Solving the Time Dependent Minimum Tour Duration and Delivery Man Problems with Dynamic Discretization Discovery

Duc Minh Vu
Cardiff Business School - Cardiff University,

Mike Hewitt
Department of Information Systems and Supply Chain Management, Quinlan School of Business, Loyola University Chicago, mhewitt3@luc.edu,

Duc D. Vu
The University of Texas at Dallas

In this paper, we present exact methods for solving the Time Dependent Minimum 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, the problems we consider in this paper involve objective function values 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 et al. 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. Thus, when travel times are very variable the function has many pieces and embedding the resulting representation in a mixed integer program, one of the standard techniques for solving routing problems, often leaves an optimization problem that is computationally difficult to solve (Montero et al. 2017).

Vu et al. (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 et al. 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 et al. (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.

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.

In a delivery context, the duration objective considered by the TD-MTDP is appropriate when optimizing from the transportation carrier's perspective. Other objectives can be optimized that instead consider the perspectives of customers. One such objective is considered by the Delivery Man Problem, which considers the amount of time a customer waits between when the vehicle departs the depot and arrives at their location. The problem then seeks to design a TSP tour that minimizes the sum of these waiting times. The same characterization of optimal solutions can be made regarding the Time Dependant Delivery Man Problem (TD-DMP). 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.

While Vu et al. (2020) propose an algorithm for solving the TD-TSPTW, they also present an adaptation of that algorithm to the TD-MTDP. That algorithm is an iterative solution approach that solves at each iteration an instance of a TD-TSPTW (or TD-MTDP) formulated on a *partially time-expanded network*, a time-expanded network wherein not every location is represented at every point in time. The approach constructs these networks so that the resulting TD-TSPTW (or TD-MTDP) instance is a relaxation of the original problem. When the solution to the relaxation can be converted to a feasible solution to the original problem, the algorithm terminates as it has found the optimal solution. When it cannot, the partially time-expanded network is refined and the algorithm continues. In addition, Vu et al. (2020) present heuristic techniques for speeding up the algorithm's search for high-quality primal solutions.

Because all optimal solutions to the TD-MTDP may require the vehicle to wait at the depot before departure, Vu et al. (2020) propose 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 the algorithm uses when solving the TD-MTDP.

To avoid this 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.

We believe this paper makes the following contributions. First, the enhancements it proposes collectively enable the resulting DDD-based algorithm to outperform all known methods from the literature for the TD-MTDP. The analogous algorithm for the TD-DMP also performs better than all known methods. Second, the techniques it proposes for dynamically adding nodes that represent departure times at the depot can be used with objective functions wherein all optimal solutions require the vehicle to wait at one or more locations other than the depot after their time window has opened. One example of such an objective is the total transportation cost associated with a tour. Third, 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 TD-TSPTW-type problems.

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 et al. (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 et al. (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 et al. (1997) and Baldacci et al. (2012). Balas and 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 and 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).

Another algorithmic strategy used to solve the TSPTW is integer programming. Ascheuer et al. (2000, 2001) present some of the first branch-and-cut-based algorithms for this class of problem. Dash et al. (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 et al. (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. Turning to MTDPs, Kara and 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 et al. (2010)

propose mixed integer programming formulations for the DMP with time windows. Both Salehipour et al. (2011) and Silva et al. (2012) propose metaheuristics for the latency objective when time windows are not present. Roberti and 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 and 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 (Gouveia and Voz 1995, Bront et al. 2014, Abeledo et al. 2013, Stecco et al. 2008, Albiach et al. 2008, Méndez-Díaz et al. 2011, Melgarejo et al. 2015). 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 piece-wise 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 et al. (2014), which solves a TD-TSP with a *Makespan* objective, and Ghiani and Guerriero (2014) which further analyzes the model of Ichoua et al. (2003). Tas et al. (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 et al. (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.

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. 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 $N = \{0, 1, 2, ..., n\}$ 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 $i = 1, \ldots, n$ 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 $t \leq t'$, we must have $t + \tau_{ij}(t) \leq t' + \tau_{ij}(t')$. A precise definition of a tour is that the vehicle must depart node 0 at time $t_0 \geq e_0$, visit every node $i \in N$ at a time t_i, wherein $e_i \leq t_i \leq l_i$, 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 $i \neq j, (i, j) \in A$, $t \geq e_i, t' = \max\{e_j, t + \tau_{ij}(t)\}$ (the vehicle can not visit early), and $t' \leq l_j$ (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

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

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

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

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:

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

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.

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.

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 Figure 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 $t' \leq \max\{e_j, t+\tau_{ij}(t)\}$ when $i \neq j$. We label an arc as *too short*

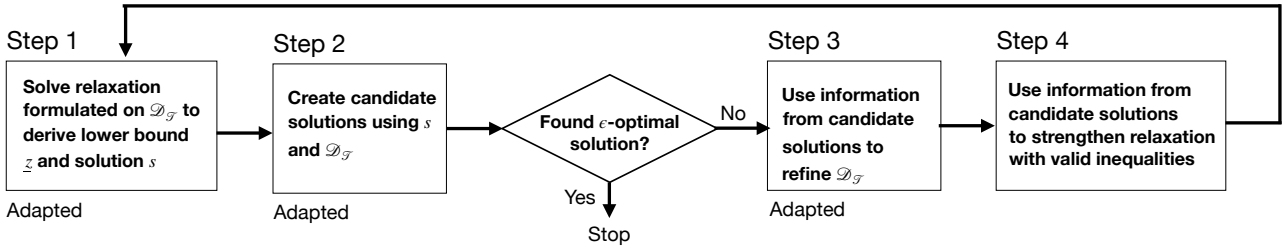


Figure 1 Iteration of DDD-based algorithm

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 $\min\{c_{((i,h)(j,t'))} | t \leq h \leq l_i \text{ and } h + \tau_{ij}(h) \leq l_j\}$.

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. $\forall (i,t) \in \mathcal{N}_{\mathcal{T}}, e_i \leq t \leq l_i$.

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 $t + \tau_{ij}(t) \leq l_j$. 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) $e_j \leq t' \leq t + \tau_{ij}(t)$. Finally, there is no $(j,t'') \in \mathcal{N}_{\mathcal{T}}$ with $t' < t'' \leq t + \tau_{ij}(t)$.

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

Vu et al. (2020) show that Properties 1 - 3 together ensure that any path $p = ((u_0,t_0),(u_1,t_1),...,(u_m,t_m))$, $(u_i,t_i) \in \mathcal{N}$ $i = 0,\ldots,m$ can be mapped to a path $p' = ((u_0,t'_0),(u_1,t'_1),...,(u_m,t'_m))$, $(u_i,t'_i) \in \mathcal{N}_{\mathcal{T}}$ where $t'_k \leq t_k$ for all $0 \leq k \leq m$. 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, $(u_0 = 0, u_1, \ldots, u_n = 0)$, 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, $t_i, i = 0, \ldots, n-1$, 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. $\mathcal{A}^1_\mathcal{T}$ is constructed to contain all 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 $t' \geq t + \tau_{ij}(t)$. Thus, any feasible solution to TD-TSPTW($\bar{\mathcal{D}_\mathcal{T}}^1$) is feasible for TD-TSPTW($\mathcal{D}$). $\mathcal{A}^2_\mathcal{T}$ is less restrictive; it is constructed to contain all 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 $t' > t$. 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($\mathcal{D}$). $\mathcal{A}^2_\mathcal{T}$ is constructed to ensure that solutions to TD-TSPTW($\mathcal{D}^2_\mathcal{T}$) 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.

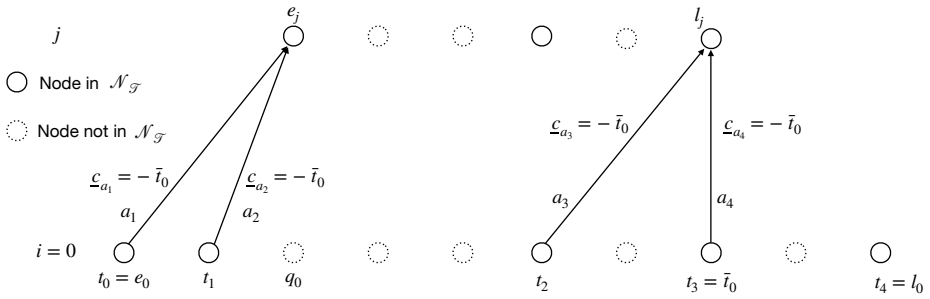
$$\sum_{((i,t)(0,t'))\in\mathcal{A}_\mathcal{T}|i\neq 0} (t+\tau_{i0}(t))x_{((i,t)(0,t'))} - \sum_{((0,t)(i,t'))\in\mathcal{A}_\mathcal{T}|i\neq 0} \bar{t}_0 x_{((0,t)(i,t'))} \tag{9}$$

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.

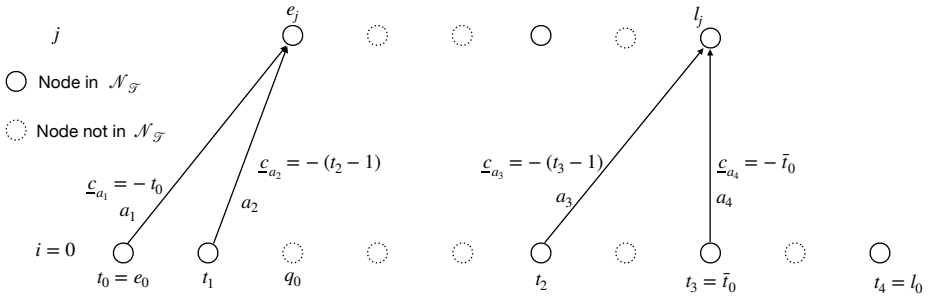
We illustrate the second term of this objective in Figure 2a, which illustrates a portion of a partially time-expanded network that consists of nodes for two locations, the depot ($i = 0$), and

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



(b) Strengthened.

Figure 2 Measuring departure time from depot.

A careful analysis yields an objective function that more accurately measures the vehicle departure time from the depot. We illustrate this analysis in Figure 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 $\mathcal{A}_\mathcal{T}$.

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

subject to

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

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

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

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 of Lemma 1 Consider a solution $s = ((0,q_0 = t),(u_1,q_1),...,(u_n = 0,q_n))$ 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 $s' = ((0,q_0^{'} = t'),(u_1,q_1^{'}),...,(u_n = 0,q_n^{'}))$ to the TD-MTDP($\mathcal{D}_\mathcal{T}$) wherein $q_i^{'} \leq q_i\ \forall i = 0,\ldots,n$.

- $(0,q_0 = t) \in \mathcal{N}_\mathcal{T}$: 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 $k = \arg\max^m_{p=0}\{t_p : t_p < q_0.\}$. 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($\mathcal{D}_\mathcal{T}$). Q.E.D.

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* $s = ((u_0 = 0, q_0 = t), (u_1, q_1), ..., (u_n = 0, q_n))$ *to TD-MTDTP($\mathcal{D}_\mathcal{T}$) 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 of Lemma 2 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.

The first mechanism takes as input a sequence of locations $(u_0 = 0, u_1, \ldots, u_n = 0)$ 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 $t_i, i = 0, \ldots, n-1$, such that $t_i + \tau_{u_i u_{i+1}}(t_i) \leq t_{i+1}, e_i \leq t_i \leq l_i$, and $e_0 \leq t_{n-1} + \tau_{u_{n-1}u_n}(t_{n-1}) \leq l_0$. 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 $\mathcal{D}_{\mathcal{T}}^{time}$. However, the arc set, $\bar{\mathcal{A}_{\mathcal{T}}}^{time} \subseteq \mathcal{A}$ consists only of arcs between subsequent locations in the sequence $(u_0 = 0, u_1, \ldots, u_n = 0)$. Formally, $\bar{\mathcal{A}_{\mathcal{T}}}^{time} = \{((p,t),(q,t')) \in \mathcal{A} : p = u_i, q = u_{i+1}, \text{ for some } i = 0, \ldots, n-1\}$. The mechanism then formulates and solves the mixed integer program TD-MTDP($\bar{\mathcal{D}_{\mathcal{T}}}^{time}$).

The second 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 et al. (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, $c_0 = (u_0^0, u_1^0, .., u_{p_0}^k = u_0^0)$, consisting of a subset of the locations, N. It then partitions the remaining set of locations, $N \setminus c_0$ into sets $L_0, \ldots, L_m,\ m \geq 0$. Specifically, we have $\cup_{i=0}^{m} L_i = N \setminus c_0$ and $L_i \cap L_j = \emptyset,\ \forall i, j = 0, \ldots, m,\ i \neq j$. It then generates three disjoint sets of arcs, $\mathcal{A}_i^{repair} \subseteq \mathcal{A}, i = 1, 2, 3$. These arc sets are defined as follows.

- $\mathcal{A}_1^{repair}$: 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 c_0. These allow for connectivity between locations in the base subtour c_0.
- $\mathcal{A}_2^{repair}$: 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 c_0 and v is in a set $L_i, i = 0, \ldots, m$. These allow the vehicle

to travel from a location in c_0 to a location in the set L_i and later return to a location in c_0. These arcs are created for nodes in each set, L_i.

- $\mathcal{A}_3^{repair}$: 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 $u, v \in L_i, i = 0,\ldots,m$. These allow for connectivity between locations not in the base subtour, c_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, $L_i, L_j, i \neq j$. Similarly, the arc set $\mathcal{A}_1^{repair}$ only contains arcs that enable the vehicle to visit locations in the base subtour, c_0, in a single sequence.

We illustrate the mechanism for creating the restricted graph in Figures 3a, 3b, and 3c. Figure 3a illustrates the solution to a TD-MTDP($\mathcal{D}_{\mathcal{T}}$) that contains two subtours. Figure 3b then depicts the base tour, c_0, and a single set of remaining locations, L_0. Figure 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 Figure 3c. For example, arc set $\mathcal{A}^2$ would also contain arcs that connect u_1 and u_6. Finally, Figure 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.

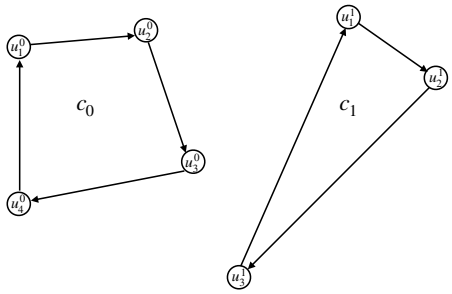
To execute these steps, the mechanism must first choose the base subtour, c_0, and then partition the remaining locations. Recall that TD-MTDP($\mathcal{D}_{\mathcal{T}}$) is a relaxation in part because the arc set $\mathcal{A}_{\mathcal{T}}$ may contain arcs that model travel times that are shorter 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 $(u_i,\ldots,u_j)$ of the sequence $(u_0,u_1,\ldots,u_q)$ as an *infeasible sequence* if there do not exist times, t_k, that satisfy the following conditions:

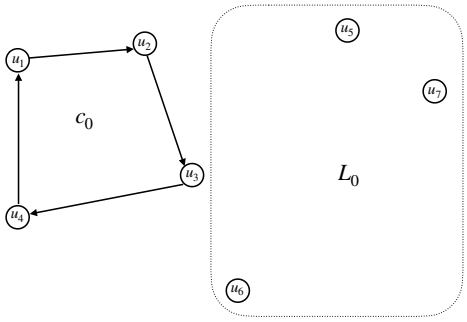
1. $e_{u_k} \leq t_k \leq l_{u_k}, \forall k = i,\ldots,j$.
2. $t_k + \tau_{u_k u_{k+1}}(t_k) \leq t_{k+1}\ \ \forall k = i,\ldots,j-1$.

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

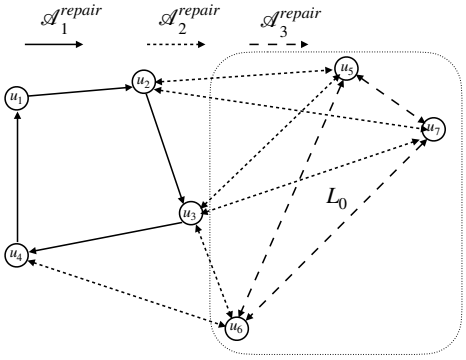
Thus, given a solution $\underline{s}$, wherein either (or both) infeasibility is present, consider a decomposition of $\underline{s}$ into the set of (subtours) $\chi_0,\ldots,\chi_r, r \geq 0$, wherein $\chi_k = (u_0^k, u_1^k, ..., u_{q_k}^k), q_k \geq 2, k = 0,\ldots,r$. In addition, assume χ_0 contains the depot. Recall that the arc set $\mathcal{A}_1^{repair}$ maintains the sequence the vehicle visits locations in χ_0 in the solution $\underline{s}$. We observe that given an infeasible sequence of nodes $(u_0,u_1,...,u_g)$ that does not visit all locations, inserting a node u into this sequence will likely yield



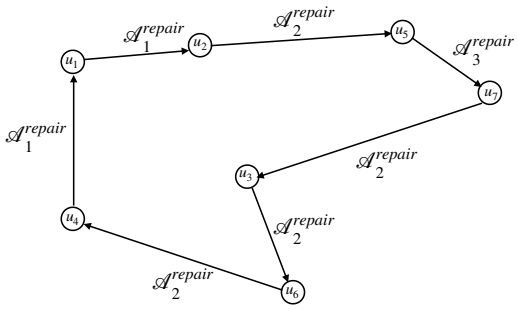
(a) Solution to TD-MTDP($\mathcal{D}_\mathcal{T}$).



(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($\mathcal{D}_\mathcal{T}^{repair}$)

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

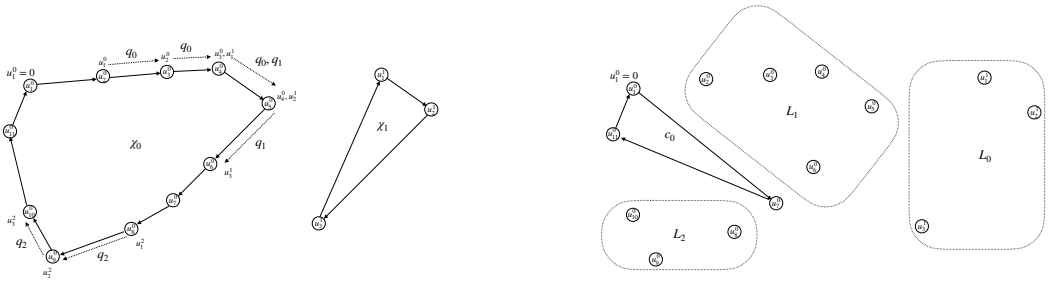
an infeasible sequence. We note that the resulting sequence is not guaranteed to violate a time window as travel times that are time dependent may not satisfy the triangle inequality. However, if χ_0 contains an infeasible sequence, the resulting TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) is likely 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 prescribes subtours (e.g. $r \geq 1$). In this case, the mechanism sets $c_0 = \chi_0$, 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 $m = 0$ and thus $\underline{s}$ 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, $q_k = (u_1^k, \ldots, u_{p_k}^k), p_k \geq 2, k = 0, \ldots, s$ 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 $L_{r+k} = (u_1^k, \ldots, u_{p_k}^k), k = 0, \ldots, s$. Finally, the mechanism creates the base subtour $c_0 = \chi_0 \setminus \cup_{k=1}^{r}((u_1^k, \ldots, u_{p_k}^k) \setminus \{0\})$. We note that the locations in the resulting base subtour c_0 are

visited in the same order as in χ_0 and c_0 is constructed to contain the depot even if it is in an infeasible sequence.

We illustrate the procedure for the second case in Figures 4a and 4b. The solution to the TD-MTDP($\mathcal{D}_\mathcal{T}$) depicted in Figure 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 c_0, L_0, L_1, and L_2 in Figure 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. For a precise description of how these steps are performed, please see Appendix 7. There,



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

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

Figure 4 **Generating node sets $c_0, L_i, i = 0, \ldots, m$ when χ_0 contains an infeasible sequence.**

Algorithm 3 describes how infeasible subsequences in the set χ_0 are detected while Algorithm 4 describes how the sets L_i are generated.

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.

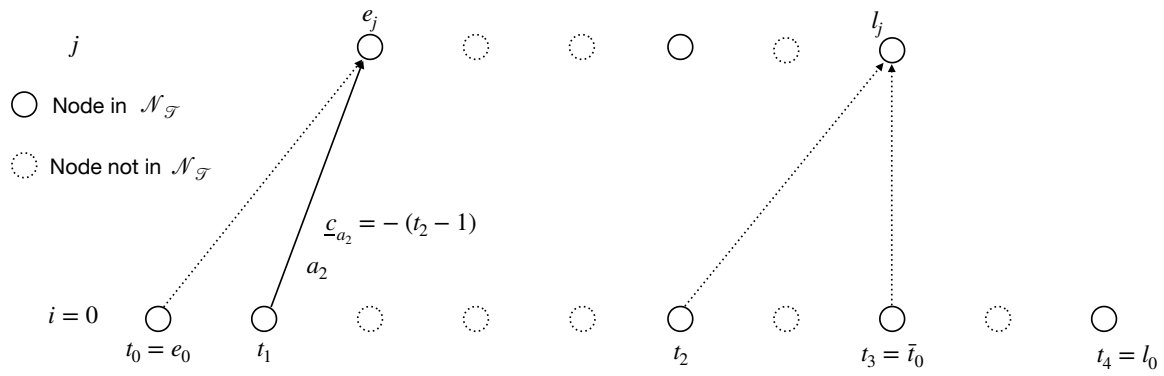
5.3. Refining $\mathcal{D}_\mathcal{T}$ to dynamically add waiting opportunities at the depot

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 $\mathcal{N}_\mathcal{T}$.

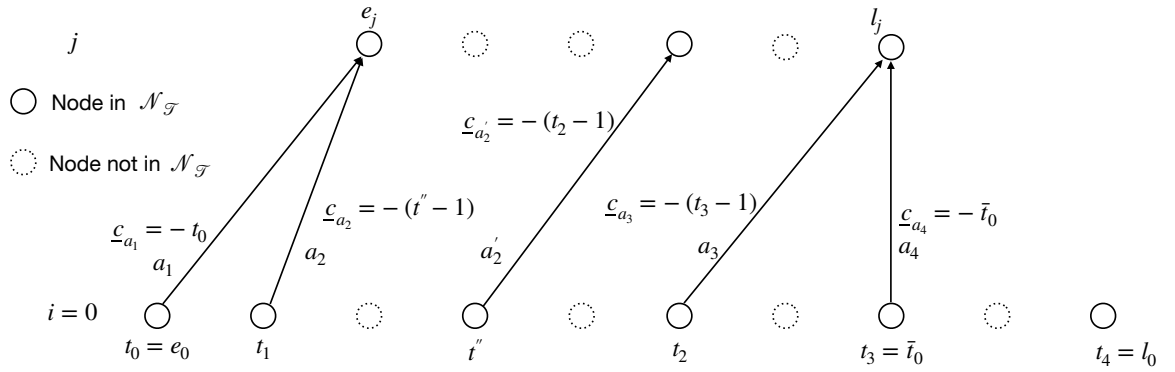
The procedure is inspired by the strengthened objective presented in Section 5.1. Specifically, presume the TD-MTDP($\mathcal{D}_\mathcal{T}$) is solved to yield a solution, s', that departs from the depot at time t_k. Assuming t_{k+1} is the next time point at the depot represented in $\mathcal{N}_\mathcal{T}$ 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 Figure 5. Returning to Figure 2b, suppose solving (10) - (13) yielded a solution wherein the vehicle departs the depot for location j at time t_1, as depicted in Figure 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.



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



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

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

We present pseudo-code of how waiting nodes are added in Algorithm 1 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 1, for each one.

Algorithm 1 Add waiting node

Require: (Sub)tour $\chi_0 = ((0, q_0^{'} = t), (u_1, q_1^{'}), ..., (u_m = 0, q_m^{'}))$ 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**

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 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 theTD-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 2 with references to the sections where steps are described in greater detail. We also note that because of how $\mathcal{D}_\mathcal{T}^2$ is constructed (Section 4.2), solving TD-MTDP($\mathcal{D}_\mathcal{T}^2$) (Step 16 of Algorithm 2) 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 2.

Lastly, we note that the analogous algorithm for solving the TD-DMP is Algorithm 2 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

Algorithm 2 DDD-TD-MTDP

Require: TD-MTDP instance (N, A), e, l, τ and c, and optimality tolerance ϵ

Perform preprocessing steps presented in Vu et al. (2020)
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).
Let $S \to \emptyset$ be the current set of feasible solutions.
while not solved **do**
  Let $\bar{S} = \emptyset$ be the set of feasible solutions found this iteration.
  Solve relaxation TD-MTDP($\mathcal{D}_\mathcal{T}$) for lower bound z and solutions $\underline{S}$. (Section 5.1)
  for all $\underline{s} \in \underline{S}$ **do**
    if $\underline{s}$ can be the basis of a solution, s, to TD-MTDP. (Section 4.2) **then**
      Add s to $\bar{S}$
    else
      Derive $\mathcal{D}_\mathcal{T}^{repair}$ from $\underline{s}$ (Section 5.2)
      Solve TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) and add solutions to $\bar{S}$ if feasible
      Generate valid inequalities corresponding to subtours and infeasible sequences in $\underline{s}$ as in Vu et al. (2020)
    end if
  end for
  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)
  for all $s \in \bar{S}$ **do**
    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).
    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)
  end for
  Compute gap δ between the best solution in S and lower bound, z.
  if $\delta \leq \epsilon$ **then**
    Stop: best solution in S is $\epsilon-$optimal for TD-MTDP.
  else
    for all $\underline{s} \in \underline{S}$ **do**
      Refine $\mathcal{D}_\mathcal{T}$ by lengthening arcs (Section 4.3)
      Refine $\mathcal{D}_\mathcal{T}$ by adding nodes to the depot (Section 5.3.)
    end for
  end if
end while

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) 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 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 setes 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. 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*. 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 2, one of the enhancements 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 2 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. As such, Steps (11) and (12) Algorithm 2 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 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($\mathcal{D}_{\mathcal{T}}^1$), or TD-MTDP($\mathcal{D}_{\mathcal{T}}^2$) 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($\mathcal{D}_{\mathcal{T}}^{repair}$) 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.

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

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 $(0,t),\ e_0 \leq t \leq l_0$, 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

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.

Table 3 Increase in network size for DDD methods

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

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($\mathcal{D}_\mathcal{T}^{repair}$), 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($\mathcal{D}_\mathcal{T}^{repair}$) in two ways. First, we compare the performance of DDD-TD-MTDP(1,3) (wherein TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) is not solved) and DDD-TD-MTDP (TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) 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.

Table 4 Impact of solving TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) when solving instances from Set MM-TDTSPTW

720 instances

	DDD-TD-MTDP(1,3)				DDD-TD-MTDP			
$\lvert N\rvert$	# 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%

We see that solving TD-MTDP($\mathcal{D}_\mathcal{T}^{repair}$) enabled DDD-TD-MTDP to produce feasible solutions 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($\mathcal{D}_\mathcal{T}^{repair}$).

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). 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 5 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).

Table 5 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

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 5), 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 6 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).

Table 6 Comparing performance on instances of TD-DMP and TD-MTDP

Instance set	TD-MTDP		TD-DMP	
	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%

Based on the results in Table 6, 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 6.
- 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 6, 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.

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. 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.

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.

Appendix. Detailed pseudo-code of steps from Section 5.2

In this section, we provide pseudo-code of how the sets c_0, L_i are generated by the heuristic presented in Section 5.2. Algorithm 3 describes how infeasible subsequences in the set χ_0 are detected. Then, Algorithm 4 describes how the sets $L_i, i = 0, \ldots, m$ are generated.

Algorithm 3 Detect infeasible subsequences - Complexity $O(n^2)$.

1. Initialize: $s_j \leftarrow j+1$ for $j = 2..p_0$ //assume no infeasible subsequence associated with j
2. For $i = 0$ to $p_0 - 2$
3. For $j = i+2$ to p_0 do
4. 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)
5. $s_j \leftarrow i$ // update the shortest infeasible subsequence associated to j
6. Break;
7. End if
8. End for
9. Stop if no infeasible subsequence starts from i
10. End For
11. $J = \cup_{j=2}^{p_0} \{j | s_j \leq j\}$
12. $\Omega = \cup_{j \in J} \{(s_j, j)\}$.
13. return J, Ω

Algorithm 4 Create artificial location sets - Complexity O(n)

1. for $j \in J$ visited in increasing order
2. $n_j \leftarrow next(j), c_j \leftarrow j$ //*next(j)*: the element after j in J
3. While $s_{nj} \leq c_j$ //overlap infeasible subsequences
4. $c_j \leftarrow n_j, n_j \leftarrow next(n_j)$
5. End While
6. $m \leftarrow m + 1$
7. $L_{m-1} \leftarrow \cup_{j' \in J | j' \leq c_j} (s_{j'}, j')$
8. Remove all j' from J where $j' \leq c_j$
9. Remove all $(s_{j'}, j')$ from Ω where $j' \leq c_j$
10. end for

References

Hernán G. Abeledo, Ricardo Fukasawa, Artur Alves Pessoa, and Eduardo Uchoa. The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation*, 5(1): 27–55, 2013.

José Albiach, José María Sanchis, and David Soler. 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, 2008.

Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, Emanuela Guerriero, and Isaac Plana. Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. *Discrete Applied Mathematics*, 2018. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2018.09.017. URL `http://www.sciencedirect.com/science/article/pii/S0166218X18304827`.

Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks*, 36(2):69–79, 2000. ISSN 1097-0037.

Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Math. Program.*, 90(3):475–506, 2001.

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

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

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

Natashia Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017a.

Natashia Boland, Mike Hewitt, Duc Minh Vu, and Martin Savelsbergh. Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 254–262. Springer International Publishing, 2017b.

Juan José Miranda Bront, Isabel Méndez-Díaz, and Paula Zabala. Facets and valid inequalities for the time-dependent travelling salesman problem. *European Journal of Operational Research*, 236(3):891–902, 2014.

Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.

Jean-François Cordeau, Gianpaolo Ghiani, and Emanuela Guerriero. Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. *Transportation Science*, 48(1):46–58, 2014.

Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

Yvan Dumas, Jacques Desrosiers, Éric Gélinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995.

Roberto De Franceschi, Matteo Fischetti, and Paolo Toth. A new ilp-based refinement heuristic for vehicle routing problems. *Mathematical Programming*, 105:471–499, 2006.

Gianpaolo Ghiani and Emanuela Guerriero. A note on the ichoua, gendreau, and potvin (2003) travel time model. *Transportation Science*, 48(3):458–462, 2014.

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

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

Géraldine Heilporn, Jean-François Cordeau, and Gilbert Laporte. The delivery man problem with time windows. *Discrete Optimization*, 7(4):269–282, 2010.

Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

Imdat Kara and Tusan Derya. Formulations for minimizing tour duration of the traveling salesman problem with time windows. *Procedia Economics and Finance*, 26:1026–1034, 2015.

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

Penélope Aguiar Melgarejo, Philippe Laborie, and Christine Solnon. A time-dependent no-overlap constraint: Application to urban delivery problems. In *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, pages 1–17, 2015.

Isabel Méndez-Díaz, Juan José Miranda Bront, Paolo Toth, and Paula Zabala. 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*, pages 198–202, 2011.

Aristide Mingozzi, Lucio Bianco, and Salvatore Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, 45(3): 365–377, 1997.

Agustın Montero, Isabel Méndez-Dıaz, and Juan José Miranda-Bront. An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research*, 88:280–289, 2017.

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

Roberto Roberti and Aristide Mingozzi. Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, 48(3):413–424, 2014.

Amir Salehipour, Kenneth Sörensen, Peter Goos, and Olli Bräysy. Efficient grasp+ vnd and grasp+ vns metaheuristics for the traveling repairman problem. *4or*, 9(2):189–209, 2011.

Martin WP Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

Marcos Melo Silva, Anand Subramanian, Thibaut Vidal, and Luiz Satoru Ochi. A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research*, 221(3): 513–520, 2012.

Gabriella Stecco, Jean-François Cordeau, and Elena Moretti. A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Comput. Oper. Res.*, 35(8):2635–2655, August 2008.

Peng Sun, Lucas P. Veelenturf, Mike Hewitt, and Tom Van Woensel. The time-dependent pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 116:1 – 24, 2018. ISSN 0191-2615.

Duygu Tas, Michel Gendreau, Ola Jabali, and Gilbert Laporte. The traveling salesman problem with time-dependent service times. *European Journal of Operational Research*, 248(2):372–383, 2016.

Christian Tilk and Stefan Irnich. Dynamic programming for the minimum tour duration problem. *Transportation Science*, 51(2):549–565, 2017.

Duc Minh Vu, Mike Hewitt, Natashia Boland, and Martin Savelsbergh. Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science*, 54 (3):703–720, 2020.