Beware of the Classical Benchmark Instances for the Traveling Salesman Problem with Time Windows

Francisco J. Soulignac
francisco.soulignac@unq.edu.ar

Universidad Nacional de Quilmes. Departamento de Ciencia y Tecnología. Bernal, Buenos Aires, Argentina.
CONICET-Universidad de Buenos Aires. Instituto de Investigación en Ciencias de la Computación (ICC). Buenos Aires, Argentina.

Abstract

We propose a simple and exact informed search method for the Traveling Salesman Problem with Time Windows and Makespan objective (TSPTW-M) that solves all instances of the classical benchmark with 50 or more customers in less than ten seconds each. Applying this algorithm as an off-the-shelf method, we also solve all but one of these instances for the Duration objective. Our main conclusion is that these instances should no longer be employed for evaluating the TSPTW-M and its Duration variant: they can be "hacked" to yield results that seem outstanding at first sight. Additionally, caution is advised when designing hard training sets for machine learning algorithms.

Keywords: traveling salesman problem with time windows, makespan objective, duration objective, informed search, benchmark instances, training datasets.

1 Introduction

In a Traveling Salesman Problem with Time Windows (TSPTW), a vehicle must visit a set of customers within their predefined time windows. The vehicle cannot arrive late to a customer, although it can arrive early and wait until the beginning of its time window. The journey begins at a depot to which the vehicle must return after visiting all the customers. Several variants of the TSPTW are defined, depending on the objective function. Three of the most studied variants involve minimizing the time to complete the route (TSPTW-M), the duration of the route (TSPTW-D), or the travel time of the vehicle, disregarding waiting times (TSPTW-TT). From here on, we use the acronym TSPTW to refer to one of these three variants.

Research on the TSPTW began over forty years ago with foundational papers, such as Christofides et al. (1981) and Baker (1983) for the TSPTW-M, Savelsbergh (1985) for the TSPTW-TT, and Savelsbergh (1992) for the TSPTW-D. Since then, numerous algorithms employing various solving strategies have been proposed for these three problem variants.

For exact approaches, significant efforts have focused on developing strong formulations suited for branch-and-bound and branch-and-cut methods (e.g. Baker, 1983; Langevin et al., 1993; Ascheuer et al., 2001; Kara et al., 2013), including time-expanded networks (e.g. Dash et al., 2012; Boland et al., 2017), dynamic programming (both pure and incomplete) (e.g. Dumas et al., 1995; Mingozzi et al., 1997; Balas and Simonetti, 2001), and anytime informed search methods (e.g. Fontaine et al., 2023; Kuroiwa and Beck, 2023). Other methods include dynamic programming within column generation and Lagrangian relaxation (e.g. Baldacci et al., 2012; Tilk and Irnich, 2017; Lera-Romero et al., 2022), constraint programming (e.g. Pesant et al., 1998; Focacci et al., 2002), and more recently, (multivalued) decision diagrams (e.g. Gillard et al., 2021; Rudich et al., 2023; Coppé et al., 2024).

In terms of heuristic and metaheuristic approaches, in addition to the pioneering works by Savelsbergh (1985) and Savelsbergh (1992), notable contributions include heuristic methods by Gendreau et al. (1998); Calvo (2000); Helsgaun (2017), tabu search by Carlton and Barnes (1996), compressed annealing by Ohlmann and Thomas (2007), and ant colony optimization (ACO) along with its Beam-ACO extensions by Favaretto et al. (2006); López-Ibáñez and Blum (2010); López-Ibáñez et al. (2013). Other significant approaches include general variable neighborhood search (e.g. da Silva and Urrutia, 2010; Mladenović et al., 2013; Amghar et al., 2019; Ye et al., 2024), iterated maximum large neighborhood search by Pralet (2023), and more recently, machine learning-based solvers (e.g. Cappart et al., 2021; Kool et al., 2022; Zheng et al., 2023; Bi et al., 2024; Chen et al., 2025; Li et al., 2025), among others.

Despite the many techniques developed for the TSPTW, the performance of most algorithms is evaluated on a subset of the *classical* benchmark instances compiled by López-Ibáñez and Blum in `https://lopez-ibanez.eu/tsptw-instances`, regardless of the objective function under consideration. The common reasoning is that the validity of an instance depends more on the space of feasible routes than on the objective function. Many machine learning algorithms replicate the generation process of some of these instances to create the large datasets of "hard" instances needed for training purposes (see e.g. Cappart et al., 2021; Kool et al., 2022; Bi et al., 2024; Li et al., 2025). A recent exception to this rule for evaluating the TSPTW is Fontaine (2023), who designs hard instances with few customers based on a benchmark by Rifki et al. (2020).

From the perspective of exact solvers, the classical benchmark is nearing the end of its useful life. Baldacci et al. (2012) solved all but one of the instances for the TSPTW-TT in at most one hour; Rudich et al. (2023); Fontaine et al. (2023) jointly solved all but five instances for the TSPTW-M in at most one hour; and Lera-Romero et al. (2022) solved all but fourteen of the considered instances for the TSPTW-D in at most three hours. A natural question then arises: how will we develop the next generation of benchmarking instances? da Silva and Urrutia (2010) addressed this question by creating a benchmark of large instances with 400 customers, using similar methods to those employed for the classical instances. Interestingly, Fontaine et al. (2023) solved the benchmark by da Silva and Urrutia (2010) in a few minutes, even though it is unable to solve some instances by Fontaine (2023); Rifki et al. (2020) with 30 customers within an hour.

The idea that the hardest instances for the TSPTW are those with loose time windows is widely accepted in the literature (see e.g. Dumas et al., 1995). What sets the benchmark proposed by Fontaine (2023) apart is its use of a parameter, β, to systematically widen the time windows, as suggested by Arigliano et al. (2019) for time-dependent problems. In a recent study, Rifki and Solnon (2025) investigate the effects of β and another parameter, α, on the feasibility and hardness of Euclidean instances for the TSPTW-M. Their main finding regarding feasibility is the presence of a pronounced phase transition between the infeasible and feasible regions. On the topic of hardness, they empirically demonstrate that the exact method proposed by Fontaine et al. (2023) benefits from tight time windows. As the authors note, their analysis is crucial for the development of next-generation of benchmark instances and for training machine learning algorithms that requires datasets with varying levels of difficulty. More recently, Chen et al. (2025) have questioned the time window generation processes used in previous machine learning studies, suggesting that they result in weak instances that are too easy to solve.

1.1 Our contributions

The main purpose of this article is to reinforce the conclusions of Rifki and Solnon (2025) and Chen et al. (2025), demonstrating that the classical benchmark instances with 50 or more customers, as well as the benchmark instances by da Silva and Urrutia (2010), should no longer be used to evaluate the TSPTW-M. This is because these instances can be solved by a simple exact algorithm in less than ten seconds each. Indeed, any method that includes a variant of our solver as a preprocessing step could yield misleading results that appear outstanding at first glance, regardless of how well does the algorithm performs after this preprocessing phase. Our method works by repeatedly running a simple informed search in a backward direction, prioritizing partial routes that reach the end depot earlier. Despite its simplicity, it is surprising that the algorithm can solve all the large instances, including the four that remained open in previous studies. However, the algorithm has serious limitations: it fails to solve some classical benchmark instances

with fewer than 50 customers and cannot solve any instance in the benchmark by Fontaine (2023); Rifki et al. (2020) with 30 or 40 customers and loose time windows.

Regarding the TSPTW-D, we show that using our solver for the TSPTW-M as an off-the-shelf method is sufficient to solve all but one of the classical benchmark instances with 50 or more customers, including the fourteen instances that remained unsolved in the study by Lera-Romero et al. (2022). This time, all instances are solved in less than 30 minutes each. These results further reinforce the notion that the classical benchmark instances are particularly easy for our informed search method and, therefore, they should not be used to evaluate the efficiency of algorithms for the TSPTW-D either.

Although our main conclusion is that the classical benchmark instances should be avoided when evaluating the TSPTW-M and the TSPTW-D, we believe that caution is also required when evaluating the TSPTW-TT. Moreover, we argue that our results should be considered when designing new benchmark sets or training datasets for machine learning methods. Neglecting to do so could lead to biased conclusions about the efficiency and effectiveness of the solvers (Chen et al., 2025). The ideas proposed by Arigliano et al. (2019), Fontaine (2023), and Rifki and Solnon (2025), which use a time window tightness parameter β to widen the time windows, provide a solid foundation for generating harder benchmark instances–even with fewer customers.

The fact that changing β yields harder instances certainly helps researchers evaluate the efficiency of their methods, but it does not imply that the synthetic time windows generated are significant for real-life situations. For example, Ascheuer (1996) discuss a real-life problem where tight time windows are designed to ensure that jobs are completed promptly in an online setting. The short question, whose answer remains unclear to us, is whether the classical benchmark instances are truly representative of real-world problems. An affirmative answer could suggest that our algorithm, while too simplistic as a general purpose approach, might still be valid as a preprocessing step for filtering out easy instances before applying more sophisticated techniques.

2 Problem Statement

Throughout this article, we write $[j] = [0, j]$, $[\![i, j]\!] = [i, j] \cap \mathbb{N}$, and $[\![j]\!] = [\![0, j]\!]$ for $i, j \in \mathbb{R}$. Consider a *transport network* described by a complete digraph D with vertex set $[\![n+1]\!]$. Vertices 0 and $n+1$ represent the *start* and *end depots*, respectively, whereas the vertices in $[\![1, n]\!]$ correspond to *customers*. Each arc $v \to w$ of D has a *travel time* $\tau(v, w)$ representing the time required to travel from v to w. Additionally, each vertex $v \in [\![n+1]\!]$ has a nonempty *time window* $[a(v), b(v)]$ in which v must be visited. We assume that $[a(v), b(v)] \subseteq [a(0), b(0)] = [a(n+1), b(n+1)] = [T]$ for $v \in [\![1, n]\!]$, where T is the *planning horizon*.

A *partial route* is a nonempty sequence of vertices in D. A partial route $R = \langle v_0, \dots v_k \rangle$ is a *route* if $v_0 = 0$ and $v_k = k = n+1$, while it is *elementary* when $v_i \neq v_j$ for $i, j \in [\![k]\!]$, $i \neq j$. Thus, R is an elementary route if and only if it visits each customer in D exactly once.

For $i \in [\![k]\!]$ and $t \in [T]$, we can compute the earliest arrival time to v_i when the vehicle traverses R departing from v_0 at time t using the following recurrence that takes waiting times into account:

$$\delta(R, t, i) = \begin{cases} \max\{t, a(v_0)\} & \text{if } i = 0, \\ \max\{\delta(R, t, i-1) + \tau(v_{i-1}, v_i), a(v_i)\} & \text{otherwise.} \end{cases}$$

The partial route R is *feasible* for time t if $t \geq a(v_0)$ and $\delta(R, t, i) \leq b(v_i)$ for every $i \in [\![k]\!]$. Let $\delta(R, t) = \delta(R, t, k)$ if R is feasible, and $\delta(R, t) = \infty$ otherwise. We refer to $\delta(R, t)$ as the *(earliest feasible) arrival time* of R *departing at time* t. The *makespan* of R is $\delta(R) = \delta(R, a(v_0))$, while its *duration* is $\Delta(R) = \min\{\delta(R, t) - t \mid t \in [T]\}$. The goal of the TSPTW-M (resp. TSPTW-D) is to find an elementary route with minimum makespan (resp. duration). Observe that the makespan (resp. duration) of such *optimum routes* is infinity when no elementary route is feasible for time 0.

Sometimes it is convenient to know the departure time of the vehicle to arrive the last node of a partial route R at a given time $t \in [T]$. We let $\delta^{-1}(R, t)$ be the maximum $t' \in [T]$ such that $\delta(R, t') = t$ ($t' = -\infty$ if no such t' exists). We refer to $\delta^{-1}(R, t)$ as the *(latest feasible) departure time* of R *arriving at time* t.

2.1 Benchmark instances

In this article, we consider 10 sets of benchmark instances, comprising a total of 1337 instances. The first 7 sets contain 467 instances and correspond to the classical instances compiled by López-Ibáñez and Blum, available at `https://lopez-ibanez.eu/tsptw-instances`. As explained in Section 1, the efficiency of most algorithms solving TSPTWs is evaluated on a subset of these instances. Following Fontaine (2023), we refer to the individual benchmarks as Asc (Ascheuer, 1996), Dum (Dumas et al., 1995), Gen (Gendreau et al., 1998), Lan (Langevin et al., 1993), Ohl (Ohlmann and Thomas, 2007), Pes (Pesant et al., 1998), and Pot (Potvin and Bengio, 1996). The travel times in Ohl are obtained by rounding the original distances to the nearest integer. The eighth benchmark, called Ohl-D, contains the same 25 instances as Ohl but with travel times rounded to the to the nearest tenth. This benchmark was considered in the context of the TSPTW-D. The ninth set, called DaS, comprises the 125 instances from da Silva and Urrutia (2010), while the last set, called Rif, contains the remaining 720 instances from Fontaine (2023), based on a benchmark by Rifki et al. (2020).

To provide some context for the results discussed in the subsequent sections, we briefly review the origins of these benchmark sets. Our primary focus is on the time windows, as their distributions appear to be more significant than the number of customers (see e.g. Rifki and Solnon, 2025). We also note that many training sets for machine learning methods are generated using procedures similar to those employed in the creation of Lan, Dum, Gen, Ohl, and DaS (e.g. Cappart et al., 2021; Kool et al., 2022; Bi et al., 2024; Li et al., 2025).

- Lan instances distribute the locations of a depot and $n \in \{20, 40, 60\}$ customers uniformly in $[\sigma] \times [\sigma]$ with $\sigma = 100$. The Euclidean distance, rounded to the nearest tenth, is used as the travel time. Time windows are set by computing a second nearest-neighbor route $R = \langle v_1, \ldots, v_{n+1} \rangle$, and assigning $a(v_i) = \delta(R, 0, i) - \mathcal{U}[0, \omega/2]$ and $b(v_i) = \delta(R, 0, i) + \mathcal{U}[0, \omega/2]$ for a width $\omega \in \{40, 60, 80\}$. These instances are now considered trivial for the TSPTW-M.

- Dum instances were created using the same procedure as Lan, but with $\sigma = 50$ for $n \in \{20, 40, 60\}$ and $\omega \in \{20, 40, 60, 80, 100\}$; $n = 80$ and $\omega \in \{20, 40, 60, 80\}$; $n \in \{100, 150\}$ and $\omega \in \{20, 40, 60\}$; and $n = 200$ and $\omega \in \{20, 40\}$. Quoting Dumas et al. (1995), "for narrow widths, [our algorithm] is less than exponential [...,] a 250-node problem with $\ell = 20$ is easy to solve in less than 10 seconds."

- Gen extends the benchmark by Dum for $n \in \{20, 40, 60, 80, 100\}$ and $\ell \in \{60, 80, \ldots, 200\}$. In their words, "[Our algorithm] would appear to be slower [...] on instances with narrow time windows. As noted by Dumas et al., the running time of their exact algorithm increases exponentially with time window width. [We generated our instances] to test the performance [...] on instances with wide time windows."

- Ohl also extends Dum for $n = 150$ and $\ell \in \{120, 140, 160\}$, and for $n = 200$ and $\ell \in \{120, 140\}$. Their justification is that their instances "show a solution method's ability to not only cope with wide time windows, but also with large numbers of customers."

- DaS was designed to "overcome the limitations of the other test sets" as da Silva and Urrutia (2010) needed "instances with more than 200 customers and wider time windows to benchmark and compare algorithms." Additionally, they believed that the method by Dumas et al. (1995) was "a biased way to generate instances since it does not reflect the real cases. Moreover, there is no challenge to build a feasible solution because the second-nearest neighbor tour already is a feasible solution." To address these concerns, they generate random routes to assign the time windows, with $n \in \{200, 250, 300, 350, 400\}$, $\sigma = 100$, and $\omega \in \{100, 200, 300, 400, 500\}$.

- Pot and Pes instances were obtained by extracting the route of a vehicle from a solution to Solomon's RC2 instances for the vehicle routing problem with time windows. They are considered hard but have at most 45 customers.

- Asc instances are derived from a real-world application that minimizes the unloaded travel time of a stacker crane in an online setting. The number of customers ranges from 10 to 231, while the travel time windows are tight to ensure jobs are not delayed for too long.

- Rifki et al. (2020) designed a benchmark set for time-dependent routing problems with travel times computed from shortest paths in the road network of Lyon, using realistic traffic simulations built from real-world data. To create Rif, Fontaine (2023) removed the time dependency from the instances with $n \in \{20, 30, 40\}$. Regarding time windows, Fontaine (2023) follows the model presented by Arigliano et al. (2019) for time-dependent instances, setting $a(v_i) = (\beta\delta(R, 0, i) - 40)$ and $b(v_i) = \delta(R, 0, i) + 40$ for a random route $R = \langle v_1, \ldots, v_{n+1} \rangle$ and $\beta \in \{0, 0.25, 0.5, 1\}$. Here, $\beta \in [1]$ represents the tightness of the time windows, from the loosest 0 to the tightest 1.

3 A Simple Solver for the TSPTW-M

The solver for the TSPTW-M builds upon the best-first search algorithm depicted in Algorithm 1. Given $t_0, ub \in [T]$, Algorithm 1 determines whether an elementary route R in the transport network D is feasible and arrives earlier than ub when departing at time t_0, i.e., whether $\delta(R, t_0) \leq ub$. If $t_0 = 0$, then the makespan of R is less than ub.

For convenience, for every partial route R starting at vertex v, we define: $\delta_{\mathrm{m}}(R) = \delta(R, \max\{t_0, a(v)\})$ as the earliest arrival time when the vehicle leaves the depot at time t_0, and $\delta_{\mathrm{m}}^{-1}(R) = \delta^{-1}(R, ub)$ as the latest departure time when the vehicle cannot arrive at the end depot after time ub.

In essence, Algorithm 1 is an informed search procedure that traverses a tree of partial routes $\mathcal{T}$ in a backward direction. The root of $\mathcal{T}$ is the partial route $\langle n + 1 \rangle$, which only visits the end depot, while the children of a partial route R are all the elementary partial routes $\langle w \rangle + R$ obtained by prepending a vertex $w \notin V(R)$ to R. Here, $V(R)$ denotes the set of vertices set already visited by R.

Within Algorithm 1, $\mathcal{R}$ maintains the *unprocessed* nodes of $\mathcal{T}$, which have already had their parent processed (Steps 1 and 12). Since $\mathcal{T}$ contains $\Theta(n!)$ nodes, two techniques are employed in Step 12 to prevent the exploration of the entire tree. First, a set of *unreachable* vertices $U(w, \delta_{\mathrm{m}}^{-1}(R))$ is computed for every partial route R that starts at vertex w, as described in Section 3.1. By construction, if $U(w, \delta_{\mathrm{m}}^{-1}(R)) \not\subseteq V(R)$, then no elementary route R' with suffix R and $\delta(R') < ub$ exists, and therefore the subtree of $\mathcal{T}$ rooted at R can be discarded. Similarly, if $\delta_{\mathrm{m}}(R) \geq ub$, then $\delta_{\mathrm{m}}(R') \geq ub$ for every route R' with suffix R, and so this subtree can also be pruned. These pruning strategies ensure that any partial route R obtained in Step 6 is feasible and has $\delta_{\mathrm{m}}(R) < ub$.

Following Fontaine et al. (2023), each iteration of the main loop (Steps 2–12) processes one route R of length k, if one exists, for every $k \in [\![n + 1]\!]$ (Steps 3–12). All extensions of R in Step 12 will have length $k + 1$, and thus can be processed within the same iteration of the main loop. The goal is to compute the output route as early as possible. To guide the search, a heuristic is applied to decide which partial route R of length k to process next (Step 5).

Note that $ub - \delta_{\mathrm{m}}(R)$ represents an upper bound on the time that can still be saved by any route with suffix R. In the extreme case where $\delta_{\mathrm{m}}(R) \geq ub$, no improvement is possible, and R is discarded (Step 12). Maximizing $ub - \delta_{\mathrm{m}}(R)$ maximizes the opportunity for the vehicle to wait at previous customers, if necessary. In event of ties, Step 5 minimizes $\delta^{-1}(R, \delta_{\mathrm{m}}(R))$ to maximize the earliest departure time of R that yields no waiting due to time windows. The goal is to allow the vehicle to wait at the previous (yet unknown) customers without significantly affecting the earliest arrival time.

Finally, Algorithm 1 avoids extending a partial route R if it can prove that it is ub-dominated. Formally, R is *ub-dominated* by a partial route Q of length $|R|$ if, for every route R' with suffix R, either the earliest arrival time of R' is $\delta_{\mathrm{m}}(R') \geq ub$, or there exists a route Q' with suffix Q whose earliest arrival time is $\delta_{\mathrm{m}}(Q') < ub$. The extension of R can be avoided when it is ub-dominated by an extended partial route $Q \neq R$. Algorithm 1 applies the following rule to discard ub-dominated routes.

Proposition 1. *Let R be a partial route from vertex v to the end depot $n+1$ with $\delta_{\mathrm{m}}^{-1}(R) \geq t_0$. If there exists a partial route $Q \neq R$ from v to $n+1$ such that its earliest arrival time is $\delta_{\mathrm{m}}(Q) < ub$, it visits the vertices $V(Q) = V(R)$, its latest departure is $\delta_{\mathrm{m}}^{-1}(Q) \geq \delta_{\mathrm{m}}^{-1}(R)$ and one of the following holds:*

1. *its latest departure is $\delta_{\mathrm{m}}^{-1}(Q) > \delta_{\mathrm{m}}^{-1}(R)$, or*
2. *$\delta(R, \delta_{\mathrm{m}}^{-1}(R)) = ub$ (i.e., R cannot arrive earlier than ub when departing as late as possible), or*
3. *$\delta(Q, \delta_{\mathrm{m}}^{-1}(Q)) < ub$ (i.e., Q arrives earlier than ub when departing as late as possible).*

then R is ub-dominated by Q.

Steps 7–11 apply Proposition 1 to determine if R is ub-dominated. A table is maintained, indexed by a vertex set V and a vertex v, to store the greatest $\delta_{\mathrm{m}}^{-1}(Q)$ among every processed route Q with $V(Q) = V$ and first vertex v. The table also stores $\delta(Q, \delta_{\mathrm{m}}^{-1}(Q))$; note that $\delta_{\mathrm{m}}(Q) < ub$ follows by invariant. Because of the search heuristic, Algorithm 1 can process an ub-dominated partial route before extending any of its dominators. We note that Algorithm 1 could discard all the routes minimizing the makespan because some of their prefixes are ub-dominated. This explains why Algorithm 1 is only a decision method rather than an optimization method, which justifies the decision to stop as soon as a route with $\delta_{\mathrm{m}}(R) < ub$ is found.

Algorithm 1 Backward Best First Search Labeling

Input: a transport network D, a departure time $t_0 \in [T]$ and a latest arrival time $ub \in [T]$.
Output: a route R with $\delta_{\mathrm{m}}(R) < ub$ or a message that no such R exists.

1: **let** $\mathcal{R} = \{\langle n+1 \rangle\}$ be a family of partial routes
2: **while** R is nonempty**:**
3: **for all** $k \in [\![n+1]\!]$**:**
4: **if** $\mathcal{R}$ has no route of length k**: continue**
5: Remove from $\mathcal{R}$ a partial route R of length k minimizing $\langle \delta_{\mathrm{m}}(R), \delta^{-1}(R, \delta_{\mathrm{m}}(R)) \rangle$
6: **if** $k = n+1$**: output** R and halt
7: **let** v be the first vertex in R and $(d, a) := \mathrm{best}(V(R), v)$
8: **if** $\delta_{\mathrm{m}}^{-1}(R) < d$**: continue**
9: **if** $\delta_{\mathrm{m}}^{-1}(R) = d$ and $a < ub$**: continue**
10: **if** $\delta_{\mathrm{m}}^{-1}(R) = d$ and $\delta(R, \delta_{\mathrm{m}}^{-1}(R)) = ub$**: continue**;
11: **else: let** $\mathrm{best}(V(R), v) = \langle \delta_{\mathrm{m}}^{-1}(R), \delta(R, \delta_{\mathrm{m}}^{-1}(R)) \rangle$
12: **for all** $w \in [\![n]\!] - V(R)$**:** insert $R' = \langle w \rangle + R$ into $\mathcal{R}$ **if** $\delta_{\mathrm{m}}(R') < ub$ and $U(w, \delta_{\mathrm{m}}^{-1}(R')) \subseteq V(R')$
13: **output** "no route R exists with $\delta_{\mathrm{m}}(R) < ub$"

To solve the TSPTW-M, we repeatedly execute Algorithm 1 as described in Algorithm 2. The output of this process is the route that minimizes the earliest arrival time, $\delta(R, t_0)$, for a given departure time $t_0 \in [T]$.

Algorithm 2 Basic solver for the TSPTW-M

Input: a transport network D and a departure time $t_0 \in [T]$
Output: a route R minimizing $\delta_{\mathrm{m}}(R) < T$ or a message no such route exists.

1: **let** $R^* = \emptyset$ and $T' = T + 1$.
2: Compute the unreachable function for Step 12 as in Section 3.1
3: **while** Algorithm 1 with input D, t_0, and $ub = T'$ outputs a route R**:**
4: **let** $T' = \delta_{\mathrm{m}}(R)$ and $R^* = R$
5: **if** $R^* \neq \emptyset$**: output** R^*, **else output** "the desired route does not exist".

Implementation details. Although our implementation of Algorithm 1 is not particularly efficient, we have taken measures to improve its running time. Each partial route R is implemented with a label $\ell(R)$ that uses $O(1)$ bits. The label $\ell(R)$ is a tuple containing the following information:

- The first vertex v of R,
- A bitset representing the set of visited vertices $V(R)$,
- The latest departure time $\delta_{\mathrm{m}}^{-1}(R)$,
- The earliest arrival time $\delta_{\mathrm{m}}(R)$,
- The other limiting times $\delta(R, \delta_{\mathrm{m}}^{-1}(R, ub))$ and $\delta^{-1}(R, \delta_{\mathrm{m}}(R))$, and
- A pointer to the label representing its parent in $\mathcal{T}$.

By traversing the branch of $\mathcal{T}$ from node R using the parents pointers, we can obtain all the vertices in R in $O(n)$ time. Each time R is extended into a a partial route R' in Step 12, the times $\delta(R', \delta_{\mathrm{m}}^{-1}(R'))$ and $\delta^{-1}(R', \delta_{\mathrm{m}}(R'))$ are computed in $O(n)$ time. Additionally, $U(w, \delta_{\mathrm{m}}^{-1}(R'))$ is computed in $O(n)$ time, as discussed in Section 3.1.

The family of routes $\mathcal{R}$ is implemented using a priority queue that holds all the unprocessed routes of length k, for every $k \in [\![n+1]\!]$. Finally, we maintain a hash table best$[v]$ for each $v \in [\![n+1]\!]$, indexed by $V(R)$, which is used in Steps 7 and 11.

3.1 The Unreachable Function

As illustrated in Algorithm 2 (Step 2), we apply a preprocessing step to compute an *unreachable function* $U \colon [\![n+1]\!] \times [T] \to 2^{[\![n+1]\!]}$, which is later used by Algorithm 1 to prune some branches of the tree $\mathcal{T}$ of partial routes. For each vertex $v \in [\![n+1]\!]$ and time $t \in [T]$, the function $U(v,t)$ returns a set of vertices that cannot appear before v in any elementary and feasible route R that visits v at a time $t' \leq t$. By construction, $U(v,t) \subseteq U(v,t')$, so U is implemented by storing $U(v,t)$ only for those times $t \in [T]$ where the function changes. As a result, each $U(v, \bullet)$ has $O(n)$ values, which allows for queries to be answered in $O(n)$ time.

The computation of U follows these steps. First, we run an all-pairs shortest path algorithm twice to obtain the following functions:

$$\mathrm{EAT}(w,v) = \min\{\delta(R, a(w)) \mid R \text{ is a feasible route in } D \text{ ending at } v\} \cup \{\infty\}$$
$$\mathrm{LDT}(w,v) = \max\{\delta^{-1}(R, b(v)) \mid R \text{ is a feasible route in } D \text{ starting at } w\} \cup \{-\infty\}$$

for every pair of vertices v, w in D. Here, $\mathrm{EAT}(w,v)$ represents the earliest arrival time at v after visiting w, and $\mathrm{LDT}(w,v)$ represents the latest departure time from w before visiting v.

Next, we compute the precedence relation $\prec$, where $v \prec w$ if and only if $\mathrm{EAT}(w,v) = \infty$ or there exists a third vertex $z \in [\![n+1]\!] - \{v, w\}$ such that:

$$\mathrm{EAT}(z,w) > \mathrm{LDT}(w,v), \quad \mathrm{EAT}(w,v) > \mathrm{LDT}(v,z), \quad \text{and} \quad \mathrm{EAT}(w,z) > \mathrm{LDT}(z,v).$$

This relation ensures that no feasible route can visit w before v when $v \prec w$. Therefore, for every $t \in \mathbb{R}$, we set $w \in U(v,t)$ if $v \prec w$. Otherwise, we set $w \in U(v,t)$ if and only if $t < \mathrm{EAT}(w,v)$.

Once the precedence relation $\prec$ is computed, we update the time windows by setting:

$$a(w) := \max\{a(w), \mathrm{EAT}(v,w)\} \quad \text{and} \quad b(v) := \min\{b(v), \mathrm{LDT}(v,w)\}$$

for every pair of vertices $v, w \in [\![n+1]\!]$ such that $v \prec w$. If any time window is modified, we recompute U and repeat the process until no further changes are made.

3.2 Solving the integer TSPTW-D

Recall that the TSPTW-D consist of finding an elementary route R with minimum duration $\Delta(R) = \min\{\delta(R,t) - t \mid t \in [T]\}$ for a given transport network D. Clearly, if the travel times and all the time windows in D are integers, then a solution to the TSPTW-D can be obtained by running a solver for the TSPTW-M $O(T)$ times. This is the approach Algorithm 3 uses to solve the TSPTW-D.

Specifically, Algorithm 3 maintains a sliding window $[\![p,q]\!]$, a route R that minimizes $\delta(R,p-1) = q$, and a route R^* with a time t^* that minimizes $\delta(R^*,t^*) - t^*$ for $t^* \in [\![p-1]\!]$. Steps 1–3 initialize these variables. Step 3 also computes last_p, which is the latest of the departure times of all the elementary routes in D. Thus, when $p > \text{last}_p$, R^* is a route with minimum duration.

Throughout the main loop (Steps 4–9) the route R is updated into a new route that minimizes $\delta(R,p) \geq q$. Note that $\delta(R,p) = q$ when Step 7 is skipped. Then, q, R^*, and t^* are updated accordingly in Steps 7, 9, and 10.

Algorithm 3 Sliding window solver for the integer TSPTW-D

Input: a transport network D in which τ and the time windows are integer
Output: a route R^* and a time t^* minimizing $\Delta(R^*) < T$ or a message that D is infeasible.

1: **let** R be a route minimizing $\delta(R)$; halt with **output** "D is infeasible" **if** $\delta(R) = \infty$
2: Heuristically find a route R^* and a time t^* with $\delta(R^*,t^*) = \delta(R)$ maximizing t^*
3: **let** $p = t^* + 1$, $q = \delta(R^*)$, and $\text{last}_p = \max\{\delta^{-1}(R,T) \mid R \text{ is an elementary route}\}$
4: **while** $p \leq \text{last}_p$**:**
5: Heuristically update R to minimize $\delta(R,p)$
6: **if** $\delta(R,p) > q$**:**
7: **let** R be a route minimizing $\delta(R,p)$ and **let** $q = \delta(R,p)$
8: Heuristically update R and p to maximize p with $\delta(R,p) = q$
9: **if** $\delta(R^*,t^*) - t^* > \delta(R,p) - p$**: let** $R^* = R$ and $t^* = p$
10: **let** $p = p + 1$
11: **output** R^* and t^*

Regarding the implementation of Algorithm 3, we run Algorithm 2 with $t_0 = 0$ and $ub = T+1$ for Step 1. Similarly, we invoke Algorithm 2 with $t_0 = p$ and $ub = \min\{\text{last}_q + \Delta(R^*), T+1\}$ for Step 7. Let D^{-1} be the *reverse* transport network with vertex set $[\![n+1]\!]$, where the travel time of an arc $v \to w$ is $\tau(w,v)$, and the time window of a vertex v is $[T - b(v), T - a(v)]$. In D^{-1}, the start depot is $n+1$ and the end depot is 0. To initialize last_p in Step 3, we run Algorithm 2 on D^{-1} with $t_0 = 0$ and $ub = T - t^*$. If a route R^{-1} in D^{-1} is found, then $\text{last}_p = T - \delta(R^{-1}, 0)$; otherwise, $\text{last}_p = t^*$. Finally, we use a local search heuristic for Steps 5 and 8, which applies the swap, 2-opt, and shift operators with a first-improvement strategy. Our implementation is not particularly efficient, as it spends $O(n^2)$ time on each operator.

4 Computational Results

We implemented Algorithms 1–3 in C++20 to evaluate their performance on the benchmark instances from Section 2.1. The code is freely available at `https://github.com/fsoulignac/tsptw-m`. We executed our experiments on a single thread of a laptop equipped with an AMD Ryzen 7 3700U CPU@2.3GHz and 6GB of physical RAM. To limit the impact of paging on the running time, we restricted the available RAM to 5GB.

4.1 Results for the TSPTW-M

For the TSPTW-M we set a time limit of 3 minutes per instance. Although this time limit is quite restrictive for an exact method, it is sufficient for our purposes and helps prevent unnecessary computation time before encountering a memory limit exception. Furthermore, this time limit is representative of a typical use case,

		Ler22		Rud23		Fon23		Algorithm 2		
	#	s^*	t_s^*	s	t_s	s^*	t_s^*	s	t_s	m_s
Asc	50	48	21	48(48)	173(109)	50(49)	18(2)	50	13	152
Asc-s	32							32	21	152
Asc-l	18							18	0	0
DaS	125	125	485	112*	615*	125(125)	13(13)	125	2	7
Dum	135	135	39	133(135)	154(111)	135(135)	0(1)	135	0	1
Gen	130	91	524	122(125)	274(142)	117(110)	111(88)	130	0	1
Lan	70	70	0	70(70)	1(1)	70(70)	0(0)	70	0	0
Ohl	25	0	—	14(22)	1888(924)	20(20)	50(9)	25	1	6
Pes	27	22	203	26(27)	175(84)	25(27)	152(66)	25	16	164
Pot	30	25	160	28(28)	185(66)	27(29)	14(83)	25	7	80
Total	592	516		553(567)		569(565)		585		

Table 1: Results of Algorithm 2 on the classical benchmarks. Entries marked with * were reported by Fontaine (2023).

where an enumerative algorithm is applied for a short duration after computing a tight lower bound (see e.g. Baldacci et al., 2012; Tilk and Irnich, 2017; Lera-Romero et al., 2022).

The results for the classical benchmarks and DaS are summarized in Table 1. We compare Algorithm 2 against the methods by Lera-Romero et al. (2022) (Ler22), Rudich et al. (2023) (Rud23), and Fontaine et al. (2023) (Fon23), which represent the state of the art for the TSPTW-M within exact methods. Ler22 and Fon23 were originally proposed for time-dependent versions of the TSPTW, and thus Lera-Romero et al. (2022) and Fontaine et al. (2023) did not report the results for the TSPTW-M. Similarly, Rudich et al. (2023) did not test Rud23 on the DaS benchmark. These missing results reported in Table 1 are taken from Fontaine (2023). We note that the results in Fontaine (2023) differ from those reported by Rudich et al. (2023) because Fontaine (2023) executed all the algorithms on the same machine and with the same configuration. Our computer has a slightly higher performance for single thread execution according to `www.cpubenchmark.net`, and, unlike Fontaine (2023), we did not disable turbo mode (we kept the ondemand governor within Ubuntu). Additionally, the time limit considered by Fontaine (2023) was one hour per instance. These discrepancies do not affect our main conclusions, and we thus disregard them.

In Table 1, column # counts the number of instances in each benchmark set, columns s indicate the number of solved instances, and columns t_s report the average time for the solved instances, rounded to the nearest second. For Algorithm 2, we also include the maximum running time among the solved instances in column m_s. Additionally, we divide the results for Asc into Asc-s and Asc-l to separately report the results for instances with fewer than 50 customers and those with 50 or more customers, respectively. Table 1 reports the results for Rud23 with the unseeded version outside parentheses and the seeded version inside parentheses. The difference between the two is that the latter is initialized with a known solution. In this regard, Algorithm 2 should be considered unseeded. Similarly, Table 1 reports the results for Fon23 for two versions of the heuristic guiding the informed search. The results for FEA are reported outside parentheses, and those for MSA are reported inside parentheses.

A quick look at Table 1 is enough to conclude that Algorithm 2 outperforms the state-of-the-art methods on the larger instances, solving the four instances that remained unsolved. Although this is strictly true, knowing the inner workings of Algorithm 2, we would never prefer it as a generic exact solver for the TSPTW-M. A deeper look at Table 1 raises an important concern: Algorithm 2 is unable to solve seven instances with 45 or fewer customers (and our unreported experiments show that memory is exhausted when the time limit is high enough). Furthermore, Algorithm 2 performs better on the larger instances in Asc than on the smaller ones. As strange as it may seem, Rifki and Solnon (2025) already noted that the ratio between the size of the time windows and the horizon could be more important than the number of customers when evaluating the efficiency of dynamic programming solvers for the TSPTW-M. Finally, we

β	n	Ler22		Rud23		Fon23		Algorithm 2		
		s	t_s	s	t_s	s	t_s	s	t_s	m_s
100	20	60	0	60	0	60	0	60	0	0
	30	60	0	60	1	60	0	60	0	0
	40	60	0	60	1	60	0	60	0	0
50	20	60	5	60	9	60	0	60	0	2
	30	60	129	58	105	60	1	59	5	125
	40	60	888	14	350	60	61	58	18	117
25	20	60	16	60	17	60	0	60	10	60
	30	60	643	27	361	60	36	0	—	—
	40	58	2318	0	—	47	1221	0	—	—
0	20	60	59	60	25	60	0	60	35	177
	30	60	1376	10	434	60	177	0	—	—
	40	34	2837	0	—	5	2011	0	—	—
Total		692		469		652		506		

Table 2: Results of Algorithm 2 on the Rif benchmark.

observe that Algorithm 2 is particularly fast on Lan, Dum, Gen, Ohl, and DaS, all of which follow the same creation procedure commonly applied in many machine learning methods to create "hard" training datasets (Section 2.1).

What we believe is happening is that Algorithm 2 takes advantage of some inherent bias in the benchmark instances resulting from the tight time windows, which yield a small search space of feasible solutions that have a well defined structure. Fon23 is another informed search that seems to have an advantage because of the structure of the instances, as depicted by the Ohl benchmark: the 20 instances it solves are completed in just a few seconds, while Ler22 is unable to solve any of them within an hour! The difference is that Ler22 was designed to solve as many as possible of the small but challenging instances. As a result, it spends a large amount of time computing penalties on a relaxed problem before running its enumeration algorithm, and it uses a pure best-first search heuristic to guide its informed search, avoiding the extension of dominated labels. Finally, Rud23 is a generic algorithm based on multivalued decision diagrams that iteratively traverses enumeration trees of different widths to obtain lower and upper bounds to "peel" the tree. Being able to find feasible solutions quickly improves the peeling process.

To present the contrasting results, we report the performance of the algorithms for the Rif benchmark in Table 2. (The results for Ler22, Rud23, and Fon23 were taken from Fontaine, 2023). Again, Algorithm 2 is among the fastest for tight time windows ($\beta \in \{50, 100\}$), but it is clearly the worst for the looser time windows ($\beta \in \{0, 25\}$), even when compared against the generic method Rud23. (We remark that increasing the time limit does not help in these cases, as the memory limit is always reached.) The clear winner in robustness is Ler22, which was specifically designed for harder instances in terms of time windows.

In our opinion, the main conclusion from the experiments in this section is that the classical benchmarks are no longer adequate for evaluating the TSPTW-M, as they can be "exploited" by a simple algorithm. Consequently, any method that incorporates a variant of Algorithm 2 in its preprocessing step would likely produce results that appear outstanding to an external reviewer.

4.2 Results for the TSPTW-D

For the TSPTW-D, we set a time limit of 30 minutes per instance. Table 3 summarizes the results for the benchmarks Asc, DaS, Gen, Ohl, and Ohl-D. It includes the results of Algorithm 3 applied to the transport network D (columns labeled Algorithm 3) and its reverse network D^{-1} (columns labeled Algorithm 3^{-1}), as the method is not symmetric, and the running times differ considerably between the two. For comparison,

		Til22		Ler22		Algorithm 3			Algorithm 3^{-1}		
	#	s	t_s	s	t_s	s	t_s	m_s	s	t_s	m_s
Asc	50	50	55	47	8	46	9	295	45	3	60
DaS	125	—	—	—	—	125	2	7	125	2	8
Dum	135	—	—	—	—	135	0	1	135	0	1
Gen	130	97/115	633	114/115	667	123	30	1279	114	3	173
Ohl	25	—	—	—	—	25	15	216	25	3	27
Ohl-D	25	1	731	12	5159	20	10	59	25	92	1093

Table 3: Results of Algorithm 3 on the classical benchmarks.

Table 3 also reports results from Tilk and Irnich (2017) and Lera-Romero et al. (2022).

We do not consider the benchmarks Pot and Pes because their travel times are rounded to 10^{-4}, and Algorithm 3 cannot handle the large number of TSPTW-M instances in the range $[10^4 T]$. This also applies to the four unsolved instances in Asc (rbg021.x for $x \in [\![6, 9]\!]$), which were derived from rbg021 by increasing both the horizon and the deadline times in the time windows. Additionally, we exclude the benchmark Lan because its instances are trivial and were not tested by Tilk and Irnich (2017) or Lera-Romero et al. (2022). We also note that Tilk and Irnich (2017); Lera-Romero et al. (2022) did not evaluate their algorithms on the Gen instances with $n \geq 80$ and $\omega \in \{80, 100\}$. It is worth mentioning that Lera-Romero et al. (2022) solved all the instances in Pot, while Tilk and Irnich (2017) solved all but three. However, both methods were tested with a time limit of 3 hours per instance.

Table 3 reinforces the conclusions from Section 4: Algorithm 3 (on either D or D^{-1}) solves all but one of the classical instances with 50 or more customers, including the 14 instances that were not solved by Tilk and Irnich (2017) or Lera-Romero et al. (2022). Furthermore, Algorithm 3 outperforms both methods on the larger instances with higher values of n.

By design, one might expect Algorithm 3 to be much less efficient than Algorithm 2, as it solves $\Theta(T)$ instances of the TSPTW-M in the worst case. However, as implied by Table 3, only a small number of these instances are actually passed to Algorithm 2 for solving in the benchmark instances. Most feasible departure times p in Loop 4–9 are either solved by the local search heuristic or ignored because of the best-known solution R^*. Moreover, last_p is typically small in the largest instances, which helps reduce the number of instances solved by Algorithm 2.

The main conclusion is that the TSPTW-D should be evaluated on "harder" instances, especially those with wider time windows, to better assess the performance of algorithms in more challenging scenarios.

5 Conclusions

We presented a simple informed search method for the TSPTW-M, which excels in solving all classical benchmark instances with 50 or more customers in under 10 seconds each. However, it struggles to solve even the smallest instances when their time windows are sufficiently loose. By running the algorithm as a preprocessing step for a limited amount of time, any method can solve these instances, meaning they should no longer be used for benchmarking purposes.

An important question that arises from this work is how valid and accurate are the conclusions drawn from methods that have been tested exclusively on these classical benchmark instances? Are these methods truly efficient in a general setting, or are they, perhaps unconsciously, exploiting some inherent bias in these benchmarks? Since these instances can be "hacked" by simpler methods, it is crucial to examine how generalizable the reported results are. Exploring this issue further could help ensure that conclusions drawn from these benchmarks are not only valid for the specific instances tested but also robust across a wider range of real-world problem scenarios.

Our method performs particularly well on the benchmarks Lan, Dum, Gen, Ohl, and DaS, all of which share the common strategy of defining a time window $[x - \omega, x + \omega]$ for a vertex visited at time x in a

given route. Here, ω is relatively small when compared to n and T. Due to the simplicity of this approach, it is commonly used to generate training datasets for many machine learning methods, which can lead to overfitting and biased conclusions. As noted by Rifki and Solnon (2025), a simple solution to make these instances more challenging for an informed search algorithm like ours is to extend the time windows. The varying levels of difficulty among these generated instances provide a better foundation for drawing deeper conclusions.

Given the exceptional performance of our method on the larger benchmark instances, it is not surprising that a trivial procedure of repeatedly running the solver across the entire horizon is sufficient to solve the TSPTW-D on most of the instances. The issue is that many of these instances are easy even for the TSPTW-D. Therefore, these instances should also be avoided when evaluating methods for the TSPTW-D. What remains to be explored is whether a similar informed search can be applied to solve the TSPTW-TT on these instances. Nonetheless, we believe it would be wise to evaluate and train future methods on instances with varying levels of time window tightness, even for the TSPTW-TT.

References

K. Amghar, J.-F. Cordeau, and B. Gendron. A general variable neighborhood search heuristic for the traveling salesman problem with time windows under completion time minimization. Technical report, CIRRELT, 2019.

A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, and I. Plana. Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. *Discret. Appl. Math.*, 261: 28–39, 2019. doi:https://doi.org/10.1016/j.dam.2018.09.017.

N. Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, Zuse Institute Berlin, 1996.

N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Math. Program.*, 90:475–506, 2001. doi:10.1007/PL00011432.

E. K. Baker. Technical note—an exact algorithm for the time-constrained traveling salesman problem. *Oper. Res.*, 31(5):938–945, 1983. doi:10.1287/opre.31.5.938.

E. Balas and N. Simonetti. Linear time dynamic-programming algorithms for new classes of restricted tsps: A computational study. *INFORMS J. Comput.*, 13(1):56–75, 2001. doi:10.1287/ijoc.13.1.56.9748.

R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS J. Comput.*, 24(3):356–371, 2012. doi:10.1287/ijoc.1110.0456.

J. Bi, Y. Ma, J. Zhou, W. Song, Z. Cao, Y. Wu, and J. Zhang. Learning to handle complex constraints for vehicle routing problems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 93479–93509. Curran Associates, Inc., 2024. doi:10.52202/079017-2964.

N. Boland, M. Hewitt, D. M. Vu, and M. Savelsbergh. Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin and M. Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 254–262. Springer International Publishing, 2017. doi:10.1007/978-3-319-59776-8_21.

R. W. Calvo. A new heuristic for the traveling salesman problem with time windows. *Transp. Sci.*, 34(1): 113–124, 2000. doi:10.1287/trsc.34.1.113.12284.

Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *Proc. AAAI Conf. Artif. Intell.*, 35 (5):3677–3687, 2021. doi:10.1609/aaai.v35i5.16484.

W. B. Carlton and J. W. Barnes. Solving the traveling-salesman problem with time windows using tabu search. *IIE Trans.*, 28(8):617–629, 1996. doi:10.1080/15458830.1996.11770707.

J. Chen, Z. Gong, L. Chen, M. Liu, J. Wang, Y. Yu, and W. Zhang. Looking ahead to avoid being late: Solving hard-constrained traveling salesman problem. In *Proceedings of the 2024 6th International Conference on Distributed Artificial Intelligences*, DAI '24, page 1–12. Association for Computing Machinery, 2025. doi:10.1145/3719545.3759088.

N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Netw.*, 11(2):145–164, 1981. doi:https://doi.org/10.1002/net.3230110207.

V. Coppé, X. Gillard, and P. Schaus. Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection. *INFORMS J. Comput.*, 36(6):1522–1542, 2024. doi:10.1287/ijoc.2022.0340.

R. F. da Silva and S. Urrutia. A general vns heuristic for the traveling salesman problem with time windows. *Discrete Optim.*, 7(4):203–211, 2010. doi:https://doi.org/10.1016/j.disopt.2010.04.002.

S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS J. Comput.*, 24(1):132–147, 2012. doi:10.1287/ijoc.1100.0432.

Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Oper. Res.*, 43(2):367–371, 1995. doi:10.1287/opre.43.2.367.

D. Favaretto, E. Moretti, and P. Pellegrini. An ant colony system approach for variants of the traveling salesman problem with time windows. *J. Inf. Optim. Sci.*, 27(1):35–54, 2006. doi:10.1080/02522667.2006.10699677.

F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS J. Comput.*, 14 (4):403–417, 2002. doi:10.1287/ijoc.14.4.403.2827.

R. Fontaine. *Exact and anytime heuristic search for the Time Dependent Traveling Salesman Problem with Time Windows*. PhD thesis, INSA Lyon, 2023.

R. Fontaine, J. Dibangoye, and C. Solnon. Exact and anytime approach for solving the time dependent traveling salesman problem with time windows. *Eur. J. Oper. Res.*, 311(3):833–844, 2023. doi:https://doi.org/10.1016/j.ejor.2023.06.001.

M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Oper. Res.*, 46(3):330–335, 1998. doi:10.1287/opre.46.3.330.

X. Gillard, V. Coppé, P. Schaus, and A. A. Cire. Improving the filtering of branch-and-bound MDD solver. In P. J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247. Springer International Publishing, 2021. doi:10.1007/978-3-030-78230-6_15.

K. Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017.

I. Kara, O. N. Koc, F. Altıparmak, and B. Dengiz. New integer linear programming formulation for the traveling salesman problem with time windows: minimizing tour duration with waiting times. *Optim.*, 62 (10):1309–1319, 2013. doi:10.1080/02331934.2013.824445.

W. Kool, H. van Hoof, J. Gromicho, and M. Welling. Deep policy dynamic programming for vehicle routing problems. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 19th International Conference, CPAIOR 2022, Los Angeles, CA, USA, June 20-23, 2022, Proceedings*, page 190–213, Berlin, Heidelberg, 2022. Springer-Verlag. doi:10.1007/978-3-031-08011-1_14.

R. Kuroiwa and J. C. Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. *Proc. Int. Conf. Autom. Plan. Sched.*, 33(1):245–253, 2023. doi:10.1609/icaps.v33i1.27201.

A. Langevin, M. Desrochers, J. Desrosiers, S. Gélinas, and F. Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Netw.*, 23(7):631–640, 1993. doi:https://doi.org/10.1002/net.3230230706.

G. Lera-Romero, J. J. Miranda Bront, and F. J. Soulignac. Dynamic programming for the time-dependent traveling salesman problem with time windows. *INFORMS J. Comput.*, 34(6):3292–3308, 2022. doi:10.1287/ijoc.2022.1236.

T. Li, H. Zou, J. Wu, and Z. Wen. Lmask: Learn to solve constrained routing problems with lazy masking, 2025. URL `https://arxiv.org/abs/2505.17938`.

M. López-Ibáñez and C. Blum. Beam-ACO for the travelling salesman problem with time windows. *Comput. Oper. Res.*, 37(9):1570–1583, 2010. doi:https://doi.org/10.1016/j.cor.2009.11.015.

M. López-Ibáñez, C. Blum, J. W. Ohlmann, and B. W. Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Appl. Soft Comput.*, 13(9): 3806–3815, 2013. doi:https://doi.org/10.1016/j.asoc.2013.05.009.

A. Mingozzi, L. Bianco, and S. Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Oper. Res.*, 45(3):365–377, 1997. doi:10.1287/opre.45.3.365.

N. Mladenović, R. Todosijević, and D. Urošević. An efficient general variable neighborhood search for large travelling salesman problem with time windows. *Yugosl. J. Oper. Res.*, 23:19–30, 2013. doi:10.2298/YJOR120530015M.

J. W. Ohlmann and B. W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS J. Comput.*, 19(1):80–90, 2007.

G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transp. Sci*, 32(1):12–29, 1998. doi:10.1287/trsc.32.1.12.

J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows part II: Genetic search. *INFORMS J. Comput.*, 8(2):165–172, 1996. doi:10.1287/ijoc.8.2.165.

C. Pralet. Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version. *Comput. Oper. Res.*, 150:106078, 2023. doi:https://doi.org/10.1016/j.cor.2022.106078.

O. Rifki and C. Solnon. On the phase transition of the euclidean travelling salesman problem with time windows. *J. Artif. Intell. Res.*, 82:2167–2188, 2025. doi:https://doi.org/10.1613/jair.1.18334.

O. Rifki, N. Chiabaut, and C. Solnon. On the impact of spatio-temporal granularity of traffic conditions on the quality of pickup and delivery optimal tours. *Transp. Res. Part E*, 142:102085, 2020. doi:https://doi.org/10.1016/j.tre.2020.102085.

I. Rudich, Q. Cappart, and L. Rousseau. Improved peel-and-bound: Methods for generating dual bounds with multivalued decision diagrams. *J. Artif. Intell. Res.*, 77:1489–1538, 2023. doi:10.1613/JAIR.1.14607.

M. W. P. Savelsbergh. Local search in routing problems with time windows. *Ann. Oper. Res.*, 4:285–305, 1985. doi:10.1007/BF02022044.

M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA J. Comput.*, 4(2):146–154, 1992. doi:10.1287/ijoc.4.2.146.

C. Tilk and S. Irnich. Dynamic programming for the minimum tour duration problem. *Transp. Sci.*, 51(2): 549–565, 2017. doi:10.1287/trsc.2015.0626.

M. Ye, E. Bartolini, and M. Schneider. A general variable neighborhood search for the traveling salesman problem with time windows under various objectives. *Discrete Appl. Math.*, 346:95–114, 2024. doi:https://doi.org/10.1016/j.dam.2023.12.006.

J. Zheng, K. He, J. Zhou, Y. Jin, and C.-M. Li. Reinforced lin–kernighan–helsgaun algorithms for the traveling salesman problems. *Knowl. Based Syst.*, 260:110144, 2023. doi:https://doi.org/10.1016/j.knosys.2022.110144.