

Solving Time Dependent Traveling Salesman Problems with Time Windows

Duc Minh Vu, Mike Hewitt¹ and Natashia Boland, Martin Savelsbergh²

¹Department of Information Systems and Supply Chain Management, Quinlan School of Business, Loyola University Chicago

²Georgia Institute of Technology, Atlanta, US

May 30, 2018

Abstract

We present a new solution approach for the Time Dependent Traveling Salesman Problem with Time Windows. This problem considers a salesman who departs from his home, has to visit a number of cities within a pre-determined period of time, and then returns home. The problem allows for travel times that can depend on the time of departure. The solution approach is based on an integer programming formulation of the problem on a time expanded network, as doing so enables time dependencies to be embedded in the definition of the network. However, as such a time expanded network (and thus the integer programming formulation) can rapidly become prohibitively large, the solution approach employs a dynamic discretization discovery framework, which has been effective in other contexts. Our computational results indicate that the solution approach outperforms the best-known methods on benchmark instances and is robust with respect to instance parameters.

Keywords: Traveling Salesman Problem, time windows, time dependent travel times, dynamic discretization discovery

1 Introduction

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem. It has been studied by researchers working in a variety of fields, including mathematics, computer science, and operations research. In the TSP, a salesman departs from his home city, must visit a set of other cities, and then return home. The objective is to minimize the total cost of traveling between cities, given a cost for travel between each city pair. In a logistics context, the salesman is replaced by a vehicle, the home city by a depot, and

the cities by customers. Introducing travel time between customer location pairs and a time window for each customer, which requires the vehicle to visit the customer during a pre-specified period of time, gives the Traveling Salesman Problem with Time Windows (TSPTW). The TSPTW is even more relevant to logistics, as customers often request, or are quoted, a delivery time window.

In this paper, we go beyond the classical TSPTW to allow travel times to depend on the time at which travel commences. This setting is particularly well-suited to logistics systems in urban areas, in which traffic patterns (and hence travel times) vary throughout the day. We present an algorithm for solving the Time Dependent Traveling Salesman Problem with Time Windows (TD-TSPTW) under one of two different objectives. The first objective is to return the vehicle to the depot as soon as possible after the start of the planning horizon. This objective is often referred to as the *makespan* objective. It has received recent attention in the literature; see, for example, Arigliano et al. (2015) and Montero et al. (2017). The second objective is for the vehicle to spend as little time away from the depot as possible, which is referred to as the *duration* objective. The TD-TSPTW with the duration objective has not yet been studied.

The algorithm we propose is based on the *Dynamic Discretization Discovery* (DDD) framework introduced in Boland et al. (2017a), where it was applied to the Service Network Design Problem. While the framework has proven to be effective, it has not yet been adapted to problems with time dependent travel times. With many companies offering tighter and tighter delivery time promises, being able to (effectively) handle travel times that vary throughout the day when planning delivery routes becomes critical. Urbanization, with the United Nations projecting that by 2050 two-thirds of the world population will live in cities (UN 2014), only exacerbates the issue, as it results in an increase in deliveries (either to individuals or retailers) in densely populated areas where congestion and variations in travel times are common. These trends are prompting research into transportation planning problems that explicitly account for time dependent travel times (Figliozzi, Gendreau et al. 2015).

The primary contribution of the research presented in this paper is a new solution method for the TD-TSPTW with a makespan objective, which outperforms the best known methods (by a wide margin) on two sets of benchmark instances. We further show that the method can be easily modified to effectively solve these instances for a duration objective. A secondary contribution is that the research shows that the DDD framework can be used to develop an algorithm that can handle (a few) problems with time dependent travel times, which suggest that the DDD framework might be used to develop effective algorithms for other transportation planning problems in which travel times are time dependent. While the computational performance of existing methods for the TD-TSPTW is highly sensitive to the frequency with which travel times change, our results indicate that the new solution method is not. We believe these results highlight a key advantage the DDD framework, which relies on integer programming formulations defined on (partially) time expanded networks: time dependency of travel times can be directly, and naturally, embedded in a

time expanded network.

The rest of this paper is organized as follows. Section 2 describes the TD-TSPTW in detail and presents a mathematical programming formulation of it. Section 3 then discusses the relevant literature. Section 4 describes the basis of the algorithm, the *partially time-expanded network*, and described how it can be constructed to accommodate time dependent travel times. Section 5 then presents the algorithm that uses partially time-expanded networks to solve instances of the makespan-minimizing TD-TSPTW. Section 6 assesses the computational performance of the algorithm. Then, in Section 7, we turn our attention to the duration objective. Finally, Section 8 summarizes the paper and presents avenues for future work.

2 Problem Setting and Formulation

We study a setting in which a vehicle departs from an initial location, called the *depot*, visits each location in a known set of locations exactly once and during a pre-defined time window, and then returns to the depot. The problem of determining a set of such movements, known as a *tour*, is referred to as the Traveling Salesman Problem with Time Windows (TSPTW). We consider situations in which the time at which travel occurs impacts the travel time, resulting in a problem that is called the Time Dependent Traveling Salesman Problem with Time Windows (TD-TSPTW).

As is commonly done, we assume travel times adhere to the *First-In, First-Out (FIFO)* property, meaning that when traveling between two locations, a later departure cannot lead to an earlier arrival, and that travel time is a piecewise linear function of departure time. We first seek to minimize a makespan objective, which seeks a tour that can be completed so as to return to the depot as early as possible after the start of the planning horizon. Under this objective and with the FIFO property, there exists an optimal solution in which the only waiting the vehicle does, if any, is when it arrives at a location before the location's time window opens. Later, we will discuss how we propose to optimize a duration objective, in which the goal is to complete the tour as quickly as possible after the departure from the depot. Under this objective, the departure time from the depot is flexible, and the tour duration may be minimized by departing from the depot at some time after the start of the planning horizon.

The TD-TSPTW is defined as follows. Let (N, A) denote a directed graph, where the node set $N = \{0, 1, 2, \dots, n\}$ includes the depot (node 0) as well as the set of locations (nodes $1, \dots, n$) that must be visited. Associated with each location $i = 1, \dots, n$ is a time window, $[e_i, l_i]$, during which the location must be visited. Note that the vehicle may arrive at location $i \in N \setminus \{0\}$ before the opening time of the window, e_i , in which case it must wait until the time window opens. There is also a time window, $[e_0, l_0]$, associated with the depot, which indicates that the tour can depart the depot no earlier than e_0 and must return to the depot no later than l_0 .

The set $A \subseteq N \times N$ consists of arcs that represent travel between locations. Associated with each arc $(i, j) \in A$ is a piecewise linear travel time function, $\tau_{ij}(t)$, which gives the travel time along arc (i, j) if the traversal of (i, j) commences at time t . Each travel time function has a finite number of breakpoints, (linear pieces), and satisfies the FIFO property, i.e., for all $(i, j) \in A$ and for times t, t' with $t \leq t'$, it must be that $t + \tau_{ij}(t) \leq t' + \tau_{ij}(t')$. This is equivalent to the requirement that all linear pieces have slope at least -1 . Thus, formally, a feasible solution to the TD-TSPTW is a sequence of node-time pairs, $((i_0, t_0), (i_1, t_1), \dots, (i_n, t_n), (i_{n+1}, t_{n+1}))$, satisfying $i_0 = i_{n+1} = 0$, $\{i_1, \dots, i_n\} = \{1, \dots, n\}$, $t_0 \geq e_0$, $t_{k+1} \geq \max\{e_{k+1}, t_k + \tau_{i_k, i_{k+1}}(t_k)\}$ for all $k = 0, \dots, n$, and $t_{n+1} \leq \ell_0$. The makespan objective minimizes t_{n+1} while the duration objective minimizes $t_{n+1} - t_0$.

We formulate the TD-TSPTW as an integer program defined on a time-expanded network, $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with finite node set \mathcal{N} and finite arc set \mathcal{A} . That such an integer program exists is based on the observation that there is a finite number of feasible tours, that each arc travel time function has a finite number of breakpoints, and that there exists an optimal tour that departs from at least one location at a breakpoint, which implies that there exist a finite time-expanded network that contains an optimal solution.

Therefore, let \mathcal{N} contain nodes (i, t) for $i \in N$ and $t \in \mathcal{T}_i$, the (finite) set of time points at location i including e_i and l_i . Furthermore, let \mathcal{A} contain arcs of the form $((i, t), (j, t'))$ with $i \neq j$, $(i, j) \in A$, $t \in \mathcal{T}_i$, $t' \in \mathcal{T}_j$ and $t' = \max\{e_j, t + \tau_{ij}(t)\}$ (the arc embeds any necessary waiting time), and $t' \leq l_j$ (the vehicle cannot arrive late).

To formulate the integer program, for each arc $((i, t), (j, t')) \in \mathcal{A}$ we define the binary variable $x_{((i,t),(j,t'))}$ to indicate whether or not the vehicle travels along that arc. As we will optimize two different objectives in this paper, we present the following formulation of the TD-TSPTW that seeks to optimize a generic objective function.

$$z = \text{minimize} \quad \sum_{((i,t),(j,t')) \in \mathcal{A}} c_{ij}(t) x_{((i,t),(j,t'))}$$

subject to

$$\sum_{((i,t),(j,t')) \in \mathcal{A}: i \neq j} x_{((i,t),(j,t'))} = 1, \quad \forall j \in N, \quad (1)$$

$$\sum_{((i,t),(j,t')) \in \mathcal{A}} x_{((i,t),(j,t'))} - \sum_{((j,\tilde{t}), (i,t)) \in \mathcal{A}} x_{((j,\tilde{t}), (i,t))} = 0, \quad \forall (i, t) \in \mathcal{N}, i \neq 0 \quad (2)$$

$$x_{((i,t),(j,t'))} \in \{0, 1\}, \quad \forall ((i, t), (j, t')) \in \mathcal{A}. \quad (3)$$

Constraints (1) ensure that the vehicle arrives at each location exactly one time during its time window. Constraints (2) then ensure that the vehicle departs every node at which it arrives (except when it returns to the depot). Finally, Constraints (3) define the decision variables' domains. We denote this formulation as TD-TSPTW(\mathcal{D}) to highlight its use of a time-expanded network.

To represent the makespan objective, we set $c_{ij}(t) = 0$ for all $((i, t), (j, t')) \in \mathcal{A}$, $j \neq 0$ and $c_{i0}(t) = t + \tau_{i0}(t)$ for all $((i, t), (0, t')) \in \mathcal{A}$ to model that costs are only incurred when returning to the depot. Note that $c_{ij}(t)$ is, in this case, both non-negative and non-decreasing in t . To represent the duration objective, the makespan cost function is modified by setting $c_{0j}(t) := -t$ for all $((0, t), (j, t')) \in \mathcal{A}$, which is non-positive and non-increasing.

Note that the impact of time windows and time dependent travel times on the feasible region of the TD-TSPTW are embedded in the time-expanded network, \mathcal{D} . That the vehicle must depart a location within its time window is handled by ensuring that \mathcal{A} does not contain any arcs that arrive at a node outside of its location's time window. Time dependent travel times are embedded in the arcs, $((i, t), (j, t')) \in \mathcal{A}$, through the choice of t' .

3 Literature review

While our paper is focused on exact methods for the TD-TSPTW under different objectives, we believe it is relevant to review literature on the TSPTW as well. Therefore, we discuss the most relevant work on exact methods for the TSPTW and the TD-TSPTW for both a makespan and duration objective. Before doing so, we recall that finding a feasible solution to the TSPTW (and thus the TD-TSPTW) is NP-Hard (Savelsbergh 1985).

The TSPTW with a makespan objective was first studied in scheduling contexts, with Christofides et al. (1981) and Baker (1983) proposing branch-and-bound-based solution methods. Other integer programming-based approaches include Langevin et al. (1993), which presents a two-commodity-based flow formulation for the makespan objective, as well as a total travel time objective.

The literature discussed so far assumes that the time to travel between locations is independent of when travel begins (often referred to as time-independent travel times). However, for various reasons, including a greater acknowledgement of the practical realities of logistics distribution in dense urban areas, researchers have begun to examine speeds, and hence travel times, that vary during the planning horizon. Some studies have focused on the Time Dependent Traveling Salesman Problem, with exact methods proposed in Picard and Queyranne (1978), Lucena (1990), Fischetti et al. (1993), Gouveia and Voz (1995), Albiach et al. (2008), Stecco et al. (2008), Méndez-Díaz et al. (2011), Bront et al. (2014), Abeledo et al. (2013), and Melgarejo et al. (2015). That said, those works do not ensure that the *FIFO* property is observed.

In contrast, Ichoua et al. (2003) proposes a travel time function that does enforce this property. This function assumes that travel speeds are constant within a time interval, but can change from one interval to the next. Ghiani and Guerriero (2014) studies properties of this travel time function, as well as proposes generalizations. This travel time function is also used in approaches for solving time dependent variants of the TSP. Related to the first problem we study, Cordeau et al. (2014) proposes an exact method for a TD-TSP based upon this travel time function in which the makespan objective is optimized. Those ideas

were then extended to the TD-TSPTW, again with the makespan objective, in Arigliano et al. (2015). More recently, Montero et al. (2017) proposes a new formulation and branch-and-cut algorithm for optimizing the TD-TSPTW under the makespan objective.

We next turn to research that proposes exact methods for optimizing the TSPTW under a duration objective. We first note that few exact methods have been proposed to date, and all presume travel times that are time independent. Kara et al. (2013) presents a new mixed integer programming formulation for instances with asymmetric travel times, whereas Kara and Derya (2015) presents a formulation for instances in which travel times are symmetric. Dynamic programming is another algorithmic strategy researchers have used to optimize the TSPTW under this objective. Tilk and Irnich (2015) extended some of the state space relaxation ideas presented in Baldacci et al. (2011) for optimizing the TSPTW under a travel time objective to develop the best-performing method for the duration objective to date. To the best of our knowledge, this is the first paper to optimize the duration objective when travel times are time dependent.

As we will discuss in more detail later, our approach is driven by the careful choice of time-expanded networks, in which not all time points are represented at all times. This time-expanded network is then used to formulate and solve an instance of the TD-TSPTW. In that sense, the method proposed by Dash et al. (2012) for optimizing the TSPTW under a travel time objective is similar, as both methods dynamically refine the granularity at which time is modeled. However, the methods differ with respect to when and how this refinement is done. Whereas the method proposed in Dash et al. (2012) refines the time periods modeled in the context of a preprocessing scheme, the method we propose does so throughout, and in a manner that guarantees convergence to an optimal solution to the TD-TSPTW. They also differ in that the method we propose accommodates time dependent travel times, whereas the method proposed by Dash et al. (2012) presumes they are constant. Finally, we note that Mahmoudi and Zhou (2016) also model time-dependent travel times on a time-expanded network, albeit in the context of solving a different problem. Furthermore, they use a dynamic programming approach, which uses dominance to control the computational effort rather than generating the time-expanded network dynamically as we do in this paper.

The algorithm presented in this paper is based upon the same framework as that used in Boland et al. (2017a) and Boland et al. (2017b). Boland et al. (2017a) first proposed this algorithmic framework, albeit in the context of solving a different problem, the *Service Network Design Problem*. Boland et al. (2017b) then illustrates how this framework may be adapted to solve instances of the TSPTW under a total travel time objective and with time independent travel times. In this paper, we go further by showing how it can be used to solve TSPTWs with time dependent travel times under either a makespan or duration objective.

4 Partially Time-Expanded Network Formulation and Properties

The formulation of the TD-TSPTW presented in Section 2 relies on a network in which each location is represented at each point in time, rendering it computationally challenging to solve for large networks and fine discretization of time. Thus we propose a solution approach that, instead of generating a time-expanded network in a static, *a priori*, fashion, does so in a dynamic and iterative manner. In this section, we show how such a network can be used to derive a lower bound on the optimal value of an instance of the TD-TSPTW when a makespan objective is optimized. Subsequently, in the next section, we discuss an algorithm for solving the TD-TSPTW that iteratively refines a partially time-expanded network and produces a sequence of improving lower bounds, as well as upper bounds, until optimality is proved.

A *partially time-expanded network*, $\mathcal{D}_{\mathcal{T}} = (\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}})$, is derived from a given subset of the timed nodes, $\mathcal{N}_{\mathcal{T}} \subseteq \mathcal{N}$. Given $\mathcal{N}_{\mathcal{T}}$, the arc set $\mathcal{A}_{\mathcal{T}} \subseteq \mathcal{N}_{\mathcal{T}} \times \mathcal{N}_{\mathcal{T}}$ consists of arcs of the form $((i, t), (j, t'))$, in which $(i, t) \in \mathcal{N}_{\mathcal{T}}$, $(j, t') \in \mathcal{N}_{\mathcal{T}}$, $i \neq j$, and $(i, j) \in A$. Like \mathcal{A} , $\mathcal{A}_{\mathcal{T}}$ consists of arcs that model travel between locations. We note that we do not require that arc $((i, t), (j, t'))$ satisfies $t' = \max\{e_j, t + \tau_{ij}(t)\}$ when $i \neq j$, but only $t' \leq \max\{e_j, t + \tau_{ij}(t)\}$. We call an arc *too short* if $t' < \max\{e_j, t + \tau_{ij}(t)\}$. We note that the choice of arc, $((i, t), (j, t'))$, implies that the departure time at i is at least t .

Regarding travel costs, for each $a = ((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$, we define $\underline{c}_{ij}(t)$ as the travel cost associated with departing from location i for location j at time t . Like travel times, we set these costs, $\underline{c}_{ij}(t)$ in such a manner that they under-estimate how the cost of such travel is represented in \mathcal{D} . Specifically, $\underline{c}_{ij}(t)$ is defined as the minimum cost of travel on (i, j) at any departure time from t until the latest possible time at which (i, j) can be traversed:

$$\underline{c}_{ij}(t) := \min\{c_{ij}(h) \mid t \leq h \leq l_i \text{ and } h + \tau_{ij}(h) \leq l_j\}. \quad (4)$$

Much of the coming discussion focuses on paths in the time-expanded network. Such a path p is identified as a sequence of timed nodes that it visits. We let c_p represent the cost of path p when evaluated with the function $c_{ij}(\cdot)$, and \underline{c}_p the cost when evaluated with the function $\underline{c}_{ij}(\cdot)$.

A crucial definition for our method is that of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), which we define as the integer program with objective

$$\min \sum_{((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}} \underline{c}_{ij}(t) x_{((i,t),(j,t'))}$$

subject to the constraints (1), (2) and (3) with \mathcal{A} replaced by $\mathcal{A}_{\mathcal{T}}$ and \mathcal{N} replaced by $\mathcal{N}_{\mathcal{T}}$. We seek to derive a lower bound on TD-TSPTW(\mathcal{D}) by solving instances of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). To do so, we construct TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) so that it is a relaxation of TD-TSPTW(\mathcal{D}). We next list properties of $\mathcal{D}_{\mathcal{T}}$ and $\{\underline{c}\}_{\mathcal{A}_{\mathcal{T}}}$ (the collection of $\underline{c}_{ij}(t)$ for

all $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ that we maintain in order to ensure that $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ is a relaxation of $\text{TD-TSPTW}(\mathcal{D})$.

Property 1. $\forall i \in N$, the nodes $(i, e_i) \in \mathcal{N}_{\mathcal{T}}$, $(i, l_i) \in \mathcal{N}_{\mathcal{T}}$ 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 $(i, j) \in A$ with $t + \tau_{ij}(t) \leq l_j$, there is a travel arc of the form $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$. Furthermore, 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, we note that $\mathcal{A}_{\mathcal{T}}$ only contains arcs $((i, t), (j, t'))$ that satisfy $t + \tau_{ij}(t) \leq l_j$.

Property 4. $\forall a = ((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$, the cost $\underline{c}_{ij}(t)$ is given by (4).

In words, Property 1 requires that $\mathcal{N}_{\mathcal{T}}$ contains nodes for the open and close of each location's time window. Property 2 states that only times within the time window of a location are represented in $\mathcal{N}_{\mathcal{T}}$. Property 3 ensures two conditions: (1) for every node $(i, t) \in \mathcal{N}_{\mathcal{T}}$, there is a copy of every arc (i, j) that emanates from i in D and can be traversed starting at time t , and, (2) the travel time associated with each arc $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ under-estimates the travel time of the arc (i, j) it models dispatching at time t . Property 4 ensures something similar, but with respect to travel costs. As the arc $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ only implies that the vehicle dispatches on arc (i, j) at a time $h \geq t$, we underestimate the cost of doing so by considering the smallest cost of doing so between t and the latest time at which such a dispatch could occur.

We illustrate Properties 3 and 4 in Figures 1a and 1b, where Figure 1a displays the time needed to travel from location i to location j for different departure times, and the number next to each arc indicates the cost associated with such travel (for ease of presentation, we assume that the time points in the time-expanded network occur at integers). Figure 1b then illustrates how, for a given node set $\mathcal{N}_{\mathcal{T}}$, arcs from i to j are created, and the costs, $\underline{c}_{ij}(t)$, associated with those arcs. We note that because arc $((i, 6), (j, 7))$ has cost 1, all copies in $\mathcal{D}_{\mathcal{T}}$ have that cost as well.

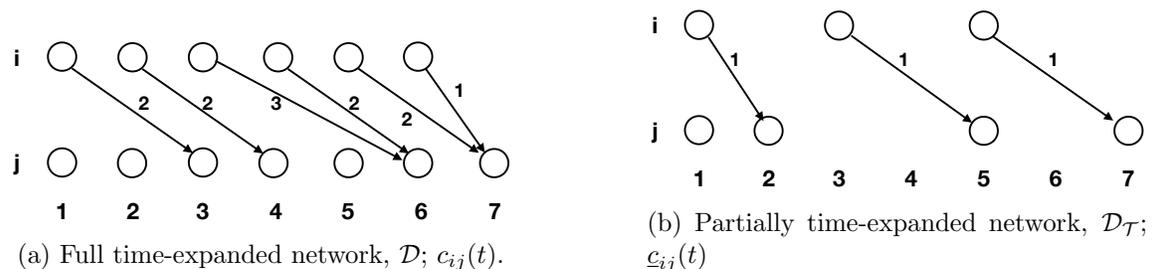


Figure 1: Properties 3, 4 for arc $(i, j) \in A$. Numbers next to arcs indicate either real costs, $c_{ij}(t)$, or under-estimates, $\underline{c}_{ij}(t)$

We next prove that $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ does indeed provide a lower bound when $\mathcal{D}_{\mathcal{T}}$ and $\{\underline{c}\}_{\mathcal{A}_{\mathcal{T}}}$ satisfy the above four properties.

Lemma 1. *If $\mathcal{D}_{\mathcal{T}}$ and $\{\underline{c}\}_{\mathcal{A}_{\mathcal{T}}}$ satisfy Properties 1, 2, 3, and 4, then the objective function value of an optimal solution to $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ is a lower bound on the objective function value of an optimal solution to $\text{TD-TSPTW}(\mathcal{D})$.*

Proof. Proof of Lemma 1 To prove this lemma we will show that each path in the complete time-expanded network, \mathcal{D} , can be mapped to a path in $\mathcal{D}_{\mathcal{T}}$ of equal or lesser cost. Note that p need not represent travel from the depot to all locations and then back to the depot.

To be precise, suppose that $p = ((u_0, t_0), (u_1, t_1), \dots, (u_m, t_m))$ is a path in \mathcal{D} , so $((u_{i-1}, t_{i-1}), (u_i, t_i)) \in \mathcal{A}$ for each $i = 1, \dots, m$. We will first show that there exists a path $p' = ((u_0, t'_0), (u_1, t'_1), \dots, (u_m, t'_m))$ that satisfies the following conditions:

- $(u_k, t'_k) \in \mathcal{N}_{\mathcal{T}}$ with $t'_k \leq t_k$, for all $k = 0, \dots, m$;
- the arc $((u_k, t'_k), (u_{k+1}, t'_{k+1})) \in \mathcal{A}_{\mathcal{T}}$ for all $k = 0, \dots, m-1$; and
- and $\underline{c}_{u_k u_{k+1}}(t'_k) \leq c_{u_k u_{k+1}}(t_k)$ for all $k = 0, \dots, m-1$.

Our proof is by induction on k . First, consider $k = 0$. By Property 1, we have that node $(u_0, e_0) \in \mathcal{N}_{\mathcal{T}}$. Thus, by setting $t'_0 = e_0$, we have that $e_0 = t'_0 \leq t_0$, as \mathcal{D} does not contain nodes (u_0, \bar{t}) with $\bar{t} < e_0$. Next, note that $\underline{c}_{u_0 u_1}(t'_0) \leq c_{u_0 u_1}(t_0)$ by Property 4 and since $t_0 + \tau_{u_0 u_1}(t_0) \leq t_{u_1} \leq l_{u_1}$, so $t_0 \leq l_{u_1} - \tau_{u_0 u_1}(t_0)$.

Next, assume $k = i$ is true. We prove that our conditions hold for $k = i + 1$. By the inductive assumption, there exists a path $((u_0, t'_0), (u_1, t'_1), \dots, (u_i, t'_i)) \in \mathcal{D}_{\mathcal{T}}$, satisfying the above conditions, and hence $(u_i, t'_i) \in \mathcal{N}_{\mathcal{T}}$ with $t'_i \leq t_i$. By Property 3 we have that the arc $((u_i, t'_i), (u_{i+1}, t''_{i+1})) \in \mathcal{A}_{\mathcal{T}}$, with $t''_{i+1} \leq t'_i + \tau_{u_i u_{i+1}}(t'_i) \leq t_i + \tau_{u_i u_{i+1}}(t_i)$, by the FIFO property. Thus $t''_{i+1} \leq t_{i+1}$, and the first and second condition are verified. Similar to our base case, and by Property 4, as $t'_i \leq t_i$, it must be that $\underline{c}_{u_i u_{i+1}}(t'_i) \leq c_{u_i u_{i+1}}(t_i)$. As we have shown that for every travel arc connecting nodes in p there is a travel arc connecting the corresponding nodes in p' that is of equal or lesser cost, we have that the cost of p' is no greater than that of p : $\underline{c}_{p'} \leq c_p$.

As noted, the above refers to any path p in \mathcal{D} , including those from which a feasible solution to $\text{TD-TSPTW}(\mathcal{D})$ can be derived. In that case, a feasible solution to $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ can be derived from the travel arcs connecting nodes in the corresponding path p' in $\mathcal{D}_{\mathcal{T}}$. As we have shown that $\underline{c}_{p'} \leq c_p$ we have that solving the $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ to optimality provides a lower bound on the optimal value of $\text{TD-TSPTW}(\mathcal{D})$. $\square \quad \square$

For a given set of timed nodes $\mathcal{N}_{\mathcal{T}}$, many different arc sets may satisfy Property 3, yielding better or worse lower bounds. We thus introduce Property 5, which states that $\mathcal{D}_{\mathcal{T}}$ contains only arcs that are as long as possible.

Property 5. $\forall a = ((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$, there is no $(j, t'') \in \mathcal{N}_{\mathcal{T}}$ with $t' < t'' \leq t + \tau_{ij}(t)$.

While we do not prove it, it is not difficult to see that Property 5 strengthens the lower bound obtained by solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) as it reduces the feasible region of that integer program.

Having established that TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) provides a lower bound on the optimal value, we now consider conditions under which it can be detected to also provide an upper bound, i.e., to prove optimality. In order to do so, we first make some observations about the structure of solutions to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). Any feasible solution for TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) must induce a path starting and ending at location 0, together with set of disjoint simple cycles in $\mathcal{D}_{\mathcal{T}}$, with each location in N appearing in exactly one cycle or appearing once in the path. We refer to the disjoint cycles as *subtours*. If the solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) does not induce any subtours in $\mathcal{D}_{\mathcal{T}}$ then it must correspond to a path. When the time discretization used to create $\mathcal{D}_{\mathcal{T}}$ is sparse, subtours may well appear in the optimal solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). Even if they don't, and so every location appears exactly once in the path corresponding to the optimal solution, the path may not correspond to a TD-TSPTW solution that is feasible with respect to time windows, since, when the correct travel times are used, the path may violate the time window constraints.

We now discuss conditions under which a solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) proves optimality: if the travel time functions are non-decreasing and if an optimal solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) consists of arcs that accurately model travel time, then it is also an optimal solution to TD-TSPTW(\mathcal{D}). We next prove this result.

Lemma 2. *Consider instances of the TD-TSPTW(\mathcal{D}) having arc cost functions non-decreasing with time, i.e., having $c_{ij}(t) \leq c_{ij}(t')$ whenever $t \leq t'$. Let $\mathcal{D}_{\mathcal{T}}$ and $\{\underline{c}\}_{\mathcal{A}_{\mathcal{T}}}$ satisfy Properties 1–4 and consider a path $p' = ((u_0, t'_0), (u_1, t'_1), \dots, (u_n, t'_n), (u_{n+1}, t'_{n+1}))$ that corresponds to an optimal solution to the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), in which $t'_i = \max\{e_{u_i}, t'_{i-1} + \tau_{u_{i-1}u_i}(t'_{i-1})\}$ for all $i = 1, \dots, n+1$. An optimal solution to TD-TSPTW(\mathcal{D}) can be derived from p' .*

Proof. Proof of Lemma 2 By the supposition that $t'_{i+1} = \max\{e_{u_{i+1}}, t'_i + \tau_{u_i u_{i+1}}(t'_i)\}$ for all $i = 0, \dots, n$, we have that the corresponding travel arcs, $((u_i, t'_i), (u_{i+1}, t'_{i+1}))$, are in \mathcal{A} , and hence p' also defines a feasible path in \mathcal{D} . As shown in Lemma 1, the cost of this path, $\underline{c}_{p'}$, is a lower bound on the cost, c_{p^*} , of the path p^* corresponding to an optimal solution of TD-TSPTW(\mathcal{D}). As travel costs are non-decreasing in t , we have that $\underline{c}_{ij}(t) = c_{ij}(t)$, and, $\underline{c}_{p'} = c_{p'}$. As such, we have a path p' , from which a feasible solution to TD-TSPTW(\mathcal{D}) can be derived that has cost equal to the value of a lower bound on the objective function value of an optimal solution to TD-TSPTW(\mathcal{D}). Hence, the solution derived from p' is optimal. \square \square

For cases where arc costs are non-decreasing, such as the makespan objective, Lemma 2 suggests a sufficient condition for when a path, p' , corresponding to an optimal solution of the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) can be used to derive an optimal solution of the TD-TSPTW. Namely, that the arcs connecting nodes in the path accurately model travel times.

In the case of the duration objective, the arc costs only fail to be non-decreasing for arcs leaving the depot, location 0; all other arcs have non-decreasing cost functions. Thus, by adding one more condition to $\mathcal{D}_{\mathcal{T}}$, that $(0, t) \in \mathcal{N}_{\mathcal{T}}$ for all t at which a feasible tour may start, the proof of Lemma 2 can easily be adapted to show that the same result holds for the duration objective.

Having detailed the theoretical properties of an appropriately-constructed partially time-expanded network, we next present an algorithm for solving the TD-TSPTW(\mathcal{D}) that relies on such networks.

5 DDD-TD-TSPTW Algorithm

The algorithm we use to solve the TD-TSPTW(\mathcal{D}) iteratively refines a partially time-expanded network, $\mathcal{D}_{\mathcal{T}}$, until it can be used to produce a provably optimal solution to the TD-TSPTW(\mathcal{D}). In this section, we describe how this algorithm, which we call DDD-TD-TSPTW, works at a high-level. We organize our discussion based upon the flow chart for the algorithm that is presented in Figure 2. We next discuss the individual steps in greater detail.

5.1 Preprocessing

The algorithm employs preprocessing techniques derived from those seen in the literature (Desrochers et al. 1992, Dash et al. 2012) on routing problems with time windows to both prune arcs from A and tighten time windows. Montero et al. (2017) also adapted these techniques to the TD-TSPTW. We present them here for the sake of completeness. These techniques are based on computing the values $\Delta_{ij}(t)$, which denote the earliest time the vehicle can depart location j , assuming it had previously departed from location i at time t . We note that if travel times satisfy the triangle inequality, then we can set $\Delta_{ij}(t) = \max\{e_j, t + \tau_{ij}(t)\}$. If they do not, then we can compute these Δ values by solving shortest path problems (note the FIFO property is important for the validity of those Δ values).

We first present rules that use these Δ values to update time windows at locations. The first two rules we present are applicable to all variants of the TSPTW and TD-TSPTW (time dependency is encoded in the computation of $\Delta_{ij}(t)$). The first rule recognizes that the vehicle cannot depart from k earlier than it can arrive there from another location, and sets $e_k = \max\{e_k, \min\{l_k, \min_{i \in \mathcal{N} \setminus k} \Delta_{ik}(e_i)\}\}$. The second rule recognizes that the vehicle can only depart a location at times that enable it to reach another location during its time window, and sets $l_k = \min\{l_k, \max\{e_k, \max_{i \in \mathcal{N} \setminus k} \max_t\{\Delta_{ki}(t) \leq l_i\}\}\}$.

The next two rules are only valid for variants of the TD-TSPTW in which waiting at a location (other than for its time window to open) is not beneficial. The first rule recognizes that it is not necessary for the vehicle to depart from location k to another location only

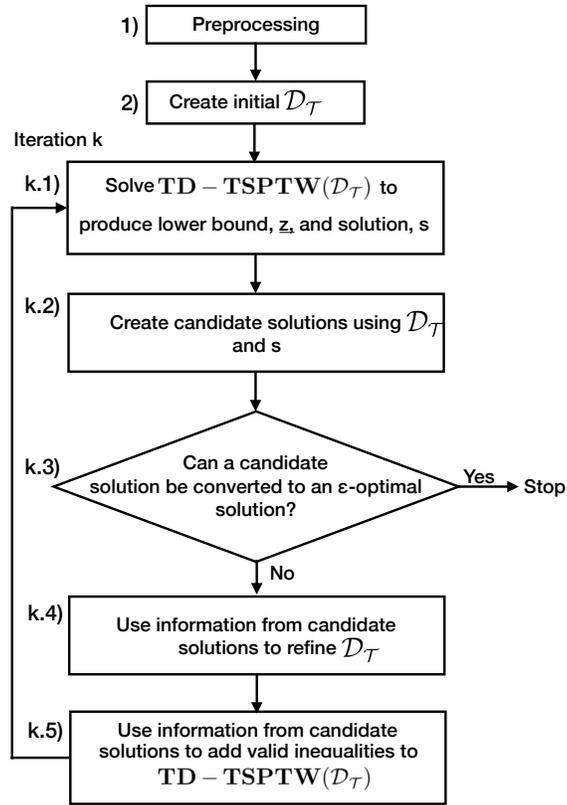


Figure 2: Flow chart of DDD-TD-TSPTW

to wait for that location's time window to open, and sets

$$e_k = \max\{e_k, \min\{l_k, \min_{i \in \mathcal{N} \setminus k} \max_t \{\Delta_{ki}(t) \leq e_i\}\}\}.$$

Similarly, the second rule recognizes that the vehicle need not depart from location k later than the latest time at which it can arrive there, and sets

$$l_k = \min\{l_k, \max\{e_k, \max_{i \in \mathcal{N} \setminus k : \Delta_{ik}(e_i) \leq l_k} \Delta_{ik}(l_i)\}\}.$$

Next, we analyze the time windows at pairs of locations in order to prune arcs from A . For example, if there is no time at which the vehicle can depart from i for j and arrive during j 's time window, then we denote the sequence $i \rightarrow j$ as infeasible. Due to the FIFO property, this is easily checked by testing if $\Delta_{ij}(e_i) > \ell_j$. We can then conclude that j must be visited before i , which we denote by $j \prec i$. Based on such a precedence relationship, we eliminate (i, j) from A . Similarly, we also analyze the relationships between time windows at three locations. If both the sequence of locations $i \rightarrow j \rightarrow k$ and $k \rightarrow i \rightarrow j$ are infeasible, then we can remove arc (i, j) from A . If the sequence of locations $i \rightarrow k \rightarrow j$ is also infeasible, then we can conclude that $j \prec i$. Finally, we note that precedences are transitive, i.e. if $i \prec j$ and $j \prec k$ then we can conclude that $i \prec k$. We present the preprocessing scheme in Algorithm 1.

Algorithm 1 PREPROCESSING

Require: Graph $G = (N, A)$ and time windows $[e_i, l_i], \forall i \in N$

- 1: Set $P = \emptyset$ { P is the set of known precedence relationships.}
- 2: **while** changes found **do**
- 3: Update time windows per rules discussed above
- 4: **for all** $(i, j) \in A$ **do**
- 5: Check if $i \rightarrow j$ is infeasible and add $j \prec i$ to P if so.
- 6: **end for**
- 7: **for all** $(i, j, k) \in N$ **do**
- 8: Check if $i \rightarrow j \rightarrow k$, $k \rightarrow i \rightarrow j$, and $i \rightarrow k \rightarrow j$, are all infeasible and add $j \prec i$ to P if so.
- 9: Check if $i \rightarrow j \rightarrow k$, and $k \rightarrow i \rightarrow j$ are both infeasible and if so, remove arc (i, j) from A
- 10: **end for**
- 11: Add transitive closure of precedences to P so that it contains all precedence pairs (if $i \prec j$ and $j \prec k$ then $i \prec k$)
- 12: Remove all arc (j, i) from A if arc $(i, j) \in P$
- 13: Remove all arcs (i, k) from A if there is j such that $(i, j), (j, k) \in P$
- 14: Remove all arcs $(i, 0)$ and $(0, j)$ from A if $(i, j) \in P$
- 15: **end while**

5.2 Creating Initial $\mathcal{D}_{\mathcal{T}}$

After preprocessing, the algorithm creates the initial partially time-expanded network, $\mathcal{D}_{\mathcal{T}} = (\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}})$. To do so, it creates the node set $\mathcal{N}_{\mathcal{T}} = \cup_{i \in N} \{e_i, l_i\}$, ensuring that $\mathcal{D}_{\mathcal{T}}$ satisfies Properties 1 and 2. Then, having created those nodes, the algorithm creates the

arc set $\mathcal{A}_{\mathcal{T}}$ to ensure that $\mathcal{D}_{\mathcal{T}}$ satisfies Property 3 (all necessary arcs are created and have travel times that may under, but never over, estimate the actual travel times), 4 (arc costs may under, but never over, estimate actual arc costs), and 5 (each created arc is as long as possible). Note that arcs satisfying these properties can be created because of the set of nodes included $\mathcal{N}_{\mathcal{T}}$.

5.3 Solve TD-TSPTW($\mathcal{D}_{\mathcal{T}}$)

As shown in Lemma 1, solving the integer program TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) for a partially time-expanded network $\mathcal{D}_{\mathcal{T}}$ that satisfies Properties 1–4 will yield a lower bound on the objective function value of an optimal solution of TD-TSPTW(\mathcal{D}). We note that TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) need not be solved to optimality to produce a lower bound on TD-TSPTW(\mathcal{D}). Instead, any lower bound on the optimal value of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) is in turn a lower bound on the optimal value of TD-TSPTW(\mathcal{D}). As a result, the solution of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) can be terminated early and the algorithm will still be able to produce a valid lower bound.

As discussed earlier, an integer solution of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) (whether optimal or not) may induce subtours and, even if it doesn't, the path it induces may violate time windows when the correct travel times are used. If neither of these cases occurs, then the solution must induce a feasible solution to the TD-TSPTW, which can be evaluated using the costs for the correct travel times, and so provide an upper bound on the value of the TD-TSPTW.

Next, we describe an enhancement to the algorithm that seeks feasible solutions to the TD-TSPTW more aggressively, in order to improve the upper bound earlier in the DDD algorithm.

5.4 Creating candidate solutions

To find a high-quality primal solution to TD-TSPTW(\mathcal{D}), at each iteration the DDD algorithm generates two “primal” partially time-expanded network, $\bar{\mathcal{D}}_{\mathcal{T}}^i = (\mathcal{N}_{\mathcal{T}}, \bar{\mathcal{A}}_{\mathcal{T}}^i)$ for $i = 1, 2$, and then solves TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^i$) for each i . The two primal networks have the same node set, $\mathcal{N}_{\mathcal{T}}$, (which is the same as the lower bound network), but differ in their arc set, $\bar{\mathcal{A}}_{\mathcal{T}}^i$, for $i = 1, 2$, respectively.

The first arc set, $\bar{\mathcal{A}}_{\mathcal{T}}^1$, consists of arcs $((i, t), (j, t'))$ that model travel times that are at least as long as the correct time (i.e., that satisfy $t' \geq t + \tau_{ij}(t)$). Specifically, for each $(i, t) \in \mathcal{N}_{\mathcal{T}}$ and each arc $(i, j) \in A$, the smallest t'' such that $(j, t'') \in \mathcal{N}_{\mathcal{T}}$ and $t'' \geq t + \tau_{ij}(t)$ is found and the arc $((i, t), (j, t''))$ is added to $\bar{\mathcal{A}}_{\mathcal{T}}^1$. As a result, any feasible solution to TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$) may be used to generate a feasible solution to TD-TSPTW(\mathcal{D}). Unlike the first, the second arc set, $\bar{\mathcal{A}}_{\mathcal{T}}^2$, includes arcs that are too short. However, it does not include arcs $((i, t), (j, t'))$ that model non-positive travel times (i.e., with $t' \leq t$). As a result, feasible solutions to TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$) cannot include subtours. Specifically, $\bar{\mathcal{A}}_{\mathcal{T}}^2$ includes all arcs $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ such that $t' > t$. Furthermore, for each arc in $\mathcal{A}_{\mathcal{T}}$ with $t' \leq t$, the smallest value $t'' > t$ such that $(j, t'') \in \mathcal{N}_{\mathcal{T}}$ is found and the arc $((i, t), (j, t''))$

is added to $\bar{\mathcal{A}}_{\mathcal{T}}^2$. We note that in both cases, the time point t'' must exist. This is because we maintain throughout the algorithm both that all arcs $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ are such that $t + \tau_{ij}(t) \leq l_j$ and that $(j, l_j) \in \mathcal{N}_{\mathcal{T}}$ for all $j \in N$.

We refer to any integer solution to the TD-TSPTW over a partially time-expanded model as a *candidate solution* to the TD-TSPTW.

5.5 Checking whether a candidate solution can be converted

DDD-TD-TSPTW attempts to convert each candidate solution to a solution to the TD-TSPTW(\mathcal{D}). As noted, candidates derived by solving TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$), are guaranteed to yield a feasible solution to TD-TSPTW(\mathcal{D}). However, candidates derived either by solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) or TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$) are not. As discussed earlier, there are two reasons why a candidate derived by solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) cannot be converted to a feasible solution to the TD-TSPTW(\mathcal{D}). Both result from how DDD-TD-TSPTW creates and refines partially time-expanded networks, $\mathcal{D}_{\mathcal{T}}$.

First, the presence of arcs $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ with $t' \leq t$ allow solutions to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) to induce subtours. Thus, when evaluating whether the candidate solution can be converted, DDD-TD-TSPTW first determines whether there is a subtour. If there is, then due to the flow balance constraints, (Constraints (2)), there must be at least two such subtours.

Second, $\mathcal{A}_{\mathcal{T}}$ can contain arcs $((i, t), (j, t'))$ with $t' < t + \tau_{ij}(t)$. These arcs enable the solution to induce a sequence of locations which, under the actual travel times, violate a time window (i.e. the earliest the vehicle could arrive at some location, given that sequence, is at a time after the end of its time window). In fact, the candidate may contain multiple such infeasible sequences. While a solution to TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$) will not contain subtours, it may yield infeasible location sequences.

Next, consider a solution that induces neither a subtour nor an infeasible location sequence. In this case, the solution corresponds to some sequence of nodes, $((v_0, t_0), \dots, (v_n, t_n), (v_{n+1}, t_{n+1}))$, each in $\mathcal{N}_{\mathcal{T}}$, which starts at the depot and in which every location appears exactly once. The solution thus prescribes a Hamiltonian path $(v_0, v_1, \dots, v_n, v_{n+1})$. In addition, because the path is feasible, there are dispatch times t'_0, \dots, t'_n such that $e_{v_i} \leq t'_i \leq l_{v_i}$, $t'_0 = e_{v_0}$, and $t'_{i+1} = \max\{e_{v_{i+1}}, t'_i + \tau_{v_i v_{i+1}}(t'_i)\}$. With these, $((v_0, t'_0), \dots, (v_i, t'_i), \dots, (v_{n+1}, t'_{n+1}))$ prescribes a solution to the TD-TSPTW(\mathcal{D}) with makespan objective value $t'_{n+1} = t'_n + \tau_{n0}(t'_n)$, in which the vehicle departs each location within its time window and as early as possible.

Whenever a candidate solution can be converted to a solution to the TD-TSPTW(\mathcal{D}), we compare the objective function value of the resulting solution to the lower bound to determine whether it is within the given optimality tolerance. Finally, we note that when solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$), or TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$) we can collect all integer solutions found by the solver (including those that are not optimal) to test for conversion to a feasible solution to the TD-TSPTW(\mathcal{D}).

5.6 Refining $\mathcal{D}_{\mathcal{T}}$

As noted above, it is the presence of arcs $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$ with $t' < t + \tau_{ij}(t)$ that cause TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) to yield an optimal solution that cannot be converted to a feasible solution to TD-TSPTW(\mathcal{D}). Thus, one way to remove the presence of such solutions from the feasible region of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) is to lengthen such “too short” arcs. In doing so, we strengthen the relaxation, TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). Of course, to ensure Lemma 1 remains valid, and the algorithm produces a valid lower bound on TD-TSPTW(\mathcal{D}) at each iteration, we must lengthen these arcs in such a way that $\mathcal{D}_{\mathcal{T}}$ retains Properties 1, 2, 3, and 4.

We lengthen arcs based on their presence in a solution $s = ((v_0, t_0), \dots, (v_{n+1}, t_{n+1}))$, in which $v_0 = v_{n+1} = 0$ and present a detailed description of how we do so in Algorithm 2. Algorithm 2 operates on a sequence of nodes visited by a path in $\mathcal{D}_{\mathcal{T}}$. Thus, when a solution does not contain a subtour, the path it prescribes can be the input to the algorithm. However, when a solution does contain a subtour, Algorithm 2 is called for a path derived from each subtour.

Algorithm 2 examines each arc $((u_i, t_i), (u_{i+1}, t_{i+1}))$ prescribed by the solution s to see if it is both too short (e.g. $t_{i+1} < t_i + \tau_{u_i u_{i+1}}(t_i)$) and can be lengthened while ensuring the vehicle arrives within the time window at the location at the head of the arc (e.g. $e_{u_{i+1}} \leq t_i + \tau_{u_i u_{i+1}}(t_i) \leq l_{u_{i+1}}$). To lengthen this arc, Algorithm 2 calls the Add-Node procedure detailed in Algorithm 3. This procedure updates any arcs that can be lengthened given the new node as well as creates new arcs, all while ensuring that $\mathcal{D}_{\mathcal{T}}$ continues to satisfy the necessary properties.

Algorithm 2 LENGTHEN-ARCS-PATH(p')

Require: Path $p' = ((u_0, t_0), \dots, (u_k, t_k), (u_{k+1}, t_{k+1}), \dots, (u_m, t_m)), (u_i, t_i) \in \mathcal{N}_{\mathcal{T}}$

- 1: **for** $i \leftarrow 0$ to $m - 1$ **do**
- 2: $t \leftarrow t_i$ {Consider departure from u_i at t_i }
- 3: **for** $j \leftarrow i + 1$ to m **do**
- 4: $t' \leftarrow \max(e_{u_j}, t + \tau_{u_{j-1} u_j}(t))$
- 5: **if** $t' > l_j$ **then**
- 6: Break
- 7: **end if**
- 8: Call Add-Node((u_j, t'))
- 9: $t \leftarrow t'$
- 10: **end for**
- 11: **end for**

Algorithm 2 does more than just lengthen arcs that are in the current solution and “too short.” In particular, after adding a node (u_j, t') , Algorithm 2 adds arcs and nodes to enable the vehicle to follow the path (in time and space) prescribed by the sequence of locations $(u_j, u_{j+1}, \dots, u_m)$, starting at u_j at time t' . We illustrate this in Figures 3a and 3b. Here, Algorithm 2 is called with the path $p = ((u_0, 1), (u_1, 1), (u_2, 2))$ where $\tau_{u_0 u_1}(t) = 1, \tau_{u_1 u_2}(t) = 2$ for all t . As the arc $((u_0, 1), (u_1, 1))$ prescribed by the solution is too short, Algorithm 2 creates node $(u_1, 2)$ so that the arc may be lengthened. However, because the original path continues on to location u_2 , the procedure also creates the node

Algorithm 3 ADD-NODE(j, t')

Require: Node (j, t')

```

1: if  $(j, t') \in \mathcal{N}_{\mathcal{T}}$  then
2:   Return
3: end if
4:  $\mathcal{N}_{\mathcal{T}} \leftarrow \mathcal{N}_{\mathcal{T}} \cup (j, t')$ 
5: for all  $k \in N$  such that  $t' + \tau_{jk}(t') \leq l_k$  do
6:    $t = \operatorname{argmax}\{\bar{t} \mid (k, \bar{t}) \in \mathcal{N}_{\mathcal{T}}, \bar{t} \leq t' + \tau_{jk}(t')\}$ .
7:   Add  $((j, t')(k, t))$  to  $\mathcal{A}_{\mathcal{T}}$  with cost  $\underline{c}_{jk}(t')$ 
8: end for
9: for all  $a \in \mathcal{A}_{\mathcal{T}}$  such that  $a = ((i, t), (j, t'')), t'' < t'$  and  $t + \tau_{ij}(t) \geq t'$  do
10:   Delete arc  $((i, t), (j, t''))$  from  $\mathcal{A}_{\mathcal{T}}$ 
11:   Add arc  $((i, t), (j, t'))$  to  $\mathcal{A}_{\mathcal{T}}$  with cost  $\underline{c}_{ij}(t)$ .
12: end for

```

$(u_2, 4)$ and arc $((u_1, 2), (u_2, 4))$ so the vehicle can visit the same sequence of locations, now on arcs that model true travel times.

In addition, Algorithm 2 considers sub-paths that begin at intermediate nodes in the path that is being examined. Continuing with our example, Algorithm 2 will consider the sub-path that departs from u_1 at time 1 and consists of the arc $((u_1, 1), (u_2, 2))$. As this arc is too short, the algorithm will create node $(u_2, 3)$ and replace arc $((u_1, 1), (u_2, 2))$ with one that models the actual travel time, $((u_1, 1), (u_2, 3))$. We note that while these “opportunistic” additions of arcs and nodes are not necessary for correctness of the algorithm, they have been computationally beneficial.

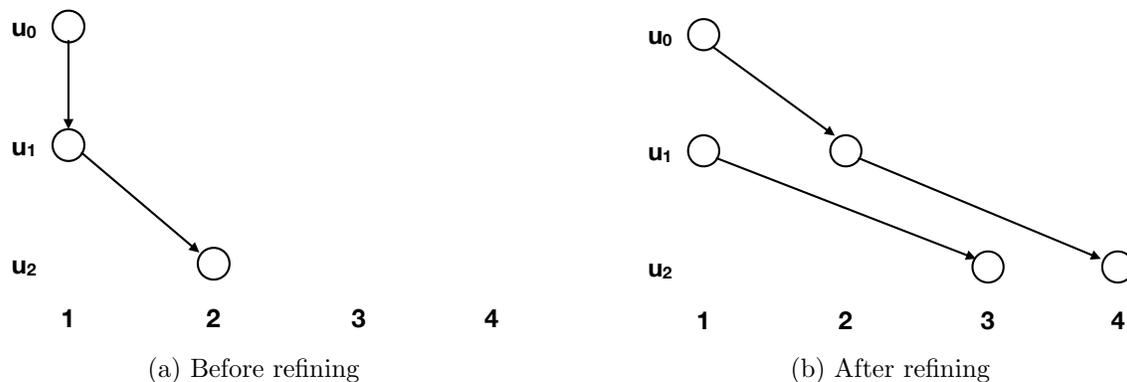


Figure 3: Refining $\mathcal{D}_{\mathcal{T}}, \tau_{u_0 u_1}(\cdot) = 1, \tau_{u_1 u_2}(\cdot) = 2$

5.7 Adding valid inequalities to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$)

As noted above, there are two reasons a candidate solution cannot be converted to an optimal solution to the TD-TSPTW(\mathcal{D}). While refining the network $\mathcal{D}_{\mathcal{T}}$ will remove a given solution from the feasible region of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), we can also render sets of

solutions with similar attributes infeasible with the use of valid inequalities. Specifically, recall that DDD-TD-TSPTW may determine that the solution, s , to the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) contains sub-tours of the form, (v_i, \dots, v_k) , in which $v_j \neq v_{j+1}, j = i, \dots, k-1$ and $v_i = v_k$. In this case, the algorithm can use valid inequalities to prevent the subtour from appearing in solutions to future instances of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). Specifically, for each subtour, let M represent the locations, (v_i, \dots, v_k) , visited in the subtour. Then DDD-TD-TSPTW adds valid inequalities of the form

$$\sum_{(i,j) \in \mathcal{A} \cap (M \times M)} \sum_{t,t':((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}} x_{((i,t),(j,t'))} \leq |M| - 1. \quad (5)$$

to the instance of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) solved in all subsequent iterations. Note that M consists of locations in N , rather than in $\mathcal{N}_{\mathcal{T}}$, and thus the inequality can be applied to the IP for any network, $\mathcal{D}_{\mathcal{T}}$: Constraint (5) does not only rule out a subtour consisting of locations in M that begins at a specific time t , it rules out all timed copies of such a subtour.

Similarly, as noted above, DDD-TD-TSPTW may determine that the solution prescribes a sequence of locations, (v_i, \dots, v_k) , that cannot appear in a feasible solution to TD-TSPTW. As for subtours, DDD-TD-TSPTW uses valid inequalities to prevent this sequence of locations from appearing in solutions to future instances of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) solved by the algorithm. Specifically, let M represent such a sequence of locations, (v_i, \dots, v_k) . The path-based valid inequalities of the form

$$\sum_{i=0}^{k-1} \sum_{t,t':((v_i,t),(v_{i+1},t')) \in \mathcal{A}_{\mathcal{T}}} x_{((v_i,t),(v_{i+1},t'))} \leq |M| - 2 \quad (6)$$

are added to the IP. Like the subtour inequalities above, Constraints (6) are determined by locations in N , and thus can be applied for any network $\mathcal{D}_{\mathcal{T}}$. Similarly, Constraints (6) forbid all timed copies of the sequence (v_i, \dots, v_k) .

We conclude that DDD-TD-TSPTW will converge to an optimal solution of TD-TSPTW(\mathcal{D}) by first noting that until the algorithm can produce a solution that is near-optimal, at each iteration it will lengthen at least one arc in $\mathcal{A}_{\mathcal{T}}$ to its correct length. As there are a finite number of arcs in \mathcal{A} , and each arc is lengthened exactly once, $\mathcal{D}_{\mathcal{T}}$ must, ultimately, be equivalent to \mathcal{D} . At that point, solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) will produce an optimal solution to TD-TSPTW(\mathcal{D}).

We note that we can further prune arcs from the network whenever a candidate solution can be converted to a feasible solution that has a lower objective function value than the best found so far. Specifically, when a new feasible solution with makespan objective function value z is found we perform the following steps: (1) we remove arcs $((i, t), (j, t'))$ with $t + \tau_{ij}(t) > z$, and, (2) we update $l_0 = z - 1$ and then repeat the preprocessing algorithm described earlier. We note that preprocessing may result in an instance of the

TD-TSPTW(\mathcal{D}_T) that is infeasible, in which case we can conclude that the best feasible solution found so far is optimal.

6 Computational Results

In this section, we assess the performance of the proposed DDD algorithm when solving TD-TSPTW with a makespan objective. We benchmark its performance against the approach proposed in Montero et al. (2017), the best-performing method in the literature. DDD-TD-TSPTW is implemented in C++, and all experiments run on a workstation with a Intel(R) Xeon (R) CPU E5-4610 v2 2.30GHz processor running the Ubuntu Linux 14.04.3 Operating System. The algorithm repeatedly solves integer programs, and the implementation used Gurobi 6.5.0 to do so. All parameter values for Gurobi were left at their default values, other than the Threads parameter, which was set to limit Gurobi to one thread of execution. The stopping conditions for the implementation of the algorithm were a one hour time limit and a provable optimality gap of $\epsilon = 10^{-4}$.

6.1 Instances

The algorithm is tested on two sets of instances, which were proposed in Cordeau et al. (2014) for the TD-TSP, extended to the TD-TSPTW in Arigliano et al. (2015), and used to test the methods proposed in Arigliano et al. (2015) and Montero et al. (2017). We note that Arigliano et al. (2015) added time windows in a way that guaranteed the existence of a feasible solution. Instances in both sets vary in the number of locations the vehicle must visit ($n = 15, 20, 30, 40$) as well as the value of a parameter Δ which represents the level of congestion, and, thus, travel times. Smaller values of Δ reflect greater congestion and longer travel times. Instances in both sets also vary depending on two traffic patterns. We refer the reader to Cordeau et al. (2014) for a detailed description of how the instances were created and their characteristics. Finally, we note that to facilitate the use of these instances when solving the *Duration* objective, we multiplied the problem data by 100 and then rounded them to yield integral data.

Recall that the travel time function proposed in Ichoua et al. (2003) assumes that travel speeds are constant within a time interval, but can change from one interval to the next. The primary difference between the two sets of instances is the number of such breakpoints. The first set, which we refer to as *Set 1*, has only three time intervals and consists of 952 instances. (It is reported that there are 960 instances, but, as experienced also by Montero et al. (2017), eight are missing.) The second set, which we refer to as *w100*, as in Montero et al. (2017), has 73 time intervals. This set does have 960 instances. While Montero et al. (2017) evaluated the performance of their algorithm on a subset of 216 of these instances, we assess the performance of our method on all of them. We note that Montero et al. (2017) observed that the number of “breakpoints” that segment time into intervals was a reliable indicator of how difficult it was to solve an instance with their algorithm.

Throughout the next discussion, we benchmark against the results reported in Montero et al. (2017) for their TTBF-CB method, and refer to the results as such. We denote the number of locations in an instance by “Nds”, the number of instances in a class by “Inst”, the number of instances solved by “Slv”, and the time needed to solve those instances by “Tme.” All reported times are in seconds, and these times are averages of the runtime required for instances that were solved.

6.2 Benchmark results

We first compare the performance of DDD-TD-TSPTW and that of TTBF-CB on both sets of instances. The results can be found in Table 1. We see that DDD-TD-TSPTW solved all instances of Set 1, where as TTBF-CB struggled with a few of them, and that DDD-TD-TSPTW is faster. Furthermore, DDD-TD-TSPTW is able to solve all *w100* instances, and in far less time. We note that the solution times reported for DDD-TD-TSPTW on *w100* instances includes times for instances that TTBF-CB did not solve.

Instance set	Traffic pattern	TTBF-CB			DDD-TD-TSPTW		
		Inst	Slv	Tme	Inst	Slv	Tme
Set 1	A	478	470	31.99	478	478	12.91
	B	474	470	20.44	474	474	8.79
Set <i>w100</i>	A	108	107	100.13	480	480	1.31
	B	108	107	93.10	480	480	1.14

Table 1: Performance on Arigliano et al. (2015) instances

We also note that the time TTBF-CB requires to solve instances from Set *w100* is often three to four times greater, on average, than when solving instances from Set 1. DDD-TD-TSPTW, however, does not exhibit this behavior; the time it requires to solve an instance from *w100* is less than, on average, the time required to solve an instance from Set 1. We attribute this to DDD-TD-TSPTW operating on a time-expanded network, as the traveling speed between locations changing based on the dispatch time only requires updating the appropriate arc in that time-expanded network. Finally, we note that while we adjusted the problem data to get integral values, the tours produced by DDD-TD-TSPTW yielded the same objective function values as reported in Montero et al. (2017) when evaluated with the original problem data.

We next focus on how the solve time for each method changes as the number of locations in an instance increases. To do so, we illustrate averages in Figure 4, by number of locations, of the time each method needs to solve instances from each set. We see that the growth rate in solve times is much smaller for DDD-TD-TSPTW than for TTBF-CB.

TTBF-CB is a branch-and-cut method, and there is typically a strong correlation between the root node gap of the search tree and the solve time, with smaller gaps leading to faster solve times and fewer nodes explored (typically). The results in Montero et al. (2017) indicate that this root node gap is much larger for instances from *w100* than those

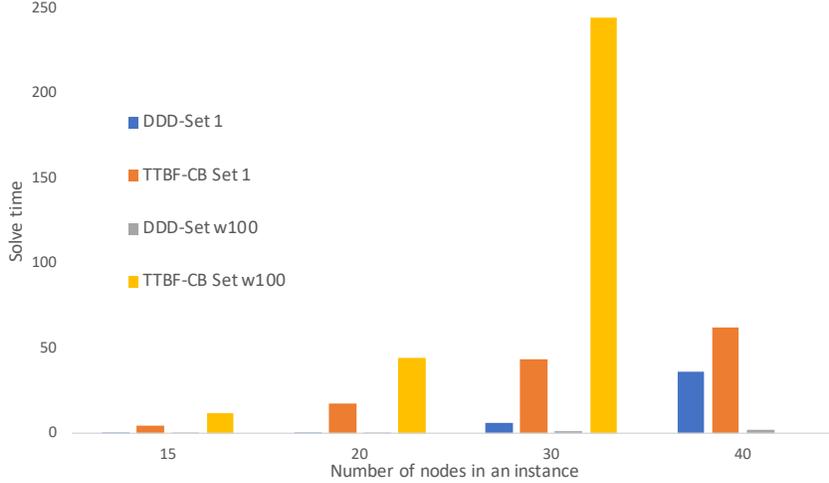


Figure 4: Solve times by number of locations

from Set 1. This suggests that the greater the number of time intervals in an instance, the weaker its linear programming relaxation and resulting dual bound, and the less likely TTBF-CB will solve the instance.

DDD does not have a root gap in the sense that TTBF-CB does. However, it is iterative in nature, and at every iteration produces a dual bound and often a primal solution. As a result, we can calculate a gap between the dual bound produced in the first and last iterations of DDD-TD-TSPTW execution. Specifically, if LB_1 represents the first dual bound produced, and LB^* the last, then we calculate “Dual gap” as $((LB^* - LB_1)/LB^*)$. Similarly, we can calculate a gap in objective function value between the first and last primal solutions found. Specifically, we calculate “Primal gap” as $((UB^* - UB_1)/UB_1)$, where UB_1 is the objective function value of the first primal solution found and UB^* is the objective function value of the last primal solution found. We illustrate these gaps, by locations in an instance and instance set in Figure 5. We observe that these gaps are not greatly impacted by the number of locations in an instance. If anything, the dual gap is decreasing. Interestingly, unlike the root gap produced by TTBF-CB, the dual gap is smaller for instances from the w100 set.

Another instance characteristic that is often considered to be a predictor of how challenging an instance will be to solve is the width of the time windows at the locations to be visited. For each instance in Set 1, we calculated the average time window width, $w = \frac{\sum_{i=0}^n (l_i - e_i)}{n}$, before preprocessing. We then computed the correlation coefficient between this average for an instance and the time DDD-TD-TSPTW needed to solve that instance over all instances in Set 1. The correlation coefficient is 0.06, suggesting that the time DDD-TD-TSPTW needs to solve an instance does not depend on time window width. We calculated a similar correlation coefficient, only with respect to the standard deviation

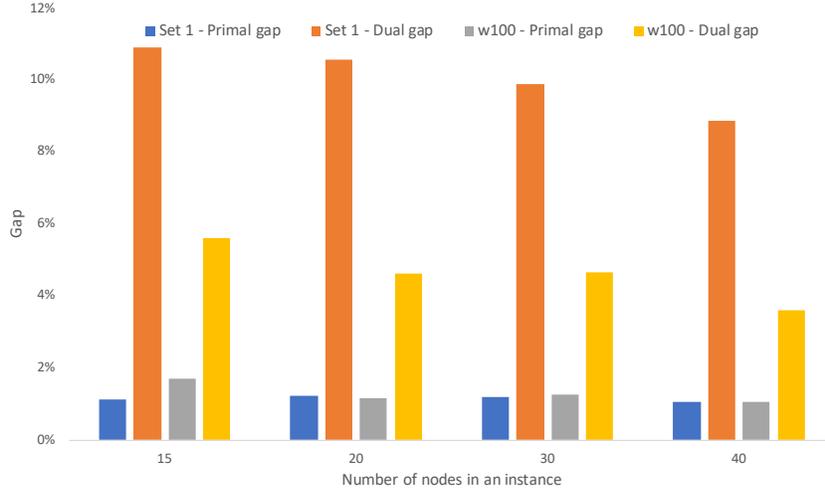


Figure 5: Gaps by number of locations

of time window widths and again saw little correlation ($\rho = -0.05$). As all locations have time windows of the same width in Set *w100*, we do not perform such an analysis for this set of instances.

Finally, we study how the performance of DDD-TD-TSPTW depends on the level of congestion (as represented by the parameter Δ). To do so, we report in Table 2 the average time DDD-TD-TSPTW needed to converge to the optimal solution for instances that model the same level of congestion. We see that instances with a congestion level of 90 have the largest solve times, on average. However, there does not appear to be a clear relationship between congestion level and DDD-TD-TSPTW solve times.

Δ	70	80	90	98
Set 1	7.02	9.77	17.04	9.59
Set w100	1.11	1.2	1.3	1.28

Table 2: DDD-TD-TSPTW solve time by level of congestion

We conclude from these results that DDD-TD-TSPTW is superior to TTBF-CB at solving instances with a makespan objective, and is robust with respect to instance parameters. We next turn to a more detailed analysis of how DDD-TD-TSPTW performs.

6.3 Detailed analysis of DDD-TD-TSPTW performance

In this section, we seek to analyze which components of DDD-TD-TSPTW contribute to its performance. Fundamentally, we focus on three components: (1) the “Base” algorithm that

iteratively solves instances of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) and refines $\mathcal{D}_{\mathcal{T}}$ until a provably ϵ -optimal solution can be found, (2) the valid inequalities (VI) that are added, and, (3) the “Primal” IPs, TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^i$) for $i = 1$ and $i = 2$, that are solved in order to speed up the discovery of high-quality primal solutions. To understand which components are impactful, we executed DD-TD-TSPTW in four configurations:

- “Base:” DD-TD-TSPTW is executed, but valid inequalities are not generated or added, and Primal IPs, TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^i$), are not solved.
- “Base + VI:” Above, but valid inequalities are generated and added
- “Base + Primal:” DDD-TD-TSPTW is executed and Primal IPs are solved.
- “Base + VI + Primal:” The full algorithm described above.

Then, for each set of instances and configuration we determined statistics like those discussed above. We also calculated the average number of iterations (Iter) the configuration needed to converge to a provably ϵ - optimal solution (when it was able to do so). We report the results in Table 3.

Method	Set 1 - 952 instances			Set <i>w100</i> - 960 instances		
	Slv	Tme	Iter	Slv	Tme	Iter
Base	950	101.19	17.46	960	3.94	9.97
Base + VI	948	110.42	15.99	960	4.09	9.65
Base + Primal	952	24.68	4.14	960	3.18	3.28
Base + VI + Primal	952	10.94	4.62	960	1.22	4.11

Table 3: Performance of different method configurations

We see that the base algorithm is quite effective, particularly on the instances from Set *w100*. However, it is quite a bit slower on instances from Set 1. At the same time, the results indicate that the valid inequalities that are added do not significantly impact the performance of DDD-TD-TSPTW. This is likely due to the fact that few inequalities are added. Specifically, when solving instances from Set 1, DDD-TD-TSPTW added, on average, only 5.08 sub-tour-based inequalities and 14.93 infeasible sequence inequalities. Fewer of each were added when DDD-TD-TSPTW solved instances from Set *w100*; only 2.34 sub-tour-based and 10.97 infeasible sequence inequalities. On the other hand, the Primal IPs dramatically reduce the time and number of iterations DDD-TD-TSPTW needs to solve instances, particularly those from Set 1. That said, we see that the addition of the Primal IPs alone does not provide the best performance; both the valid inequalities and Primal IPs are necessary to achieve the fastest run-times.

Similarly, we report in Table 4, for each set of instances and configuration, the average number of nodes in the partially time-expanded network ($|\mathcal{N}_{\mathcal{T}}|$) when DDD-TD-TSPTW

terminates. Given that nodes are added at each iteration (when arcs are lengthened), it is not surprising to see that the Primal IPs enabled DDD-TD-TSPTW to converge with fewer nodes.

Nodes	Set 1				Set <i>w100</i>			
	Base	Base + VI	Base + Primal	Base + VI + Primal	Base	Base + VI	Base + Primal	Base + VI + Primal
15	62.40	56.73	66.39	37.32	89.34	91.17	99.80	83.56
20	133.47	127.03	125.52	72.96	146.50	144.61	158.58	124.30
30	407.33	384.03	312.85	201.27	300.50	298.70	339.35	249.88
40	998.76	970.51	637.00	437.38	497.08	474.78	460.80	306.87

Table 4: Network size ($|\mathcal{N}_{\mathcal{T}}|$) at termination, by configuration

Relatedly, as DDD-TD-TSPTW draws on three sources to generate primal solutions, we next study which source yields the best solution the most often. Specifically, we report in Table 5, by instance set, the percentage of instances in which the best primal solution was derived by solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), by solving TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$), and by solving TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$). We see that the primal partially time-expanded networks often yield the best solution. However, we also note that the relaxation TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) produces the best solution much more often for Set *w100* than it does Set 1. We also see that while each primal partially time-expanded network contributes to the search for good solutions, $\bar{\mathcal{D}}_{\mathcal{T}}^1$ does so slightly more often.

	TD-TSPTW($\mathcal{D}_{\mathcal{T}}$)	TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$)	TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$)
Set 1	4.21%	53.76%	42.04%
Set <i>w100</i>	23.41%	39.95%	36.64%

Table 5: % of best solutions found, by source

Of course, there may have been instances in which the optimal solution was found by solving TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$) simply because it was solved before TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$). As a result, we next ran DDD-TD-TSPTW again, with only one of the Primal IPs enabled, and report the corresponding results in Table 6 (and for ease of comparison we repeat the results when both primal IPs were used). We see that when only one of the Primal IPs

Which Primal IP	Set 1 - 952 instances			Set <i>w100</i> - 960 instances		
	Slv	Tme	Iter	Slv	Tme	Iter
Both	952	10.94	4.62	960	1.22	4.11
TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$)	952	14.01	7.81	960	1.37	6.68
TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^2$)	952	26.79	5.83	960	3.01	5.13

Table 6: Performance when using only one Primal IP

is solved, DDD-TD-TSPTW is still able to solve each instance, although it requires more

time and iterations. We also see that just solving TD-TSPTW($\bar{\mathcal{D}}_{\mathcal{T}}^1$), in which the travel time of each arc is modeled as being at least as long as its actual travel time and thus its solution is feasible for the TD-TSPTW(\mathcal{D}), is slightly more effective. However, we conclude that it is best to solve both Primal IPs, as that yields the best results.

7 Duration objective

We next study how DDD-TD-TSPTW can be used to minimize the total time spent away from the depot (the duration objective). Mathematically, this corresponds to the objective function

$$\sum_{((i,t),(0,t')) \in \mathcal{A}} (t + \tau_{i0}(t))x_{((i,t),(0,t'))} - \sum_{((0,t),(i,t')) \in \mathcal{A}} tx_{((0,t),(i,t'))}.$$

Recall that when optimizing the makespan objective with travel times that adhere to the FIFO property, there exists an optimal solution in which the only waiting the vehicle may need to do is when it arrives at a location before its time window opens. Because of this, when creating and refining partially time-expanded networks, DDD-TD-TSPTW did not need to add nodes to $\mathcal{N}_{\mathcal{T}}$ that model waiting. With the duration objective, the situation is similar with regards to waiting at locations other than the depot, but it may be beneficial not to depart from the depot as early as possible, i.e., to wait at the depot. However, by changing how DDD-TD-TSPTW creates the initial partially time-expanded network, $\mathcal{D}_{\mathcal{T}}$, it can be used to optimize this objective as well.

Specifically, before creating the initial partially time-expanded network, we calculate the latest time, \bar{e}_0 , at which the vehicle might depart from the depot in a feasible solution, i.e.,

$$\bar{e}_0 = \min_{i \in \mathcal{N} \setminus \{0\}} \max_t \{\Delta_{0i}(t) \leq l_i\}.$$

As with the makespan objective, we add to $\mathcal{N}_{\mathcal{T}}$ the nodes (i, e_i) and (i, l_i) for all locations (including the depot). Now, however, we also add to $\mathcal{N}_{\mathcal{T}}$ the nodes $(0, t)$, $t = e_0 + \epsilon, e_0 + 2\epsilon, \dots, \bar{e}_0$. Here, ϵ represents the unit of time we consider when modeling the vehicle waiting at the depot. We illustrate the initial node sets, $\mathcal{N}_{\mathcal{T}}$, for each objective and an example with two locations in Figures 6a and 6b. In this example, $l_1 = 5$ and $l_2 = 7$, whereas $\tau_{01}(\cdot) = 1$ and $\tau_{02}(\cdot) = 2$. As a result, the latest time at which the vehicle can depart the depot in a feasible solution, \bar{e}_0 , is 4.

In addition to the arcs normally created to form $\mathcal{A}_{\mathcal{T}}$, we also add arcs of the form $((0, t), (0, t'))$, with t and t' consecutive time points in \mathcal{T}_0 , to represent waiting at the depot. The cost underestimate, $\underline{c}_{ij}(t)$ for each $((i, t), (j, t')) \in \mathcal{A}_{\mathcal{T}}$, is set exactly as for the makespan objective, except in the case that $i = 0$. For the new, waiting, arcs, as there is no penalty for waiting at the depot, we associate the cost $\underline{c}_{00}(t) = 0$ for all $t = e_0 + 1, \dots, \bar{e}_0$. We also set $\underline{c}_{0i}(t) = -t$ for all $((0, t), (i, t')) \in \mathcal{A}_{\mathcal{T}}$ with $i \neq 0$. Thus, for each arc leaving the depot, the cost “underestimate” is accurate.

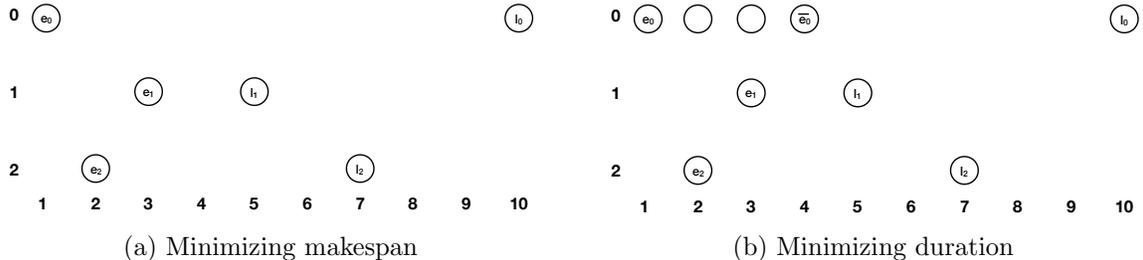


Figure 6: Initial $\mathcal{N}_{\mathcal{T}}$.

With this definition of $\mathcal{D}_{\mathcal{T}}$ and $\underline{c}_{\mathcal{A}_{\mathcal{T}}}$, it is not difficult to prove that $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ yields a lower bound on the original problem for the duration objective. The proof of Lemma 1 may be adapted by only considering paths in \mathcal{D} that correspond to feasible tours. (The current proof of Lemma 1 is more general, but the restriction to feasible tours is sufficient to yield the lower bound result.) Only the $k = 0$ case of the induction needs to be modified. In this case, since the path corresponds to a feasible tour, its first arc $((u_0, t_0), (u_1, t_1))$ has $u_0 = 0$ and $t_0 \leq t_0$. Thus $(u_0, t_0) = -t'_0 = -t_0 = c_{u_0 u_1}(t_0)$. The remainder of the proof is the same. It is, furthermore, not difficult to verify that if the solution to $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$, with $\mathcal{D}_{\mathcal{T}}$ and $\underline{c}_{\mathcal{A}_{\mathcal{T}}}$ constructed as described above, uses only arcs of the correct length, then, similar to Lemma 2, it must provide an optimal solution to the problem with the duration objective.

To optimize the duration objective we adapted DDD-TD-TSPTW in three other ways. First, the Primal IPs, $\text{TD-TSPTW}(\bar{\mathcal{D}}_{\mathcal{T}}^1)$ and $\text{TD-TSPTW}(\bar{\mathcal{D}}_{\mathcal{T}}^2)$ are solved with a duration objective (the networks are created the same way). Second, when solving $\text{TD-TSPTW}(\mathcal{D}_{\mathcal{T}})$ returns a solution, $((v_0, t_0), \dots, (v_n, t_n))$ that does not contain a sub-tour, we evaluate its duration under all potential departure times from the depot. As waiting at locations other than the depot is not necessary in an optimal solution to the $\text{TD-TSPTW}(\mathcal{D})$, this can be done by enumerating departure times from the depot and evaluating that tour in a greedy fashion for each departure time. Third, when a new feasible solution with objective function value z is found, we update the close of the time window at the depot, l_0 , to limit the search to solutions with a lower objective function value. Specifically, we set l_0 to $z - 1 + t_0$, where t_0 represents the departure time of the vehicle from the depot in that solution. We then repeat the preprocessing algorithm with this new value for l_0 .

To test the effectiveness of DDD-TD-TSPTW when solving instances with the *Duration* objective, we consider the same set of instances as those reported on above. We note that as we have adjusted the data associated with each instance to integral values, it was natural to set $\epsilon = 1$. We report statistics regarding DDD-TD-TSPTW's ability to solve instances with either objective in Table 7. We see that while DDD-TD-TSPTW can no longer solve every instance from each set, it can solve nearly all of them. We also note that it takes DDD-TD-TSPTW longer to solve instances when optimizing this objective.

However, we also note that by running DDD-TD-TSPTW for two hours (instead of one), it was able to solve all instances from both sets, with the remaining 22 instances from Set 1 requiring 3,647.47 seconds, on average, and the remaining 5 instances from Set *w100* requiring 3,688.40 seconds, on average.

Objective	Set 1 - 952 instances			Set <i>w100</i> - 960 instances		
	Slv	Tme	Iter	Slv	Tme	Iter
makespan	952	10.94	4.62	960	1.22	4.11
duration	930	143.45	9.77	955	106.20	9.87

Table 7: Performance of DDD-TD-TSPTW on different objectives

One of the primary challenges with this objective is that the additional nodes added to $\mathcal{N}_{\mathcal{T}}$ require DDD-TD-TSPTW to solve larger integer programs at each iteration. To illustrate, we present in Table 8, for each objective, the average size of both the partially time-expanded network, $\mathcal{D}_{\mathcal{T}}$, when DDD-TD-TSPTW begins (labeled “Initial”) and ends (labeled “Final”). We see that optimizing the duration objective with DDD-TD-TSPTW by adding to $\mathcal{N}_{\mathcal{T}}$ a node for each possible departure time from the depot often leads to over twice as many nodes in both networks.

Objective	Set 1 - 952 instances		Set <i>w100</i> - 960 instances	
	Initial $ \mathcal{N}_{\mathcal{T}} $	Final $ \mathcal{N}_{\mathcal{T}} $	Initial $ \mathcal{N}_{\mathcal{T}} $	Final $ \mathcal{N}_{\mathcal{T}} $
makespan	53.34	188.80	53.50	191.15
duration	111.88	452.24	94.50	438.89

Table 8: Network size in first and last iterations, for different objectives

8 Conclusions and Future Work

In this paper, we presented an algorithm, DDD-TD-TSPTW, based on the *Dynamic Discretization Discovery* (DDD) framework that can solve instances of the TD-TSPTW when either one of two objectives is to be optimized: (1) a makespan objective, and (2) a duration objective. This algorithm differs from existing applications of the DDD framework in that it can accommodate time dependent travel times. In addition, the techniques underlying this algorithm can be used to apply DDD to other transportation planning problems in which travel times are time dependent. The results of our computational study on benchmark instances indicate that the algorithm we proposed outperforms the best performing algorithm in the literature. We also saw that, unlike existing methods, the algorithm we propose is robust with respect to all instance parameters, particularly the degree of travel time variability.

In the case of the duration objective, the ratio of number of nodes in the final time-expanded node to the number of nodes in the initial time-expanded network is approximately the same as the ratio for the makespan objective, yet both numbers for the duration objective are about double their counterpart for the makespan objective. This motivates a scheme that adds time-expanded network nodes at the depot dynamically, instead of in an *a priori* and enumerative fashion. Such a scheme may accelerate solution for the duration objective case, and is a topic for future study.

The success of DDD-TD-TSPTW is, in part, due to the fact that the combination of a makespan objective and travel times satisfying the FIFO property, implies that the travel costs are non-decreasing in t , which meant that the cost underestimate $\underline{c}_{ij}(t)$ is never an underestimate, but always accurate. For other objectives, e.g., minimizing total travel time, this is no longer the case, and additional techniques are necessary to ensure acceptable computational performance. We are currently investigating this setting. Other, related problems we intend to study are those in which travel times are stochastic.

ACKNOWLEDGMENT

We would like to thank the authors of Arigliano et al. (2015) and Montero et al. (2017) for the exchange of test instances and ideas.

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, and Emanuela Guerriero. Time dependent traveling salesman problem with time windows: Properties and an exact algorithm. Technical report, Optimization Online, 2015.
- Edward K. Baker. Technical note - an exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5):938–945, 1983.
- Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.
- 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 *14th Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*, volume Lecture Notes in Computer Science. Springer, 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.
- Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.
- Miguel Andres Figliozzi. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, (3):616 – 636. ISSN 1366-5545.
- Matteo Fischetti, Gilbert Laporte, and Silvano Martello. The delivery man problem and cumulative matroids. *Operations Research*, 41(6):1055–1064, 1993.
- Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: A review. *Computers & Operation Research*, 64:189–197, 2015.
- Gianpaolo Ghiani and Emanuela Guerriero. A note on the ichoua, gendreau, and potvin (2003) travel time model. *Transportation Science*, 48(3):458–462, 2014.
- 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.
- 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.
- Imdat Kara, Ozge Nimet Koc, Fulya Altparmak, and Berna Dengiz. New integer linear programming formulation for the traveling salesman problem with time windows: minimizing tour duration with waiting times. *Optimization*, 62(10):1309–1319, 2013.
- André Langevin, Martin Desrochers, Jacques Desrosiers, Sylvie Gélinas, and François Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, 23(7):631–640, 1993.
- Abilio Lucena. Time-dependent traveling salesman problem-the deliveryman case. *Networks*, 20(6):753–763, 1990.
- Monirehalsadat Mahmoudi and Xuesong Zhou. Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological*, 89:19–42, 2016.
- 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.
- Agustín Montero, Isabel Méndez-Díaz, 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.
- Martin Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.
- 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.
- Christian Tilk and Stefan Irnich. Dynamic programming for the minimum tour duration problem. *Transportation Science*, 2015. Artices in Advance.
- UN. Our urbanizing world. Technical report, United Nations Department of Economic and Social Affairs, 2014.