Transportation Science

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

Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows

Duc Minh Vu, Mike Hewitt, Natashia Boland, Martin Savelsbergh

To cite this article:
Duc Minh Vu, Mike Hewitt, Natashia Boland, Martin Savelsbergh (2020) Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows. Transportation Science 54(3):703-720.
https://doi.org/10.1287/trsc.2019.0911

http://pubsonline.informs.org/journal/trsc

TRANSPORTATION SCIENCE
Vol. 54, No. 3, May–June 2020, pp. 703–720
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows

Duc Minh Vu,[a] Mike Hewitt,[b] Natashia Boland,[c] Martin Savelsbergh[c]

[a] Université François-Rabelais de Tours, CNRS, LI EA 6300, ROOT ERL CNRS 7002, Tours, France; [b] Department of Information Systems and Supply Chain Management, Quinlan School of Business, Loyola University Chicago, Chicago, Illinois 60660; [c] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

Contact: ducminh.vu@univ-tours.fr, http://orcid.org/0000-0001-5882-3868 (DMV); mhewitt3@luc.edu, http://orcid.org/0000-0002-9786-677X (MH); natashia.boland@isye.gatech.edu, http://orcid.org/0000-0002-6867-6125 (NB); martin.savelsbergh@isye.gatech.edu, http://orcid.org/0000-0001-7031-5516 (MS)

Received: July 16, 2018
Revised: December 12, 2018
Accepted: January 19, 2019
Published Online in Articles in Advance: August 8, 2019

https://doi.org/10.1287/trsc.2019.0911

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 predetermined period of time, and then, returns home. The problem allows for travel times that can depend on the time of departure. We consider two objectives for the problem: (1) a *makespan* objective that seeks to return the salesman to his home as early as possible and (2) a *duration* objective that seeks to minimize the amount of time that he is away from his home. The solution approach is based on an integer programming formulation of the problem on a time-expanded network, because doing so enables time dependencies to be embedded in the definition of the network. However, because such a time-expanded network (and thus, the integer programming formulation) can rapidly become prohibitively large, the solution approach uses 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.

History: This paper has been accepted for the *Transportation Science* Special Section on Urban Freight Transportation and Logistics. Michel Gendreau served as the guest editor-in-chief for this article.
Funding: This material is based on work supported by the National Science Foundation under the Division of Civil, Mechanical and Manufacturing Innovation [Grants CMMI-1435625 and 1662848].

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, returns 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 is replaced by a depot, and the cities are replaced 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 prespecified period of time, gives the traveling salesman problem with time windows (TSPTW). The TSPTW is even more relevant to logistics, because customers often request or are quoted a delivery time window.

In this paper, we go beyond the classical TSPTW to allow for 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, Montero, Méndez-Dıaz, and Miranda-Bront 2017 and Arigliano et al. 2018). 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 that 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. Although 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, because it results in an increase in deliveries (to either 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 2012, Gendreau, Ghiani, and Guerriero 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 suggests that the DDD framework might be used to develop effective algorithms for other transportation planning problems in which travel times are time dependent. Although 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 that these results highlight a key advantage of 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. Sec tion 2 describes the TD-TSPTW in detail and presents a mathematical programming formulation of it. Sec tion 3 then discusses the relevant literature. Section 4 describes the basis of the algorithm, the *partially time-expanded network*, and describes 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 predefined 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 TSPTW. We consider situations in which the time at which travel occurs impacts the travel time, resulting in a problem that is called the TD-TSPTW.

As is commonly done, we assume that 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 that 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 $N = \{0, 1, 2, \ldots, n\}$ includes the depot (node 0) as well as the set of locations (nodes $1, \ldots, n$) that must be visited. Associated with each location $i = 1, \ldots, n$ 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 $t \le t'$, it must be that $t + \tau_{ij}(t) \le t' + \tau_{ij}(t')$). 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, $((i_0, t_0), (i_1, t_1), \ldots, (i_n, t_n), (i_{n+1}, t_{n+1}))$, satisfying $i_0 = i_{n+1} = 0$, $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$, $t_0 \ge e_0$, $t_{k+1} \ge \max\{e_{i_{k+1}}, t_k + \tau_{i_k, i_{k+1}}(t_k)\}$, $t_k \le l_{i_k}$ for all $k = 0, \ldots, n$, and $t_{n+1} \le \ell_0$. The makespan objective minimizes t_{n+1}, whereas the duration objective minimizes $t_{n+1} - t_0$.

We formulate the TD-TSPTW as an integer program (IP) defined on a time-expanded network $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with finite node set $\mathcal{N}$ and finite arc set $\mathcal{A}$. Because

departure times are a continuous quantity, the existence of an integer program defined over a time-expanded network with a finite set of nodes and a finite set of arcs that may be solved to derive a continuous time optimal solution to the TD-TSPTW is not necessarily obvious. However, one can prove its existence by observing 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.

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 $i \neq j, (i,j) \in A, t \in \mathcal{T}_i, t' \in \mathcal{T}_j, t' = \max\{e_j, t + \tau_{ij}(t)\}$ (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 the vehicle travels along that arc. Because we optimize two different objectives in this paper, we present the following formulation of the TD-TSPTW that seeks to optimize a generic objective function:

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

subject to

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

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

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

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

In this paper, we seek to optimize the following two objectives, with the first representing the makespan of the tour and the second representing its duration:

$$\sum_{((i,t),(0,t'))\in\mathcal{A}} (t + \tau_{i0}(t))x_{((i,t),(0,t'))} \tag{4}$$

$$\sum_{((i,t),(0,t'))\in\mathcal{A}} (t + \tau_{i0}(t))x_{((i,t),(0,t'))} - \sum_{((0,t),(i,t'))\in\mathcal{A}} t x_{((0,t),(i,t'))}. \tag{5}$$

To represent the makespan objective (4), 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 nonnegative and nondecreasing in t. To represent the duration objective (5), the makespan cost function is modified by setting $c_{0j}(t) := -t$ for all $((0,t),(j,t')) \in \mathcal{A}$, which is nonpositive and nonincreasing.

Note that the impact of time windows and time-dependent travel times on the feasible region of the TD-TSPTW is embedded in the time-expanded network, $\mathcal{D}$. The fact 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

Although our paper is focused on exact methods for the TD-TSPTW under different objectives, we believe that 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 objective and a 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, Mingozzi, and Toth (1981) and Baker (1983) proposing branch-and-bound–based solution methods. Other integer programming-based approaches include the work by 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, Laporte, and Martello (1993), Gouveia and Voz (1995), Albiach, Sanchis, and Soler (2008), Stecco, Cordeau, and Moretti (2008), Méndez-Díaz et al. (2011), Abeledo et al. (2013), Bront, Méndez-Díaz, and Zabala (2014), and Melgarejo, Laborie, and Solnon (2015). That said, those works do not ensure that the FIFO property is observed.

In contrast, Ichoua, Gendreau, and Potvin (2003) propose 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) study properties of this travel time function as well as propose generalizations. This travel time function is also used in approaches for solving time-dependent variants of the TSP. Related to the first problem that we study, Cordeau, Ghiani, and Guerriero (2014) propose an exact method for a time-dependent traveling salesman problem (TD-TSP) based on 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. (2018). More recently, Montero, Méndez-Dıaz, and Miranda-Bront (2017) propose 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 that all presume travel times that are time independent. Kara et al. (2013) present a new mixed integer programming formulation for instances with asymmetric travel times, whereas Kara and Derya (2015) present a formulation for instances in which travel times are symmetric. Dynamic programming is another algorithmic strategy that researchers have used to optimize the TSPTW under this objective. Tilk and Irnich (2017) extended some of the state space relaxation ideas presented in Baldacci, Mingozzi, and Roberti (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, because 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 that 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 that we propose accommodates time-dependent travel times, whereas the method proposed by Dash et al. (2012) presumes that 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 on the same framework as that used in Boland et al. (2017a, b). 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 illustrate 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. Such as is commonly done in the literature, we assume that the FIFO property holds for the arc travel time functions, which implies that it is not possible to arrive at the location at the head of an arc by departing later at the tail of the arc. For the two objectives that we consider (i.e., makespan and duration), this implies that it is never advantageous to wait at a location (other than at the depot in case of the 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 require that $t' \leq \max\{e_j, t + \tau_{ij}(t)\}$. In addition, we do not require that $t' > t$. In fact, arc $((i,t),(j,t'))$ may model negative (or zero) travel time by having $t' < t$ (or $t' = t$). 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 underestimate 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:

$$\underline{c}_{ij}(t) := \min\{c_{ij}(h) \mid t \le h \le l_i \text{ and } h + \tau_{ij}(h) \le l_j\}. \quad (6)$$

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 $\underline{c}_p$ represent 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)–(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 to ensure that TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) is a relaxation of 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. *$\forall (i,t) \in \mathcal{N}_{\mathcal{T}}, e_i \le t \le l_i$.*

Property 3. *$\forall (i,t) \in \mathcal{N}_{\mathcal{T}}$ and arc $(i,j) \in A$ with $t + \tau_{ij}(t) \le l_j$; 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) $e_j \le t' \le t + \tau_{ij}(t)$. Finally, we note that $\mathcal{A}_{\mathcal{T}}$ only contains arcs $((i,t),(j,t'))$ that satisfy $t + \tau_{ij}(t) \le l_j$.*

Property 4. *$\forall a = ((i,t)(j,t')) \in \mathcal{A}_{\mathcal{T}}$, and the cost $\underline{c}_{ij}(t)$ is given by (6).*

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 $\mathcal{D}$ and can be traversed starting at time t, and (2) the travel time associated with each arc $((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}$ underestimates the travel time of the arc (i,j) that it models dispatching at time t. Property 4 ensures something similar but with respect to travel costs. Because the arc $((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}$ only implies that the vehicle dispatches on arc (i,j) at a time $h \ge t$, 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.

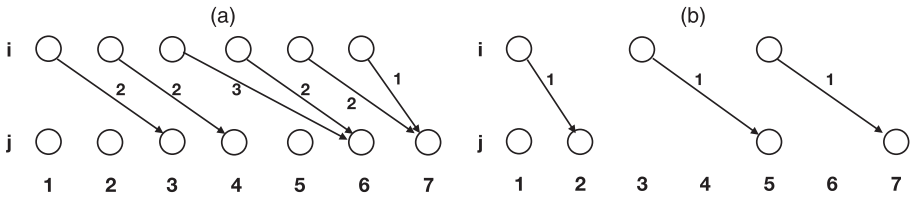
We illustrate Properties 3 and 4 in Figure 1, where Figure 1(a) displays the time needed to travel from location i to location j for different departure times, and the number next to each arc indicates the cost associated with such travel (for ease of presentation, we assume that the time points in the time-expanded network occur at integers). Figure 1(b) 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.

We next prove that 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 1–4, *then the objective function value of an optimal solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) is a lower bound on the objective function value of an optimal solution to TD-TSPTW($\mathcal{D}$).*

Proof of Lemma 1. To prove this lemma, we 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.

Figure 1. Properties 3 and 4 for Arc $(i,j) \in A$



Full time-expanded network, $\mathcal{D}$; $c_{ij}(t)$ Partially time-expanded network, $\mathcal{D}_{\mathcal{T}}$; $\underline{c}_{ij}(t)$

Note. Numbers next to arcs indicate either real costs, $c_{ij}(t)$, or underestimates, $\underline{c}_{ij}(t)$.

Note that p need not represent travel from the depot to all locations and then back to the depot.

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

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

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 $e_0 = t'_0 \le t_0$, because $\mathcal{D}$ does not contain nodes $(u_0, \bar{t})$ with $\bar{t} < e_0$. Second, note that $\underline{c}_{u_0 u_1}(t'_0) \le c_{u_0 u_1}(t_0)$ by Property 4 and because t_0 is one of the time points considered when the value $\underline{c}_{u_0 u_1}(t'_0)$ was determined. This can be seen by observing that $t_0 + \tau_{u_0 u_1}(t_0) \le t_{u_1} \le l_{u_1}$, and thus, $t_0 \le l_{u_1} - \tau_{u_0 u_1}(t_0)$.

Next, assume that $k = i$ is true. We prove that our conditions hold for $k = i + 1$. By the inductive assumption, there exists a path $((u_0, t'_0), (u_1, t'_1), \ldots, (u_i, t'_i)) \in \mathcal{D}_\mathcal{T}$, satisfying the above conditions, and hence, $(u_i, t'_i) \in \mathcal{N}_\mathcal{T}$ with $t'_i \le t_i$. By Property 3, we have that the arc $((u_i, t'_i), (u_{i+1}, t''_{i+1})) \in \mathcal{A}_\mathcal{T}$, with $t''_{i+1} \le t'_i + \tau_{u_i u_{i+1}}(t'_i) \le t_i + \tau_{u_i u_{i+1}}(t_i)$, by the FIFO property. Thus, $t''_{i+1} \le t_{i+1}$, and the first and second conditions are verified. Similar to our base case and by Property 4, because $t''_i \le t_i$, it must be that $\underline{c}_{u_i u_{i+1}}(t'_i) \le c_{u_i u_{i+1}}(t_i)$. Because 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: $\underline{c}_{p'} \le c_p$.

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

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 $t' < t'' \le t + \tau_{ij}(t)$.*

Although we do not prove it, it is not difficult to see that Property 5 strengthens the lower bound obtained by solving TD-TSPTW($\mathcal{D}_\mathcal{T}$), because 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). 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 the 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 do not and therefore, 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, because 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 nondecreasing 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 nondecreasing with time: that is, having $c_{ij}(t) \le c_{ij}(t')$ whenever $t \le t'$. Let $\mathcal{D}_\mathcal{T}$ and $\{\underline{c}\}_{\mathcal{A}_\mathcal{T}}$ satisfy Properties 1–4, and consider a path $p' = ((u_0, t'_0), (u_1, t'_1), \ldots, (u_n, t'_n), (u_{n+1}, t'_{n+1}))$ 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 $i = 1, \ldots, n+1$, and $u_{n+1} = u_0$. An optimal solution to TD-TSPTW($\mathcal{D}$) can be derived from p'.*

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 $i = 0, \ldots, n$, we have that the corresponding travel arcs, $((u_i, t'_i), (u_i, t'_{i+1}))$, 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($\mathcal{D}$). Because travel costs are nondecreasing in t, we have that $\underline{c}_{ij}(t) = c_{ij}(t)$ and that $\underline{c}_{p'} = c_{p'}$. 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. □

For cases where arc costs are nondecreasing, 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.

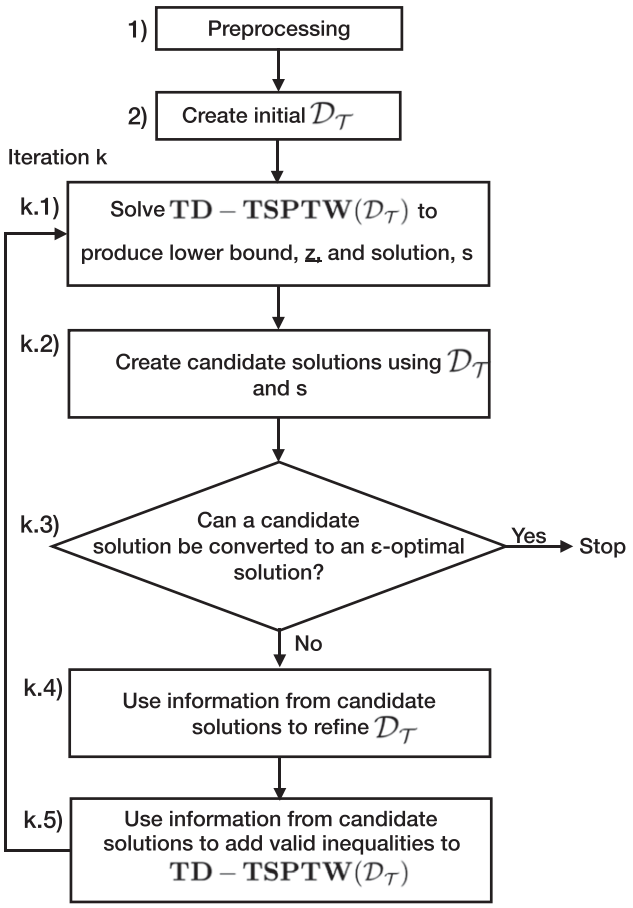
In the case of the duration objective, the arc costs only fail to be nondecreasing for arcs leaving the depot, location 0; all other arcs have nondecreasing cost functions. Thus, by adding one more condition to $\mathcal{D}_{\mathcal{T}}$, that $(0,t) \in \mathcal{N}_{\mathcal{T}}$ for all t at which a feasible tour may start, 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 that 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 around the flowchart for the algorithm that is presented in Figure 2. That flowchart illustrates the steps that the algorithm takes during its execution. The individual steps are then discussed in detail in the following subsections. The final subsection discusses correctness of the algorithm.

Figure 2. Flowchart of DDD-TD-TSPTW



5.1. Preprocessing

The algorithm uses preprocessing techniques derived from those seen in the literature (Desrochers, Desrosiers, Solomon 1992, Desrosiers et al. 1995, Dash et al. 2012) on routing problems with time windows to both prune arcs from A and tighten time windows. Montero, Méndez-Dıaz, and Miranda-Bront (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 that the vehicle can depart location j assuming that 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 that 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 that 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 it 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 it sets $l_k = \min\{l_k \max\{e_k, \max_{i \in \mathcal{N}\setminus k} \max_t\{\Delta_{ki}(t) \le l_i\}\}\}$.

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

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

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

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

Next, we analyze the time windows at pairs of locations 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 $i \to j$ as infeasible. Because of 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 sequences of locations $i \to j \to k$ and $k \to i \to j$ are infeasible, then we can remove arc (i,j) from A. If the sequence of locations $i \to k \to j$ is also infeasible, then we can conclude that $j \prec i$. Finally, we note that precedences are transitive: that is, 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 $i \to j$ 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 $i \to j \to k, k \to i \to j$, and $i \to k \to j$ are all infeasible, and add $j \prec i$ to P if so
9: Check if $i \to j \to k$ and $k \to i \to j$ 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 underestimate, but never overestimate, the actual travel times), Property 4 (arc costs may underestimate, but never overestimate, actual arc costs), and Property 5 (each created arc is as long as possible). Note that arcs satisfying these properties can be created, because 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 does not, the path that 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 therefore, 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 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 networks, $\bar{\mathcal{D}}^i_{\mathcal{T}} = (\mathcal{N}_{\mathcal{T}}, \bar{\mathcal{A}}^i_{\mathcal{T}})$ for $i = 1, 2$, and then, solves TD-TSPTW($\bar{\mathcal{D}}^i_{\mathcal{T}}$) 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, $\bar{\mathcal{A}}^i_{\mathcal{T}}$, for $i = 1, 2$, respectively.

The first arc set, $\bar{\mathcal{A}}^1_{\mathcal{T}}$, 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 $\bar{\mathcal{A}}^1_{\mathcal{T}}$. As a result, any feasible solution to TD-TSPTW($\bar{\mathcal{D}}^1_{\mathcal{T}}$) may be used to generate a feasible solution to TD-TSPTW($\mathcal{D}$). Unlike the first, the second arc set, $\bar{\mathcal{A}}^2_{\mathcal{T}}$, includes arcs that are too short. However, it does not include arcs $((i,t),(j,t'))$ that model nonpositive travel times (i.e., with $t' \leq t$). As a result, feasible solutions to TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$) cannot include subtours. Specifically, $\bar{\mathcal{A}}^2_{\mathcal{T}}$ 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 $\bar{\mathcal{A}}^2_{\mathcal{T}}$. 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 $t + \tau_{ij}(t) \le l_j$ 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($\bar{\mathcal{D}}^1_{\mathcal{T}}$) 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($\bar{\mathcal{D}}^2_{\mathcal{T}}$) 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 $t' \le t$ allows 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 because of the flow balance constraints (constraint (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 that, 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. Although a solution to TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$) will not contain subtours, it may yield infeasible location sequences.

Third, 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 and ends 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 $e_{v_i} \le t'_i \le l_{v_i}$, $t'_0 = e_{v_0}$, 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 the 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 with 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($\bar{\mathcal{D}}^1_{\mathcal{T}}$), or TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$), 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 causes 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 that Lemma 1 remains valid and that 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 1–4.

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 that 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 that the vehicle arrives within the time window at the location at the head of the arc (e.g., $e_{u_{i+1}} \le t_i + \tau_{u_i u_{i+1}}(t_i) \le l_{u_{i+1}}$). 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 $p' = ((u_0,t_0),\dots,(u_k,t_k),(u_{k+1},t_{k+1}),\dots,(u_m,t_m)),(u_i,t_i) \in \mathcal{N}_{\mathcal{T}}$

```
for i ← 0 to m − 1, do
    t ← t_i {Consider departure from u_i at t_i}
    for j ← i + 1 to m, do
        t' ← max{e_{u_j}, t + τ_{u_{j−1}u_j}(t)}
        if t' > l_j, then
            Break
        end if
        Call Add-node((u_j, t'))
        t ← t'
    end for
end for
```

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 $t' + \tau_{jk}(t') \le l_k$, **do**
6: $t = \text{argmax}\{\bar{t} | (k,\bar{t}) \in \mathcal{N}_{\mathcal{T}}, \bar{t} \le t' + \tau_{jk}(t')\}$
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 $t + \tau_{ij}(t) \ge t'$, **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**

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 $(u_j,u_{j+1},\ldots,u_m)$ starting at u_j at time t'. We illustrate this in Figure 3. Here, Algorithm 2 is called with the path $p = ((u_0,1),(u_1,1),(u_2,2))$, where $\tau_{u_0u_1}(t) = 1$, $\tau_{u_1u_2}(t) = 2$ for all t. Because 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 $(u_2,4)$ and arc $((u_1,2),(u_2,4))$ so that the vehicle can visit the same sequence of locations, now on arcs that model true travel times.

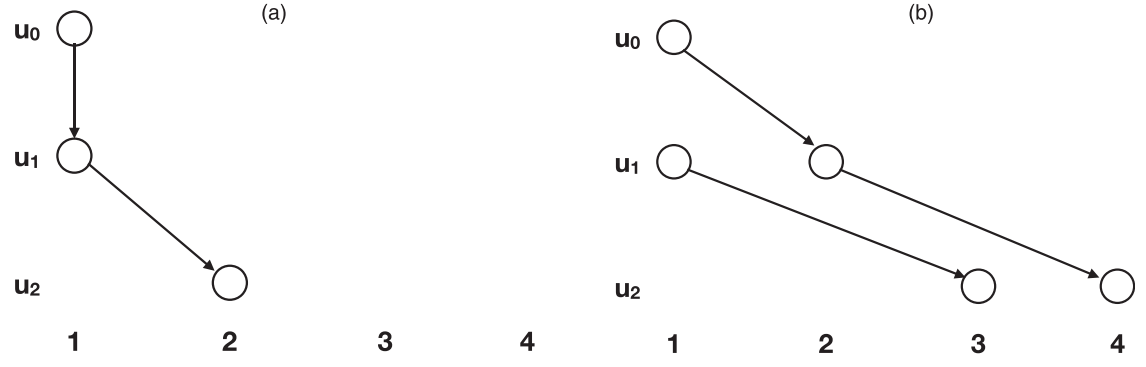
In addition, Algorithm 2 considers subpaths that begin at intermediate nodes in the path that is being examined. Continuing with our example, Algorithm 2 considers the subpath that departs from u_1 at time 1 and consists of the arc $((u_1,1),(u_2,2))$. Because this arc is too short, the algorithm creates node $(u_2,3)$ and replaces arc $((u_1,1),(u_2,2))$ with one that models the actual travel time, $((u_1,1),(u_2,3))$. As will be discussed later in this section, lengthening one arc in each iteration guarantees correctness of the algorithm. However, we have found that these "opportunistic" additions of arcs and nodes have been computationally beneficial. We hypothesize that this is because each lengthened arc renders the sets $\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}$ to be closer approximations of the sets $\mathcal{N}, \mathcal{A}$ that are used to formulate the TD-TSPTW($\mathcal{D}$), which in turn, strengthens the bound produced by solving the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) at an iteration.

We note that we can prune arcs from the network, $\mathcal{D}_{\mathcal{T}}$, 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 $t + \tau_{ij}(t) \ge z$, and (2) we update $l_0 = z - 1$ and then, repeat the preprocessing algorithm described earlier. We note that this preprocessing may result in an instance of the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) that is infeasible, in which case we can conclude that the best feasible solution found so far is optimal.

5.7. Adding Valid Inequalities to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$)

As noted above, there are two reasons that a candidate solution cannot be converted to an optimal solution to the TD-TSPTW($\mathcal{D}$). Although refining the network $\mathcal{D}_{\mathcal{T}}$ removes 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 (VIs). Specifically, recall that DDD-TD-TSPTW may determine that the solution, s, to the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) contains subtours of the form, $(v_i,\ldots,v_k)$, in which $v_j \ne v_{j+1}, j = i,\ldots,k-1$ 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

Figure 3. Refining $\mathcal{D}_{\mathcal{T}}, \tau_{u_0u_1}(\cdot) = 1, \tau_{u_1u_2}(\cdot) = 2$



Notes. (a) Before refining. (b) After refining.

locations, $(v_i, \ldots, v_k)$, visited in the subtour. Then, DDD-TD-TSPTW adds valid inequalities of the form

$$\sum_{(i,j)\in A\cap(M\times M)} \sum_{t,t':((i,t),(j,t'))\in\mathcal{A}_{\mathcal{T}}} x_{((i,t),(j,t'))} \leq |M| - 1 \qquad (7)$$

to the instance of TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) solved in all subsequent iterations. Note that M consists of locations N rather than $\mathcal{N}_{\mathcal{T}}$, and thus, the inequality can be applied to the IP for any network, $\mathcal{D}_{\mathcal{T}}$. Constraint (7) not only rules out a subtour consisting of locations in M that begin at a specific time t but also, 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, $(v_i, \ldots, v_k)$, 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, $(v_i, \ldots, v_k)$. The path-based valid inequalities of the form

$$\sum_{i=0}^{k-1} \sum_{t,t':((v_i,t),(v_{i+1},t'))\in\mathcal{A}_{\mathcal{T}}} x_{((v_i,t),(v_{i+1},t'))} \leq |M| - 2 \qquad (8)$$

are added to the IP. Like the subtour inequalities above, constraint (8) is determined by locations in N, and thus, it can be applied for any network $\mathcal{D}_{\mathcal{T}}$. Similarly, constraint (8) forbids all timed copies of the sequence $(v_i, \ldots, v_k)$.

5.8. Correctness of DDD-TD-TSPTW

We next discuss why DDD-TD-TSPTW will terminate in a finite number of iterations with an optimal solution (to some prespecified tolerance ϵ) of the TD-TSPTW($\mathcal{D}$) as well as a proof that that solution is optimal. First, we note that, when DDD-TD-TSPTW terminates, it does so with a solution that is optimal, because the solution to TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) is feasible for TD-TSPTW($\mathcal{D}$). Second, we note that, when the algorithm does not terminate in an iteration, the network $\mathcal{D}_{\mathcal{T}}$ is refined by identifying an arc $((i,t),(j,t'))$ such that $t' < t + \tau_{ij}(t)$ and $e_j \leq t + \tau_{ij}(t)\big) \leq l_j$. This implies that $(j, t + \tau_{ij}(t)) \in \mathcal{N}$ and $((i,t),(j,t+\tau_{ij}(t))) \in \mathcal{A}$. Third, the node $(j, t + \tau_{ij}(t))$ is added to $\mathcal{N}_{\mathcal{T}}$, and the arc $((i,t),(j,t+\tau_{ij}(t)))$ is added $\mathcal{A}_{\mathcal{T}}$. Given that nodes are never removed from $\mathcal{N}_{\mathcal{T}}$ and that arcs are never removed from $\mathcal{A}_{\mathcal{T}}$, after a finite number of iterations of DDD-TD-TSPTW, $\mathcal{N}_{\mathcal{T}} = \mathcal{N}$ and $\mathcal{A}_{\mathcal{T}} = \mathcal{A}$. At that iteration, by solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$), DDD-TD-TSPTW is in fact solving TD-TSPTW($\mathcal{D}$) and will terminate after doing so.

6. Computational Results

In this section, we assess the performance of the proposed DDD algorithm when solving TD-TSPTW with a makespan objective. We first benchmark its performance against the approach proposed in Montero, Méndez-Dıaz, and Miranda-Bront (2017), the best-performing method in the literature. We then analyze its performance in greater detail to understand how its components impact its performance. We finish this section with an assessment of its performance on a new set of instances that are larger than what has been considered in the literature to date. DDD-TD-TSPTW is implemented in C++, and all experiments were run on a workstation with an Intel(R) Xeon (R) CPU E5-4610 v2 2.30 GHz processor running the Ubutu 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 other than the following two were left at their default values: (1) the Threads parameter was set to limit Gurobi to one thread of execution, and (2) the MIPGap parameter was set so that Gurobi would terminate when it found a solution with a provable optimality gap no greater than 10^{-4}. 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 first tested on two sets of instances from the literature, which were proposed in Cordeau, Ghiani, and Guerriero (2014) for the TD-TSP, extended to the TD-TSPTW in Arigliano et al. (2018), and used to test the methods proposed in Montero, Méndez-Dıaz, and Miranda-Bront (2017) and Arigliano et al. (2018). We note that Arigliano et al. (2018) added time windows in a way that guaranteed the existence of a feasible solution. Instances in both sets vary in the number of locations that the vehicle must visit ($n = 15, 20, 30, 40$). In these instances, customers are randomly distributed across three concentric circular zones, wherein the innermost zone represents a city center, the middle zone represents the outskirts of a city, and the outermost zone represents a residential area. Traffic density and thus, travel speed on an arc depend in part on the zone in which the arc originates. These instances also vary in the value of a parameter Δ that represents the degradation in travel speed because of traffic density. Smaller values of Δ reflect greater congestion and longer travel times. Regarding time dependency of travel times, these instances model a day that is divided into three periods. The first and third periods represent morning and evening rush hours, whereas the second period represents the middle of the day

when traffic is lighter. Instances in both sets also vary depending on two traffic patterns. We refer the reader to Cordeau, Ghiani, and Guerriero (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, Gendreau, and Potvin (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, Méndez-Dıaz, and Miranda-Bront (2017), 8 are missing.) The second set, which we refer to as *w100*, such as in Montero, Méndez-Dıaz, and Miranda-Bront (2017), has 73 time intervals. This set does have 960 instances. Although Montero, Méndez-Dıaz, and Miranda-Bront (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, Méndez-Dıaz, and Miranda-Bront (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, Méndez-Dıaz, and Miranda-Bront (2017) for their branch-and-cut method, which they label as TTBF-BC, and refer to the results as such. Their results were achieved on a Workstation with an Intel Core i7-2600 3.4 GHz CPU and 16 Gb RAM. Their approach is also an integer programming-based approach, and they used CPLEX 12.5 as a solver.

We denote 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 "Time." 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-BC 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, whereas TTBF-BC 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 include times for instances that TTBF-BC did not solve.

Table 1. Performance on the Instances of Arigliano et al. (2018)

		TTBF-BC			DDD-TD-TSPTW		
Instance set	Traffic pattern	Inst	Slv	Time	Inst	Slv	Time
Set 1	A	478	470	31.99	478	478	12.91
Set 1	B	474	470	20.44	474	474	8.79
Set w100	A	108	107	100.13	480	480	1.31
Set w100	B	108	107	93.10	480	480	1.14

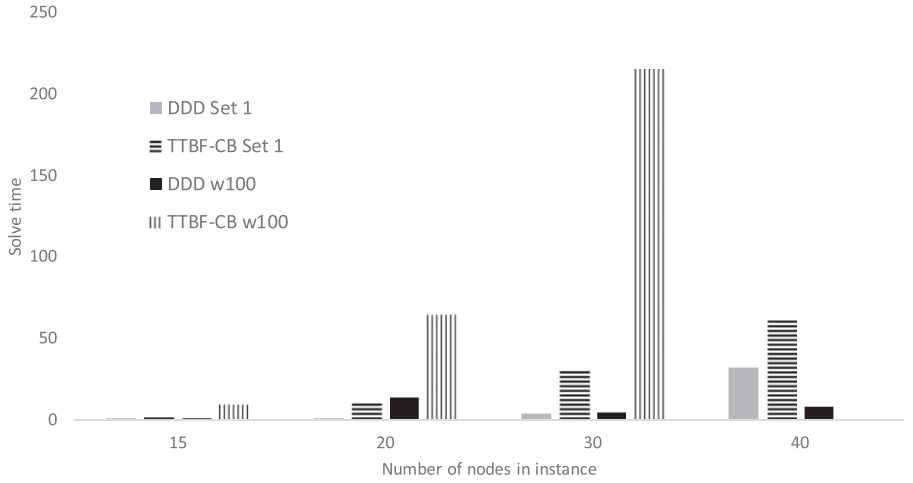
We also note that the time that TTBF-BC requires to solve instances from Set w100 is often three to four times greater, on average, than when solving instances from Set 1. DDD-TD-TSPTW, however, does not exhibit this behavior; the time that it requires to solve an instance from w100 is less than, on average, the time required to solve an instance from Set 1. We attribute this to DDD-TD-TSPTW operating on a time-expanded network, because 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, although 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, Méndez-Dıaz, and Miranda-Bront (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 that each method needs to solve instances from each set. We note that Montero, Méndez-Dıaz, and Miranda-Bront (2017) did not report results for their method when applied to instances from Set w100 with 40 nodes. We see that the growth rate in solve times is much smaller for DDD-TD-TSPTW than for TTBF-BC.

TTBF-BC 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, Méndez-Dıaz, and Miranda-Bront (2017) indicate that this root node gap is much larger for instances from w100 than those 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-BC is to solve the instance.

DDD does not have a root gap in the sense that TTBF-BC does. However, it is iterative in nature, and at every iteration, it 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,

Figure 4. Solve Times by Number of Locations



if LB_1 represents the first dual bound produced and LB^* represents 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 $(UB_1 - UB^*)/UB_1$, 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-BC, 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 is 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, $w = \frac{\sum_{i=0}^{n}(l_i - e_i)}{n}$, after preprocessing. We then computed the correlation coefficient between this average for an instance and the time that DDD-TD-TSPTW needed to solve that instance over all instances in Set 1. The correlation coefficient is 0.01, suggesting that the time that DDD-TD-TSPTW needs to solve an instance does not depend on time window width. Because 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 that DDD-TD-TSPTW

Figure 5. Gaps by Number of Locations

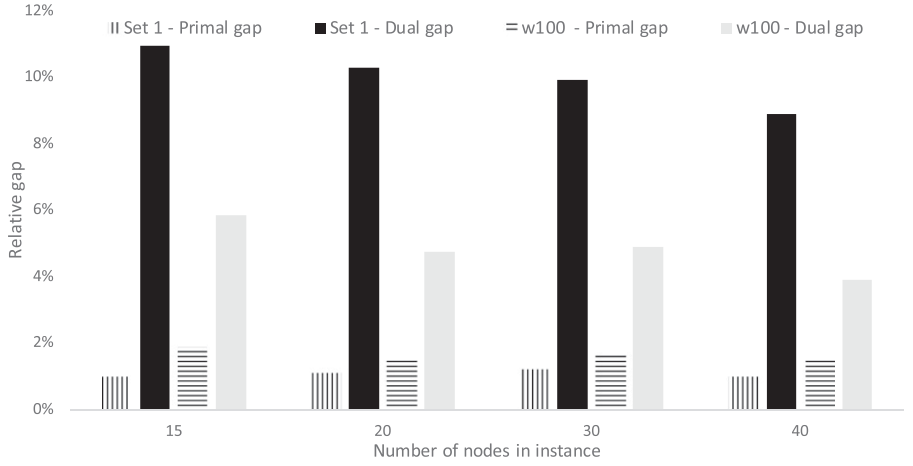


Table 2. DDD-TD-TSPTW Solve Time by Level of Congestion

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

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 seem to be a clear relationship between congestion level and DDD-TD-TSPTW solve times.

We conclude from these results that DDD-TD-TSPTW is superior to TTBF-BC at solving instances with a makespan objective and that it 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 $\epsilon-$optimal solution can be found, (2) the VIs that are added, and (3) the "Primal" IPs, TD-TSPTW($\bar{\mathcal{D}}^i_{\mathcal{T}}$) for $i = 1$ and $i = 2$, that are solved 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($\bar{\mathcal{D}}^i_{\mathcal{T}}$), are not solved
- "Base + VI": The same as 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 (Iters) that the configuration needed to converge to a provably $\epsilon-$ optimal solution (when it was able to do so). We report the results in Table 3.

Table 3. Performance of Different Method Configurations

	Set 1: 952 instances			Set w100: 960 instances		
Method	Slv	Time	Iter	Slv	Time	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

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 because of the fact that few inequalities are added. Specifically, when solving instances from Set 1, DDD-TD-TSPTW added, on average, only 5.08 subtour-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 subtour-based and 10.97 infeasible sequence inequalities were added. However, the Primal IPs dramatically reduce the time and number of iterations that DDD-TD-TSPTW needs to solve instances, particularly those from Set 1. We partially attribute this to the fact that DDD-TD-TSPTW is able to quickly produce strong dual bounds (as seen in Figure 5). As such, after the algorithm finds a (near-)optimal primal solution, it can accurately assess the solutions' quality and terminate. We also attribute this to the fact that improved primal solutions enable DDD-TD-TSPTW to prune arcs from $\mathcal{D}_{\mathcal{T}}$, which in turn, reduces the time need to solve subsequent instances of the TD-TSPTW($\mathcal{D}_{\mathcal{T}}$). 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.

Relatedly, because 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($\mathcal{D}_{\mathcal{T}}$), TD-TSPTW($\bar{\mathcal{D}}^1_{\mathcal{T}}$), and TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$). 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 for Set 1. We also see that, although each primal partially time-expanded network contributes to the search for good solutions, $\bar{\mathcal{D}}^1_{\mathcal{T}}$ does so slightly more often.

Of course, there may have been instances in which the optimal solution was found by solving TD-TSPTW($\bar{\mathcal{D}}^1_{\mathcal{T}}$),

Table 4. Network Size ($|\mathcal{N}_{\mathcal{T}}|$) at Termination by Configuration

	Set 1				Set w100			
Nodes	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

simply because it was solved before TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$). As a result, we next ran DDD-TD-TSPTW again with only one of the Primal IPs enabled, and we report the corresponding results in Table 6 (for ease of comparison, we repeat the results when both primal IPs were used).

We see that, when only one of the Primal IPs 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($\bar{\mathcal{D}}^1_{\mathcal{T}}$), 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, because that yields the best results.

6.4. Performance on Larger Instances

Having benchmarked and analyzed the performance of DDD-TD-TSPTW on instances from the literature, we next turn our attention to how well the algorithm performs when solving instances with greater numbers of locations. To do so, we generated instances with a method similar to the one described in Arigliano et al. (2018), albeit with 60, 80, or 100 customer locations. Like the benchmark instances, we generated locations in concentric circular zones. For a given instance, all customers were assigned time windows of the same width, and we considered six values for that width when generating an instance: 40, 60, 80, 100, 120, and 150. We considered the two traffic patterns (A and B) used to generate the w100 set of instances and the same four values for the congestion parameter Δ. All told, there are four parameter values that determine a class of instance: (1) the number of locations, (2) the time window width, (3) the traffic pattern, and (4) the value of Δ. For each class of instance, we randomly generated five instances. In summary, we generated 240 instances for each number of locations and 720 instances in total.

Table 5. Percentages of Best Solutions Found by Source

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

We executed DDD-TD-TSPTW for one hour and with a 1% optimality tolerance, and we report summary statistics based on the results of these experiments in Table 7. As in our previous tables, we report on the number of instances solved as well as the time and number of iterations that DDD-TD-TSPTW needed, on average, to solve an instance to within a 1% optimality tolerance. In addition, we also report the average optimality gap calculated as $(UB^* - LB^*)/UB^*$ reported by the algorithm for those instances that it could not solve ("Gap unsolved").

We see that, although DDD-TD-TSPTW is able to solve nearly all of the instances with 60 or 80 locations, it takes much more time to do so than the instances from the literature with 40 locations. Because the number of iterations to converge is roughly the same, we conclude that the increased time is because of the fact that the integer programs that the algorithm solves at an iteration are larger and therefore, harder to solve. We also observe that, for the few instances that DDD-TD-TSPTW could not solve, the optimality gap at the end of the one-hour runtime was quite small (less than 3% on average). Although we conclude from these results that DDD-TD-TSPTW performs well on instances with 60 or 80 locations, we see that it begins to struggle on instances with 100 locations. In addition to solving fewer instances, the optimality gap of those unsolved is quite large. That said, we also believe that, as the performance of integer programming solvers improves, DDD-TD-TSPTW will be able to solve more of the instances with 100 locations.

7. Duration Objective

We next study how DDD-TD-TSPTW can be used to minimize the total time spent away from the depot

Table 6. Performance When Using Only One Primal IP

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

Table 7. Performance on Instances with More Customer Locations

Locations	Inst	Slv	Time	Iter	Gap unsolved, %
60	240	239	54.26	3.78	1.56
80	240	235	239.15	4.58	2.84
100	240	227	321.48	4.69	11.13

(the duration objective). Mathematically, this corresponds to the objective function

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

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: that is,

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

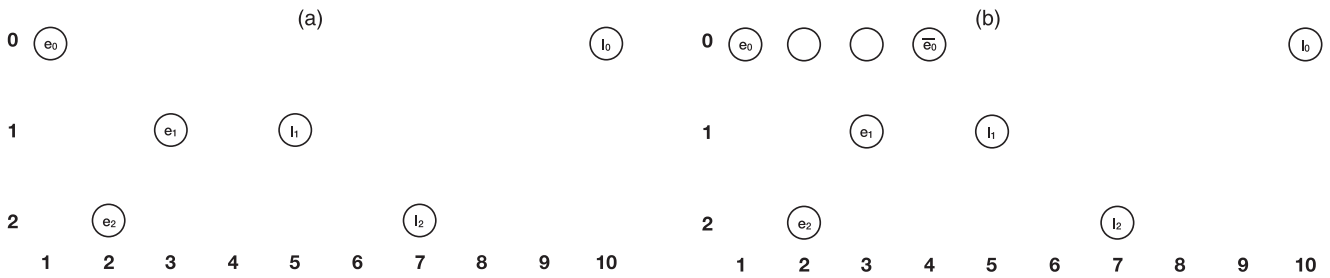
As with the makespan objective, we add to $\mathcal{N}_{\mathcal{T}}$ the nodes (i,e_i) and (i,l_i) for all locations (including the depot). Now, however, we also add to $\mathcal{N}_{\mathcal{T}}$ the nodes $(0,t), t = e_0+\epsilon, e_0+2\epsilon,\ldots,\bar{e}_0$. Here, ϵ represents the unit of time that 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 Figure 6. 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 four.

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' as 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, because there is no penalty for waiting at the depot, we associate the cost $\underline{c}_{00}(t) = 0$ for all $t = e_0+1,\ldots,\bar{e}_0$. 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.

With this definition of $\mathcal{D}_{\mathcal{T}}$ and $\underline{c}_{\mathcal{A}_{\mathcal{T}}}$, it is not difficult to prove that TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) yields a lower bound on the original problem for the duration objective. Proof of Lemma 1 may be adapted by only considering paths in $\mathcal{D}$ that correspond to feasible tours. (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, because the path corresponds to a feasible tour, its first arc $((u_0,t_0),(u_1,t_1))$ has $u_0 = 0$ and $t_0 \le l_0$. Thus, we can set $\underline{c}_{u_0u_1}(t_0') = -t_0' = -t_0 = c_{u_0u_1}(t_0)$. The remainder of the proof is the same. It is, furthermore, not difficult to verify that, if the solution to 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, TD-TSPTW($\bar{\mathcal{D}}^1_{\mathcal{T}}$), and TD-TSPTW($\bar{\mathcal{D}}^2_{\mathcal{T}}$) are solved with a duration objective (the networks are created the same way). Second, when solving TD-TSPTW($\mathcal{D}_{\mathcal{T}}$) returns a solution $((v_0,t_0),\ldots,(v_n,t_n))$ that does not contain a subtour, we evaluate its duration under all potential departure times from the depot.

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



Notes. (a) Minimizing makespan. (b) Minimizing duration.

Table 8. Performance of DDD-TD-TSPTW on Different Objectives

	Set 1: 952 instances			Set w100: 960 instances		
Objective	Slv	Time	Iter	Slv	Time	Iter
Makespan	952	10.94	4.62	960	1.22	4.11
Duration	930	143.45	9.77	955	106.20	9.87

Because waiting at locations other than the depot is not necessary in an optimal solution to the 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, because 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 8. We see that, although 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.

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 9 for each objective the average size of 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.

Table 9. Network Size in First and Last Iterations for Different Objectives

	Set 1: 952 instances		Set w100: 960 instances	
Objective	Initial $\lvert\mathcal{N}_{\mathcal{T}}\rvert$	Final $\lvert\mathcal{N}_{\mathcal{T}}\rvert$	Initial $\lvert\mathcal{N}_{\mathcal{T}}\rvert$	Final $\lvert\mathcal{N}_{\mathcal{T}}\rvert$
Makespan	53.34	188.80	53.50	191.15
Duration	111.88	452.24	94.50	438.89

8. Conclusions and Future Work

In this paper, we presented an algorithm, DDD-TD-TSPTW, based on the DDD framework that can solve instances of the TD-TSPTW when either one of two objectives is to be optimized: (1) a makespan objective or (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 that we proposed outperforms the best-performing algorithm in the literature. We also saw that, unlike existing methods, the algorithm that 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, but 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 the solution for the duration objective case, and it is a topic for future study.

The success of DDD-TD-TSPTW is, in part, because of the fact that the combination of a makespan objective and travel times satisfying the FIFO property implies that the travel costs are nondecreasing 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 that we intend to study are those in which travel times are stochastic.

Acknowledgments

The authors thank Arigliano et al. (2018) and Montero, Méndez-Dıaz, and Miranda-Bront (2017) for the exchange of test instances and ideas.

References

Abeledo HG, Fukasawa R, Pessoa AA, Uchoa E (2013) The time dependent traveling salesman problem: Polyhedra and algorithm. *Math. Programming Comput.* 5(1):27–55.

Albiach J, Sanchis JM, Soler D (2008) An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *Eur. J. Oper. Res.* 189(3):789–802.

Arigliano A, Ghiani G, Grieco A, Guerriero E, Plana I (2018) Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. Working paper, Università del Salento, Lecce, Italy.

Baker EK (1983) Technical note—an exact algorithm for the time-constrained traveling salesman problem. *Oper. Res.* 31(5):938–945.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.* 59(5):1269–1283.

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

Boland N, Hewitt M, Vu DM, Savelsbergh M (2017b) Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. Salvagnin D, Lombardi M, eds. *Integration of AI and OR Techniques in Constraint Programming*, Theoretical Computer Science and General Issues, vol. 10335 (Springer International Publishing, Cham, Switzerland), 254–262.

Bront JM, Méndez-Díaz I, Zabala P (2014) Facets and valid inequalities for the time-dependent travelling salesman problem. *Eur. J. Oper. Res.* 236(3):891–902.

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

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

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

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* 40(2):342–354.

Desrosiers J, Dumas Y, Solomon MM, Soumis F (1995) Time constrained routing and scheduling. Ball MO, Magnanti TL, Monma CL, Nemhauser GI, eds. *Network Routing*, Handbooks in Operations Research and Management Science, vol. 8 (Elsevier, Amsterdam), 35–139.

Figliozzi MA (2012) The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Res. Part E: Logist. Trans. Rev.* 48(3):616–636.

Fischetti M, Laporte G, Martello S (1993) The delivery man problem and cumulative matroids. *Oper. Res.* 41(6):1055–1064.

Gendreau M, Ghiani G, Guerriero E (2015) Time-dependent routing problems: A review. *Comput. Oper. Res.* 64:189–197.

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

Gouveia L, Voz S (1995) A classification of formulations for the (time-dependent) traveling salesman problem. *Eur. J. Oper. Res.* 83(1): 69–82.

Ichoua S, Gendreau M, Potvin J-Y (2003) Vehicle dispatching with time-dependent travel times. *Eur. J. Oper. Res.* 144(2):379–396.

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

Kara I, Koc ON, Altparmak F, Dengiz B (2013) New integer linear programming formulation for the traveling salesman problem with time windows: Minimizing tour duration with waiting times. *Optimization* 62(10):1309–1319.

Langevin A, Desrochers M, Desrosiers J, Gélinas S, Soumis F (1993) A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks* 23(7): 631–640.

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

Mahmoudi M, Zhou X (2016) 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 Res. Part B: Methodological* 89:19–42.

Melgarejo PA, Laborie P, Solnon C (2015) A time-dependent no-overlap constraint: Application to urban delivery problems. Michel L, ed. *Integration of AI and OR Techniques in Constraint Programming*, Lecture Notes in Computer Science, vol. 9075 (Springer, Cham, Switzerland), 1–17.

Méndez-Díaz I, Juan JMB, Toth P, Zabala P (2011) Infeasible path formulations for the time-dependent TSP with time windows. Adacher L, Flamini M, Leo G, Nicosia G, Pacifici A, Piccialli V, eds. *Proc. 10th Cologne-Twente Workshop Graphs Combinatorial Optim., Frascati, Itality*, 198–202.

Montero A, Méndez-Dıaz I, Miranda-Bront JJ (2017) An integer programming approach for the time-dependent traveling salesman problem with time windows. *Comput. Oper. Res.* 88:280–289.

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

Savelsbergh M (1985) Local search in routing problems with time windows. *Ann. Oper. Res.* 4(1):285–305.

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

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

UN (2014) Our urbanizing world. Technical report, United Nations Department of Economic and Social Affairs, New York.