vehicle departs the depot and arrives at their location. Thus, if we let t_k represent the time at which the vehicle arrives at customer $k \in N \setminus \{0\}$, the objective can be written as

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

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

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

The first term in (7) represents the arrival time at customer j. The second term computes the departure time from the depot term, aggregated over all customers. The TD-DMP seeks to minimize this objective subject to constraints (3) - (5).

4. Overview of dynamic discretization discovery of Vu et al. (2020) for solving the TD-TSPTW

The methods we present in this paper for solving the TD-MTDP and TD-DMP follow a similar framework as the approach presented in Vu et al. (2020) for optimizing the TD-TSPTW under the Makespan objective function. Thus, in this section we review the main steps of that method as well as the principles that ensure it will converge to a provably optimal solution.

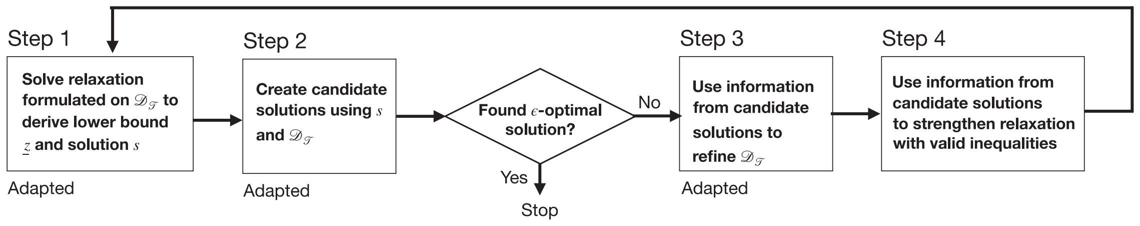

Step 1
Solve relaxation formulated on $\mathcal{D}_\mathcal{T}$ to derive lower bound $\underline{z}$ and solution s
Adapted
Step 2
Create candidate solutions using s and $\mathcal{D}_\mathcal{T}$
Adapted
Found ϵ-optimal solution?
Yes
Stop
No
Step 3
Use information from candidate solutions to refine $\mathcal{D}_\mathcal{T}$
Adapted
Step 4
Use information from candidate solutions to strengthen relaxation with valid inequalities

Fig. 1. Iteration of DDD-based algorithm.

A formulation similar to the one presented in Section 3.1 can be used for the TD-TSPTW under the Makespan objective. Namely, the same network, $\mathcal{D}$, is used, but arc costs are instead

$$c_{a=((i,t),(j,t'))} = \begin{Bmatrix} t+\tau_{ij}(t), & \text{for } j=0 \\ 0, & \text{otherwise} \end{Bmatrix} \quad \forall a \in \mathcal{A}. \tag{8}$$

We refer to the resulting formulation as the TD-TSPTW($\mathcal{D}$). One of the main computational challenges encountered when solving such a TD-TSPTW formulation is that the underlying time-expanded network, $\mathcal{D}$, may be very large, which will in turn yield an integer program that is too large to be solved in a reasonable runtime. Dynamic Discretization Discovery (DDD) mitigates this issue by instead solving instances of a TD-TSPTW formulated on a *partially time-expanded network*, wherein not every location is represented at every point in time.

That DDD is guaranteed to find a provably optimal solution is based in part on constructing a *partially time-expanded network* such that a TD-TSPTW formulated on that network is a relaxation of the original problem. DDD is an iterative procedure, wherein each iteration begins by solving a partially time-expanded network-induced relaxation. The solution to that relaxation is then examined to determine whether it can be converted to a provably optimal solution to the original problem. If it can be, the algorithm terminates. Otherwise, the partially time-expanded network is refined and the algorithm continues. As we will next discuss, by maintaining certain properties, DDD is guaranteed to terminate with a provably optimal solution. We illustrate in Fig. 1 the steps taken at an iteration, also indicating those that are adapted to solve the problem considered in this paper. We next discuss how Steps 1,2, and 3 are performed in greater detail. As Step 4 is not adapted, nor is it necessary to guarantee that DDD will converge to an optimal solution, we refer the reader to Vu et al. (2020) for details regarding how it is performed.

4.1. Step 1: formulating a relaxation with a partially time-expanded network

Formally, a *partially time-expanded network*, $\mathcal{D}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T})$, is based on a node set, $\mathcal{N}_\mathcal{T}$, that is a subset of the nodes $\mathcal{N}$ defining $\mathcal{D}$ (i.e. $\mathcal{N}_\mathcal{T} \subseteq \mathcal{N}$). The arc set $\mathcal{A}_\mathcal{T}$ contains arcs of the form $((i,t),(j,t'))$, wherein $(i,t) \in \mathcal{N}_\mathcal{T}$, $(j,t') \in \mathcal{N}_\mathcal{T}$, $i \neq j$, and $(i,j) \in A$. Such arcs in $\mathcal{A}_\mathcal{T}$ model travel between locations, albeit never with travel times that are "too long." Namely, $\mathcal{A}_\mathcal{T}$ is constructed with arcs $((i,t),(j,t'))$ such that $t' \le \max\{e_j, t+\tau_{ij}(t)\}$ when $i \neq j$. We label an arc as *too short* if $t' < \max\{e_j, t+\tau_{ij}(t)\}$. Similar to how arcs $((i,t),(j,t'))$, $i \neq j$ can underestimate actual travel times $\tau_{ij}(t)$, associated with arcs in $\mathcal{A}_\mathcal{T}$ are costs, $\underline{c}_a$ that underestimate the actual travel costs, c_a. Specifically, $\underline{c}_{a=((i,t),(j,t'))}$ is set to $\min\{c_{((i,h)(j,t'))} | t \le h \le l_i \text{ and } h+\tau_{ij}(h) \le l_j\}$.

The TD-TSPTW($\mathcal{D}_\mathcal{T}$) is defined to be the TD-TSPTW formulated on the network $\mathcal{D}_\mathcal{T}$ and optimized with respect to the costs $\underline{c}_a$. We next state properties of $\mathcal{D}_\mathcal{T}$ that together ensure that TD-TSPTW($\mathcal{D}_\mathcal{T}$) is a relaxation of TD-TSPTW($\mathcal{D}$).

Property 1. *$\forall i \in N$, both nodes (i,e_i) and (i,l_i) are in $\mathcal{N}_\mathcal{T}$.*

Property 2. *$\forall (i,t) \in \mathcal{N}_\mathcal{T}, e_i \le t \le l_i$.*

Property 3. *$\forall (i,t) \in \mathcal{N}_\mathcal{T}$ and arc $(i,j) \in A$, there is an arc of the form $((i,t)(j,t')) \in \mathcal{A}_\mathcal{T}$ if $t+\tau_{ij}(t) \le l_j$. In addition, every arc $((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}$ must have either (1) $t+\tau_{ij}(t) < e_j$ and $t' = e_j$, or (2) $e_j \le t' \le t+\tau_{ij}(t)$. Finally, there is no $(j,t'') \in \mathcal{N}_\mathcal{T}$ with $t' < t'' \le t+\tau_{ij}(t)$.*

Property 4. *$\forall a = ((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}$, $\underline{c}_a = \min\{c_{((i,h),(j,t'))} | t \le h \le l_i$ and $h+\tau_{ij}(h) \le l_j\}$.*

Vu et al. (2020) show that Properties 1–3 together ensure that any path $p = ((u_0,t_0),(u_1,t_1),\dots,(u_m,t_m))$, $(u_i,t_i) \in \mathcal{N}$ $i = 0,\dots,m$ can be mapped to a path $p' = ((u_0,t'_0),(u_1,t'_1),\dots,(u_m,t'_m))$, $(u_i,t'_i) \in \mathcal{N}_\mathcal{T}$ where $t'_k \le t_k$ for all $0 \le k \le m$. Note such a path p need not begin and end at the depot. That such a mapping exists is critical for showing that TD-TSPTW($\mathcal{D}_\mathcal{T}$) is a relaxation of TD-TSPTW($\mathcal{D}$).

Even though there may be many nodes in $\mathcal{D}$ that are not present in $\mathcal{D}_\mathcal{T}$, any feasible solution s to TD-TSPTW($\mathcal{D}$) may be mapped to a feasible solution s' to TD-TSPTW($\mathcal{D}_\mathcal{T}$) in which departures occur at earlier times. Given the definition of the arc costs $\underline{c}_a$ the cost of the solution s' with respect to the costs $\underline{c}_a$ can not exceed the cost of the solution s with respect to the costs c_a. In short, Vu et al. (2020) show that every solution to TD-TSPTW($\mathcal{D}$) can be mapped to a solution to TD-TSPTW($\mathcal{D}_\mathcal{T}$) of lesser or equal cost. Thus, TD-TSPTW($\mathcal{D}_\mathcal{T}$) is the relaxation of TD-TSPTW($\mathcal{D}$) that DDD solves at each iteration. Vu et al. (2020) also show that as TD-TSPTW($\mathcal{D}_\mathcal{T}$) is a relaxation induced by arcs that are "too short", if its optimal solution at an iteration consists only of arcs that model the correct length (i.e. they are of the form $((i,t),(j,t+\tau_{ij}(t)))$ then that solution is optimal for TD-TSPTW($\mathcal{D}$).

We note that the DDD-based solution approach presented in Vu et al. (2020) begins with various pre-processing procedures. It then creates the initial partially time-expanded network $\mathcal{D}_\mathcal{T}$ that has the fewest possible nodes while still satisfying Properties 1–3. Namely, it sets $\mathcal{N}_\mathcal{T} = [\cup_{i \in N}\{(i,e_i)\}] \cup [\cup_{i \in N}\{(i,l_i)\}]$. The arc set $\mathcal{A}_\mathcal{T}$ is then constructed to satisfy Property 3.

4.2. Step 2: creating candidate solutions

There are three procedures DDD employs to try and create candidate solutions to the TD-TSPTW($\mathcal{D}$) from a solution s to the relaxation TD-TSPTW($\mathcal{D}_\mathcal{T}$) solved at an iteration. First, the solution s is examined to see whether it can be used to derive a solution to the TD-TSPTW($\mathcal{D}$). For example, if s consists solely of arcs of the form $((i,t),(j,t+\tau_{ij}(t))$, then it is also a feasible (and in fact optimal) solution to TD-TSPTW($\mathcal{D}$). Even if s does contain arcs that are "too short," the sequence, $(u_0 = 0, u_1, \dots, u_n = 0)$, in which s visits locations may be the basis of a feasible solution to the TD-TSPTW($\mathcal{D}$). To determine this, DDD tries to associate departure times, $t_i, i = 0,\dots,n-1$, associated with each location that satisfy the following properties:

- $t_0 = e_0$: the vehicle departs from the depot at the beginning of its time window.

- $t_i + \tau_{u_i u_{i+1}}(t_i) \leq t_{i+1}$: the departure times agree with travel times.
- $e_i \leq t_i \leq l_i, \ldots, i = 0, \ldots, n-1$: the departure times occur during each locations's time window.
- $t_{u_{n-1} u_n} + \tau_{u_{n-1} u_n}(t_{u_{n-1}}) \leq l_n$: the vehicle arrives at the depot before the end of its time window.

Such a set of departure times, along with the sequence $(u_0 = 0, u_1, \ldots, u_n = 0)$, are then used to generate a solution to the TD-TSPTW($\mathcal{D}$).

The second and third procedures are both integer programming-based and involve generating a partially time-expanded network $\bar{\mathcal{D}_\mathcal{T}}^i = (\mathcal{N}_\mathcal{T}, \bar{\mathcal{A}_\mathcal{T}}^i), i = 1, 2,$ such that solving TD-TSPTW($\bar{\mathcal{D}_\mathcal{T}}^i$) is likely to yield a feasible solution to TD-TSPTW($\mathcal{D}$). $\bar{\mathcal{D}_\mathcal{T}}^1$ and $\bar{\mathcal{D}_\mathcal{T}}^2$ share the node set $\mathcal{N}_\mathcal{T}$ but differ in their arc sets. $\mathcal{A}_\mathcal{T}^1$ is constructed to contain arcs of the form $((i,t),(j,t')),\ (i,j) \in A, i \neq j, (i,t) \in \mathcal{N}_\mathcal{T}, (j,t') \in \mathcal{N}_\mathcal{T}$ such that $t' \geq t + \tau_{ij}(t)$. When there are multiple nodes (j,t') such that $t' \geq t + \tau_{ij}(t)$, the one that minimizes t' is chosen. Thus, any feasible solution to TD-TSPTW($\bar{\mathcal{D}_\mathcal{T}}^1$) is feasible for TD-TSPTW($\mathcal{D}$). $\mathcal{A}_\mathcal{T}^2$ is less restrictive; it is constructed to contain arcs $((i,t),(j,t')),\ (i,j) \in A, i \neq j, (i,t) \in \mathcal{N}_\mathcal{T}, (j,t') \in \mathcal{N}_\mathcal{T}$ such that $t' > t$. Like before, when there are multiple nodes (j,t') such that $t' > t$, the one that minimizes t' is chosen. As observed in Vu et al. (2020), the presence of arcs $((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}$ such that $t' \leq t$ allows for cycles to exist in $\mathcal{D}_\mathcal{T}$ and solutions to TD-TSPTW($\mathcal{D}_\mathcal{T}$) that consist of subtours and thus cannot be used to construct a solution to TD-TSPTW($\mathcal{D}$). $\mathcal{A}_\mathcal{T}^2$ is constructed to ensure that solutions to TD-TSPTW($\mathcal{D}_\mathcal{T}^2$) do not contain subtours.

4.3. Step 3: refining $\mathcal{D}_\mathcal{T}$

Vu et al. (2020) show that the only reason solving TD-TSPTW($\mathcal{D}_\mathcal{T}$) may not yield a feasible solution to TD-TSPTW($\mathcal{D}$) is the presence of "short arcs," $((i,t),(j,t')), t' < t + \tau_{ij}(t)$, in $\mathcal{A}_\mathcal{T}$. As such, at an iteration DDD will refine $\mathcal{D}_\mathcal{T}$ to "lengthen" such short arcs. Specifically, it will first create the node $(j, t + \tau_{ij}(t))$. It will then delete the arc $((i,t),(j,t'))$ and add the arc $((i,t),(j, t + \tau_{ij}(t)))$. Then, additional arcs are added to $\mathcal{A}_\mathcal{T}$ to ensure the refined $\mathcal{D}_\mathcal{T}$ satisfies Property 3. We note that DDD does not lengthen all short arcs at an iteration and refer the reader to Vu et al. (2020) for a description of how the arcs to lengthen are chosen.

Termination of DDD is guaranteed because $\mathcal{A}$ is of finite cardinality and at least one arc in $\mathcal{A}_\mathcal{T}$ is lengthened at each iteration. Thus, after a finite number of iterations, $\mathcal{A}_\mathcal{T}$ will contain sufficient arcs from $\mathcal{A}$ such that solving TD-TSPTW($\mathcal{D}_\mathcal{T}$) is equivalent to solving TD-TSPTW($\mathcal{D}$). However, in the computational results presented in Vu et al. (2020), the algorithm only requires a small number of iterations to produce a solution that is provably within 1% of optimal.

5. Enhanced dynamic discretization discovery

We propose three types of enhancements to DDD, one for each of the steps described above, to enhance its performance when solving instances of the TD-MTDP or the TD-DMP. We present these enhancements in the context of solving the TD-MTDP. That they all naturally apply when solving the TD-DMP can be easily shown. The first enhancement we present is to strengthen the relaxation DDD solves at an iteration. The second is new schemes for creating candidate solutions that solve restricted integer programs that recognize waiting opportunities. The third is a modification of DDD's refinement procedure that dynamically adds nodes to model waiting at the depot. We next describe each of these enhancements in detail and the order in which they are executed by DDD. We finish the section with a complete presentation of the enhanced algorithm.

5.1. Strengthening the relaxation solved at an iteration

Similar to the TD-TSPTW, we define the TD-MTDP($\mathcal{D}_\mathcal{T}$) as the MTDP model presented in Section 3.1, albeit formulated on the partially time-expanded network, $\mathcal{D}_\mathcal{T}$. When applied in this context, Property 4 yields a TD-MTDP with the following objective function.

$$\sum_{((i,t)(0,t')) \in \mathcal{A}_\mathcal{T} | i \neq 0} (t + \tau_{i0}(t)) x_{((i,t)(0,t'))} - \sum_{((0,t)(i,t')) \in \mathcal{A}_\mathcal{T} | i \neq 0} \bar{t}_0 x_{((0,t)(i,t'))} \tag{9}$$

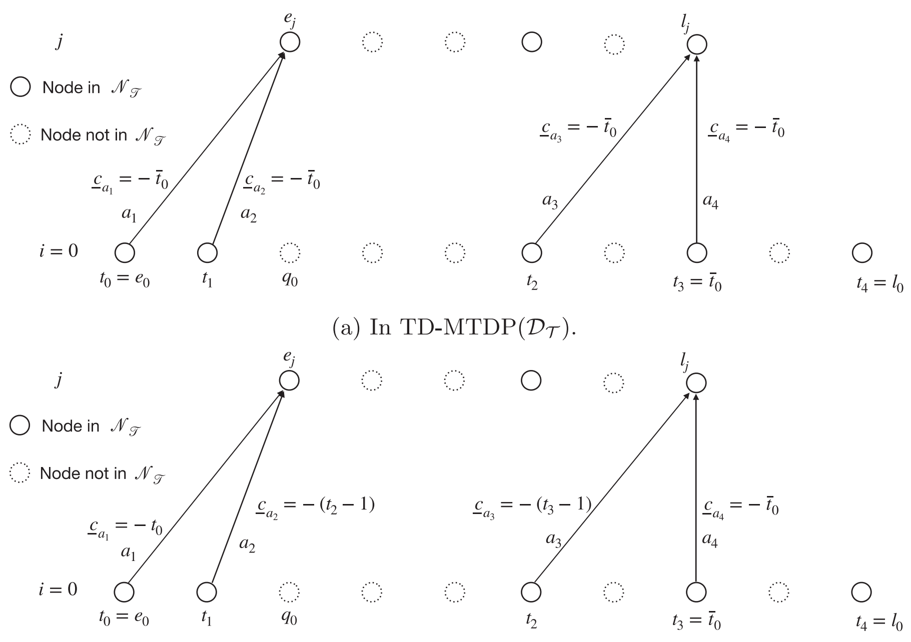
The first term in this objective reflects when the vehicle returns to the depot whereas the second measures when it departs. Recognizing that the vehicle must depart the depot at a time that enables it to visit each location during its time window, the coefficient associated with variables in the second term are $\bar{t}_0$, the largest value t such that $t + \tau_{0i}(t) \leq \min_{j \in N} l_j$ for some $i \in N$. With arguments analogous to those used in Vu et al. (2020) for the TD-TSPTW, one can show that the TD-MTDP($\mathcal{D}_\mathcal{T}$) is a relaxation of the TD-MTDP. We propose strengthening this relaxation by increasing the values of the cost coefficients in second term of this objective function.

We illustrate the second term of this objective in Fig. 2a, which illustrates a portion of a partially time-expanded network that consists of nodes for two locations, the depot ($i = 0$), and another location (j). The figure illustrates that the objective function proposed in Vu et al. (2020) measures all departures from the depot, even those at the open of its time window, as occuring much later, at the time $\bar{t}_0$.

As an example, consider a solution to the TD-MTDP that departs the depot for node j at time q_0, wherein $t_1 < q_0 < t_2$. That the TD-MTDP($\mathcal{D}_\mathcal{T}$) is a relaxation of the TD-MTDP relies on the same reasoning mentioned in Section 4.1. Namely, that you can map the solution to the TD-MTDP to a solution to the TD-MTDP($\mathcal{D}_\mathcal{T}$) wherein the locations are visited in the same order and always at the same time or earlier. With the network partially depicted in Fig. 2a, the departure time q_0 can not be represented. Instead, one would need to map the solution to the TD-MTDP($\mathcal{D}_\mathcal{T}$) as starting at time t_1. In fact, all departures at times t, $t_1 < t < t_2$ can be represented as occuring at time t_1. Then, to ensure the objective function value of the resulting solution to the TD-MTDP($\mathcal{D}_\mathcal{T}$) does not exceed the value of the original solution to the TD-MTDP, the method in Vu et al. (2020) would measure the departure as occuring at time $\bar{t}_0$. Doing so significantly weakens the provable lower bound DDD generates by solving such a relaxation.

A careful analysis yields an objective function that more accurately measures the vehicle departure time from the depot. We illustrate this analysis in Fig. 2b. Namely, the presence of node $(0, t_2) \in \mathcal{N}_\mathcal{T}$ implies that one need not measure the departure as occuring at time $\bar{t}_0$ to have a relaxation but instead at time $t_2 - 1$. Given that $t_2 < \bar{t}_0$, the objective function value of the mapped solution under this objective is at least as great as the objective considered in the TD-MTDP($\mathcal{D}_\mathcal{T}$).

Formally, consider the times $T^0 = \{t_0, t_1, \ldots, t_m = \bar{t}_0\}$ such that at an iteration of DDD, the partially time-expanded network contains the nodes $\{(0,t_0), (0,t_1), \ldots, (0, t_m = \bar{t}_0)\}$. In other words, T^0 represents the set of time points at which the depot is represented in the current partially time-expanded network. As such, we propose solving the following relaxation in the context of executing DDD. We note that the sets $\delta^+_{(i,t)}$ and $\delta^-_{(i,t)}$ are defined as in Section 3.1, albeit with respect to the arc set $\mathcal{A}_\mathcal{T}$.



j
e_j
l_j
Node in $\mathcal{N}_\mathcal{T}$
Node not in $\mathcal{N}_\mathcal{T}$
$\underline{c}_{a_1} = -\bar{t}_0$
$\underline{c}_{a_2} = -\bar{t}_0$
$\underline{c}_{a_3} = -\bar{t}_0$
$\underline{c}_{a_4} = -\bar{t}_0$
a_1
a_2
a_3
a_4
$i = 0$
$t_0 = e_0$
t_1
q_0
t_2
$t_3 = \bar{t}_0$
$t_4 = l_0$
j
e_j
l_j
Node in $\mathcal{N}_\mathcal{T}$
Node not in $\mathcal{N}_\mathcal{T}$
$\underline{c}_{a_1} = -t_0$
$\underline{c}_{a_2} = -(t_2 - 1)$
$\underline{c}_{a_3} = -(t_3 - 1)$
$\underline{c}_{a_4} = -\bar{t}_0$
a_1
a_2
a_3
a_4
$i = 0$
$t_0 = e_0$
t_1
q_0
t_2
$t_3 = \bar{t}_0$
$t_4 = l_0$

(a) In TD-MTDP($\mathcal{D}_\mathcal{T}$).

(b) Strengthened.

Fig. 2. Measuring departure time from depot.

$$\text{minimize} \sum_{((i,t)(0,t'))\in\mathcal{A}_\mathcal{T} | i\neq 0} (t + \tau_{i0}(t)) x_{((i,t)(0,t'))} - \sum_{t_k\in T^0 : t_k+1\in T^0} \sum_{((0,t_k),(i,t'))\in\mathcal{A}_\mathcal{T}} t_k x_{((0,t_k),(i,t'))} - \sum_{t_k\in T^0 : t_k+1\notin T^0} \sum_{((0,t_k),(i,t'))\in\mathcal{A}_\mathcal{T}} (t_{k+1} - 1) x_{((0,t_k),(i,t'))} \tag{10}$$

subject to

$$\sum_{(i,t)\in\mathcal{N}_\mathcal{T}} \sum_{a\in\delta^+_{(i,t)}} x_a = 1, \quad \forall i \in N, \tag{11}$$

$$\sum_{a\in\delta^+_{(i,t)}} x_a - \sum_{a\in\delta^-_{(i,t)}} x_a = 0, \quad \forall (i,t) \in \mathcal{N}_\mathcal{T},\ i \neq 0, \tag{12}$$

$$x_a \in \{0, 1\}, \quad \forall a \in \mathcal{A}_\mathcal{T}. \tag{13}$$

Lemma 1. *The mathematical program (10)–(13) is a relaxation of the TD-MTDP that is stronger than the TD-MTDP($\mathcal{D}_\mathcal{T}$).*

Proof. Consider a solution $s = ((0, q_0 = t), (u_1, q_1), \dots, (u_n = 0, q_n))$ to the TD-MTDP that has objective function value $q_n - q_0$. We consider two cases that differ based on whether the departure time from the depot, t, in the solution s is represented in $\mathcal{D}_\mathcal{T}$. In both cases we use the result from Vu et al. (2020) that one can map the solution s to the TD-MTDP to a solution $s' = ((0, q'_0 = t'), (u_1, q'_1), \dots, (u_n = 0, q'_n))$ to the TD-MTDP($\mathcal{D}_\mathcal{T}$) wherein $q'_i \leq q_i \ \forall i = 0, \dots, n$.

- $(0, q_0 = t) \in \mathcal{N}_\mathcal{T}$: In this case, one can construct a solution s' with $q'_0 = q_0$. Thus, s' has objective function value $q'_n - q_0$ in (10)–(13), which is no greater than $q_n - q_0$, the objective function value of s in TD-MTDP.
- $(0, q_0 = t) \notin \mathcal{N}_\mathcal{T}$: In this case, let $k = argmax_{p=0}^{m}\{t_p : t_p < q_0.\}$. Namely, k is the largest index such that $t_k \in T^0$ and $t_k < q_0$. That t_k exists is guaranteed because $\mathcal{N}_\mathcal{T}$ contains the node $(0, e_0)$. In this case, one can construct a solution s' with $q'_0 = t_k$. Thus, s' has objective function value $q'_n - (t_{k+1} - 1)$ in (10)–(13), which, given that $t_k < q_0 < t_{k+1}$, is no greater than $q_n - q_0$, the objective function value of s in TD-MTDP.

Lastly, we note that $q'_n - q'_0 \geq q'_n - \bar{t}_0$, the objective function value of s' in TD-MTDP($\mathcal{D}_\mathcal{T}$). Thus, solving (10)–(13) will yield at least as great a lower bound as solving the TD-MTDP($\mathcal{D}_\mathcal{T}$). □

Recall that when solving the TD-TSPTW, DDD can terminate when the optimal solution to the TD-TSPTW($\mathcal{D}_\mathcal{T}$) consists only of arcs of the form $((i,t),(j,t'))$ where $t' = \max\{e_j, t + \tau_{ij}(t)\}$ as such a solution is optimal for the TD-TSPTW($\mathcal{D}$). Such a solution to the TD-MTDP($\mathcal{D}_\mathcal{T}$) may not be optimal for the TD-MTDP as waiting at the depot may yield an improving solution and the node set $\mathcal{N}_\mathcal{T}$ may not contain the nodes necessary to capture all waiting opportunities. However, a similar stopping criteria can be used, which is given in the following lemma.

Lemma 2. *Suppose the optimal solution $s = ((u_0 = 0, q_0 = t), (u_1, q_1), \dots, (u_n = 0, q_n))$ to TD-MTDP($\mathcal{D}_\mathcal{T}$) satisfies the following properties:*

Property 5. *All arcs $((u_i, t_i), (u_{i+1}, t_{i+1}))$ are such that $t_{i+1} = \max\{e_{i+1}, t_i + \tau_{u_i u_{i+1}}(t_i)\}$,*

Property 6. *The node $(0, q_0 + 1)$ is in $\mathcal{N}_\mathcal{T}$.*

Then s is optimal for the TD-MTDP.

Proof. Property 5 implies that s is also feasible for the TD-MTDP. Property 6 implies that the objective function value of s when evaluated as a solution to the TD-MTDP is the same as its objective function value when evaluated as a solution to the relaxation TD-MTDP($\mathcal{D}_\mathcal{T}$). Thus, s is optimal for the TD-MTDP. □

Lastly, we note that when solving the TD-MTDP, DDD also includes the node $(0, \bar{t}_0)$ in the initial partially time-expanded network, $\mathcal{D}_\mathcal{T}$.

5.2. Additional mechanism for discovering candidate solutions

To solve the TD-MTDP, we first note that we adapt the integer programming-based heuristics presented in Section 4.2 for solving the TD-TSPTW. Specifically, we construct the same partially time-expanded networks, $\mathcal{D}^1_\mathcal{T}, \mathcal{D}^2_\mathcal{T}$, described there. However, we formulate and solve integer programs based on the Duration objective.

We refer to the resulting integer programs as TD-MTDP($\mathcal{D}^1_{\mathcal{T}}$) and TD-MTDP($\mathcal{D}^2_{\mathcal{T}}$).

We propose two additional heuristic mechanisms for producing candidate solutions to the TD-MTDP. Given the nature of the Duration objective, the first mechanism takes as input a feasible solution to the TD-MTDP and constructs a restricted integer program that optimizes its departure time from the depot. The second mechanism instead seeks to repair solutions to the relaxation, TD-MTDP($\mathcal{D}_{\mathcal{T}}$), from which a feasible solution to the TD-MTDP cannot be derived, and solves a restricted integer program to do so. Like the heuristics proposed in Vu et al. (2020) for the TD-TSPTW, both mechanisms search for improving solutions to the TD-MTDP by creating a partially time-expanded network, $\bar{\mathcal{D}_{\mathcal{T}}}$, and solving the resulting instance of the TD-MTDP($\bar{\mathcal{D}_{\mathcal{T}}}$). We next discuss each mechanism in detail.

New mechanism 1:

This mechanism takes as input a sequence of locations $(u_0 = 0, u_1, \dots, u_n = 0)$ that can induce a feasible solution to the TD-MTDP. Specifically, the sequence itself is a TSP tour (i.e. it does not contain any subtours), and there exists a departure time for each location in that sequence but the last from which a feasible solution to the TD-MTDP can be derived. Formally, there exist departure times $t_i, i = 0, \dots, n-1$, such that $t_i + \tau_{u_i u_{i+1}}(t_i) \le t_{i+1}, e_i \le t_i \le l_i$, and $e_0 \le t_{n-1} + \tau_{u_{n-1} u_n}(t_{n-1}) \le l_0$. Given such a sequence of locations the mechanism creates a partially time-expanded network, $\bar{\mathcal{D}_{\mathcal{T}}}^{time}$, to find departure times for that sequence that yield the smallest duration. The same node set, $\mathcal{N}$, used to create the complete time-expanded network, $\mathcal{D}$, is used to construct $\mathcal{D}^{time}_{\mathcal{T}}$. However, the arc set, $\bar{\mathcal{A}_{\mathcal{T}}}^{time} \subseteq \mathcal{A}$ consists only of arcs between subsequent locations in the sequence $(u_0 = 0, u_1, \dots, u_n = 0)$. Formally, $\bar{\mathcal{A}_{\mathcal{T}}}^{time} = \{((p,t),(q,t')) \in \mathcal{A} : p = u_i, q = u_{i+1}, \text{ for some } i = 0, \dots, n-1\}$. The mechanism then formulates and solves the mixed integer program TD-MTDP($\bar{\mathcal{D}_{\mathcal{T}}}^{time}$).

New mechanism 2:

This mechanism seeks to repair a solution, $\underline{s}$, to the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) from which a feasible solution to the TD-MTDP can not be immediately derived. The heuristic we propose operates in a manner similar to the heuristic proposed in Franceschi, Fischetti, & Toth (2006) for the Distance-Constrained Capacitated Vehicle Routing Problem (DCVRP). That method derives a neighborhood of a solution to the DCVRP by removing from each route in the solution a subset of the locations visited by that route. Doing so leaves a set of "holes" in each route. Then, an integer program is solved to re-insert the removed locations into these holes.

Our method is similar in structure, albeit for a single route/tour. We note that as the operations of this mechanism do not depend on vehicle departure or arrival times, we initially describe it in "space," i.e. not in terms of a time-expanded network. Like the first heuristic, this mechanism creates a partially time-expanded network, $\bar{\mathcal{D}_{\mathcal{T}}}^{repair}$, based on the same node set, $\mathcal{N}$, used to create the complete time-expanded network, $\mathcal{D}$. It differs from the first heuristic in how it creates the arc set, $\bar{\mathcal{A}_{\mathcal{T}}}^{repair}$.

The mechanism begins by deriving from $\underline{s}$ a base subtour, $b_0 = (u^0_0, u^0_1, .., u^k_{p_0} = u^0_0)$, consisting of a subset of the locations, N. It then partitions the remaining set of locations, $N \setminus b_0$ into sets $L_0, \dots, L_m$, $m \ge 0$. Specifically, we have $\cup_{i=0}^m L_i = N \setminus b_0$ and $L_i \cap L_j = \emptyset$, $\forall i, j = 0, \dots, m$, $i \ne j$. It then generates three disjoint sets of arcs, $\mathcal{A}^{repair}_i \subseteq \mathcal{A}, i = 1, 2, 3$. These arc sets are defined as follows.

- $\mathcal{A}^{repair}_1$: Arcs $a = ((u,t),(v, \max\{e_v, t + \tau_{uv}(t)\})$ such that location v is visited directly after location u in the base subtour b_0. These allow for connectivity between locations in the base subtour b_0.
- $\mathcal{A}^{repair}_2$: Arcs $a = ((u,t),(v, \max\{e_v, t + \tau_{uv}(t)\})$ and $a = ((v,t),(u, t + \max\{e_u, t + \tau_{vu}(t)\}))$ such that location u is in the base subtour b_0 and v is in a set $L_i, i = 0, \dots, m$. These allow the vehicle to travel from a location in b_0 to a location in the set L_i and later return to a location in b_0. These arcs are created for nodes in each set, L_i.
- $\mathcal{A}^{repair}_3$: Arcs $a = ((u,t),(v, \max\{e_v, t + \tau_{uv}(t)\})$ and $a = ((v,t),(u, t + \max\{e_u, t + \tau_{vu}(t)\}))$ such that $u, v \in L_i, i = 0, \dots, m$. These allow for connectivity between locations not in the base subtour, b_0, but in the same set, L_i.

We note that to manage the size and computational complexity of the resulting integer program, the mechanism does not add to $\bar{\mathcal{A}_{\mathcal{T}}}^{repair}$ arcs between nodes in different sets, $L_i, L_j, i \ne j$. Similarly, the arc set $\mathcal{A}^{repair}_1$ only contains arcs that enable the vehicle to visit locations in the base subtour, b_0, in a single sequence.

We illustrate the mechanism for creating the restricted graph in Fig. 3a–c. Fig. 3a illustrates the solution to a TD-MTDP($\mathcal{D}_{\mathcal{T}}$) that contains two subtours. Fig. 3b then depicts the base tour, b_0, and a single set of remaining locations, L_0. Fig. 3c illustrates the graph the mechanism generates, and on which it formulates and solves the TD-MTDP($\bar{\mathcal{D}_{\mathcal{T}}}^{repair}$). We note that not all arcs are depicted in Fig. 3c. For example, arc set $\mathcal{A}^2$ would also contain arcs that connect u_1 and u_6. Finally, Fig. 3d depicts the solution produced by the heuristic by solving the resulting TD-MTDP($\bar{\mathcal{D}_{\mathcal{T}}}^{repair}$), with an indication of which set each arc in the solution came from.

To execute these steps, the mechanism must first choose the base subtour, b_0, and then partition the remaining locations. Recall that TD-MTDP($\mathcal{D}_{\mathcal{T}}$) is a relaxation in part because the arc set $\mathcal{A}_{\mathcal{T}}$ may contain arcs that model travel times that are shorter than actual travel times. The presence of such "short" arcs allows for solutions to the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) to contain structures that render them infeasible for the TD-MTDP. Namely, they can contain subtours and/or they can violate time windows at locations when evaluated with respect to actual travel times.

To be precise, we refer to a sub-sequence $(u_i, \dots, u_j)$ of the sequence $(u_0, u_1, \dots, u_q)$ as an *infeasible sequence* if there do not exist times, t_k, that satisfy the following conditions:

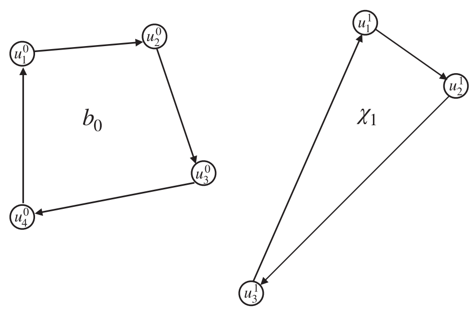
1. $e_{u_k} \le t_k \le l_{u_k}, \forall k = i, \dots, j$.
2. $t_k + \tau_{u_k u_{k+1}}(t_k) \le t_{k+1} \ \ \forall k = i, \dots, j-1$.

In words, there is no way the vehicle can visit the sequence of locations $(u_i, \dots, u_j)$ in that order without violating the time window associated with at least one of the locations. We refer to an infeasible sequence $(u_i, \dots, u_j)$ as a *minimal* infeasible subsequence if there is no other infeasible subsequence $(u_p, \dots, u_q)$ such that $i \le p < q \le j$ and either $p > i$ or $q < j$.

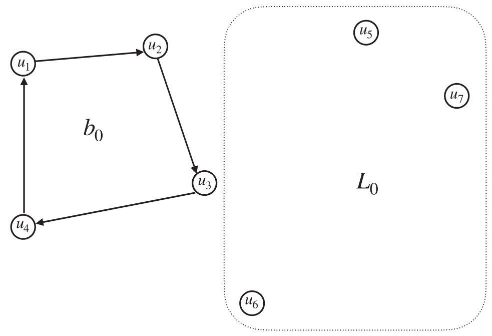
Thus, given a solution $\underline{s}$, wherein either (or both) infeasibility is present, we identify in $\underline{s}$ the set of (sub)-tours $\chi_0, \dots, \chi_r, r \ge 0$, wherein $\chi_k = (u^k_0, u^k_1, \dots, u^k_{q_k}), q_k \ge 2, k = 0, \dots, r, u^k_{q_k} = u^k_0$. In addition, we label χ_0 as the (sub)-tour that contains the depot. If the solution $\underline{s}$ prescribes a single tour, then $r = 0$ and χ_0 is in fact a tour.

Recall that the arc set $\mathcal{A}^{repair}_1$ maintains the sequence the vehicle visits locations in χ_0 in the solution $\underline{s}$. We observe that given an infeasible sequence of nodes $(u_0, u_1, \dots, u_g)$ that does not visit all locations, inserting a node u into this sequence will likely yield an infeasible sequence. We note that the resulting sequence is not guaranteed to violate a time window as travel times that are time dependent may not satisfy the triangle inequality. However, if χ_0 contains an infeasible sequence, the resulting TD-MTDP($\mathcal{D}^{repair}_{\mathcal{T}}$) is likely to be infeasible. Thus, the mechanism behaves differently depending on whether χ_0 contains such a sequence.

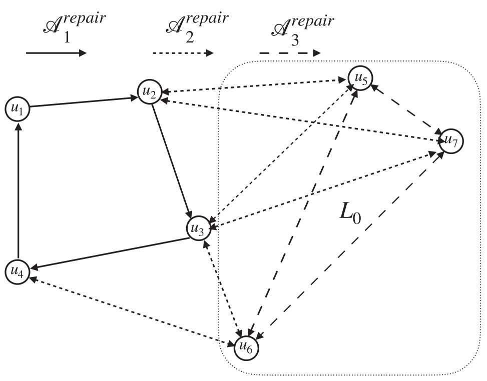
1. χ_0 does not contain an infeasible sequence: In this case, a solution to the TD-MTDP can not be derived from $\underline{s}$ because it pre-



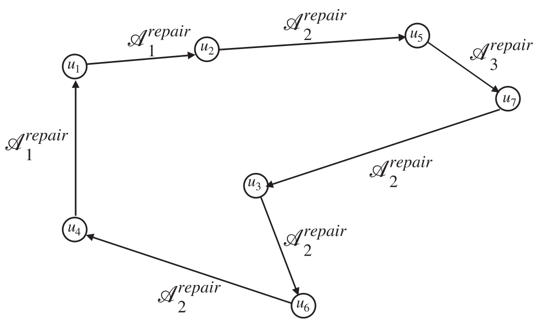
(a) Solution to TD-MTDP($\mathcal{D}_{\mathcal{T}}$).



(b) Node sets for restricted graph (in space) used to formulate TD-MTDP($\mathcal{D}_{\mathcal{T}}^{repair}$)



(c) Restricted graph used to formulate TD-MTDP($\mathcal{D}_{\mathcal{T}}^{repair}$), not all arcs shown.



(d) Solution to TD-MTDP found by solving TD-MTDP($\mathcal{D}_{\mathcal{T}}^{repair}$)

Fig. 3. IP-based heuristic for repairing solutions to relaxation TD-MTDP($\mathcal{D}_{\mathcal{T}}$).

scribes subtours (e.g. $r \geq 1$). In this case, the mechanism sets $b_0 = \chi_0$, and $L_{i-1} = \chi_i, i = 1, \ldots, r$.

2. χ_0 does contain an infeasible sequence. We note that this also includes cases wherein $r = 0$ and thus $\underset{\sim}{s}$ does not prescribe subtours. However, when it does, as in the previous case, it creates the sets $L_{i-1} = \chi_i, i = 1, \ldots, r$. Next, the mechanism identifies all minimal infeasible sequences, $q_k = (v_0^k, \ldots, v_{p_k}^k)$, $p_k \geq 2, k = 0, \ldots, s$ within χ_0. Next, when sequences overlap (e.g. $q_k \cap q_l \neq \emptyset$), their union is taken. Then, for each remaining sequence or union of sequences, the mechanism creates the set $L_{r+k} = (v_0^k, \ldots, v_{p_k}^k), k = 0, \ldots, s$. Finally, the mechanism creates the base subtour $b_0 = \chi_0 \setminus \cup_{k=1}^{r}((v_0^k, \ldots, v_{p_k}^k) \setminus \{0\})$. We note that the locations in the resulting base subtour b_0 are visited in the same order as in χ_0 and b_0 is constructed to contain the depot even if it is in an infeasible sequence.

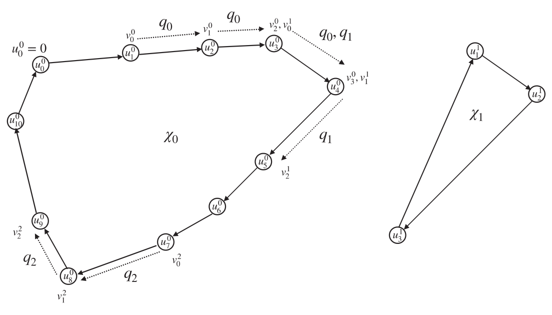
We illustrate the procedure for the second case in Fig. 4a and 4 b. The solution to the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) depicted in Fig. 4a prescribes two subtours. In addition, the subtour that contains the depot, χ_0, contains three infeasible sequences, q_0, q_1, q_2. The infeasible sequences q_0 and q_1 intersect. We depict the corresponding node sets b_0, L_0, L_1, and L_2 in Fig. 4b. The subtour χ_1 is used to generate its own node set, L_0. As they intersect, the infeasible sequences q_0 and q_1 are combined to create the node set L_1. Lastly, the infeasible sequence q_2 is used to create the node set L_2. Each node set in Fig. 4b is labeled with its source from Fig. 4a.

We next provide a more precise description and pseudo-code of how infeasible subsequences in the set χ_0 are detected and the sets L_i are generated. Regarding infeasible sequences, given the sequence the locations in χ_0 are visited, Algorithm 1 examines each subsequences of that sequence to see if it is feasible. Specifically,

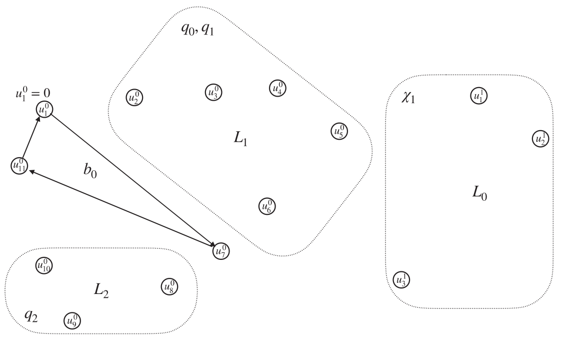
Algorithm 1 Detect infeasible subsequences - Complexity O(n^2).

```
1. Require: Solution $\underset{\sim}{s}$ to relaxation
2. Require: Set $\chi_0 \in N$, ordered based on solution $\underset{\sim}{s}$
3. Initialize: $p_0 = |\chi_0|$
4. Initialize: $s_j \leftarrow j+1$ for $j = 2.p_0$ //assume no infeasible subsequence associated with $j$
5. For $i = 0$ to $p_0 - 2$
6.     For $j = i+2$ to $p_0$ do
7.         If traveling from $u_i^0$ to $u_j^0$ at time $e_{u_i^0}$ violates time window at $u_j^0$ //O(1)
8.             $s_j \leftarrow i$ // update the shortest infeasible subsequence associated to $j$
9.             Break;
10.        End if
11.    End for
12.    Stop if no infeasible subsequence starts from $i$
13. End For
14. $J = \cup_{j=2}^{p_0}\{j | s_j \leq j\}$
15. $\Omega = \cup_{j \in J}\{(s_j, j)\}$.
16. return $J$, $\Omega$, and labels $s$
```

for each subsequence it determines whether visiting those locations in the order visited in $\underset{\sim}{s}$ will violate a time window if each location is departed as early as possible. We note that because we assume travel times observe the FIFO property, it is sufficient to determine when a time window is violated when locations are departed as early as possible.



(a) Solution to TD-MTDP($\mathcal{D}_\mathcal{T}$).



(b) Node sets for restricted graph (in space) used to formulate TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$)

Fig. 4. Generating node sets $b_0, L_i, i = 0, \ldots, m$ when χ_0 contains an infeasible sequence.

Algorithm 1 associates a label, s_j, with each location $j \in \chi_0$ that serves two purposes: (1) if location j is part of an infeasible subsequence, and if so, (2) the start node i of that subsequence. The label s_j for each location $j \in \chi_0$ is initialized with the value $j+1$. Recall that the start node i of an infeasible subsequence containing j must precede j in the solution $\underline{s}$. Thus, when s_j has a value that is greater than j, this indicates that it is **not** part of an infeasible subsequence.

Returning to the example depicted in Fig. 4a, presume that departing u_1^0 at its earliest departure time and following the sequence u_1^0, u_2^0, u_3^0 and u_4^0 would violate the time window at u_4^0. In this case, Algorithm 1 would begin by searching for an infeasible subsequence that begins at u_0^0 (the depot). It would discover that the time window at u_4^0 is violated and update the label s_4 of node u_4^0 from 5 to 0. It would then search for an infeasible subsequence that begins at u_1^0. In doing so, it would identify that the time window at node u_4^0 would be violated and thus update the label s_4 of node u_4^0 from 0 to 1. Similarly, presume that departing u_3^0 at the opening of its time window violates the time window at u_5^0. As such, the algorithm will first update the label s_5 of u_5^0 to 2 when it searches for infeasible subsequences that originate at u_2^0. It will then update the label s_5 again to 3 when searching for infeasible subsequences that begin at u_3^0 Lastly, presuming departing u_7^0 at the opening of its time window violates the time window at u_9^0, at termination the label s_9 of u_9^0 will have value 7.

Algorithm 1 returns two sets. The first, J, is a set that contains all the locations in χ_0 that are the end points of an infeasible subsequence. The set Ω contains for each such location j the start location of that infeasible subsequence. Returning to the example depicted in Fig. 4a, J would consist of the nodes (4,5,9). The set Ω would consist of the tuples $\{(1,4), (3,5), (7,9)\}$.

Algorithm 2 scans the locations Algorithm 1 marked as being contained in an infeasible subsequence to determine the sets $L_i, i = 0, \ldots, m$. As noted, the algorithm is designed to combine infeasible sequences that overlap (i.e. their node sets have a non-empty intersection) into the same set L_i. It does so by scanning the locations that are marked as being included in an infeasible subsequence in order to extend each infeasible subsequence to include subsequent infeasible subsequences it overlaps with. It uses the label c to record the position of the last examined location in the infeasible subsequence currently being extended. Then, as a new location is scanned, the label s that indicates the starting position of the infeasible sequence containing that location is compared with c to see if the new location can be included in the subsequence being extended.

Continuing with our example, Algorithm 2 would scan the set J in the order (4,5,9). In the first iteration of the for loop, $j = 4, n =$

Algorithm 2 Create artificial location sets - Complexity O(n).

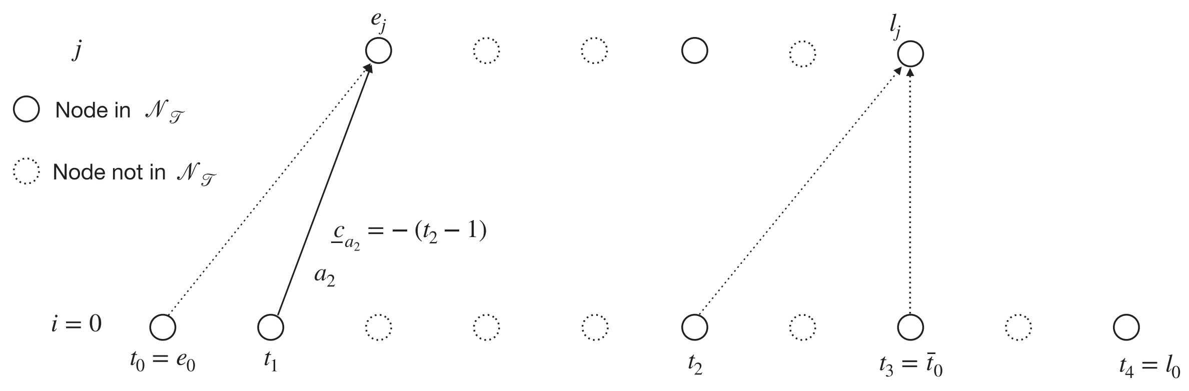
```
1.  Require: $J, \Omega$, and labels $s$ //All outputs from Algorithm 1
2.  Require: Solution $\underline{s}$ to relaxation
3.  Initialize: $m \leftarrow 0$
4.  for $j \in J$ visited in increasing order in $\underline{s}$
5.      $n \leftarrow next(j), c \leftarrow j$ //next(j): the element after $j$ in $J$
6.      While $s_n \le c$ //overlap infeasible subsequences
7.          $c \leftarrow n, n \leftarrow next(n_j)$
8.      End While
9.      $m \leftarrow m + 1$
10.     $L_{m-1} \leftarrow \cup_{j' \in J | j' \le c}(s_{j'}, j')$
11.     Remove all $j'$ from $J$ where $j' \le c$
12.     Remove all $(s_{j'}, j')$ from $\Omega$ where $j' \le c$
13. end for
```

5, and $c = 4$. We see that $s_5 = 3 \le 4$, or, in words, the starting location of the infeasible sequence containing the location in position 5 is before the most recent location examined. The condition in the while loop is satisfied and thus the current infeasible subsequence can be extended. Next, c is set to 5 and n is set to 9. We have $s_9 = 7$ which violates the while condition, indicating that the current infeasible subsequence cannot be extended to include this location. At this point, the set L_1 is created with all locations with s label that is less than or equal to 5, which is locations (1,2,3,4). These locations are removed from further consideration and the for loop would continue with the location in position 7.

We also note that many mixed integer programming solvers will return multiple integer feasible solutions discovered in the course of searching the branch-and-bound tree. In our implementation, we collect such solutions from solving the TD-MTDP($\mathcal{D}_\mathcal{T}$). We then formulate and solve a TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) for each solution that satisfies the two stated conditions.

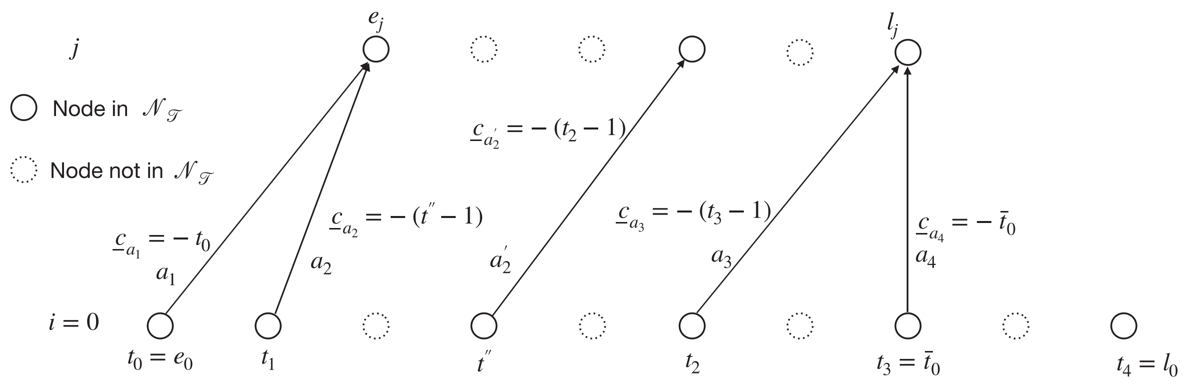
5.3. Refining $\mathcal{D}_\mathcal{T}$ to dynamically add waiting opportunities at the depot

DDD begins with a partially time-expanded network, $\mathcal{D}_\mathcal{T}$, wherein the only nodes representing the depot in $\mathcal{N}_\mathcal{T}$ are $(0, e_0)$, $(0, \bar{t}_0)$, and $(0, l_0)$. However, for some instances of the TD-MDTP all optimal solutions may require the vehicle to wait at the depot until departing at time t, $e_0 < t < l_0$. Thus, to guarantee DDD will produce such a solution, we enhance its refinement procedure (Step 3, Section 4.3) to dynamically add nodes of the form $(0, t)$, $e_0 < t < l_0$, to $\mathcal{N}_\mathcal{T}$.



j
e_j
l_j
Node in $\mathcal{N}_{\mathcal{T}}$
Node not in $\mathcal{N}_{\mathcal{T}}$
$\underline{c}_{a_2} = -(t_2 - 1)$
a_2
$i = 0$
$t_0 = e_0$
t_1
t_2
$t_3 = \bar{t}_0$
$t_4 = l_0$

(a) Solution to TD-MTDP($\mathcal{D}_{\mathcal{T}}$) at iteration uses arc $((0, t_1), (j, e_j))$



j
e_j
l_j
Node in $\mathcal{N}_{\mathcal{T}}$
Node not in $\mathcal{N}_{\mathcal{T}}$
$\underline{c}_{a'_2} = -(t_2 - 1)$
$\underline{c}_{a_2} = -(t'' - 1)$
$\underline{c}_{a_1} = -t_0$
a_1
a_2
a'_2
$\underline{c}_{a_3} = -(t_3 - 1)$
a_3
$\underline{c}_{a_4} = -\bar{t}_0$
a_4
$i = 0$
$t_0 = e_0$
t_1
t''
t_2
$t_3 = \bar{t}_0$
$t_4 = l_0$

(b) Adding node to bisect interval $[t_1, t_2]$

Fig. 5. Dynamically adding a node that models waiting at the depot based on a solution to TD-MTDP($\mathcal{D}_{\mathcal{T}}$).

The procedure is inspired by the strengthened objective presented in Section 5.1. Specifically, presume the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) is solved to yield a solution, s', that departs from the depot at time t_k. Assuming t_{k+1} is the next time point at the depot represented in $\mathcal{N}_{\mathcal{T}}$ and $t_{k+1} > t_k + 1$, this departure time is under-estimated in the objective function (10). We thus add a time point t" that bisects the interval $[t_k, t_{k+1}]$, as doing so minimizes the largest under-estimate associated with departing at times t_k or t" in the next iteration of the algorithm.

We illustrate this procedure in Fig. 5. Returning to Fig. 2b, suppose solving (10)–(13) yielded a solution wherein the vehicle departs the depot for location j at time t_1, as depicted in Fig. 5a. Such a departure reduced the objective function value associated with the solution by $(t_2 - 1)$. Bisecting the interval $[t_1, t_2]$ yields the time point t" and thus the node $(0, t)$" is added to $\mathcal{N}_{\mathcal{T}}$. With this node, in future iterations departures at time t_1 will reduce the objective function value by $(t'' - 1)$. As $(t'' - 1) < (t_2 - 1)$ even if the solution in the next iteration departs at time t_1 and returns to the depot at the same time as in the solution s', the objective function value of the solution will strictly increase.

We present pseudo-code of how waiting nodes are added in Algorithm 3 and note that this procedure is executed after the usual refinement procedures performed in Step 3 (Section 4.3). As with the procedure for creating candidate solutions, we collect solutions found when solving the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) and refine $\mathcal{D}_{\mathcal{T}}$, including performing Algorithm 3, for each one.

We note that by using the strengthened objective (10) in the TD-MTDP($\mathcal{D}_{\mathcal{T}}$), adding node $(0, t'')$ to $\mathcal{N}_{\mathcal{T}}$ will increase the objective function coefficients associated with arcs of the form $((0, t), (i, \tilde{t}))$ in subsequent instances of the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) solved by DDD. This is in contrast to the original objective, (9), wherein

Algorithm 3 Add waiting node.

```
Require: (Sub)tour $\chi_0 = ((0, q'_0 = t), (u_1, q'_1), \ldots, (u_m = 0, q'_m))$ derived from solution to TD-MTDP($\mathcal{D}_{\mathcal{T}}$)
1: Let $t'$ be the smallest value such that $t' > t$ and $(0, t') \in \mathcal{N}_{\mathcal{T}}$
2: if $t' > t + 1$ then
3:     Let $t'' = \lfloor (t + t')/2 \rfloor$
4:     Add node $(0, t'')$ to $\mathcal{N}_{\mathcal{T}}$
5:     Add arcs emanating from $(0, t'')$ to satisfy Property 3.
6: end if
```

the objective function coefficient associated with arcs of the form $((0, t), (i, \tilde{t}))$ remain $-\bar{t}_0$. In fact, one can present examples that DDD will not converge to the optimal solution if nodes that model waiting at the depot are added dynamically and the objective (9) is used to formulate the TD-MTDP($\mathcal{D}_{\mathcal{T}}$) solved at an iteration. Thus, in addition to strengthening the relaxation TD-MTDP($\mathcal{D}_{\mathcal{T}}$), the objective (10) is necessary for correctness of the algorithm.

5.4. Proposed algorithm

We next present a high-level description of the complete algorithm, which we refer to as DDD-TD-MTDP. Regarding instance parameters, we denote the set of times at which location time windows open by e. Similarly, the set of times at which location time windows close is denoted by l. We let τ denote the travel time function and c represent the vector of arc costs. A high-level description of the proposed algorithm is provided in Algorithm 4 with references to the sections where steps are described in greater detail. We also note that because of how $\mathcal{D}^2_{\mathcal{T}}$ is constructed (Section 4.2), solving TD-MTDP($\mathcal{D}^2_{\mathcal{T}}$) (Step 16 of

Algorithm 4 DDD-TD-MTDP.

```
Require: TD-MTDP instance $(N, A)$, $e$, $l$, $\tau$ and $c$, and optimality tolerance $\epsilon$
1: Perform preprocessing steps presented in Vu et~al. (2020)
2: Create initial partially time-expanded network $\mathcal{D}_{\mathcal{T}}$ as in Vu et~al. (2020), albeit with node $(0, \bar{t}_0)$ (Section 5.1).
3: Let $S \rightarrow \emptyset$ be the current set of feasible solutions.
4: while not solved do
5:     Let $\bar{S} = \emptyset$ be the set of feasible solutions found this iteration.
6:     Solve relaxation TD-MTDP($\mathcal{D}_{\mathcal{T}}$) for lower bound $z$ and solutions $\underline{S}$. (Section 5.1)
7:     for all $\underline{s} \in \underline{S}$ do
8:         if $\underline{s}$ can be the basis of a solution, $s$, to TD-MTDP. (Section 4.2) then
9:             Add $s$ to $\bar{S}$
10:        else
11:            Derive $\mathcal{D}_{\mathcal{T}}^{repair}$ from $\underline{s}$ (Section 5.2, New mechanism 2)
12:            Solve TD-MTDP($\mathcal{D}_{\mathcal{T}}^{repair}$) and add solutions to $\bar{S}$ if feasible
13:            Generate valid inequalities corresponding to subtours and infeasible sequences in $\underline{s}$ as in Vu et~al. (2020)
14:        end if
15:    end for
16:    Solve TD-MTDP($\mathcal{D}_{\mathcal{T}}^{1}$), TD-MTDP($\mathcal{D}_{\mathcal{T}}^{2}$) based on $\mathcal{D}_{\mathcal{T}}$ and, when feasible, add their solutions to $\bar{S}$. (Section 4.2)
17:    for all $s \in \bar{S}$ do
18:        Generate and solve $TD-MTDP(\mathcal{D}_{\mathcal{T}}^{time})$ based on sequence locations visited in $s$ and add solution to $S$ (Section 5.2, New mechanism 1).
19:        Generate valid inequality that will prevent solutions to TD-MTDP($\mathcal{D}_{\mathcal{T}}$) in future iterations prescribing the same sequence of locations as in $s$ as in Vu et~al. (2020)
20:    end for
21:    Compute gap $\delta$ between the best solution in $S$ and lower bound, $z$.
22:    if $\delta \leq \epsilon$ then
23:        Stop: best solution in $S$ is $\epsilon-$optimal for TD-MTDP.
24:    else
25:        for all $\underline{s} \in \underline{S}$ do
26:            Refine $\mathcal{D}_{\mathcal{T}}$ by lengthening arcs (Section 4.3)
27:            Refine $\mathcal{D}_{\mathcal{T}}$ by adding nodes to the depot (Section 5.3.)
28:        end for
29:    end if
30: end while
```

Algorithm 4) may yield solutions that are not feasible for TD-MTDP. As such, these solutions are also used to refine $\mathcal{D}_{\mathcal{T}}$ in Step 26 of Algorithm 4.

Lastly, we note that the analogous algorithm for solving the TD-DMP is Algorithm 4 only with instances of the TD-DMP solved instead of instances of the TD-MTDP in Steps 6, 12, 16, and, 18. We refer to this algorithm as DDD-TD-DMP.

6. Computational results

In this section, we first discuss how our computational study was performed. We then assess the impact of the enhancements proposed in Section 5 on the ability of DDD-TD-MTDP to solve instances of the TD-MTDP. To do so, we first benchmark the enhanced method against both the method proposed in Vu et al. (2020) and the commercial mixed integer programming solver Gurobi (version 8.11) solving the integer program (2)–(5) formulated on a complete time-expanded network. Comparisons are done on the instances that were considered in Vu et al. (2020). Results regarding the performance of the method proposed in Vu et al. (2020) are taken from that paper. After studying the performance of each method at a summary level, we then analyze the impact of the enhancements on the performance of DDD-TD-MTDP in more detail. Next, we assess the performance of the enhanced DDD-TD-MTP approach on larger TD-MTDP instances than those considered by Vu et al. (2020). Then, to understand the robustness of the proposed enhancements, we assess the ability of the DDD-TD-DMP to solve instances of the TD-DMP. We finish the section with a comparison of the ability of DDD-TD-MTDP to solve instances of the TD-MTDP with the ability of DDD-TD-DMP to solve instances of the TD-DMP.

6.1. Setup of computational study

We consider three sets of instances, all of which are from the literature and have been used to assess the performance of methods for solving the TD-TSPTW. However, as the TD-MTDP and TD-DMP differ from the TD-TSPTW only in the objective function, these instances can also be used to study the performance of algorithms for these problems. We next give a brief overview of these sets of instances. All three sets of instances are discussed in Vu et al. (2020) in greater detail, and we refer the interested reader to that paper for a more detailed description.

The first two sets, labeled *Set 1* and *Set w100*, were proposed in Arigliano et al. (2018) and originally generated as instances of the TD-TSP. Arigliano et al. (2018) added time windows to these instances in a manner that guaranteed their feasibility. Both sets contain instances that consider either 10, 20, 30, or 40 locations. The primary difference between these sets is the variability of travel times in the instances they contain. Recall that Ichoua et al. (2003) models travel times as a piecewise linear function of departure times. The 952 instances in *Set 1* consider travel times that can be modeled by three linear segments while those in *Set w100* (960 instances) consider travel times that require 73 linear segments. The third set of instances, which we refer to as *Set MM-TDTSPTW*, was first proposed in Vu et al. (2020). Instances in that set were generated with a methodology similar to the one used for the first two sets and parameter values that are mostly similar to those used to generate *Set w100*. Regarding travel time variability, these instances also consider travel times that require 73 linear segments. However, these instances are larger, as they consider either 60, 80, or 100 locations. The set consists of 240 instances for each number of locations and thus 720 instances in total.

To assess the impact of the proposed enhancements when solving the TD-MTDP, we execute DDD-TD-MTDP in two different configurations of enhancements. Specifically, we execute the following configurations:

- DDD-TD-MTDP(1,3): This variant involves executing the DDD of Vu et al. (2020), albeit having it solve a tighter relaxation at each iteration (the enhancement discussed in Section 5.1), solve TD-MTDP($\mathcal{D}_{\mathcal{T}}^{time}$) to search for improved primal solutions (Step 18 of Algorithm 4, the enhancement **New mechanism 1** discussed in Section 5.2), and dynamically add nodes that represent waiting at the depot (the enhancement discussed in Section 5.3). In this variant, Steps (11) and (12) of Algorithm 4 are not executed.
- DDD-TD-MTDP: This is the variant above, DDD-TD-MTDP(1,3), albeit also solving TD-MTDP($\mathcal{D}_{\mathcal{T}}^{repair}$) as discussed in Section 5.2, **New mechanism 2**. As such, Steps (11) and (12) of Algorithm 4 are executed.

Regarding implementation, the algorithm is implemented in C++, and all experiments were run on a workstation with a Intel(R) Xeon (R) CPU E5-4610 v2 2.30GHz processor running the Ubutu Linux 14.04.3 Operating System. We next describe implementation

Table 1
Performance when solving TD-MTDP for Arigliano et al. (2018) instances.

	Set 1 - 952 instances		*Set w100* - 960 instances	
Method	# Solved	Time	# Solved	Time
Gurobi	788	950.48	942	784.23
DDD of Vu et al. (2020)	952	224.43	960	124.86
DDD-TD-MTDP(1,3)	952	70.16	960	6.01

details for DDD-TD-MTDP; the same configuration was used when executing DDD-TD-DMP.

DDD-TD-MTDP repeatedly solves integer programs, and the implementation uses Gurobi 8.11 to do so. When solving instances of TD-MTDP($\mathcal{D}_\mathcal{T}$), TD-MTDP($\mathcal{D}^1_\mathcal{T}$), or TD-MTDP($\mathcal{D}^2_\mathcal{T}$) the time limit was set to three minutes. If, when solving TD-MTDP($\mathcal{D}_\mathcal{T}$), the solver had not found a feasible solution by the three minute time limit, it was allowed to continue until such a solution was found. DDD-TD-MTDP solved TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$) when there were no more than five sets, L_i, and did so to a time limit of five minutes. All such integer programs were solved to an optimality tolerance of 1%.

In all experiments reported on below, the stopping conditions for Gurobi, DDD-TD-MTDP (either configuration), and DDD-TD-DMP were a two hour time limit and a provable optimality gap of $\epsilon = 1\%$. Whenever executing Gurobi, the Threads parameter of Gurobi was set to limit Gurobi to one thread of execution. We note that all reported times are in seconds.

6.2. Solving the TD-MTDP

We first compare the performance of DDD-TD-MTDP(1,3) with that of the two benchmark methods with respect to the following summary statistics: (1) the number of instances solved within the given time limit (two hours), and, (2) the average time the method took to solve an instance. We report these statistics in Table 1.

We see that the DDD of Vu et al. (2020) outperformed Gurobi, both in terms of the number of instances it can solve and the average time it needs to do so. However, we also see that the enhancements enabled DDD-TD-MTDP(1,3) to solve instances in significantly less time. In fact, the time needed to solve instances is reduced by at least a factor of three.

We also observe in Table 1 that the proposed enhancements enabled DDD-TD-MTDP(1,3) to solve instances from *Set w100* in much less time than those in *Set 1*. Relatedly, DDD-TD-MTDP(1,3) needed fewer iterations, on average, to solve instances from *Set w100* (4.35 iterations) than those from *Set 1* (6.35 iterations). To understand this, we study the quality of the dual bound produced at the first iteration of DDD-TD-MTDP(1,3)'s execution. To measure that quality, we compute the gap between the dual bound, $\underline{z}^F$, DDD-TD-MTDP(1,3) produced at the first iteration, and the dual bound, $\underline{z}^L$, it produced at the last iteration of its execution. The gap is calculated as $(\underline{z}^L - \underline{z}^F)/\underline{z}^L$. For instances from *Set 1*, the average of these gaps is 10.56% while it is 6.16% for instances from *Set w100*. We conclude that the stronger dual bounds DDD-TD-MTDP(1,3) was able to produce early in its execution when solving instances from *Set w100* are one reason it was able to solve those instances so quickly.

The DDD of Vu et al. (2020) modeled the vehicle waiting at the depot by creating an initial partially time-expanded network, $\mathcal{D}_\mathcal{T}$, that contained all potential waiting nodes. One of the enhancements proposed in this paper is to instead generate such nodes dynamically. To assess the impact of doing so, we first study the number of nodes of the form $(0, t)$, $e_0 \le t \le l_0$, in the network, $\mathcal{D}_\mathcal{T}$, at termination of both DDD-based methods. We report in Table 2 the number of such nodes, on average, created by each method and for each set of instances.

Table 2
Number of depot nodes in $\mathcal{D}_\mathcal{T}$ at termination of DDD method.

	Set 1	*Set w100*
Method	# Nodes of form $(0, t)$	# Nodes of form $(0, t)$
DDD of Vu et al. (2020)	58.58	41.05
DDD-TD-MTDP(1,3)	7.53	7.00

Table 3
Increase in network size for DDD methods.

	Set 1	*Set w100*
Method	$\lvert\mathcal{N}^{Final}_\mathcal{T}\rvert - \lvert\mathcal{N}^{Initial}_\mathcal{T}\rvert$	$\lvert\mathcal{N}^{Final}_\mathcal{T}\rvert - \lvert\mathcal{N}^{Initial}_\mathcal{T}\rvert$
DDD of Vu et al. (2020)	340.36	344.39
DDD-TD-MTDP(1,3)	119.66	58.60

We see that, on average, DDD-TD-MTDP(1,3) created 12.85% of the nodes the DDD of Vu et al. (2020) created for instances from *Set 1* and 17.05% of such nodes for instances from *Set w100*. In short, by adding nodes that model waiting at the depot dynamically, DDD-TD-MTDP(1,3) created far fewer such nodes than the DDD of Vu et al. (2020).

Turning to network size overall, we present in Table 3 the average increase in the cardinality of the node set, $\mathcal{N}_\mathcal{T}$, from the first iteration to the last for each DDD-based method. We see that by dynamically adding nodes that model waiting at the depot, DDD-TD-MTDP(1,3) generated significantly smaller partially time-expanded networks overall. As DDD-TD-MTDP(1,3) solves instances of the TD-MTDP($\mathcal{D}_\mathcal{T}$) at each iteration and the time required to do so is correlated to the size of the network $\mathcal{D}_\mathcal{T}$, these smaller networks enabled DDD-TD-MTDP(1,3) to converge in much less time.

We conclude from these results that the enhancements present in DDD-TD-MTDP(1,3) greatly improve the performance of DDD when solving instances of the TD-MTDP. Thus, we next turn our attention to the impact of solving TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$), as proposed in Section 5.2. Given the performance of DDD-TD-MTDP(1,3) on the Arigliano et al. (2018) instances reported on in Table 1, for this analysis we consider the set of larger instances, *Set MM-TDTSPTW*.

We study the impact of solving TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$) in two ways. First, we compare the performance of DDD-TD-MTDP(1,3) (wherein TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$) is not solved) and DDD-TD-MTDP (TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$) is solved). In addition to the summary statistics presented above, we consider two others. First, we present the number of instances for which a method was able to produce a feasible solution (# Found). Second, we present the average optimality gap a method reported at termination for the instances that method was not able to solve but was able to produce a feasible solution. We present results for both methods in Table 4, by number of locations in the instance.

We see that solving TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$) enabled DDD-TD-MTDP to produce feasible solutions for and solve more instances. Even though DDD-TD-MTDP solved more instances, the average amount of time it needed to do so is only slightly greater. As a second assessment of the effectiveness of the proposed heuristic, we observe that for 317 of the 592 instances for which DDD-TD-MTDP produced a primal solution, the best primal solution it found was produced by solving TD-MTDP($\mathcal{D}^{repair}_\mathcal{T}$).

To get another perspective on the computational difficulty associated with solving these kinds of problems, we consider the results reported in Dabia et al. (2013) for a branch-and-price method for solving a time dependent vehicle routing problem with time windows wherein the total duration of routes is minimized. We present in Table 5 summaries of the results presented in that paper for the variant that required the pricing problem to yield elementary paths (the better performing of the two proposed variants).

Table 4
Impact of solving TD-MTDP($\mathcal{D}_T^{repair}$) when solving instances from *Set MM-TDTSPTW* 720 instances, 240 for each value of $|N|$.

	DDD-TD-MTDP(1,3)				DDD-TD-MTDP			
$\|N\|$	# Found	# Solved	Time	Gap unsolved	# Found	# Solved	Time	Gap unsolved
60	204	198	797.95	2.44%	238	209	738.95	4.19%
80	160	144	1,202.31	2.14%	211	153	1,313.00	3.90%
100	71	64	1,928.09	1.86%	143	97	1,903.54	4.22%
Summary	435	406	1,119.52	2.14%	592	459	1,176.41	4.07%

Table 5
Performance of branch-and-price method of Dabia et al. (2013) for solving minimum route duration TD-VRPTW.

# Customers	# Solved	# Unsolved	Average time to solve
25	35	2	1,550.85
50	19	11	2,292.90
100	3	18	3,150.17

Table 6
Performance of DDD-TD-DMP.

Instances	Method	Solved	Time
Set 1	DDD-TD-DMP	952	67.81
952 instances	Gurobi	949	506.28
Set w100	DDD-TD-DMP	960	65.61
960 instances	Gurobi	960	301.76
Set MM-TDTSPTW	DDD-TD-DMP	691	498.00
720 instances	Gurobi	666	1,472.52

As the optimization problems are different, a direct comparison of the algorithm performance results presented in Tables 4 and 5 is not meaningful. However, we do conclude from the results in Table 5 that instances from the class of problems wherein duration objectives are optimized and travel times are time dependent are computationally challenging to solve. Yet, from Table 4 we see that DDD-TD-MTDP was able to solve 63.75% of instances from this class and in less than 20 minutes (on average).

Ultimately, we conclude that all proposed enhancements, together, yield a DDD algorithm, DDD-TD-MTDP that is significantly more effective at solving instances of the TD-MTDP than the DDD of Vu et al. (2020) and the best-performing method in the literature to date. As a result, we next study whether the analogous algorithm for the TD-DMP, the DDD-TD-DMP, performs similarly.

6.3. Solving the TD-DMP

As of this writing, we are unaware of any methods for solving the TD-DMP. Thus, we benchmark DDD-TD-DMP against the performance of Gurobi 8.11 on the same three sets of instances considered for the TD-MTDP. We report in Table 6 two of our summary statistics: (1) the number of instances a method was able to solve, and, (2) the average time it took to do so. Considering *Set 1* and *Set w100*, we observe that DDD-TD-DMP was able to solve every instance while Gurobi 8.11 was able to solve all but three. However, DDD-TD-DMP was much faster than Gurobi, often solving instances in less than a fifth of the time. Turning to instances from *Set MM-TDTSPTW*, we see that while DDD-TD-DMP could not solve every instance it solved more than Gurobi and in much less time (often in less than a third of the time).

We next note that DDD-TD-DMP was able to find a feasible solution for all instances from *Set MM-TDTSPTW* and the average optimality gap associated with those it could not solve is only 1.73%. Gurobi was able to find a feasible solution for 670 (of the 720) instances. Of the four that Gurobi found a feasible solution but did not solve, the average optimality gap is 1.96%. In short, we conclude that DDD-TD-DMP is the most computationally effective method for solving the TD-DMP.

6.4. Comparing solving the TD-MTDP and solving the TD-DMP

Considering solving the TD-MTDP (Table 1) and solving the TD-DMP (Table 6), we observe that DDD-TD-DMP was often able to solve instances from *Set 1* and *Set w100* of the TD-DMP in less time than DDD-TD-MTDP(1,3) could solve corresponding instances of the TD-MTDP. Similarly, for larger underlying networks (e.g. *Set MM-TDTSPTW*), DDD-TD-DMP was able to solve more instances of the TD-DMP than DDD-TD-MTDP could the TD-MTDP.

To try and understand this difference in performance, we next compare the performance of the two algorithms on the same instances, albeit used to formulate the two different optimization problems. We report in Table 7 two summary statistics: (1) the average time the method needed to find the first primal solution (Time to first), and, (2) the average gap between the dual bound the method produced in its first iteration and its last (Dual increase).

Based on the results in Table 7, we observe that DDD-TD-DMP is more able to quickly produce both feasible solutions and strong dual bounds for the TD-DMP than DDD-TD-MTDP can for the TD-MTDP. As the only difference in the two sets of experiments is the objective function considered in the optimization problem, we formulate the following hypotheses:

- Feasible solutions: The delivery man objective encourages the vehicle to arrive at a location early. As the relaxation solved by DDD-TD-DMP may consider “short” arcs, arriving to a location early in a solution to the relaxation increases the chances that solution can be the basis of a feasible solution to the original problem. As evidence of this hypothesis, we note the Time to first columns in Table 7.
- Dual bounds: The duration objective allows for multiple solutions, each involving visiting locations in different sequences, to yield the same objective function value. Thus, DDD-TD-MTDP may need to execute more iterations to increase the dual bound. As evidence of this hypothesis in addition to the Dual increase columns in Table 7, we note that for instances from *Set MM-TDTSPTW*, DDD-TD-MTDP needed, on average, 10.12 iterations to solve an instance of the TD-MTDP but DDD-TD-DMP needed only 5.35 iterations to solve an instance of TD-DMP.

To summarize, we conclude that the proposed enhancements yield the best-performing algorithms to date at solving the TD-MTDP and TD-DMP. We also conclude that the algorithm for solving the TD-DMP, the DDD-TD-DMP, is particularly effective.

6.5. Solving the MTDP and DMP

Finally, to get another perspective on the performance of the DDD-based algorithm described in Algorithm 4, we executed it on variants of the two problems considered in this paper with static (i.e. not time-dependent) travel times. More specifically, we execute it to solve instances of the Minimum Tour Duration Problem

Table 7
Comparing performance on instances of TD-DMP and TD-MTDP.

Instance	TD-MTDP		TD-DMP	
set	Time to first	Dual increase	Time to first	Dual increase
Set 1	57.00	10.56%	0.96	4.37%
Set w100	3.00	6.16%	0.05	7.96%
Set MM-TDTSPTW	858.98	4.68%	45.72	1.54%
Average	306.33	7.13%	15.58	4.62%

Table 8
Comparing performance of DDD-TD-MTDP and algorithm of Tilk & Irnich (2017) on MTDP.

	# instances	Tilk & Irnich (2017) Solved	Tilk & Irnich (2017) Time	DDD-TD-MTDP Solved	DDD-TD-MTDP Time
Easy	32	20	3.60	29	860.84
Hard	18	16	137.4	3	6,629.67

Table 9
Comparing performance of DDD-TD-DMP and method of Heilporn et al. (2010) on DMP.

Attempted in Heilporn et al. (2010)	# instances	Heilporn et al. (2010) Solved	Heilporn et al. (2010) Time	DDD-TD-DMP Solved	DDD-TD-DMP Time
Yes	20	16	844.59	11	3,605.50
No	30	N/A	N/A	19	3,243.33

(MTDP) and the Delivery Man Problem (DMP). For either problem, Algorithm 4 is executed as presented, albeit the appropriate MIP solved for that problem. For the MTDP, we compare the performance of the proposed algorithm with that of the method proposed in Tilk & Irnich (2017). For the DMP, we compare the performance of the proposed algorithm with that of the method proposed in Heilporn et al. (2010). To the best of our knowledge, these are the best-performing methods for these two problems to date. All results reported below for these methods are from the papers themselves.

Both papers study the performance of the respective proposed methods on multiple sets of instances. One set considered in both is the set of 50 asymmetric TSPTW instances proposed in Ascheuer, Fischetti, & Grötschel (2001). As such, for brevity, we consider these instances in this section. The node sets, N, in these instances range in size from 12 to 233. Lastly, we note that the literature often partitions these instances into a set of 32 “easy” instances and 18 “hard” instances. We do the same.

We first consider the MTDP. We benchmark against the hybrid algorithm presented in Tilk & Irnich (2017) that is labeled Algorithm 3 in that paper. This algorithm combines multiple techniques, including preprocessing, neighborhood search, column generation, and dynamic programming. The computational results reported in that paper for a given instance are achieved by initializing the algorithm with the optimal TSPTW tour with respect to total travel time for that instance. Details regarding the computational setting can be found in Tilk & Irnich (2017). DDD-TD-MTDP was executed on those same instances with the same parameter values as discussed above. We report in Table 9 the number of instances each method could solve and the average time it executed before termination. We report results for the Easy and Hard instances separately.

We observe that DDD-TD-MTDP was able to solve 9 more of the Easy instances. However, it solved 13 fewer of the Hard. We also observe that DDD-TD-MTDP is slower, on average, than the algorithm proposed in Tilk & Irnich (2017). Lastly, we note that the average provable optimality gap for the instances DDD-TD-MTDP could not solve was 2.93%.

We next consider the DMP. We benchmark against the method presented in Heilporn et al. (2010), which consists of using a mixed integer proramming solver to solve a provably tighter integer programming formulation of the DMP than a formulation based on classical techniques for this class of problem. Results reported in Heilporn et al. (2010) were achieved by solving the formulation with ILOG CPLEX with a one hour time limit and all other parameters left to their default values. While Ascheuer et al. (2001) propose 50 instances, Heilporn et al. (2010) report results for 20 instances that have node sets of cardinality 67 or less. DDD-TD-DMP was executed on all 50 instances with the same parameter values as discussed above. We report results for the instances attempted in Heilporn et al. (2010) separately from those not attempted.

We observe that DDD-TD-DMP solved five fewer of the instances attempted in Heilporn et al. (2010). However, it was able to solve 19 of the 30 instances not considered in that paper, many of which consider more nodes than 67. More specifically, DDD-TD-DMP was able to solve instances with 152, 172, 193, and 201 locations, and required on average 134.30 seconds to do so. While DDD-TD-DMP could not solve the instance with 233 locations, it terminated with a primal solution and dual bound that together yielded a provable optimality gap of 1.89%.

From these experimental results, we conclude that the DDD-based algorithmic framework proposed in this paper is computationally effective for both these problems. Overall, for instances it could not solve within two hours it was often able to produce solutions and dual bounds that together yielded an optimality gap that was less than 3%.

7. Conclusions and future work

In this paper, we studied TD-TSPTW-type problems wherein the objective function value depends in part on the departure time of the vehicle from the depot. As such, optimizing these problems requires optimizing the time the vehicle begins its tour. We proposed an iterative *Dynamic Discretization Discovery*-based algorithm wherein some waiting opportunities at the depot may not be explicitly represented at a given iteration. Instead, representations of such opportunities are added dynamically based on solution process information. One motivation for the design of this algorithm was that time-expanded networks are an effective modeling construct for time-dependent travel times. In addition, we proposed new heuristic techniques for finding high-quality primal solutions, one of which was designed for objective functions that encourage waiting at the depot. Ultimately, the enhancements we proposed yielded algorithms that outperformed all known methods for the TD-MTDP or TD-DMP on instances taken from the literature. To assess the computational effectiveness of the DDD framework more generally, we also studied its performance when solving problems with static travel times. While designed for a different class of problem, the framework performed reasonably well in comparison to methods that were specifically designed for this class of problem. Ultimately, the DDD-based framework proposed in this paper can be executed, *nearly unchanged*, on four classes of TSPTW-type problems, and is the state of the art for those wherein travel times are time-dependent. The only change needed when considering a different one of these variants is the mixed integer program solved in the course of executing the framework.

Regarding future work, the objective functions we considered only necessitated that the vehicle wait at the depot (other than for a time window to open at another location). Optimal solutions to TD-TSPTWs that minimize objective functions such as total transportation costs may also require the vehicle to wait at one or more other locations. One avenue for future work is to adapt the techniques proposed in this paper to such problems. To date, the capabilities of DDD at solving routing problems have been studied in the context of single vehicle problems. Another avenue for future work is to adapt the ideas in this paper to problems that involve a fleet of multiple vehicles, such as the (Time Dependent) Vehicle Routing Problem with Time Windows. As such problems are typically best solved with methods such as branch-and-price, this avenue would likely require developing a hybrid method that combines the steps of DDD with those of branch-and-price.

References

Abeledo, H. G., Fukasawa, R., Pessoa, A. A., & Uchoa, E. (2013). The time dependent traveling salesman problem: Polyhedra and algorithm. *Mathematical Programming Computation, 5*(1), 27–55.

Albiach, J., Sanchis, J. M., & Soler, D. (2008). An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *European Journal of Operational Research, 189*(3), 789–802.

Archetti, C., & Bertazzi, L. (2021). Recent challenges in routing and inventory routing: E-commerce and last-mile delivery. *Networks, 77*, 255–268.

Arigliano, A., Ghiani, G., Grieco, A., Guerriero, E., & Plana, I. (2018). Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. *Discrete Applied Mathematics*. https://doi.org/10.1016/j.dam.2018.09.017. http://www.sciencedirect.com/science/article/pii/S0166218X18304827

Ascheuer, N., Fischetti, M., & Grötschel, M. (2000). A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks, 36*(2), 69–79.

Ascheuer, N., Fischetti, M., & Grötschel, M. (2001). Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming, 90*(3), 475–506.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals OR, 86*, 529–558.

Balas, E., & Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing, 13*(1), 56–75.

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing, 24*(3), 356–371.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2017a). The continuous-time service network design problem. *Operations Research, 65*(5), 1303–1321.

Boland, N., Hewitt, M., Vu, D. M., & Savelsbergh, M. (2017b). Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin, & M. Lombardi (Eds.), *Integration of AI and OR techniques in constraint programming* (pp. 254–262). Springer International Publishing.

Boland, N., Hewitt, M., Vu, D. M., & Savelsbergh, M. (2017c). Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin, & M. Lombardi (Eds.), *Integration of AI and OR techniques in constraint programming* (pp. 254–262). Cham:Springer International Publishing.

Boland, N. L., & Savelsbergh, M. W. (2019). Perspectives on integer programming for time-dependent models. *Top, 27*, 147–173.

Bront, J. J. M., Méndez-Díaz, I., & Zabala, P. (2014). Facets and valid inequalities for the time-dependent travelling salesman problem. *European Journal of Operational Research, 236*(3), 891–902.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks, 11*(2), 145–164.

Clautiaux, F., Hanafi, S., Macedo, R., Voge, M.-E., & Alves, C. (2017). Iterative aggregation and disaggregation algorithm for pseudo-polynomial network ow models with side constraints. *European Journal of Operational Research, 258*, 467–477.

Cordeau, J.-F., Ghiani, G., & Guerriero, E. (2014). Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. *Transportation Science, 48*(1), 46–58.

Dabia, S., Ropke, S., Van Woensel, T., & De Kok, T. (2013). Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science, 47*(3), 380–396.

Dash, S., Günlük, O., Lodi, A., & Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing, 24*(1), 132–147.

Dumas, Y., Desrosiers, J., Gélinas, E., & Solomon, M. M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research, 43*(2), 367–371.

Franceschi, R. D., Fischetti, M., & Toth, P. (2006). A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming, 105*, 471–499.

Ghiani, G., & Guerriero, E. (2014). A note on the Ichoua, Gendreau, and Potvin (2003) travel time model. *Transportation Science, 48*(3), 458–462.

Goel, A. (2012). The minimum duration truck driver scheduling problem. *EURO Journal on Transportation and Logistics, 1*(4), 285–306.

Gouveia, L., & Voz, S. (1995). A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research, 83*(1), 69–82.

Heilporn, G., Cordeau, J.-F., & Laporte, G. (2010). The delivery man problem with time windows. *Discrete Optimization, 7*(4), 269–282.

Hewitt, M. (2019). Enhanced dynamic discretization discovery for the continuous time load plan design problem. *Transportation Science, 53*, 1731–1750.

Hewitt, M., Boland, N., Savelsbergh, M., & Hewitt, M. (2021). Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science, 55*, 29–51.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research, 144*(2), 379–396.

Kara, I., & Derya, T. (2015). Formulations for minimizing tour duration of the traveling salesman problem with time windows. *Procedia Economics and Finance, 26*, 1026–1034.

Lucena, A. (1990). Time-dependent traveling salesman problem-the deliveryman case. *Networks, 20*(6), 753–763.

Medina, J., Hewitt, M., Lehud, F., & Pton, O. (2019). Integrating long-haul and local transportation planning: the service network design and routing problem. *EURO Journal on Transportation and Logistics, 8*, 119–145.

Melgarejo, P. A., Laborie, P., & Solnon, C. (2015). A time-dependent no-overlap constraint: Application to urban delivery problems. In *Proceedings of the 12th international conference on integration of AI and OR techniques in constraint programming, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015* (pp. 1–17).

Méndez-Díaz, I., Bront, J. J. M., Toth, P., & Zabala, P. (2011). Infeasible path formulations for the time-dependent TSP with time windows. In *Proceedings of the 10th cologne-Twente workshop on graphs and combinatorial optimization. Extended abstracts, villa Mondragone, Frascati, Italy, June 14-16, 2011* (pp. 198–202).

Mingozzi, A., Bianco, L., & Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research, 45*(3), 365–377.

Montero, A., Méndez-Dıaz, I., & Miranda-Bront, J. J. (2017). An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research, 88*, 280–289.

Picard, J.-C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research, 26*(1), 86–110.

Roberti, R., & Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science, 48*(3), 413–424.

Salehipour, A., Sörensen, K., Goos, P., & Bräysy, O. (2011). Efficient GRASP+ VND and GRASP+ VNS metaheuristics for the traveling repairman problem. *4or, 9*(2), 189–209.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing, 4*(2), 146–154.

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

Silva, M. M., Subramanian, A., Vidal, T., & Ochi, L. S. (2012). A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research, 221*(3), 513–520.

Stecco, G., Cordeau, J.-F., & Moretti, E. (2008). A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Computers & Operations Research, 35*(8), 2635–2655.

Sun, P., Veelenturf, L. P., Hewitt, M., & Van Woensel, T. (2018). The time-dependent pickup and delivery problem with time windows. *Transportation Research Part B: Methodological, 116*, 1–24.

Tas, D., Gendreau, M., Jabali, O., & Laporte, G. (2016). The traveling salesman problem with time-dependent service times. *European Journal of Operational Research, 248*(2), 372–383.

Tilk, C., & Irnich, S. (2017). Dynamic programming for the minimum tour duration problem. *Transportation Science, 51*(2), 549–565.

Vu, D. M., Hewitt, M., Boland, N., & Savelsbergh, M. (2020). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science, 54*(3), 703–720.