Dynamic programming for the time-dependent traveling salesman problem with time windows[*]

Gonzalo Lera-Romero[†‡] Juan José Miranda-Bront[§¶] Francisco J. Soulignac[†‡]

gleraromero@dc.uba.ar, jmiranda@utdt.edu, fsoulign@dc.uba.ar

Abstract

The recent growth of direct-to-consumer deliveries has stressed the importance of last-mile logistics, becoming one of the critical factors in city planning. One of the key factors lies in the last-mile deliveries, reaching in some cases nearly 50% of the overall parcel delivery cost. Different variants of the the well-known Traveling Salesman Problem (TSP) arise naturally at the core of complex decision making processes within last mile logistics. The Time-Dependent Traveling Salesman Problem with Time Windows (TDTSPTW) is one of such variants, where the travel time between two customers is not assumed to be constant. The TDTSPTW is particular suited for distribution problems in large cities, as it effectively accounts the effects of congestion at the planning level, yielding more accurate distributions plans. In this paper, we develop a labeling-based algorithm for the TDTSPTW that incorporates state-of-the-art components from the related literature. We propose a new state-space relaxation specifically designed for the time-dependent context. Extensive computational experiments show the effectiveness of the overall approach and the impact of the new relaxation, outperforming several recent algorithms proposed for the TDTSPTW. In addition, we provide evidence showing that our approach also improves the recent results reported for the Minimum Tour Duration Problem.

1 Introduction

The recent growth of direct-to-consumer deliveries has stressed the importance of last-mile logistics, becoming one of the critical factors in city planning. According to different studies, one of the key factors lies in the last-mile deliveries, reaching in some cases nearly 50% of the overall parcel delivery cost (see, e.g. Savelsbergh and Van Woensel (2016); Joerss et al. (2016)). The *Traveling Salesman Problem* (TSP) arises naturally at the core of complex decision making processes within last-mile logistics. The goal of the TSP is to minimize the total distance traveled by a salesman who has to depart from a depot, visit a group of customers exactly once, and return back to the depot. To account for additional operational constraints, different variants of the TSP have been proposed in the literature during the last decades. Among them, one of the most relevant is the *TSP with Time Windows* (TSPTW), in which each customer must be visited within a specific time interval. When restricting to distribution problems in large cities, the congestion of the road network is another key aspect with a significant practical impact. Accounting effectively for the effects of congestion at the planning level results in more accurate distributions plans, while neglecting its impact can lead to infeasible routes violating the time windows. The *Time-Dependent TSPTW* (TDTSPTW) is a generalization of the TSPTW in which the travel time between two customers is not assumed to remain constant over the planning horizon.

[*]Supported by PICT ANPCyT grants PICT-2016-2677 and PICT-2018-2961 and Google LARA award.

[†]Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales. Departamento de Computación. Buenos Aires, Argentina.

[‡]CONICET-Universidad de Buenos Aires. Instituto de Investigación en Ciencias de la Computación (ICC). Buenos Aires, Argentina.

[§]Universidad Torcuato Di Tella, Buenos Aires, Argentina.

[¶]Consejo Nacional de Investigaciones Científicas y Técnicas.

To describe the TDTSPTW, let $D = (V, A)$ be a digraph with a set of vertices $V = \{0, 1, \ldots, n+1\}$. Vertices 0 and $n+1$ represent the starting and ending depot, respectively, while $V \setminus \{0, n+1\}$ represent the customers. Operations in the network D take place within a *planning horizon* $[0, T]$. Associated to each $i \in V$ there is a time interval $[a_i, b_i] \subseteq [0, T]$ that specifies the *time window* in which i must be visited; a_i and b_i are referred to as the *release* and *deadline* of i, respectively. To include time dependency into the network we consider the travel time model proposed by Ichoua et al. (2003), in which a *travel time function* τ_{ij} with domain $[0, T]$ is defined for each arc $(i, j) \in A$. For each $t \in [0, T]$, $\tau_{ij}(t)$ represents the time required to travel from i to j when i is departed from at time t. Each τ_{ij} is continuous, piecewise linear, and satisfies the *first-in first-out* (FIFO) *property*, i.e., $t + \tau_{ij}(t) \leq t' + \tau_{ij}(t')$ for every $0 \leq t \leq t' \leq T$. By considering the preprocessing techniques discussed by Lera-Romero and Miranda-Bront (2019), we also assume that the *no-waiting property* holds for τ_{ij}, meaning that $t + \tau_{ij}(t) \geq a_j$ for every $t \in [0, T]$. The family of travel time functions, however, need not satisfy the *triangle inequality*; thus, $\tau_{ik}(t)$ could be greater than $\tau_{ij}(t) + \tau_{jk}(t + \tau_{ij}(t))$ for some i, j, k.

Let $p = (v_0, \ldots, v_k)$ be a path of D and define the function $\delta_p^0(t) = t$ with domain $[0, T]$. By the FIFO and no-waiting properties, the earliest time at which v_{i+1} $(0 \leq i < k)$ can be reached, when a traversal of p is started at time $t \in [0, T]$, is given by the function $\delta_p^{i+1}(t) = \delta_p^i(t) + \tau_{i(i+1)}(\delta_p^i(t))$. By definition, δ_p^i is continuous, piecewise linear, and non-decreasing for every $0 \leq i \leq k$. For the sake of notation, we define $\delta_p = \delta_p^k$ as the *arrival function* of p. Thus, $\delta_p(t)$ is the earliest time at which v_k can be reached if v_0 is departed from at time t when traversing p. Say that $t \in [0, T]$ is a *feasible departing time* (for p) if $t \geq a_{v_0}$ and $\delta_p^i(t) \leq b_{v_i}$ for every $0 \leq i \leq k$. When t is a feasible departing time, $\delta_p(t)$ is a *feasible arrival time* (for p). Moreover, as it is always possible to wait at v_k, we also consider that t' is a *feasible arrival time* (for p) for every $\delta_p(t) < t' \leq b_{v_k}$. (This is nothing more than a technical requirement for the correct definition of the backward instance in Section 2.2. In short, as it is possible to reach a vertex before its release and then wait —as encoded by the travel time functions—, for symmetry it should be possible to reach a vertex and then wait.) By the no-waiting property, t is a feasible departing time if all the vertices of p are visited within their time windows when a traversal of p starts at time t. Similarly, t is a feasible arriving time if $t \leq b_{v_k}$ and all the vertices of p can be visited within their time windows in a traversal of p that ends at a time $t' \leq t$. Thus, the set $\mathrm{FDT}(p)$ of feasible departing times is included in $[a_{v_0}, b_{v_0}]$, while the set $\mathrm{FAT}(p)$ of feasible arrival times is included in $[a_{v_k}, b_{v_k}]$. Moreover, by the FIFO property, $\mathrm{FDT}(p)$ and $\mathrm{FAT}(p)$ are both intervals, either $\mathrm{FDT}(p)$ is empty or contains a_{v_0}, and either $\mathrm{FAT}(p)$ is empty or contains b_{v_k}. As a counterpart of δ_p, the *leave time function* $\lambda_p(t) = \max\{t' \in \mathrm{FDT}(p) \mid \delta_p(t') \leq t\}$ on the domain $\mathrm{FAT}(p)$ denotes the latest time at which v_0 must be departed from to reach v_k at a time $t' \leq t$ when traversing p. It is not hard to see that λ_p is also a continuous, piecewise linear, and non-decreasing function. We say that p is: *feasible* if $\mathrm{FDT}(p) \neq \emptyset$ (equivalently, $\mathrm{FAT}(p) \neq \emptyset$); a *tour* if $v_0 = 0$, $v_k = n+1$, and $k = n+1$; and *elementary* if p is feasible and $v_i \neq v_j$ for every $0 \leq i < j \leq k$. When p is a feasible path, its *makespan* is $m_p = \min \mathrm{FAT}(p) = \delta_p(a_{v_0})$, its *duration when* v_0 *is departed from at time* $t \in \mathrm{FDT}(p)$ is $\Gamma_p(t) = \delta_p(t) - t$, its *duration when* v_k *is visited at time* $t \in \mathrm{FAT}(p)$ is $\Delta_p(t) = t - \lambda_p(t)$, and its *duration* is $c_p = \min\{\Gamma_p(t) \mid t \in \mathrm{FDT}(p)\} = \min\{\Delta_p(t) \mid t \in \mathrm{FAT}(p)\}$.

With the above terminology, the goal in the TDTSPTW is to find an elementary tour of minimum duration. Note that the makespan can be minimized by letting $b_0 = a_0$; for the sake of notation, we refer to this restricted problem as the TDTSPTW$_m$. The *time-independent* version of the TDTSPTW, in which all the travel time functions are constant functions, is known as the *Minimum Tour Duration Problem* (MTDP), whereas the time-independent version of the TDTSPTW$_m$, where the cost function is given by the travel times, is the classical TSPTW.

1.1 Literature review

The TSP and TSPTW have been extensively studied in the literature along with other less popular variants, such as the MTDP. In general, the most effective approach for solving the TSP in an exact manner is the use of tailored *branch and cut* (BC) algorithms that apply families of inequalities specifically defined for the problem (Dantzig et al., 1954; Applegate et al., 2007). Regarding the TSPTW, BC algorithms have also been

applied successfully. Ascheuer et al. (2000) present a polyhedral study for the TSPTW, that is evaluated experimentally in Ascheuer et al. (2001) within three different BC algorithms. The best configuration is capable of solving real-world instances with up to 233 vertices. More recently, Dash et al. (2012) proposed the *time bucket formulation* based on a partition of the time windows into the so-called *buckets*. They derive new families of valid inequalities generalizing the ones considered in Ascheuer et al. (2001), that produce stronger lower bounds and allow the solution to optimality of several previously unsolved benchmark instances.

Alternative techniques such as Dynamic Programming (DP) have also been successfully implemented for the TSPTW. Christofides et al. (1981) introduced the so-called *state-space relaxations* to compute fast lower bounds for the TSPTW and other routing problems. Savelsbergh (1992) designed an algorithm for the MTDP and conducted a comparative with the TSPTW and other variants. Later on, Dumas et al. (1995) proposed the first DP algorithm for the TSPTW, including an extensive analysis in order to reduce the number of enumerated states. The results were promising as they solved instances with tight time windows and up to 200 vertices. Mingozzi et al. (1997) proposed a different DP algorithm for the TSPTW with Precedence Constraints (TSPTW-PC) that builds upon the idea of Christofides et al. (1981) and uses lower bounds to reduce the number of explored states; this idea is known as *bounding*. They were able to solve randomly generated instances with up to 120 vertices.

The *ng-path* state-space relaxation is introduced by Baldacci et al. (2011) to solve the *Vehicle Routing Problem with Time Windows* (VRPTW). This relaxation became a standard for computing lower bounds for routing problems, as it provides tight lower bounds with a considerable reduction in computation times. Baldacci et al. (2012) adapt the *ng*-path relaxation to solve the TSPTW, where they introduced the *ng-tour* and *ngL-tour* relaxations. Also, they propose a *penalty method*, based on a Lagrangean relaxation, to improve the lower bounds and reduce the state-space graph of the exact DP. Combining these ideas, they are able to solve all but one instance in the literature for the TSPTW. In a follow up paper, Tilk and Irnich (2017) adapt this scheme to tackle the MTDP. They introduce two new relaxations, called 1*res* and 2*res*, that can be combined with the *ng*-tour and *ngL*-tour. Moreover, they improved the general framework by applying a technique called *Dynamic Neighbour Augmentation* (DNA) to obtain similar lower bounds in less computation time.

There have been several attempts to capture congestion using different models; see e.g. Gendreau et al. (2015). The *travel speed model* introduced by Ichoua et al. (2003) has been widely adopted by the routing optimization community. This model assumes that operations take place on a planning horizon that is partitioned into time zones where the average travel speed of each arc remains constant. The travel time function of each arc is then obtained by processing the speeds of the different time zones. As discussed in Section 1, each travel time function is continuous, piecewise linear, and satisfies the FIFO property. Although the recent literature on time-dependent problems includes an increasing number of different approaches and variants, we restrict mainly to those involving exact algorithms.

Regarding multi-vehicle variants, Dabia et al. (2013) develop a *branch and price* (BP) algorithm for the Capacitated TDVRPTW (TDCVRPTW). The pricing problem is a Time Dependent Elementary Shortest Path Problem with Resource Constraints (TDESPPRC), and is tackled with a DP algorithm that incorporates the bidirectional search proposed by Righini and Salani (2006), being able to solve consistently instances with 50 customers. Lera-Romero et al. (2019) improve this algorithm by incorporating cutting planes, partial domination and other techniques that significantly improve the performance of the DP algorithm. The approach shows good results on instances with up to 100 customers, solving 82 previously unsolved instances. Sun et al. (2018b) extend the original approach in Dabia et al. (2013) to solve the TDCVRPTW with Pickup and Deliveries. The TDESPPRC resulting from the pricing problem is studied in detail by Sun et al. (2018a), who conclude that the DP approach outperforms a MILP solved by a general-purpose solver.

Single-vehicle variants have also received some attention recently. A first stream of research is motivated by the ideas proposed in Cordeau et al. (2014) for the TDTSP$_m$ (i.e., the TDTSPTW$_m$ with no time windows). The methods build on the solution of an auxiliary time-independent TSP to compute a lower bound, that is later incorporated into a BC algorithm. Tight lower bounds are obtained under certain traffic conditions, allowing to solve instances with up to 40 vertices. Arigliano et al. (2018a) and recently Adamo et al. (2020) improve the original method to compute tighter lower bounds within a new BB algorithm.

A similar sequence is observed for the TDTSPTW$_m$, where the approach by Cordeau et al. (2014) is first adapted by Arigliano et al. (2015) and later improved by Arigliano et al. (2018b), being able to solve instances with up to 40 customers. Simultaneously, more general models for the TDTSPTW have been studied, focusing not only on the makespan but also the duration of the route. Montero et al. (2017) improve the MILP proposed in Sun et al. (2018a) and develop a BC algorithm incorporating well-known valid inequalities from the TSPTW. Computational experiments over the instances used in Arigliano et al. (2015) demonstrate that the approach solves more instances in less computation time. Vu et al. (2019) propose a *Dynamic Discretization Discovery* (DDD) algorithm for the TDTSPTW and the TDTSPTW$_m$. They are able to solve all instances to optimality when minimizing makespan, and most of the instances when minimizing duration, in small computation times. Recently, Lera-Romero and Miranda-Bront (2018, 2019) extend the approach by Montero et al. (2017) by reducing the size of the MILP formulation while preserving the quality of the LP relaxation, and derive new families of valid inequalities specifically crafted to exploit the time-dependency structure. The resulting BC algorithm is evaluated on benchmark instances for the TDTSPTW, TDTSPTW$_m$, and other *profitable* variants. The algorithm outperforms the one by Vu et al. (2019) for the TDTSPTW instances, while it remains comparable for the TDTSPTW$_m$. Overall, the algorithms devised by Arigliano et al. (2018b), Vu et al. (2019) and Lera-Romero and Miranda-Bront (2019) represent so far the state-of-the-art regarding exact algorithms for the TDTSPTW and the TDTSPTW$_m$. Due to simultaneity, there is no complete benchmark for these methods over all the available instances. However, the results suggest that the methods in Vu et al. (2019) and Lera-Romero and Miranda-Bront (2019) are more robust on instances having tighter time windows, while the BB developed by Arigliano et al. (2018a) is more suitable for instances with larger time windows. Additionally, to the best of our knowledge, no DP algorithm was proposed to tackle either the TDTSPTW or the TDTSP, as opposed to their time-independent counterparts.

Finally, we highlight some recent developments proposed for other relevant time-dependent problems. Gmira et al. (2019a) and Gmira et al. (2019b) investigate static and dynamic TDVRPs on a road network, instead of considering a customer-based graph. Additionally, Taş et al. (2016) and Cacchiani et al. (2019) study the TSP with (eventually non-linear) time-dependent service times.

1.2 Contributions

In this article we propose a new labeling-based algorithm to solve the TDTSPTW that incorporates many state-of-the-art features from the literature. Our algorithm builds upon the work by Tilk and Irnich (2017) for the MTDP that, in turn, is based on the work by Baldacci et al. (2012) for the TSPTW. As argued by Tilk and Irnich (2017), adapting the resource extension functions to handle the duration objective instead of the makespan objective is not an easy task. One of the main problems is that more resources are required to describe the duration of a path. Further extending these functions to incorporate time-dependency is by no means trivial, as the duration depends on the departure time at each vertex of a path. Moreover, most of the techniques involved in these algorithms must be adapted to handle the time-dependent scenario. As a result, more complex algorithms have to be described and implemented.

Beside extending the algorithm to deal with time-dependent costs, we also propose three additional improvements over the algorithm by Tilk and Irnich (2017). First, we introduce the *ti*-relaxation that is specifically designed to decrease the execution time for time-dependent instances, trying to compromise as little as possible the quality of the bounds. The computational results show that this new relaxation is fundamental to solve some of the most difficult instances, and it even allows the solution of instances without time windows. Second, we define new feasibility rules for the time-dependent *ngL*-tour relaxation, that apply to the time-independent case as well. These rules increase the number of infeasible labels that may be safely discarded, leading to better bounds and faster running times. Finally, we propose a different scheme for running the labeling algorithms that is robust when lower and upper bounds are loose. The algorithms by Baldacci et al. (2012) and Tilk and Irnich (2017) are very sensible to the quality of the lower bound lb obtained by the relaxations, as they guess an upper bound ub close to lb to phatom many states of the labeling algorithm while applying tight completion bounds. If the guess of ub is incorrect, then lb and ub are updated and the algorithm is restarted. Thus, if the gap between lb and the optimal value is large,

several iterations of the algorithm are required. Obtaining a tight lower bound in the time-dependent case is harder than in its time-independent counterpart, thus many iterations should be expected when this scheme is applied. The algorithm by Tilk and Irnich (2017) is also highly dependent on the quality of the upper bound, as it is used to discard labels when the dynamic neighbor augmentation is applied. In particular, the labeling algorithm for this step is monodirectional, and labels are discarded by using completion bounds that surpass the upper bound. Tilk and Irnich (2017) obtain such an upper bound by running a good (and costly) heuristic before executing their algorithm. For the time-dependent case, these tight upper bounds are not available. Thus, in this article we avoid bounding in the dynamic neighbor augmentation step and, instead, we apply a bidirectional labeling algorithm. On the other hand, once the lower bound is known after applying relaxations, we run the exact monodirectional labeling algorithm only once, using the completion bounds to select the order in which the labels are enumerated. Our experiments show that the obtained algorithm is robust with respect to the quality of the upper bound.

As discussed in Section 1, our algorithm can be readily applied to the MTDP by letting each travel time function be a constant. The obtained algorithm is bloated, as tailored implementations can avoid the maintenance of heavier labels and data structures. Nevertheless, our algorithm is able to solve some previously unsolved instances. We believe that the efficiency of our algorithm is not only the result of the improvements described in the previous paragraph, but also on how the labels are represented within the labeling algorithm. As shown in Section 2.1.2, the partial dominance criterion applied by our algorithm is also effective to discard some labels that are not discarded by the algorithm by Tilk and Irnich (2017).

The remaining of the paper is organized as follows. In Section 2 we introduce the exact forward labeling algorithm for solving the TDTSPTW, its backward counterpart, and the bidirectional search. In Section 3 we describe a preprocessing step used to reduce the size of the instance and to compute some additional information that is required when completion bounds are calculated. The implementation of the completion bound function is defined in Section 4, together with some acceleration techniques. The numerical experiments and their results are reported in Section 6. Finally, in Section 7 the final conclusions are presented along with future research lines that open after this work.

2 Exact dynamic programming algorithm

Forward labeling algorithms for time-independent problems usually work by maintaining a single label for each elementary path that begins at the depot 0. Starting from an initial label that represents the path (0), the algorithm iteratively extends each enumerated label to consider all the elementary paths that visit an additional vertex. A label L that represents a path p ending at a vertex v is defined by v, the information required to rebuild p, and a set of resources r that can be used to decide if L is *dominated*. Specifically, L is dominated by a label M representing a path p_1 when $p_1 + p_2$ is an elementary tour with a cost no greater than the cost of $p + p_2$, for every elementary tour $p + p_2$. Those labels that are dominated can be safely discarded from the enumaration without affecting the output of the algorithm. Deciding if L is dominated by M can be as hard as enumerating all the possible extensions of L. For this reason, efficient domination rules that are sufficient to determine that L is dominated are applied.

A simple way to generalize the above labeling algorithm to the time-dependent case is to consider that p is encoded by a *full label* L. Recall that $\text{FAT}(p)$ is the set of feasible times at which the last vertex v of p can be visited. A full label L is a function with domain $\text{FAT}(p)$ such that $L(t)$ is the time-independent label that represents the state of p when v is visited at time t. In this setting, the concept of domination can be generalized to that of *full domination*: L is fully dominated by a label M encoding a path q when $\text{FAT}(p) \subseteq \text{FAT}(q)$ and $L(t)$ is dominated by $M(t)$ for every $t \in \text{FAT}(p)$. (Luo et al. (2017) refer to full domination as pair domination.) If L is fully dominated by M, then it can be safely discarded. This is the approach used by Dabia et al. (2013) to solve the TDESPPRC for the pricing problem of the TDCVRPTW. In the particular case of the TDTSPTW, the full label representing p is a tuple $L = (\text{prev}_p, v, S, \Delta_p)$, where prev_p is a pointer to the label encoding $p - v$ and S is the set of vertices visited by p. Thus, L is fully dominated by $M = (\text{prev}_q, v, S, \Delta_q)$ if $\text{FAT}(p) \subseteq \text{FAT}(q)$ and $\Delta_q(t) \leq \Delta_p(t)$ for every $t \in \text{FAT}(p)$.

Clearly, L can be safely discarded if there exists a set of labels $\mathcal{M}$ such that, for every $t \in \text{FAT}(p)$,

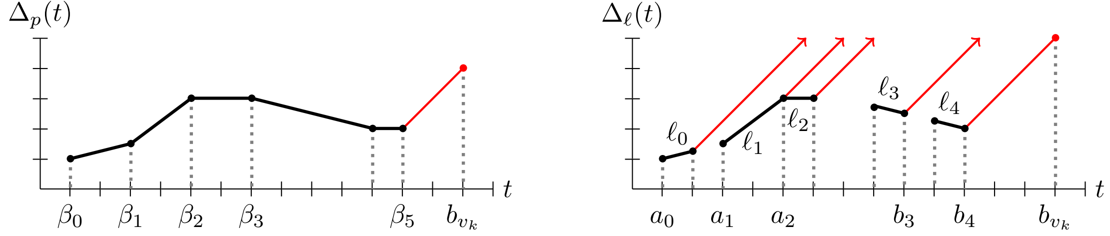


Figure 1: To the left, the duration function for a path $p = (0, v_1, \ldots, v_k)$ having breakpoints $\beta_0, \ldots, \beta_6$ and a red waiting piece; by definition, $\beta_0 = \delta_P(\min \mathrm{FAT}(P))$, $\beta_5 = \delta_P(\max \mathrm{FAT}(P))$, and $\beta_6 = b_{v_k}$. To the right, the duration function of five labels $\ell_0, \ldots, \ell_4$ for p, as maintained at some step of the algorithm. Here, $\mathrm{dom}(\ell_i) = [a_i, b_i]$, the black portion of each function depicts the duration within dom, and each red piece of ℓ_i shows the function from b_i to b_{v_k}.

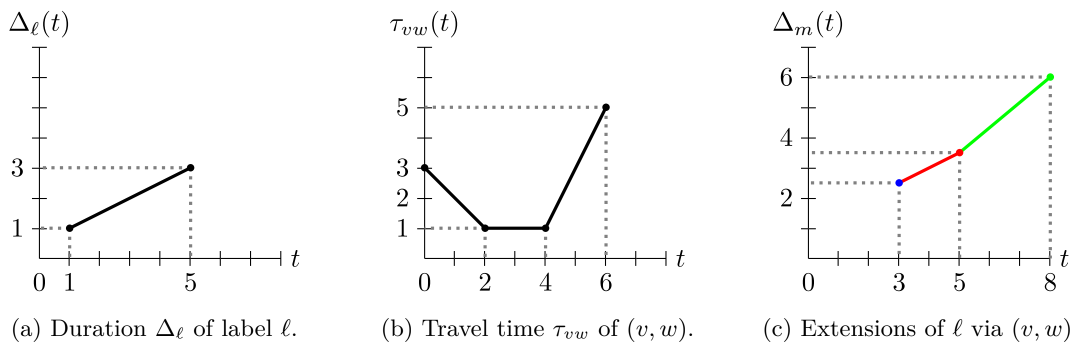
$L(t)$ is dominated by $M(t)$ for some $M \in \mathcal{M}$. Yet, L is not discarded when a full domination criterion is applied unless one of the labels in $\mathcal{M}$ fully dominates L. In a *partial dominance criterion*, $t \in \mathrm{FAT}(p)$ can be discarded from the domain $\mathrm{dom}(L)$ of L when $L(t)$ is dominated, even if $L(t')$ is not dominated for some $t' \in \mathrm{FAT}(p)$. In this case, $\mathrm{dom}(L)$ is a subset of $\mathrm{FAT}(p)$ that need not be a unique interval, but instead a collection of disjoint intervals. If $\mathrm{dom}(L)$ is finally emptied, then L is discarded. Lera-Romero et al. (2019) describe three ways to implement a partial dominance criterion, as proposed by different authors (Ioachim et al., 1998; Liberatore et al., 2011; Luo et al., 2017; Spliet et al., 2018). Each of these ways is equivalent in the sense that they discard the same points from $\mathrm{dom}(L)$. However, each implementation has its own strengths and weaknesses regarding the efficiency of the final algorithm. Ultimately, each implementation depends on how the labels are defined. The algorithm that we implement to solve the TDTSPTW relies on the so-called linear labels, as they yield a simple and convenient way to implement the bounding technique.

We begin this section by describing an exact forward labeling algorithm for the TDTSPTW. For this, we specify how linear labels are defined, what are the extension rules to ensure that only labels representing elementary paths are propagated, and what are the domination rules applied to decrease the number of enumerated labels. A comparison between our algorithm and the one devised by Tilk and Irnich (2017) is also included. Next, we show how an instance of the TDTSPTW can be transformed to generate elementary backward paths while using our same forward algorithm and, finally, we describe an exact bidirectional algorithm for the problem.

2.1 Forward algorithm

Let $p = (v_0, \ldots, v_k)$ be an elementary path extended from the depot $v_0 = 0$. Recall that λ_p is a continuous and piecewise function, thus $\Delta_p(t) = t - \lambda_p(t)$ is continuous and piecewise linear as well. Every continuous and piecewise linear function f is characterized by a sequence of *breakpoints* $\beta_0 < \ldots < \beta_j$ such that the restriction f_i of f to the domain $[\beta_i, \beta_{i+1}]$ is linear for $0 \leq i < j$. We refer to f_i as being a *piece* of f. Figure 1 depicts Δ_p for a path p having seven breakpoints, where the red piece corresponds to a mandatory waiting arising from $\delta_p(t) < b_{v_k}$ for $t = \max \mathrm{FDT}(p)$. If existing, we refer to the piece of Δ_p in which waiting is mandatory as being a *waiting piece*. By definition, the slope of the waiting piece is always 1.

A *forward linear label* of p is a tuple $\ell = (\mathrm{prev}, v_k, S, \Delta_\ell)$ where prev is a pointer to a forward linear label of $(v_0, \ldots, v_{k-1})$, $S = \{v_0, \ldots, v_k\}$ is the set of visited vertices, and Δ_ℓ is the restriction of a piece of Δ_p to a nonempty domain $\mathrm{dom}(\ell) = [a(\ell), b(\ell)]$ (Figure 1). Note that $a(\ell)$ and $b(\ell)$ denote the earliest and latest times at which v_k can be reached with respect to ℓ. For the sake of notation, we use $p(\ell)$, $\mathrm{prev}(\ell)$, $v(\ell)$ and $S(\ell)$ to denote p, prev, v_k, and S, respectively. Moreover, we simply refer to ℓ as being a *forward label*, omitting the term linear, and we also drop the term forward when no confusions are possible. By definition, $p(\ell)$ admits an infinite number of labels. The algorithm ensures that the label m referenced by $\mathrm{prev}(\ell)$ is such that ℓ can be obtained from m via an extension (see below). This is not a fundamental property, as



(a) Duration Δ_ℓ of label ℓ. (b) Travel time τ_{vw} of (v, w). (c) Extensions of ℓ via (v, w).

Figure 2: Extension of a label ℓ through an arc $(v, w) \in A$, for $v = v(\ell)$. Three labels, generically denoted by m, are obtained.

the sole purpose of $\text{prev}(\ell)$ is to be able to rebuild $p(\ell)$. Thus, as specified, $\text{prev}(\ell)$ can reference any label of $(v_0, \ldots, v_{k-1})$.

For each elementary path p, the labeling algorithm keeps a sequence $L = \ell_0, \ldots, \ell_j$ of labels of p such that $b(\ell_i) \leq a(\ell_{i+1})$ for every $1 \leq i < j$ (Figure 1). Moreover, either ℓ_i and ℓ_{i+1} belong to different pieces of Δ_p or $b(\ell_i) < a(\ell_{i+1})$. Thus, every $t \in \text{FAT}(p)$ belongs to at most two labels in L and t belongs to two labels only if t is a breakpoint of Δ_p. As conceptually described at the beginning of this section, L can be though of as a function whose domain is $\text{dom}(L) = \bigcup_i \text{dom}(\ell_i) \subseteq \text{FAT}(p)$. In this interpretation, $L(t)$ is a *time-independent label* $(\text{prev}(\ell), v(\ell), S(\ell), \Delta_\ell(t))$, where ℓ is a label such that $t \in \text{dom}(\ell)$. (This interpretation holds for ℓ as well, as $\ell(t)$ is a time-independent label.) Then, taking into account that $\text{dom}(L)$ need not be equal to $\text{FAT}(p)$, we refer to L as being a *partial label* representing p. We remark, however, that we use partial labels as a concept; the algorithm has no knowledge about which labels correspond to the same partial label. Figure 1 shows a partial label with five black labels representing an elementary path p. The labeling algorithm never keeps a label whose domain is included in the waiting piece. Instead, we just assume that every label ℓ can be continued with a waiting piece that has slope 1, as depicted by the red lines in Figure 1. Thus, even though $\text{dom}(\ell)$ is included in the domain of a piece, the set $\text{FAT}(\ell) = [a(\ell), b_{v(\ell)}]$ of *feasible arriving times* for ℓ always contains $b_{v(\ell)}$. Therefore, $\Delta_\ell(t) = \Delta_\ell(b(\ell)) + (t - b(\ell))$ represents the duration of ℓ when $v(\ell)$ is reached at $t > b(\ell)$.

Starting with the *initial label* $\ell_0 = (\bot, 0, \{0\}, \Delta_{(0)})$, where $\Delta_{(0)}(t) = 0$ for $t \in [a_0, b_0]$, the algorithm iteratively *extends* each label ℓ through all the arcs $(v(\ell), w) \in A$. The result of the extension of ℓ through the arc $(v(\ell), w)$ is a set of labels L, each of which is extended later. The set L is empty when $w \in S(\ell)$, as $p(\ell) + w$ is not elementary. If $w \notin S(\ell)$, then a label $m = (\ell, w, S \cup \{w\}, \Delta_m)$ is generated for each piece $\tau \in \tau_{v(\ell)w}$ with domain $[a(\tau), b(\tau)]$ such that $[a(\tau), b(\tau)] \cap \text{dom}(\ell)$ is a nonempty interval $[a, b]$ with $\Delta_\ell(a) + \tau(a) \leq b_w$. Specifically, Δ_m is a function with domain $[a + \tau(a), \min\{b + \tau(b), b_w\}]$ such that, for t in such domain, $\Delta_m(t) = t - \lambda_{p(\ell+w)}(t) = \Delta_\ell(t') + \tau(t')$ where $t' \in [a, b]$ is the maximum such that $t' + \tau(t') \leq t$. In colloquial terms, the domain of Δ_m corresponds to the set of feasible arriving times to w in which v is visited inside $\text{dom}(\ell)$, while the image of Δ_m is the minimum duration required to arrive w at each feasible arriving time.

An example of the extension of a label ℓ through an arc $(v(\ell), w) \in A$ is depicted in Figure 2. Figure 2a shows Δ_ℓ, whereas Figure 2b illustrates the travel time function $\tau_{v(\ell)w}$. As in Figure 2c, three labels m_1 (blue), m_2 (red) and m_3 (green) are obtained after the extension. Note that Δ_{m_1} is a single point overlapping Δ_{m_2}. This happens because if v is departed from at any time $t \leq 2$, then w is reached at time 3. By the FIFO property, the duration at the feasible arriving time 3 is minimized when $v(\ell)$ is visited at time 2, because this is the latest time that we can visit $v(\ell)$ to reach w at time 3. Therefore, $\Delta_{m_1}(t) = \Delta_\ell(2) + \tau_{v(\ell)w}(2) = 2.5$. Regarding m_2, observe that the domain of $\Delta_{m_2}(t)$ is the interval $[3, 5]$ which are the feasible arrival times at vertex w when restricted to the second piece. For a reference, $\Delta_{m_2}(5) = 3.5 = \Delta_\ell(4) + \tau_{v(\ell)w}(4)$.

For $t \in \text{dom}(\ell)$, let $\mathcal{E}_t(\ell)$ be the set of paths such that $p \in \mathcal{E}_t(\ell)$ if $p(\ell) + p$ is an elementary tour and

$t \in \mathrm{FDT}(p)$. That is, $p \in \mathcal{E}_t(\ell)$ when $p(\ell)$ can be extended into an elementary tour that visits v at time t. Say that t is *tour-infeasible* if $\mathcal{E}_t(\ell) = \emptyset$. If $\mathcal{E}(\ell) = \bigcup\{\mathcal{E}_t(\ell) \mid t \in \mathrm{dom}(\ell)\}$ is empty, then ℓ is *tour-infeasible*. Clearly, tour-infeasible times can be safely removed from $\mathrm{dom}(\ell)$, while tour-infeasible labels can be completely discarded. Recall that, in our problem, the triangle inequality need not be satisfied by the travel time functions. For vertices $v, w \in V$, let $\mathrm{LDT}(v, w)$ be the latest time v can be departed from to reach w before its deadline through any path. Clearly, if $w \notin S(\ell)$ is such that $t > \mathrm{LDT}(v(\ell), w)$, then t is tour-infeasible. Therefore, we can apply the next rule to reduce the size of $\mathrm{dom}(\ell)$.

Rule 1 (Tour-feasibility)**.** *For every label ℓ, $\mathrm{dom}(\ell)$ can be restricted to $[0, t]$ for $t = \min\{\mathrm{LDT}(v(\ell), w) \mid w \in V \setminus S(\ell)\}$.*

Rule 1 is a generalization of the classic feasibility rule for the time independent problem. Indeed, t is discarded from ℓ when the time-independent label $\ell(t)$ cannot be extended into a feasible tour. As noted by Baldacci et al. (2012), this rule tends to be highly effective in presence of tight time windows. Observe that if $\mathrm{dom}(\ell)$ is emptied by the application of this rule, then ℓ is tour-infeasible, thus it is discarded.

As Rule 1, the rule for partial domination is obtained by generalizing the rules for domination from the time-independent context. Briefly, ℓ is *dominated* by m at $t \in \mathrm{FAT}(\ell)$ if $t \in \mathrm{FAT}(m)$ and $\ell(t)$ is dominated by $m(t)$. We stress that t need not belong to $\mathrm{dom}(\ell) \cup \mathrm{dom}(m)$, i.e., the visit of $v(\ell)$ at time t can occur by reaching before t and waiting for either ℓ or m. That is, $p(m) + p$ is an elementary tour having less duration than $p(\ell) + p$ when visited at time t, including waiting times, for every path p such that $p(\ell) + p$ is an elementary tour that traverses $v(\ell)$ at time t. As computing every feasible extension of ℓ is expensive, we apply the following rule to decide if ℓ is dominated at a time t.

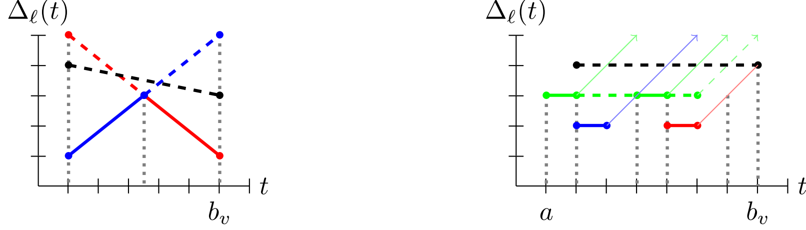
Rule 2 (Partial domination)**.** *A label ℓ is dominated by another label m at time $t \in \mathrm{FAT}(\ell)$ if $t \in \mathrm{FAT}(m)$, $v(m) = v(\ell)$, $S(m) = S(\ell)$, and $\Delta_m(t) \leq \Delta_\ell(t)$.*

From now on, and for the sake of simplicity, whenever we say that ℓ is dominated by m at time t we mean that m satisfies the conditions in Rule 2. Each label ℓ kept by the algorithm is such that ℓ is not dominated by m at time t, for every $t \in \mathrm{dom}(\ell)$. In other words, if a label ℓ that is obtained after an extension is dominated by another label m at a time t, then t is discarded from $\mathrm{dom}(\ell)$ and ℓ is ultimately discarded if $\mathrm{dom}(\ell)$ gets emptied. Consequently, for $v \in V$ and $S \subseteq V$, if $L = \ell_0, \ldots, \ell_j$ is the sequence of non-discarded labels with $v(\ell_i) = v$ and $S(\ell_i) = S$ such that $a(\ell_i) \leq a(\ell_{i+1})$ $(0 \leq i < j)$, then $b(\ell_i) \leq a(\ell_{i+1})$ and, hence $\mathrm{dom}(\ell_i) \cap \mathrm{dom}(\ell_{i+1})$ has at most one point.

If a label ℓ is not dominated at every $t \in \mathrm{dom}(\ell)$ by any other single label, then all of $\Delta_{p(\ell)}$ is kept under a full domination criterion, even if ℓ is dominated at every $t \in \mathrm{dom}(\ell)$ by the union of two or more labels. Moreover, even if ℓ cannot be completely discarded, many of the pieces of $\Delta_{p(\ell)}$ are removed with partial domination, effectively reducing the number of enumerated pieces (Figure 3a). On the contrary, if some portions of $\mathrm{dom}(\ell)$ are dominated, then ℓ must be split into several disjoint labels to fulfill the invariant of discarding dominated times (Figure 3b). Even if this seems as a drawback at first, the effectiveness of partial domination outperforms the exponential growth of labels resulting from a weaker domination rule.

2.1.1 Implementation

Let the *tour-duration* of a label ℓ be $\bar{c}(\ell) = \min\{\Delta_\ell(t) + \Gamma_p(t) \mid t \in \mathrm{dom}(\ell) \text{ and } p \in \mathcal{E}_t(\ell)\}$, where $\bar{c}(\ell) = \infty$ if $\mathcal{E}(\ell) = \emptyset$. Colloquially, $\bar{c}(\ell)$ is the minimum duration of a tour that traverses $p(\ell)$ and visits $v(\ell)$ at a time $t \in \mathrm{dom}(\ell)$. The labeling algorithm, depicted in Algorithm 1, receives two parameters in addition to the input instance: cb and *ub*. The *completion bound* cb is a function that, given a label ℓ, returns a lower bound of $\bar{c}(\ell)$ that is greater than or equal to $c_{p(\ell)} = \min\{\Delta_\ell(t) \mid t \in \mathrm{dom}(p)\}$. Thus, if $p(\ell)$ is a tour for which its duration is minimized when $n + 1$ is arrived at a time $t \in \mathrm{dom}(\ell)$, then $c_{p(\ell)} \leq \mathrm{cb}(\ell) \leq \bar{c}(\ell) = \Delta_\ell(t) = c_{p(\ell)}$ is the duration of $p(\ell)$. The *upper bound* parameter *ub* can be any value, but it is supposed to be an upper bound of the duration c^* of any optimal elementary tour. By Rule 3, assuming $ub \geq c^*$, the algorithm discards any label ℓ with $\mathrm{cb}(\ell) > ub$ to reduce the number of enumerated states (Step 14). However, if $ub < c^*$, then every label representing a tour is discarded and, hence, no optimal elementary tour is found.



(a) Reduction of domains via domination (b) Fragmentation of labels by domination

Figure 3: Under full domination, all the labels in a and b are kept untouched. Under partial domination, the dashed portions are removed, thus black labels are completely discarded while the green label in b is partitioned into three labels because of partial domination. In b, thin grayed lines correspond to waiting pieces that do not belong to the domain. Thus, $\bigcup_\ell \text{dom}(\ell)$ is not an interval, although $\bigcup_\ell \text{FAT}(\ell) = [a, b_v]$.

Section 4 discusses how to compute good completion bounds; ub can be obtained by executing any heuristic. Yet, to obtain a correct algorithm, it suffices to consider $\text{cb}(\ell) = c_{p(\ell)}$ and $ub = \infty$.

Rule 3 (Bounding)**.** *If ub is an upper bound on the optimal tour duration and $\text{cb}(\ell) > ub$, then $p(\ell) + p$ is not an optimal tour for any $p \in \mathcal{E}(\ell)$, thus ℓ can be safely discarded.*

Algorithm 1 stores labels in a map $\mathcal{L}$, where each label is augmented with an additional field for a completion bound as described above. Let $(|S(\ell)|, v(\ell), S(\ell))$ be the *core* of a label ℓ. For each core C, $\mathcal{L}(C)$ keeps a sequence $\ell_0, \ldots, \ell_j$ of labels with core C, where $b(\ell_i) \leq a(\ell_{i+1})$ for $0 \leq i < j$. Recall that, by Rule 2, a label can only be dominated by another label with its same core. At each iteration (Steps 4–15), the algorithm simultaneously *processes* the sequence L of not-yet-processed labels with minimum completion bound b and core $C = (k, v, S)$. By the way b is defined, if C is the first core processed with $k = n + 1$, then every label in L represents a tour with duration c^*. Thus, the algorithm is stopped (Step 8) and no label ℓ with $\text{cb}(\ell) > c^*$ is ever processed. Efficiency is the reason why all the labels in L are processed simultaneously. Indeed, the *extension* (Step 10) is linear on $|L|$ and the number of pieces in τ_{vw}, *domination* (Step 13) is linear on $|M|$ and $|\mathcal{L}(k + 1, w, S + \{w\})|$, and the analogous occurs with bounding (Step 14). Thus, we seek to minimize the number of times these operations are executed.

Algorithm 1. Forward labeling algorithm

Input: An instance $(D = (V, A), o = 0, d = n + 1, a, b, T, \tau)$, an upper bound ub, and a completion bound cb.

Output: If existing, an elementary tour p of minimum duration $c_p \leq ub$; otherwise, an error message.

1: Let $\ell_0 = (\bot, o, \{o\}, [0, T] \to 0)$ be the initial label, updated with $\text{cb}(\ell_0)$.

2: Let q be a priority queue initialized with $(\text{cb}(\ell_0), 1, o, \{o\})$.
// Tuples from q are popped in lexicographic ascending order.

3: Let $\mathcal{L}$ be a map initialized with $(1, o, \{0\}) \to \{\ell_0\}$.

    //$\mathcal{L}(C)$ *returns a sequence of labels* $\ell_0, \ldots, \ell_j$ *with core* C*, where* $b(\ell_i) \leq a(\ell_{i+1})$ *for* $0 \leq i < j$.
4: **While** $q \neq \emptyset$:
5: Pop (b, k, v, S) from the top of q.
6: Let L be the subsequence of labels ℓ in $\mathcal{L}(k, v, S)$ with $\text{cb}(\ell) = b$.
7: If $L = \emptyset$, then go to the next iteration.
8: If $k = n + 1$, then halt and return a tour represented by a label in L.
9: **For each** $(v, w) \in A$ such that $w \notin S$:
10: Let M be the sequence of labels that are obtained by extending the labels in L.
11: **For each** $m \in M$: shrink $\text{dom}(m)$ with Rule 1, and discard m if $\text{dom}(m) = \emptyset$.
12: Merge M into N, where $\mathcal{L} = \mathcal{L}(k + 1, w, S + \{w\})$.
13: **For each** $\ell \in N$: apply Rule 2 to remove all the dominated times from $\text{dom}(\ell)$.
14: **For each** $m \in M \cap N$: update m with $\text{cb}(m)$ and discard m if $\text{cb}(m) > ub$.
15: Insert $(\text{cb}(m), k + 1, w, S + \{w\})$ into q for every $m \in M \cap N$.
16: Inform that no elementary tour p with $c_p \leq ub$ exists.

Observe that there could exist two labels ℓ and m that dominate each other at different points of their respective domains (Figure 3a). Let C be the core of ℓ (and m). If ℓ is inserted first into $\mathcal{L}$, then ℓ is *corrected* when m is inserted (Step 13), to keep the ordering invariant of $\mathcal{L}(C)$. As a result of this correction, ℓ gets transformed into several labels, one for each undominated piece of its domain (Figure 3b). By definition, $\text{cb}(\ell') \geq \text{cb}(\ell)$ for each label ℓ' so obtained. Yet, as recomputing $\text{cb}(\ell')$ is a costly step, we simply set $\text{cb}(\ell') = \text{cb}(\ell)$ assuming these bounds are close to each other. Thus, completion bounds are computed only when a label is extended, at Step 14.

We remark that ℓ could have been processed before it is corrected by the extension of a label m, because labels are processed in the order given by cb. The extensions arising from ℓ at the dominated times will be later dominated by the extensions of m, meaning that we spent time extending useless portions of some labels. The time required to extend ℓ is independent of the size of $\text{dom}(\ell)$. Thus, as far as none of the extensions arising from ℓ is completely discarded, the amount of time spent in useless extension of Δ_ℓ is not significant. If cb is *perfect*, meaning that $\text{cb}(\ell) = \bar{c}(\ell)$ for every label ℓ, then no processed label is ever discarded. This implies that our algorithm is sensible to the quality of the completion bounds. A label setting alternative, in which correction can be completely avoided at the expense on maintaining more labels, is discussed in Section 4.

2.1.2 Relation with the time independent algorithm

Let p be an elementary path starting from the depot 0 and ending at a vertex v. To establish the relation between Algorithm 1 and the one proposed by Tilk and Irnich (2017), recall that Δ_p is a continuous and piecewise linear function. As travel time functions are constant for a time-independent problem, Δ_p is composed by at most two pieces Δ_1 and Δ_2 with domains $[a, t^*]$ and $[t^*, b(v)]$, respectively, where $t^* = \delta_p(\max \text{FDT}(p))$. That is, Δ_1 corresponds to the feasible arriving times that do not incur in an unnecessary waiting, whereas Δ_2 is the waiting piece. This is the case depicted in Figure 3b. Therefore, taking into account that waiting pieces are implicitly encoded, if domination is ignored, then our algorithm keeps one label ℓ for p with $\Delta_\ell = \Delta_1$. An alternative representation of a label can be obtained using three constant resources. Following the notation of Tilk and Irnich (2017), let $T_{\text{time}} = a(\ell) = \min \text{FAT}(p)$ be the earliest feasible arrival time, $T_{\text{last}} = b(\ell) = t^*$ be the latest feasible arrival time without any unnecessary waiting, and $T_{\text{dur}} = \min\{\Delta_p(t) \mid t \in \text{dom}(\ell)\}$. Tilk and Irnich (2017) keep three resources that together encode Δ_p: T_{time}, T_{dur}, and $T_{\text{help}} = T_{\text{last}} - T_{\text{dur}} = \lambda_p(T_{\text{last}})$ (i.e., T_{help} is the latest feasible departing time). Thus, p is represented by a label $\hat{\ell} = (\text{prev}(\hat{\ell}), v(\hat{\ell}), S(\hat{\ell}), T_{\text{time}}(\hat{\ell}), T_{\text{dur}}(\hat{\ell}), T_{\text{help}}(\hat{\ell}))$.

The above observation highlights that our algorithm generalizes the one proposed by Tilk and Irnich (2017), as far as domination is not considered. However, Tilk and Irnich (2017) implement a full dominance criterion: $\hat{\ell}$ is an alternative representation of a full label, since Δ_1 is encoded in all of its domain by the resources of $\hat{\ell}$. Thus, as depicted by the black label in Figure 3b, $\hat{\ell}$ may not be discarded even though it is

dominated at every feasible arriving time. In contrast, our algorithm algorithm keeps more than one label ℓ per path and, therefore, Δ_ℓ is indeed a restriction of Δ_1 to a subdomain.

Recall that, because labels are processed in the order given by cb, our algorithm can correct a label ℓ after it was processed. To avoid this problem, an alternative is to apply a *core-ordering* version of Algorithm 1 in which labels are processed in the order given by their cores. In this case, as ℓ can only be corrected by a label visiting $|S(\ell)|$ vertices, no extension of ℓ was processed the moment ℓ is corrected. As a byproduct, all the labels with the minimum core are simultaneously extended at each iteration (Steps 4–15), improving the overall efficiency of the algorithm. For this algorithm to be correct, however, every label ℓ with $c^* < \mathrm{cb}(\ell) \leq ub$ and $|S(\ell)| \leq n$ must be processed. Hence, the efficiency of this method depends on the quality of ub.

A third approach, proposed by Baldacci et al. (2012) and applied by Tilk and Irnich (2017), is to guess ub by considering that $ub = (1+\epsilon)lb$, where $\varepsilon \approx 0$ and lb is a lower bound of c^*, e.g., $lb = \mathrm{cb}(\ell_0)$ for the initial label ℓ_0. If $ub \geq c^*$, then the algorithm finds the optimal tour. Otherwise, lb and ub are updated to ub and $(1+\varepsilon)ub$, respectively, and a new iteration is executed. This approach, as well as ours, in sensible to the quality of the lower bound $lb = \mathrm{cb}(\ell_0)$. However, it also depends on $c^* - lb$ and the magnitude of ε. If $(c^* - \mathrm{cb}(\ell_0))/\varepsilon$ is large, then many iterations that repeatedly compute the same labels are executed, while if $(c^* - \mathrm{cb}(\ell_0))/\varepsilon$ is small, then many labels with $\mathrm{cb} > c^*$ are examined. Thus, a good choice of ε for one instance can be a bad choice for the others. In this sense, Algorithm 1 is more robust because it adapts itself to each instance, which is particularly important in our context, as the computed bounds are not as tight as in time-independent problems.

2.2 Backward algorithm

By definition, a label ℓ always represents a path p that starts at the depot 0 and is extended by inserting vertices after the last visited vertex. In a backward labeling algorithm, *backward labels* that represent paths ending at the depot $n+1$ are constructed, by inserting vertices before the first visited vertex. A backward label, representing a path $p = (v_0, \ldots, v_k)$ ending at the depot $v_k = n+1$, is a tuple $\ell = (\mathrm{next}, v_0, \{v_0, \ldots, v_k\}, \Gamma_\ell)$, where next is a pointer to a backward label representing $(v_1, \ldots, v_k)$ and $\Gamma_\ell(t)$ is the restriction of a piece of Γ_p to an interval $\mathrm{dom}(\ell) = [a(\ell), b(\ell)]$. As in the forward case, we write $v(\ell) = v_0$, $S(\ell) = \{v_0, \ldots, v_k\}$, and $\mathrm{FDT}(\ell) = [a(v_0), b(\ell)]$.

The backward labeling algorithm can be obtained by adapting Algorithm 1, together with the tour-feasibility and partial domination rules, to work with backward labels. There are at least two inconveniences with this approach. On the one hand, we have to restate the extension function and the rules for discarding labels, obtaining a large description of a similar algorithm. On the other hand, two similar DP algorithms have to be implemented, leading to a larger and error-prone code base. Instead of adapting Algorithm 1, in this section we discuss how to transform the instance to simulate the backward algorithm by running Algorithm 1 as is.

Recall that an instance of TDTSPTW is a tuple $\mathcal{I} = (D, o, d, a, b, T, \tau)$, where D is a digraph (V, A) having two distinguished vertices $o = 0$ and $d = n+1$ that represent the depot, a and b are the vectors of release and deadline times of the vertices, $[0, T]$ is the planning horizon, and τ is the vector of travel times of the arcs. The *backward instance* of $\mathcal{I}$ is the tuple $B(\mathcal{I}) = (D', d, o, a', b', T, \tau')$ such that D' is the digraph obtained by inverting all the arcs in D, $a'_i = T - b_i$, $b'_i = T - a_i$, and the travel time function τ'_{ji} of τ' corresponding to the arc (j, i) of D' is given by

$$\tau'_{ji}(T - t) = t - \max\{t' \in [0, T] \mid t' + \tau_{ij}(t') \leq t\}, \tag{1}$$

for every $t \in [0, T]$. Note that $B(\mathcal{I})$ is an instance of TDTSPTW and, moreover, $B(B(\mathcal{I})) = \mathcal{I}$. The following proposition shows the relation between the instances.

Proposition 1. *If $p = (v_1, \ldots, v_k)$ is a path of D, then $q = (v_k, \ldots, v_1)$ is a path of D'. Moreover, $\mathrm{FDT}(q) = \{T - t \mid t \in \mathrm{FAT}(p)\}$, $\mathrm{FAT}(q) = \{T - t \mid t \in \mathrm{FDT}(q)\}$, and $\Delta_q(T - t) = \Delta_p(t)$ for every $t \in \mathrm{FAT}(p)$.*

Intuitively, if $p = (v_1, \ldots, v_k)$ is a path ending at the depot $v_k = n+1$, then $q = (v_k, \ldots, v_1)$ is a path of $B(\mathcal{I})$ that starts at the source depot. Moreover, if $\mathrm{FDT}(p) = [t_1, t_2]$, then $\mathrm{FAT}(Q) = [T - t_2, T - t_1]$.

Thus, if we run Algorithm 1 on $B(\mathcal{I})$, then each forward label ℓ *represents* the backward label m such that $\text{next}(m) = \text{prev}(\ell)$, $v(m) = v(\ell)$, $S(m) = S(\ell)$, $\text{dom}(m) = \{T - t \mid t \in \text{dom}(\ell)\}$, and $\Gamma_m(t) = \Delta_\ell(T - t)$ for every $t \in \text{dom}(m)$.

We remark that the same solution is obtained when Algorithm 1 is executed on either $\mathcal{I}$ or $B(\mathcal{I})$. Yet, the former alternative is more efficient when the makespan is minimized. Just observe that $a(0) = b(0)$, thus the domain of every label is a point when Algorithm 1 is run on $\mathcal{I}$. Instead, the initial label for $B(\mathcal{I})$ has a non-singleton domain that is extended throughout the algorithm.

2.3 Bidirectional algorithm

Bidirectional search for labeling algorithms was originally proposed by Righini and Salani (2006) and is widely used across different variants of the vehicle routing problem. The idea is to execute the forward and backward labeling algorithms to generate non-dominated labels that represent paths reaching some "middle point". Then, forward and backward labels are *merged* to generate full paths. Bidirectional search is correct only if the optimal solution can be obtained by merging the computed non-dominated labels, and is usually faster than its monodirectional counterparts when the examination of many pairs of forward and backward labels is avoided at the merging step.

By definition, every tour has exactly $|V|$ vertices. Thus, $p(\ell) + p(m)$ is a tour, for a forward label ℓ and a backward label m, only if $|S(\ell)| + |S(m)| = |V| + 1$. In our bidirectional algorithm, the core-ordering version of Algorithm 1 with no bounding is invoked to extend all the forward and backward labels visiting $\lfloor(|V|+1)/2\rfloor$ and $\lceil(|V|+1)/2)\rceil$, respectively. Two sets F and B with forward and backward non-dominated labels, respectively, are thus obtained. Suppose $t \in \text{dom}(\ell)$ is tour-feasible for a label $\ell \in F$, and consider a path $p \in \mathcal{E}_t(\ell)$ with $\bar{c}(\ell) = \Delta_\ell(t) + \Gamma_p(t)$. By definition, there exists a backward label m for p with $t \in \text{FDT}(m)$ and $\Gamma_p(t) = \Gamma_m(t)$, that need not belong to B. (Note that $t \in \text{FDT}(p)$ does not imply $t \in \text{dom}(m)$ because t could be a time in the waiting piece of m. For backward labels, the waiting piece —if existing— is the initial piece of $\Delta(p)$ that has slope -1 that arises because of the no-waiting property. Thus, $t \in \text{FDT}(m) \setminus \text{dom}(m)$ when waiting is mandatory.)

After computing F and B, our bidirectional algorithm traverses every label in $\ell \in F$ to locate a label $m \in B$ such that $\bar{c}(\ell) = \min\{\Delta_\ell(t) + \Gamma_m(t) \mid t \in \text{dom}(\ell) \text{ and } p(m) \in \mathcal{E}_t(\ell)\}$. By the discussion above, m is always found when ℓ is tour-feasible. The algorithm then outputs $p = p(\ell) + p(m)$ for the pair (ℓ, m) minimizing $c_p = \bar{c}(\ell)$. The algorithm is correct because F always contains a tour-feasible label ℓ with an optimal tour duration. The next proposition discusses how to determine if $m \in \mathcal{E}_t(\ell)$.

Proposition 2. *Let ℓ be a forward label, m be a backward label, and $t \in \text{dom}(\ell)$. Then, $p(m) \in \mathcal{E}_t(\ell)$ if and only if $v(\ell) = v(m)$, $S(\ell) \cap S(m) = \{v(\ell)\}$, $S(\ell) \cup S(m) = V$, and $a(\ell) \le t \le b(m)$.*

3 Preprocessing techniques

In this section we present the preprocessing techniques that we apply to transform the instance into an equivalent one with the same optimal solution. Two main purposes are pursued with the preprocessing of an instance. On the one hand, the goal is to reduce the number of edges and obtain tighter time windows to help in the efficiency of the labeling algorithms. On the other hand, we compute some information required by the relaxation of the labeling algorithm that allows us to improve the completion bounds that are later consumed by Algorithm 1.

Preprocessing is composed by three interrelated steps, that are applied iteratively until no new modifications are produced. For the first of these steps, aimed at tightening the time windows, we apply the techniques by Montero et al. (2017) and Lera-Romero and Miranda-Bront (2019) that extend rules proposed by Ascheuer et al. (2001) for the TSPTW. Due to space limitations, we refer to the aforementioned articles for a detailed description.

The second step involves the inference of the precedence relation $\prec$ such that $v \prec w$ when v is visited before w in every elementary tour. If $v \prec w$, then v is a *predecesor* of w and w is a *succesor* of v. Clearly,

$0 \prec n+1$ and $0 \prec v \prec n+1$ for $v \in V \setminus \{0, n+1\}$. In practice, a subset of $\prec$ is inferred using the following rule. Here, $\mathrm{EAT}(v, w)$ is the earliest time w can be arrived to when v is departed from at time a_v. Recall that the triangle inequality does not necessarily holds for the TDTSPTW, but $\mathrm{EAT}(v, w)$ can be efficiently precomputed using Dijkstra's algorithm given that the FIFO property holds on the travel times.

Rule 4 (Precedence)**.** *If* $\mathrm{EAT}(w, v) > b_v$*, then* $v \prec w$*. If* $v \prec z$*,* $\mathrm{EAT}(w, v) > \mathrm{LDT}(v, z)$ *and* $\mathrm{EAT}(v, z) > \mathrm{LDT}(z, w)$*, then* $v \prec w \prec z$*.*

Finally, the third standard component of the preprocessing is the removal of infeasible arcs. Clearly, (v, w) is infeasible when either $w \prec v$ or $a_v + \tau_{vw}(a_v) > b_w$. We remark that the latter condition does not imply $w \prec v$ because the triangle inequality need not be satisfied. We remark that $\mathrm{LDT}(v, w) = -\infty$ if $w \in \pi(v)$, and therefore Algorithm 1 applies an analogous filter regarding predecessors during the extension step compared to Tilk and Irnich (2017).

The time windows can be further exploited to reduce the number of extensions of labels within the labeling algorithm. Rule 4 optimistically supposes that v can be departed from at its release a_v when computing $\mathrm{EAT}(v, w)$. During the enumeration of labels, however, v is sometimes reached at a time $t > a_v$. Thus, additional precedences, that remain valid only under this assumption, can be inferred. Motivated by the ILP formulation introduced by Dash et al. (2012) for the TSPTW, where buckets (i.e., time intervals) can precede vertices and vice versa, we propose new precedence relations between vertices that remain valid only within a specific time interval, although the precedence may not hold in general..

For a label ℓ, define $\prec_\ell$ as the relation such that $v \prec_\ell w$ when v is visited before w in $p(\ell) + p$, for every $p \in \mathcal{E}(\ell)$. Certainly, ℓ can be discarded if $w \notin S(\ell)$ for $w \prec_\ell v(\ell)$. Furthermore, it suffices to extend ℓ only to the minimum vertices with respect to $\prec_\ell$. We further generalize the set of predecessors and successors of a vertex to become *label-dependent*, that is $\pi(v, \ell) = \{w \in V \mid w \prec_\ell v\}$ and $\sigma(v, \ell) = \{w \in V \mid v \prec_\ell w\}$. Recall the initial label ℓ_0 and note that $v \prec w$ iff $v \prec_{\ell_0} w$, $\pi(w) = \pi(w, \ell_0)$ and $\sigma(v) = \sigma(v, \ell_0)$.

Unfortunately, we cannot precompute a good subset of $\prec_\ell$ because the algorithm can generate an exponential number of labels. Also, the high cost of computing a good subset of $\prec_\ell$, for a given label ℓ, prevents its calculation each time a label is extended. The alternative that we propose is to relax $\prec_\ell$ to account only for time at which $v(\ell)$ is visited, despite the other relevant information in ℓ. For $v \in V$ and $t \in [0, T]$, let $\prec_v^t$ be the relation such that $w \prec_v^t z$ when w is visited before z in every feasible tour that arrives at v at time t.

The following result connects these generalizations regarding precedences.

Proposition 3. *For every label* ℓ*,* $\prec \subseteq \prec_{v(\ell)}^{a(\ell)} \subseteq \prec_\ell$*.*

In our implementation, Rule 5 below is applied to compute a set of predecesors $\pi(v, t) \subseteq \{w \in V \mid w \prec_v^t v\}$ and of successors $\sigma(v, t) \subseteq \{w \in V \mid v \prec_v^t w\}$ for each $v \in V$ and $t \in [0, T]$.

Rule 5 (Time precedence)**.** *If* $\mathrm{LDT}(v, w) < t$*, then* $w \prec_v^t v$ *and* $w \in \pi(v, t)$*, while if* $\mathrm{EAT}(w, v) > t$*, then* $v \prec_v^t w$ *and* $w \in \sigma(v, t)$*.*

Again, note that $\pi(w) = \pi(w, 0)$ and $\sigma(v) = \sigma(v, T)$. Clearly, $\pi(v, t) \subseteq \pi(v, t')$ for $t \leq t'$, thus at most $n - 1$ different sets of predecessors exist for a given $v \in V$. These sets can be pre-computed by calculating the breakpoints t that induce a change in $\pi(v, t)$. Then, $\pi(v, t)$ can be queried efficiently in $\mathcal{O}1$ for every $t \in [0, T]$. Sets $\sigma(v, t)$ are computed in a similar fashion.

We remark that Rule 1 implicitly encodes the use of $\pi(v(\ell), t)$ within Algorithm 1. In addition, note that having $w \in S(\ell) \cap \sigma(v(\ell), t)$ would imply $\mathrm{dom}(\ell) = \emptyset$. The purpose of the enhanced preprocessing, however, is to accelerate the execution of the relaxed algorithms, as explained in the next section.

4 Computing completion bounds

Recall that a completion bound cb is a function such that $c_{p(\ell)} \leq \mathrm{cb}(\ell) \leq \bar{c}(\ell)$ for every label ℓ. The discussion in Section 2.3 implies a simple algorithm to compute completion bounds. First, the core-ordering version of Algorithm 1 with no bounding is run to obtain a set of non-dominated backward labels B. Then,

$\mathrm{cb}(\ell)$ is obtained by computing $\bar{c}(\ell) = \min\{\Delta_\ell(t) + \Gamma_m(t) \mid t \in \mathrm{dom}(\ell) \text{ and } p(m) \in \mathcal{E}_t(\ell)\}$. This algorithm is correct because B contains every non-dominated backward label, regardless of the number of visited vertices.

The completion bound above is perfect as $\mathrm{cb}(\ell) = \bar{c}(\ell)$ for every label ℓ. When applied in Algorithm 1, no extended label is ever discarded, although some labels are corrected. An alternative approach is to change the meaning of cb to make it time-dependent, i.e., $\mathrm{cb}(\ell)$ is computed as the piecewise linear function $\Delta_\ell(t)+\Gamma_m(t)$. By doing so, every linear label in Algorithm 1 can be further sub-divided into a *bounded* label in which $\mathrm{cb}(\ell)$ linear. When applied to bounded labels, the correcting step of Algorithm 1 has no effects and can be removed to obtain a label setting algorithm. However, a reason to avoid the use of bounded labels is that more labels have to be maintained for an algorithm that already consumes exponential space.

Although correct, the algorithm with perfect bounds is useless because all the (non-dominated) exact backward labels have to be enumerated. Thus, the optimal solution to the TDTSPTW problem can be readily obtained by looking at the backward labels reaching 0. Instead of running an exact algorithm to enumerate backward labels, a relaxation is used to reduce computation time at the expense of obtaining slightly worse bounds. The purpose of this section is to describe alternative relaxations and how they are applied to compute the completion bounds.

4.1 The *ngL*-tour relaxation

The idea behind relaxations is to enlarge the set of enumerated tours by allowing non-elementary tours. The most efficient state-of-the-art algorithms for the TSPTW and MTDP use a variation of the *ng-tour relaxation*, originally proposed for the VRP by Baldacci et al. (2011). In this relaxation, the repeated traversal of a vertex v is admitted only if the tour visited a vertex outside the neighborhood of v since its last visit. Specifically, a *neighborhood* $N(v) \subseteq V$ containing v is associated to each vertex $v \in V$. Then, a path $p = (v_0, \dots, v_k)$ is an $ng(N)$*-path* if it contains no $ng(N)$*-cycles*, where $(v_i, \dots, v_j)$ $(0 \le i < j \le k)$ is an $ng(N)$-cycle if $v_i = v_j$ and $\{v_i, \dots, v_j\} \subseteq N(v_j)$. Those $ng(N)$-paths that are tours are called $ng(N)$*-tours*. Clearly, every $ng(N')$-tour is an $ng(N)$-tour when $N(v) \subseteq N'(v)$ for every $v \in V$, thus the duration c of an optimal $ng(N)$-tour is a lower bound on the duration of an optimal $ng(N')$-tour. Since every $ng(N')$-tour is elementary when $N'(v) = V$ for every $v \in V$, then c is a lower bound of the optimal elementary tour duration as well. In general, larger neighborhoods yield higher lower bounds but required more computational time. From now on, N is dropped from $ng(N)$ when no confusions are possible.

The *ngL-tour* relaxation was proposed by Baldacci et al. (2012) to reduce the number of labels that must be explored, while still reducing the number of feasible *ng*-tours. This relaxation is suitable only for those problems in which all the vertices must be visited exactly once. Let $\prec$ be a precedence relation on V such that $v \prec w$ if and only if v is traversed before w in every tour of D. By definition, $0 \prec v \prec n+1$ for every $v \in V$, thus 0 and $n+1$ are the minimum and maximum of $\prec$, respectively. For a maximal chain L of $\prec$ and a neighborhood vector N, say that an $ng(N)$-path p is an $ng(L,N)$*-path* if p visits each vertex of L at most once in the same order as they appear in L. If p is also a tour, then p is an $ng(L,N)$*-tour*; clearly, every elementary tour is an $ng(L,N)$-tour. From now on, L is fixed as a longest chain of $\prec$, thus those $ng(L,N)$-paths and tours are referred to as $ngL(N)$-paths and tours. As above, we drop N when it is clear by context.

Labeling algorithms are applied to compute optimal *ngL*-tours. To specify these algorithms we must define what a label is, how a label is extended, and how a time in the domain of a label is discarded because of a (partial) domination. Let $p = (0 = v_0, \dots, v_k)$ be an *ngL*-path. A (*forward*) *ngL-label* of p a is a tuple $\ell = (\mathrm{prev}, k, h, v_k, S, \Delta_\ell)$ such that prev is a pointer to an *ngL*-path representing $(v_0, \dots, v_{k-1})$, $h = |p \cap L|$ is the number of vertices visited in L, $S \subseteq V$ is the set of vertices that form an *ng*-cycle when inserted after v_k, and Δ_ℓ is the duration function as defined in Section 2.1 for linear labels. *Backward ngL-labels* are defined analogously. For the sake of simplicity, we extend the notation and terminology of linear labels to *ngL*-labels, while we also let $k(\ell) = k$ and $h(\ell) = h$. The *initial* label is $\ell_0 = (\perp, 1, 1, 0, \{0\}, [0,T] \to 0)$.

Suppose $L = (0 = z_1 \prec \dots \prec z_{|L|} = d)$ and define $V_h = \{u \in V \mid u \not\prec z_h \text{ and } z_{h+1} \not\prec u\}$ for every $0 \le h < |L|$. By the definition of $\prec$, $u \in V_h$ if there exists a tour in which u is visited after z_h and another (possibly equal) tour in which u is visited before z_{h+1}. Therefore, V_h is a superset of the vertices that can be visited between z_h and z_{h+1} in any tour. Consequently, a label ℓ can be extended via an arc

$(v, w) \in A$ only if $w \in V_{h(\ell)}$. Combining this fact with the time window, *ng*, and precedence constraints, we say that an extension of ℓ via (v, w) is *feasible* when $a(\ell) + \tau_{vw}(a(\ell)) \leq b_w$, $w \in V_{h(\ell)} \setminus S$, $|\pi(w)| \leq k(\ell) + 1$, $|\sigma(w)| \leq n - k(\ell) + 2$, and $|\sigma(z_{h(\ell)+1})| \leq n - k(\ell) + 1$ if $w \neq z_{h(\ell)+1}$. To the best of our knowledge, the last two conditions have not been enforced in the literature yet and, according to our experiments, they considerably reduce the number of feasible labels, yielding better bounds and faster execution times.

The new preprocessing described in Section 3 allows to strengthen some of these conditions during the extension by including the time dimension. Consider extending ℓ via arc $(v(\ell), w)$, and that all the previous conditions are met. Let m be a label obtained as the extension of ℓ by a piece of $\tau_{v(\ell)w}$. Recall the definition of $\pi(w, t)$, $\sigma(w, t)$, and that $\pi(w, t) \subseteq \pi(w, t')$ and $\sigma(w, t) \supseteq \sigma(w, t')$ for $t \leq t'$, inducing an interval of time where is feasible for vertex w to have exactly $k(\ell)$ predecessors (eventually, the entire horizon $[0, T]$). Specifically, $t \in \text{dom}(m)$ can be discarded $t < \arg\min_t\{|\pi(w, t)| \geq k(\ell) + 1\}$ or $t > \arg\max_t\{|\sigma(w, t)| \leq n - k(\ell) + 2\}$, and m can be discarded if the adjusted $\text{dom}(m) = \emptyset$. Note that this represents an adaptation of Rule 1 to the *ngL* context, where the paths enumerated are not necessarily elementary. In our implementation, these limits are precomputed during the preprocessing phase and later used within Algorithm 1 during the extension.

When the extension of ℓ via (v, w) is infeasible, the extension of ℓ is an empty set. Otherwise, a set of *ngL*-labels is obtained, one for each piece of τ_{vw} that intersects $\text{dom}(\ell)$, as explained in Section 2.1 for the exact algorithm. The computation of each extension m of ℓ is exactly the same as the one described in Section 2.1 for linear labels, with the exception that $S(m) = (S(\ell) \cup \{w\}) \cap N(w)$, $k(m) = k(\ell) + 1$ and $h(m)$ is either equal to $h(\ell)$ or $h(\ell) + 1$ according to whether $w \neq z_{h(\ell)+1}$ or not, respectively. The domination rule is also modified to handle the possible repetition of vertices in $p(\ell)$, as follows.

Rule 6 (*ngL*-partial domination)**.** *An* ngL*-label ℓ is dominated by another* ngL*-label m at a time $t \in \text{dom}(\Delta_L)$ if $t \in \text{FAT}(m)$, $k(m) = k(\ell)$, $v(m) = v(\ell)$, $h(m) = h(\ell)$, $S(m) \subseteq S(\ell)$, and $\Delta_\ell(t) \leq \Delta_\ell(t)$.*

Algorithm 1 can be adapted to compute an optimal *ngL*-tour. When the core-ordering version is applied, a set of non-dominated *ngL*-labels can be obtained. Moreover, a set of backward *ngL*-labels B is obtained when the algorithm is executed on the backward instance. Let M be a neighborhood vector with $M(v) \supseteq N(v)$ for every $v \in V$. For an *ngL*(M)-label ℓ, let $\mathcal{E}^N_t(\ell)$ be the subset of B such that $m \in \mathcal{E}^N_t(\ell)$ if and only if $v(\ell) = v(m)$, $S(\ell) \cap S(m) = \{v\}$, $k(\ell) + k(m) = n + 1$, and $a(\ell) \leq t \leq b(m)$. Since B is obtained by running the algorithm until no unprocessed labels remain, a discussion similar to the one in Section 2.3 is enough to conclude that:

1. $\mathcal{E}^M_t(\ell) \neq \emptyset$ when ℓ can be extended into an *ngL*(M)-tour that visits $v(M)$ at a time $t \in \text{dom}(\ell)$, and
2. $\mathcal{E}^N_t(\ell) \neq \emptyset$ when $\mathcal{E}^M_t(\ell) \neq \emptyset$.

The former implies that we can apply a bidirectional algorithm to compute an *ngL*(M)-tour with the optimal duration. Since every *ngL*(M)-tour is an *ngL*(N)-tour, the latter means that $\text{cb}(\ell) = \min\{\Delta_\ell(t) + \Gamma_m(t) \mid m \in \mathcal{E}^N_t(\ell) \text{ and } t \in \text{dom}(\ell)\}$ is a lower bound on the optimal duration of those *ngL*(M)-tours that traverse ℓ and visit v at a time $t \in \text{dom}(\ell)$. Function cb is the completion bound that we apply for Algorithm 1 and, as discussed, it is correct when the algorithm is run to solve either the exact problem or the *ngL*-tour relaxation.

4.2 Dynamic Neighbourhood Augmentation

The *dynamic neighborhood augmentation* (DNA, see e.g. Tilk and Irnich (2017)) builds the neighborhood of each vertex by iteratively extending them to remove an *ng*-cycle from an optimal *ngL*-tour. If p is an optimal *ngL*(N)-tour for some initial vector of neighborhoods N, then either p is elementary or it contains an *ng*-cycle $(v_0, \dots, v_j)$. In the former case, p is an optimal solution for the exact problem. In the latter case, v_0 is inserted into $N(v_i)$, for $0 < i < j$, to "forbid" $(v_0, \dots, v_j)$. The neighborhood augmentation process is repeated until either an optimal elementary tour is found or the neighborhoods reach a maximum size K.

We implement the DNA method proposed by Tilk and Irnich (2017), although we initilize $N(v)$ with v and three vertices closer to v. At each step, an optimal *ngL*(N)-tour p is computed. Then, a maximal family

C of disjoint ng-cycles $(v_0, \ldots, v_j)$, such that $|N(v_i)| < K$ for every $0 < i < j$, is generated. If $C = \emptyset$, then the DNA is halted; otherwise, v_0 is inserted into $N(v_i)$ for every $(v_0, \ldots, v_j) \in C$.

4.3 Penalty method

The behavior of the completion bound cb has a major impact on the efficiency of Algorithm 1. Indeed, when cb is perfect, only those labels that can be extended into an optimal tour are processed. On the contrary, many labels are processed when cb is loose. The function cb obtained with the ngL-tour relaxation is not tight enough to solve the most difficult instances. Yet, cb can be tightened if the cycles found by the ngL-tour relaxation are penalized using an appropriate penalty for each vertex. The *penalty method*, applied by Baldacci et al. (2012) for the TSPTW and by Tilk and Irnich (2017) for the MTDP, implements this idea. In this section we adapt the penalty method for Algorithm 1, pointing out only the modifications that are required; for a detailed explanation, refer to Baldacci et al. (2012).

Let Ω be the set of feasible tours for a given relaxation. Denote by $\rho_{ip} \in \mathbb{N}$ the number of times vertex $i \in V$ is visited by tour $p \in \Omega$, let $\nu_i \in \mathbb{R}$ be the *penalty* associated to i, and define $\nu_p = \sum_{i \in V}(\rho_{ip} - 1)\nu_i$ as the *penalty* of a path p. Clearly, $\nu_p = 0$ when p is an elementary tour. Therefore, $lb(\nu) = \min\{c_p - \nu_p \mid p \in \Omega\}$ is a lower bound for the TDTSPTW. As shown by Baldacci et al. (2012), $lb(\nu)$ is maximized by the vector ν of dual variables corresponding to the following linear programming problem:

$$\begin{aligned} \min \sum_{p \in \Omega} c_p y_p \\ s.t. \sum_{p \in \Omega} \rho_{ip} y_p = 1, i \in V \\ y_p \geq 0, p \in \Omega. \end{aligned}$$

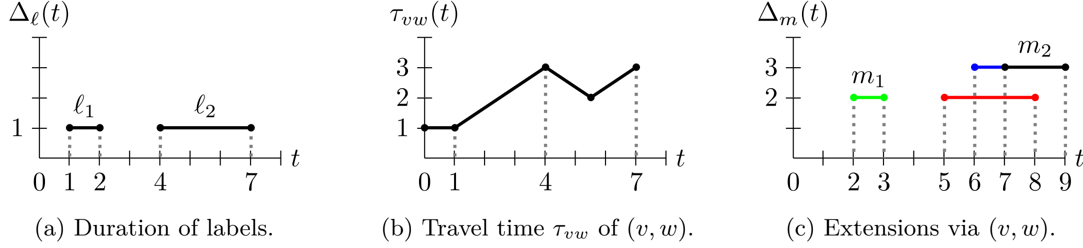
The variables of the program above cannot be explicitly enumerated because Ω has an exponential number of tours. Hence, we solve this problem via column generation. The pricing problem is to find a (relaxed) tour $p \in \Omega$ such that $c_p - \sum_{i \in V} \rho_{ip}\nu_i < 0$, where ν_i is the value of the dual variable associated to constraint i in the restricted master problem. The pricing problem can be solved with labeling algorithms similar to those discussed in Section 2, adapted to the relaxation at hand. In these algorithms, a penalty $\nu_{p(\ell)}$ is computed for each label ℓ, by adding ν_i for each visited vertex i. Then, the cost $\Delta_\ell - \nu_{p(\ell)}$ is minimized and, therefore, the domination rule is adapted to discard t from $\text{dom}(\ell)$ only when ℓ is dominated by a label m at t (Rule 2) and $\Delta_\ell(t) - \nu_{p(\ell)} \geq \Delta_m(t) - \nu_{p(m)}$.

To apply the penalty method for bounding, first a set of labels B is computed by running a monodirectional labeling algorithm with a vector of penalties ν, as in the pricing problem above. Then, a monodirectional labeling algorithm with no penalties is executed in the opposite direction. As observed by Baldacci et al. (2012), $\text{cb}(\ell) = \min\{\Delta_\ell(t) + \Gamma_m(t) + \nu_{p(m)} \mid t \in \text{dom}(m) \text{ and } m \in \mathcal{E}_t(\ell)\}$ is a completion bound, for each label ℓ of the second run.

4.4 The *ti*-tour relaxation

When the time required to solve the penalty method to optimality surpasses its assigned limit, ν is obtained from the last iteration of the column generation. As discussed by Tilk and Irnich (2017), better penalties are obtained when more iterations of the column generation are executed in a more relaxed problem. On the hardest instances, the penalty method ends before reaching optimality, even when small neighborhoods are considered for the ngL-tour relaxation. In this section we discuss a further relaxation to ngL, specifically designed for time-dependent instances, that allows more columns to be generated.

In a time-dependent context, maintaining several labels for each feasible path degrades the performance of the algorithm. When solving an ngL-tour relaxed instance $\mathcal{I}$, new labels are generated at two different steps: extension and domination. The creation of labels at the domination step corresponds to the removal of dominated times, thus partial domination has a positive impact as fewer labels are enumerated (see Section 2.1.2). On the contrary, the labels created at the extension step have to be independently processed,



(a) Duration of labels. (b) Travel time τ_{vw} of (v, w). (c) Extensions via (v, w).

Figure 4: $\tau'_{vw} = 1$ in the straightforward relaxation $\mathcal{I}'$. Thus, the green and red labels in c are obtained when ℓ_1 and ℓ_2 are extended. In the *ti*-relaxation, instead, $\hat{\tau}^{\ell_1}_{vw} = 1$ and $\hat{\tau}^{\ell_2}_{vw} = 2$, thus m_1 and the blue plus black label are obtained after adding these times. However, as $\tau_{vw}(4) = 3$, the blue portion is discarded from m_2.

thus more time is required when more labels are generated. Let $\mathcal{I}'$ be the time-independent MTDP instance obtained from $\mathcal{I}$ by replacing τ_{vw} with $\tau'_{vw} = \min\{\tau_{vw}(t) \mid t \in [a_v, b_v]\}$ for every arc $(v, w) \in A$. Clearly, the optimal solution of $\mathcal{I}'$ is a lower bound of the optimal solution of $\mathcal{I}$. Thus, $\mathcal{I}'$ is a simple and straightforward relaxation of $\mathcal{I}$ in which only one label is generated at each extension step that, as discussed in Section 2.1.2, can be solved with a labeling algorithm that applies a partial dominance criterion.

Unfortunately, the lower bounds obtained by solving $\mathcal{I}'$ are of a poor quality when the travel time functions of the original instance $\mathcal{I}$ present a high variance. Moreover, as the travel times at the releases of the vertices are underestimated, the labels generated while solving $\mathcal{I}'$ need not correspond to feasible *ngL*-tours. The *ti-tour* relaxation we propose is designed to overcome these issues (Figure 4). The *ti*-tour relaxation is obtained by executing Algorithm 1, adapted to the corresponding *ngL*-tour relaxation, with the following two modifications. First, each time a label ℓ is extended via an arc (v, w), the travel time function τ_{vw} is replaced with the constant (function) $\hat{\tau}^{\ell}_{vw} = \min\{\tau_{vw}(t) \mid t \in \text{dom}(\ell)\}$. Thus, only one label m is obtained because of the extension and, moreover, Δ_m is a constant function. With this modification alone, the domain of m would be $[a(\ell) + \hat{\tau}^{\ell}_{vw}, b(\ell) + \hat{\tau}^{\ell}_{vw}]$. By definition, $\hat{\tau}^{\ell}_{vw} \leq \tau_{vw}(a(\ell))$. Thus, considering that w cannot be reached before $a(\ell) + \tau_{vw}(a(\ell))$ when traversing $p(\ell)$, the second modification is to restrict $\text{dom}(m)$ to $[a(\ell) + \tau_{vw}(a(\ell)), b(\ell) + \hat{\tau}^{\ell}_{vw}]$. No further modifications are performed on Algorithm 1.

Since the *ti*-tour relaxation is obtained by underestimating the travel times, the optimal solution of the *ti*-tour relaxation is a lower bound of the optimal solution to the *ngL*-tour relaxation. Moreover, the extension step generates only one label because constant travel times are used. Thus, the two goals pursued when defining the straightforward relaxation $\mathcal{I}'$ are achieved by the *ti*-tour relaxation. We highlight three interesting advantages of the *ti*-tour relaxation over $\mathcal{I}'$ (Figure 4). First, note that the time $t' = \arg\min\{\tau_{vw}(t) \mid t \in [a_v, b_v]\}$ minimizing the travel time between v and w need not belong to $\text{dom}(\ell)$ for a given label ℓ. Thus, $\hat{\tau}^{\ell}_{vw}(t) \geq \tau_{vw}(t') = \tau'_{vw}$ for $t \in \text{dom}(\ell)$. Consequently, the duration of ℓ in $\mathcal{I}'$ is a lower bound on the duration of ℓ in the *ti*-tour relaxation, hence tighter bounds are produced by the *ti*-tour relaxation. These bound are further improved when we recall that domains are restricted because of the partial domination criterion. Hence, as a second advantage, the use of $\hat{\tau}^{\ell}_{vw}$ during the extension step partially captures the variability of the travel times, as the removal of time-dependency is restricted to $\text{dom}(\ell)$. Thus, two different labels ℓ and m extended over the same arc (v, w) will consider different travel times $\hat{\tau}^{\ell}_{vw}$ and $\hat{\tau}^{m}_{vw}$, corresponding to the minimums in their respective domains. Finally, it can be observed by induction that $a(\ell) = \min \text{FAT}(p(\ell))$ for every label ℓ of the *ti*-relaxation, i.e. ℓ represents a feasible *ngL*-path. Consequently, fewer labels, producing tighter bounds, are enumerated.

5 Overall Algorithm

Algorithm 1 is a template that we adapt to implement three different methods: LBL, LBL_{ngL}, and LBL_{ti}. These methods receive a *direction tag* $d \in \{fwd, bwd, bi\}$ as input, to indicate if a forward (*fwd*), backward (*bwd*), or bidirectional (*bi*) search is applied. The other parameters of these algorithms are the instance $\mathcal{I}$,

upper bound ub, completion bound cb, and vectors of neighborhoods N, predecesors π, and succesors σ, and penalties ν. Let L be a pre-computed longest chain of $\prec$ (Section 4.1). Then:

- LBL($\mathcal{I}$, ub, cb, ν) solves $\mathcal{I}$ with a forward search that bounds with ub and cb,
- LBL$_{ngL}$($\mathcal{I}$, d, N, π, σ, ν) solves the $ng(L, N)$-tour relaxation of $\mathcal{I}$ with a search in direction d that applies no bounding and processes labels with a core-ordering, and
- LBL$_{ti}$($\mathcal{I}$, N, π, σ, ν) solves the ti-tour relaxation of the $ng(L, N)$-tour relaxation of $\mathcal{I}$ with a forward search that applies no bounding and processes labels with a core-ordering.

Each of the above methods is adapted to consider the penalty vector ν, as discussed in Section 4.3.

Algorithm 2. TI method to solve the TDTSPTW

Input: An instance $\mathcal{I} = (D = (V, A), o = 0, d = n + 1, a, b, T, \tau)$ and an elementary tour p.

Output: An elementary tour p of minimum duration.

1: Preprocess $\mathcal{I}$ to obtain π and σ and let $ub = c_p$.
2: For each $v \in V$, let $N(v)$ contain v and the three vertices closer to v.
3: Let lb_0 be the lower bound obtained from LBL$_{ti}$($\mathcal{I}$, N, π, σ, 0).
4: Let ν and lb_1 be the penalty vector and best lower bound obtained by solving the penalty method. Each pricing iteration is solved with LBL$_{ti}$($\mathcal{I}$, N, π, σ, ν).
5: If $lb_0 > lb_1$, then let $\nu = 0$.
6: Augment N with the DNA method using $K = 14$ as the maximum neighborhood size. Each iteration is solved with LBL$_{ngL}$($\mathcal{I}$, N, π, σ, bi, ν).
7: Let cb be the completion bound obtained from LBL$_{ngL}$($\mathcal{I}$, N, π, σ, bwd, ν).
8: Output the path given by LBL($\mathcal{I}$, ub, cb, ν).

Algorithm 2 takes an elementary tour p that provides an upper bound $ub = c_p$ (Step 1). The bound ub serves two purposes. First, if a lower bound $lb = ub$ is found, then the algorithm is immediately halted and p is given as output. Second, the labels with a completion bound greater than ub are discarded at Step 8, as discussed in Algorithm 1. To compute p, we run an exact algorithm in a depth first search until an elementary tour is found. As expected, p provides a weak bound that has a minor impact on the algorithm. When Algorithm 2 is applied on an MTDP instance, however, we create an artificial path whose cost corresponds to the upper bound reported by Tilk and Irnich (2017). This setting provides a fair comparison between our algorithm and the one implemented by Tilk and Irnich (2017).

Any invocation to a labeling algorithm can fail to terminate within a reasonable amount of time. Thus, Algorithm 2 has three time limits, that are imposed to Steps 3+4, 6, and 7+8, respectively. The penalty method is initialized with the input path p. So, even though an optimal penalty vector yields tighter bounds than the vector 0, several iterations of the column generation are required before a penalty vector ν improving 0 is found. Moreover, the algorithm can fail to find ν within its time limit. Thus, Step 3 allows Algorithm 2 to halt on the easiest instances when a lower bound equal to ub is found, while it provides a better penalty vector on the hardest instances (Step 5).

There are two differences between the labeling algorithms executed for the penalty method (Step 4) and the DNA method (Step 6). First, the ti-tour relaxation is applied in the former, whereas the ngL-tour relaxation is used in the latter. As explained in Section 4.3, the ti-tour allows a faster resolution of the pricing problem, thus more columns are generated and better penalties are obtained on the hardest instances. On the other hand, the DNA method usually requires fewer iterations to end, thus it rarely reaches its time limit. Moreover, the computed neighborhoods should provide the tightest completion bounds that are used to solve the exact instance. The second difference between Steps 4 and 6 is that the former uses a forward search and the latter applies a bidirectional search. When neighborhoods are small, fewer labels representing large paths exist and, thus, the number of pairs to test at the merge step of a bidirectional search surpasses the number of enumerated labels. The situation is the opposite when larger neighborhoods are considered.

Tilk and Irnich (2017) propose a forward search for the DNA method (Step 6), in replace of our bidirectional search. To avoid an explosion of labels, a completion bound arising from the previous iteration

is applied. As discussed in Section 4.1, this approach is also valid for our algorithm. Note, however, that the efficiency of this variant strongly depends on the tightness of ub. If $ub = \infty$, then no label is discarded because of the completion bounds (see Algorithm 1). On the other hand, if ub is closer to the completion bounds, then this approach can be faster than the bidirectional search. This is particularly true for the later iterations of the DNA method, that consume more time but provide better completion bounds.

In view of the above observation, we remark that ub plays a minor role in our algorithm for those hardest cases in which Step 8 is executed. As observed in Algorithm 1, the sole purpose of ub is to discard those labels having a completion bound greater than ub. We remark that, by the way in which the labels are processed, no such label will ever be processed. We conclude, therefore, that our algorithm is more robust in the sense that it depends less on the tightness of ub. Finally, we remark that the direction of the labeling algorithms at Steps 7 and 8 can be inverted. This would yield an equally efficient algorithm for the TDTSPTW. However, when the TDTSPTW$_m$ is solved, the forward algorithm runs faster. Therefore, it is convenient to apply the inefficient backward algorithm on the faster relaxed instance (Step 7) and the efficient forward algorithm on the slower exact instance (Step 8).

6 Computational results

We conducted computational experiments in order to evaluate the performance of our approach. The algorithm is coded in C++, using CPLEX 12.9 as an LP solver. The experiments are run on single thread in a Workstation running Ubuntu 18.04 with an Intel Core i7-8700 3.20GHz CPU and 32GB of RAM. We evaluate the algorithm on benchmark instances from three different problems: MTDP, TDTSPTW and TDTSP. We report in each case a direct comparison with recent algorithms proposed in the literature to provide substantial evidence regarding the effectiveness of our algorithm. Overall, our approach is tested on 6100 benchmark instances from the related literature.

Regarding the evaluation, we consider the following algorithms:

- FWD: is the straightforward application of the forward algorithm with no bounding. FWD is the weakest of the algorithms implemented and is evaluated to provide a baseline for the labeling algorithm when compared against the state-of-the-art BC algorithms.
- NGL: is obtained by replacing LBL$_{ti}$ with LBL$_{ngL}$ at Steps 3 and 4 of Algorithm 2. NGL allows the evaluation of the impact of the ti-tour relaxation within the penalty method.
- TI: the method we propose in Algorithm 2.

The experiments are organized as follows. First, we conducted some preliminary experiments on a randomly selected subset of 1600 TDTSPTW$_{(m)}$ proposed in Arigliano et al. (2018b) to calibrate our algorithm. Additionally, we use this reduced benchmark to establish a comparison among FWD, NGL and TI. Afterwards, we compare the results obtained by TI for the TDTSPTPW$_{(m)}$, the TDTSP$_{(m)}$ and the MTDP, with the best algorithm in the literature in each particular case. Specific details about the instances and the experiments are given in the corresponding sections.

For each time-dependent instance, we set a time limit of 3600 seconds. For NGL and TI, the time is distributed in the following fashion: 1200 seconds for the penalty method (steps 3 and 4), 1200 seconds for the DNA (Step 6) and 1200 seconds for the exact labeling (steps 7 and 8), giving a total of 3600 seconds. For the MTDP, to match the time limit considered in Tilk and Irnich (2017), these values are adjusted to 3600 seconds for each stage. In the tables, we report aggregated results with the following metrics: inst indicates the total number of instances considered; opt the number of instances solved to optimality within the time limit; %G$_{\text{cg}}$ reports the % gap of the lower bound obtained after the penalty method; T$_{\text{cg}}$ is the time required to compute the penalties; %G$_{\text{dna}}$ is the % gap of the lower bound obtained after the DNA; T$_{\text{dna}}$ is the time to compute the DNA; and T$_{\text{tot}}$ is the overall computing time. The % gap of the lower bound lb is computed as $100 \times (z^* - lb)/lb$, where z^* stands for an upper bound on the instances. Gaps and times are reported as the average over the instances solved to optimality by the corresponding method.

n	inst	FWD		TI						NGL					
		opt	T_{tot}	opt	$\%G_{cg}$	T_{cg}	$\%G_{dna}$	T_{dna}	T_{tot}	opt	$\%G_{cg}$	T_{cg}	$\%G_{dna}$	T_{dna}	T_{tot}
15	400	400	0.03	400	0.49	0.39	0.00	0.44	0.83	400	0.49	0.64	0.00	0.52	1.15
20	400	400	0.45	400	0.84	4.35	0.17	10.00	15.82	400	0.84	7.87	0.17	10.13	19.18
30	400	374	158.74	400	0.83	97.39	0.41	101.57	219.36	400	0.83	188.19	0.40	100.62	309.08
40	400	143	126.55	374	1.35	533.17	0.82	324.21	925.87	350	1.26	743.57	0.86	342.74	1171.92

Table 1: Results for the reduced set of TDTSPTW_m instances.

6.1 Evaluation of the Relaxations

Preliminary experiments show that FWD instantly solves the instances reported by Lera-Romero and Miranda-Bront (2019), that include those by Arigliano et al. (2015) and form the core in Vu et al. (2019). This includes instances having $n \leq 40$, in their versions minimizing the makespan as well as the duration of the route. Due to space limitations, a direct comparison among TI and the aforementioned algorithms is omitted.

We consider the complete dataset for the TDTSPTW_m proposed in Arigliano et al. (2018b), that considers wider time windows and more time periods. The dataset contains 30 different instances for each combination of $n = 15, 20, 30, 40$, two traffic scenarios (B_1 and B_2), five congestion levels and four different scenarios for the size of the time windows $\beta = 0, 0.25, 0.50, 1.00$ – where a larger value of β indicates tighter time windows– resulting in 4800 instances in total. We randomly selected 10 out of the 30 instances for each combination of the parameters, obtaining a reduced dataset with 1600 of these instances. The results are shown in Table 1.

Table 1 encodes several messages. FWD produces reasonable results in general, solving all instances in the dataset for $n = 10, 20$. As expected, its performance deteriorates as n increases due to the large number of labels enumerated, although it is able to solve a reasonable proportion of the instances for $n = 40$ compared to other algorithms in the literature. Aligned with the time-independent variants, the framework described in Section 5 increases the number of instances solved to optimality. Both NGL and TI, by embedding tight completion bounds within the exact DP algorithm, are able to successfully solve almost every instance in this reduced set.

The main message in Table 1 is that TI solves to optimality 24 more instances than NGL in less average time. A detailed analysis of the results in Table 1 provides further insights regarding the methods. Both TI and NGL reach good quality lower bounds, where $\%G_{dna}$ is on average below 1% over the solved instances. Recall, however, that the difference between these two methods lies in the computation of the penalties, where TI applies the *ti*-tour relaxation. As can be retrieved from the results, $\%G_{cg}$ is nearly the same for both methods, even when *ti*-tour are a relaxation of *ngL*-tours. In addition, we highlight that the difference in the execution times in favour of TI is mainly a consequence of the reduction of the computation times in the column generation. Overall, this experiment shows the impact of the *ti*-tour relaxation, reaching a good tradeoff between computation times and the quality of the lower bounds obtained in a time-dependent context. Finally, we note that the memory limit is usually reached by the unsolved instances, for both TI and NGL.

6.2 TDTSPTW

In this section we compare the results obtained by TI with the ones reported by Arigliano et al. (2018b) on the complete dataset for the TDTSPTW_m. For the sake of conciseness, we present the results aggregated in a different fashion compared to Arigliano et al. (2018b). Following their notation, we present the results for all instances in Table 2, where each is a combination of $n = 15, 20, 30, 40$ and $\beta = 0, 0.25, 0.50, 1.00$, aggregated by the congestion level. This is done because our method is sensitive to the size of the time windows (Section 3), but not to the congestion level, despite the potential degradation on the computed lower bounds.

The results clearly state that TI outperforms their approach, increasing the number of instances solved to optimality by a 35% and solving 4724 of the 4800 available instances. Notably, TI is able to solve all

β	n	inst	Arigliano et al. (2018b) opt	T_{tot}	TI opt	$\%G_{cg}$	T_{cg}	$\%G_{dna}$	T_{dna}	T_{tot}
0.00	15	300	287	298.86	300	0.54	0.81	0.00	0.84	1.66
	20	300	248	659.60	300	1.49	9.57	0.41	29.97	45.64
	30	300	155	1631.16	300	1.58	219.72	1.02	217.78	488.56
	40	300	110	2263.33	237	2.27	1115.59	1.62	763.55	2119.87
0.25	15	300	299	143.43	300	0.88	0.54	0.00	0.62	1.17
	20	300	286	382.64	300	1.18	5.55	0.24	13.19	20.92
	30	300	199	1273.53	300	0.97	133.80	0.43	138.34	299.96
	40	300	132	1954.10	287	1.83	924.22	1.07	592.85	1654.10
0.50	15	300	299	25.96	300	0.69	0.17	0.00	0.37	0.55
	20	300	296	289.13	300	0.63	2.19	0.01	3.86	6.32
	30	300	157	1433.36	300	0.94	48.61	0.22	61.39	115.67
	40	300	55	2275.56	300	1.63	418.19	0.90	182.63	619.69
1.00	15	300	300	1.58	300	0.00	0.00	0.00	0.00	0.00
	20	300	300	29.49	300	0.00	0.00	0.00	0.00	0.01
	30	300	233	597.23	300	0.00	0.01	0.00	0.00	0.01
	40	300	106	527.74	300	0.00	0.08	0.00	0.01	0.10

Table 2: Results for Arigliano et al. (2018b) benchmark.

instances with $n \leq 30$. Despite different computers were used, the total average execution times (T_{tot}) are considerably reduced, specially for $\beta = 1$ where all instances are solved instantly. This is aligned with our previous observation, where FWDalready already outperform the previous best algorithms when the time windows are tight.

In general, the behavior of TI is consistent with the results reported in Section 6.1. The optimality gaps $\%G_{cg}$ and $\%G_{dna}$ are small in general, showing slightly larger values for $n = 40$. As expected, TI improves its performance as the time windows become more narrow (i.e., a larger value of β). The computation of the penalties is the most demanding in terms of the computation times.

6.3 TDTSP

In this section we evaluate the performance of TI on the TDTSP_m, i.e., when no time windows are present and the customers may be visited at any time during the planning horizon. The TDTSP_m is originally tackled in Cordeau et al. (2014), and subsequently refined recently in Arigliano et al. (2018a) and Adamo et al. (2020). The results reported suggest that the BB algorithm proposed in the latter is the most effective one, being able to solve larger instances with more time periods than the others. Therefore, for our experiment we consider the instances proposed in Adamo et al. (2020) for $n = 15, 20, 25, 30, 35, 40$, and establish a comparison between TI and their approach. To the best of our knowledge, recent exact-DP based approaches proposed in Baldacci et al. (2012) and Tilk and Irnich (2017) have not been tested previously on instances without time windows since the TSP can be efficiently solved by other techniques (see e.g. Applegate et al. (2007)). Currently, the TDTSP_m remains a very challenging for algorithms based on ILP, even for small values of n.

Table 3 displays the aggregated results for the TDTSP_m instances. Similarly to the TDTSPTW, the TI outperforms the approach in Adamo et al. (2020). Specifically, TI solves 897 out of 1080 instances, surpassing the 518 solved by the BB algorithm proposed in Adamo et al. (2020). Total average computation times T_{tot} are higher for TI, but recall that the averages are computed over the instances solved by the corresponding method.

Considering similar values of n, TDTSP_m instances seem to be harder to solve than for the TDTSPTW_m, while the average optimality gaps $\%G_{cg}$ and $\%G_{dna}$ remain below 2%. The increment in the computation times is the expected behavior, as the preprocessing described in Section 3 has no impact due to the absence of time windows. Furthermore, no precedences among vertices are inferred, resulting in a weaker version of the feasibility rule of the *ngL*-relaxation. Consequently, the labeling algorithm enumerates more labels

		Adamo et al. (2020)		TI					
n	inst	opt	T_{tot}	opt	$\%G_{cg}$	T_{cg}	$\%G_{dna}$	T_{dna}	T_{tot}
15	180	176	240.93	180	1.01	4.47	0.01	32.28	38.11
20	180	141	615.37	178	1.33	36.09	0.42	133.63	220.10
25	180	84	1030.38	164	1.83	220.29	1.02	360.22	778.73
30	180	60	1379.24	151	0.82	802.53	0.41	551.54	1543.86
35	180	35	1426.46	145	0.94	1202.83	0.43	1070.82	2616.96
40	180	22	1278.48	78	1.38	1203.76	0.74	1206.04	2890.87

Table 3: Results for Adamo et al. (2020) benchmark.

and becomes more expensive from a computational standpoint, even when applying different relaxation techniques. This is also reflected in the relative proportion of time consumed by the DNA stage compared to the TDTSPTW instances, which in some cases even exceeds the time spent during the column generation. Instances become harder for $n = 35, 40$, where in general the column generation and the DNA do not converge within the time limit. Overall, the TI algorithm stands as the best approach in the literature so far for the TDTSP_m, being able to solve instances with up to 40 vertices.

6.4 MTDP

Finally, we evaluate our approach on the (time independent) MTDP instances, and compare the results with the ones reported in Tilk and Irnich (2017). This comparison enables to measure the impact of the enhancements incorporated to the algorithm despite its adaptation to the time dependent context (i.e., partial dominance, new preprocessing, etc.). The MTDP instances are well known TSPTW benchmarks (i.e. Ascheuer et al. (2001), Potvin and Bengio (1996), Gendreau et al. (1998) and Ohlmann and Thomas (2007)), where the duration is minimized instead of costs. We group the results by the type of instances and report the results similarly to Tilk and Irnich (2017). Due to space limitations, we refer the reader to Tilk and Irnich (2017) for a detailed description on the benchmarks.

These instances assume integer input data, a fact that is exploited in Tilk and Irnich (2017) by rounding fractional values to the nearest integer to accelerate convergence or, eventually, decide that the solution is optimal. Optimal solutions for the TDTSPTW need not to be integer, thus our algorithm does not take advantage of this valuable information. Similarly to Tilk and Irnich (2017), our algorithm also considers a hard time limit of 10800 for the overall execution.[1]

Tables 4 – 7 show the results for each dataset. The same pattern can be observed in all cases: TI is able to solve more instances, and in some cases with significant reductions –sometimes reaching 70%– in the computation times. The method TI solves all the `Ascheuer` instances, all but one of the `Gendreau` instances (with 17 new optimal solutions), all the `Potvin+Bengio` instances (with 3 new optimal solutions), and 12 `Ohlmann` instances (including 11 new optimal solutions). Overall, 31 new optimal solutions to the MTDP dataset proposed by Tilk and Irnich (2017) were obtained in less computation times.

Our approach produces significant improvements in the lower bounds compared to Tilk and Irnich (2017). Note that TI consistently reduces the $\%G_{cg}$ and, consequently, the $\%G_{dna}$. This effect can be better observed in the `Potving+Bengio` dataset, where the $\%G_{cg}$ is always below 1.5%. Several reasons motivate these improvements. The partial domination criterion and the new preprocessing clearly contribute in this aspect. However, we believe a key difference is that we enumerate ngL-tours, whereas Tilk and Irnich (2017) enumerate $ngL2res$-tours that further relax feasibility. We can apply the former because partial domination, combined with the new feasibility rules, accelerate the execution of the ngL-tour relaxation, improving the trade-off between the computation times and the quality of the lower bounds. Labels in $ngL2res$ are obtained by neglecting T_{time} (Section 2.1.2), thus they do not encode the domain of Δ_ℓ and, consequently, partial dominance cannot be incorporated. The computation times are in some cases reduced to a 30%, both for

[1]According to `cpubenchmark.net`, our processor is approximately 35% faster as the one used in Tilk and Irnich (2017) for single-threaded processes.

the penalty method and the overall TI algorithm. These improvements are justified by the above discussion and the fact that our approach enumerates fewer labels.

		Tilk and Irnich (2017)						TI					
type	inst	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}
easy	32	32	0.07	3.58	0.04	0.09	3.70	32	0.00	0.94	0.00	1.42	2.37
hard	18	18	0.03	137.37	0.01	7.65	145.03	18	0.02	16.36	0.00	2.64	19.13

Table 4: Results for the MTDP, Ascheuer et al. (2001) benchmark.

		Tilk and Irnich (2017)						TI					
n	inst	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}
20	25	25	1.28	0.57	0.39	0.09	0.68	25	0.23	0.90	0.00	0.08	0.99
40	25	24	0.63	99.24	0.32	16.50	121.50	25	0.23	147.16	0.06	29.00	178.97
60	25	23	0.27	581.29	0.15	16.95	736.87	25	0.16	338.30	0.03	47.82	392.05
80	25	13	0.13	1821.23	0.08	130.25	1951.52	24	0.26	1740.31	0.18	290.59	2165.09
100	15	12	0.01	1343.93	0.01	2.48	1346.41	15	0.01	318.68	0.00	262.53	652.62

Table 5: Results for the MTDP, Gendreau et al. (1998) benchmark.

		Tilk and Irnich (2017)						TI					
type	inst	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}	opt	%G_{cg}	T_{cg}	%G_{dna}	T_{dna}	T_{tot}
rc201	4	4	0.06	0.13	0.00	0.03	0.15	4	0.00	0.00	0.00	0.00	0.01
rc202	4	4	0.99	6.58	0.79	0.43	7.10	4	0.24	3.77	0.00	1.98	5.76
rc203	4	4	1.00	92.53	0.79	25.65	119.90	4	0.69	53.49	0.14	42.53	100.92
rc204	3	2	2.01	1001.30	2.01	32.70	1097.50	3	0.52	356.88	0.23	72.00	435.08
rc205	4	4	2.20	5.13	2.19	0.15	5.33	4	0.48	0.24	0.00	0.07	0.32
rc206	4	4	2.79	5.18	1.89	3.40	8.75	4	0.52	5.41	0.00	2.62	8.05
rc207	4	4	5.19	16.20	4.83	18.45	411.88	4	1.10	29.43	0.08	16.91	49.21
rc208	3	1	3.93	7.90	3.32	41.90	57.90	3	1.44	80.71	0.56	32.67	117.43

Table 6: Results for the MTDP, Potvin and Bengio (1996) benchmark.

7 Conclusions and Future research

In this paper, we present an exact DP based algorithm developed mainly the TDTSPTW, but flexible enough to be applied to other variants. The framework incorporates some state of the art features originally proposed for the TSPTW and the MTDP, which are generalized and enhanced to handle more complex, time-dependent, travel time and cost functions. From a practical perspective, efficient algorithms for kind of problems are critical to obtain better distribution plans in congested networks.

We proposed several algorithmic contributions to exploit the time-dependent nature of the problem. First, we introduced new preprocessing rules that apply within the DP algorithm. Second, we approached the time-dependency using *linear labels*, *partial dominance*, and time-dependent feasibility rules to compute tight completion bounds in a context requiring more complex labels and data structures to be handled. Third, we partly re-designed the time-independent framework to make it less sensitive to the availability of good quality upper bounds. Finally, we proposed the *ti*-tour relaxation, that decreases the execution times while reaching very tight lower bounds for time-dependent problems.

From a practical standpoint, we conducted extensive computational experiments over more than 6100 instaces from three different problems. Briefly, our algorithm outperforms tailored approaches for the TDTSPTW, TDTSP, and the MTDP recently proposed in the literature, providing many new optimal solutions

		Tilk and Irnich (2017)						TI					
n	inst	opt	%G$_{cg}$	T$_{cg}$	%G$_{dna}$	T$_{dna}$	T$_{tot}$	opt	%G$_{cg}$	T$_{cg}$	%G$_{dna}$	T$_{dna}$	T$_{tot}$
150	15	1	0.00	729.1	0.00	3.3	732.4	8	0.79	2421.67	0.20	1333.22	4115.00
200	10	0	–	–	–	–	–	4	0.54	2422.12	0.26	1596.61	7246.04

Table 7: Results for the MTDP, Ohlmann and Thomas (2007) benchmark.

for previously unsolved instances of the three problems. Overall, the combination among the algorithmic contributions and the strong evidence provided by the experiments suggests a new research direction towards DP-based algorithms as one of the state of the art techniques for time-dependent problems.

We believe that our paper could motivate several research lines. Considering further operational constraints, such as pickup and delivery, is always interesting. The flexibility observed regarding the time windows make the approach appealing to more challenging problems, such as soft time windows or consistent variants of the TDTSP. Finally, alternative methods to compute completion bounds that complement the current approach are worth investigating, as this represents one of the key ingredients within the framework.

Acknowledgements

This research has been founded by FONCyT grants PICT-2016-2677 and PICT-2018-2961 from the Ministry of Science, Argentina, and by the Google Latin America Research Award 2019.

References

Tommaso Adamo, Gianpaolo Ghiani, and Emanuela Guerriero. An enhanced lower bound for the Time-Dependent Traveling Salesman Problem. *Comput. Oper. Res.*, 113:104795, 2020.

David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, USA, 2007.

Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Time dependent traveling salesman problem with time windows: Properties and an exact algorithm. Technical report, 2015.

Anna Arigliano, Tobia Calogiuri, Gianpaolo Ghiani, and Emanuela Guerriero. A branch-and-bound algorithm for the time-dependent travelling salesman problem. *Networks*, 72(3):382–392, 2018a.

Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, Emanuela Guerriero, and Isaac Plana. Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. *Discrete App. Math.*, 261:28–39, 2018b.

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

Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut. *Math. Program.*, (3):475–506, 2001.

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

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

Valentina Cacchiani, Carlos Contreras-Bolton, and Paolo Toth. Models and algorithms for the traveling salesman problem with time-dependent service times. *Eur. J. Oper. Res.*, 2019. In press.

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

Jean-François Cordeau, Gianpaolo Ghiani, and Emanuela Guerriero. Analysis and Branch-and-Cut Algorithm for the Time-Dependent Travelling Salesman Problem. *Transport. Sci.*, 48(1):46–58, 2014.

Said Dabia, Stefan Ropke, Tom Van Woensel, and Ton G. de Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transp. Sci.*, 47(3):380–396, 2013.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.*, 2(4):393–410, 1954.

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

Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M. Solomon. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Oper. Res.*, 43(2):367–371, 1995.

Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.

Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: a review. *Comput. Oper. Res.*, 64:189–197, 2015.

Maha Gmira, Michel Gendreau, Andrea Lodi, and Jean-Yves Potvin. Tabu Search for the Time-Dependent Vehicle Routing Problem with Time Windows on a Road Network. Technical report, CIRRELT-2019-32, 2019a.

Maha Gmira, Michel Gendreau, Andrea Lodi, and Jean-Yves Potvin. Managing in Real-Time a Vehicle Routing Plan with Time-Dependent Travel Times on a Road Network. Technical report, CIRRELT-2019-45, 2019b.

Soumia Ichoua, Michel Gendreau, and Jean Yves Potvin. Vehicle dispatching with time-dependent travel times. *Eur. J. Oper. Res.*, 144(2):379–396, 2003.

Irina Ioachim, Sylvie Gélinas, François Soumis, and Jacques Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204, 1998.

Martin Joerss, Jürgen Schröder, Florian Neuhaus, Chirstoph Klink, and Florian Mann. Parcel delivery: The future of last mile. Technical report, McKinsey&Company, 2016.

Gonzalo Lera-Romero and Juan José Miranda-Bront. Integer programming formulations for the time-dependent elementary shortest path problem with resource constraints. *Electron. Notes Discrete Math.*, 69:53–60, 2018.

Gonzalo Lera-Romero and Juan José Miranda-Bront. A branch and cut algorithm for the time-dependent profitable tour problem with resource constraints. *Eur. J. Oper. Res.*, 2019. In press.

Gonzalo Lera-Romero, Juan José Miranda-Bront, and Francisco Juan Soulignac. An enhanced branch-cut and price algorithm for the time-dependent vehicle routing problem with time windows. Technical report, Optimization Online, 2019.

Federico Liberatore, Giovanni Righini, and Matteo Salani. A column generation algorithm for the vehicle routing problem with soft time windows. *4OR-Q J. Oper. Res.*, 9(1):49–82, 2011.

Zhixing Luo, Hu Qin, Wenbin Zhu, and Andrew Lim. Branch and price and cut for the split-delivery vehicle routing problem with time windows and linear weight-related cost. *Transp. Sci.*, 51(2):668–687, 2017.

Aristide Mingozzi, Lucio Bianco, and Salvatore Ricciardelli. Dynamic Programming Strategies for the Traveling Salesman Problem with Time Window and Precedence Constraints. *Oper. Res.*, 45(3):365–377, 1997.

Agustín Montero, Isabel Méndez-Díaz, and Juan José Miranda-Bront. An integer programming approach for the time-dependent traveling salesman problem with time windows. *Comput. Oper. Res.*, 88:280–289, 2017.

Jeffrey W Ohlmann and Barrett W Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007.

Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part ii: genetic search. *INFORMS journal on Computing*, 8(2):165–172, 1996.

Giovanni Righini and Matteo Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optim.*, 3(3):255–273, 2006.

Martin W. P. Savelsbergh. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. *ORSA J. Comput.*, 4(2):146–154, 1992.

Martin W. P. Savelsbergh and Tom Van Woensel. 50th anniversary invited article—city logistics: Challenges and opportunities. *Transp. Sci.*, 50(2):579–590, 2016.

Remy Spliet, Said Dabia, and Tom Van Woensel. The time window assignment vehicle routing problem with time-dependent travel times. *Transp. Sci.*, 52(2):261–276, mar 2018.

Peng Sun, Lucas P. Veelenturf, Said Dabia, and Tom Van Woensel. The time-dependent capacitated profitable tour problem with time windows and precedence constraints. *Eur. J. Oper. Res.*, 264(3):1058–1073, 2018a.

Peng Sun, Lucas P Veelenturf, Mike Hewitt, and Tom Van Woensel. The time-dependent pickup and delivery problem with time windows. *Transport. Res. B-Meth.*, 116:1–24, 2018b.

Duygu Taş, Michel Gendreau, Ola Jabali, and Gilbert Laporte. The traveling salesman problem with time-dependent service times. *Eur. J. Oper. Res.*, 248(2):372–383, 2016.

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

Duc Minh Vu, Mike Hewitt, Natashia Boland, and Martin Savelsbergh. Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows. *Transport. Sci.*, 2019. In press.