Iterated Maximum Large Neighborhood Search for the Traveling Salesman Problem with Time Windows and its Time-Dependent Version

Cédric Pralet

Iterated Maximum Large Neighborhood Search for the Traveling Salesman Problem with Time Windows and its Time-Dependent Version

Cédric Pralet

ONERA, Université de Toulouse, 2 av. Edouard Belin, BP 74025 F-31055 Toulouse Cedex 4, France

Abstract

This article introduces a new algorithm for finding feasible or makespan-optimal solutions of Traveling Salesman Problems with Time Windows (TSPTWs) and Time-Dependent TSPTWs (TDTSPTWs). The algorithm starts from a sequence of visits of the customers involved in the problem, uses destroy and repair operations to iteratively improve this sequence, and applies perturbations to diversify search. For the destroy phase, customers are removed from the current sequence of visits as long as a parameter called the insertion-width is not too high. For the repair phase, the customers removed are reinserted for the best based on a dynamic programming procedure whose complexity is only linear in the number of customers. For the perturbation phase, some customers are randomly shifted in the sequence of visits. The algorithm obtained is called Iterated Maximum Large Neighborhood Search (ImaxLNS). On seven standard TSPTW benchmarks, it returns the best-known solution for each instance in less than one second on average. On two TDTSPTW benchmarks related to urban logistics, it provides new feasible solutions and best solutions. On a TDTSPTW benchmark related to Earth observing satellites, it solves most of the instances in less than a second.

Keywords: Traveling Salesman Problem with Time Windows, Time-Dependent Transitions, Large Neighborhood Search, Makespan Minimization, Solution Feasibility

1. Introduction

In a Traveling Salesman Problem with Time Windows (TSPTW), a vehicle must visit a set of customers within allowed time windows while minimizing some objective function. The latter can be the sum of the travel times (TSPTW-TT) or travel costs (TSPTW-TC), the time at which the sequence of visits is completed (TSPTW-M), or the duration spent by the vehicle outside of its depot (TSPTW-D). Several extensions of TSPTW were also introduced over the years to answer application needs. Some extensions add precedence constraints between customer visits. Other extensions like Time-Dependent TSPTW (TDTSPTW) take into account travel times or travel costs depending on the time at which the vehicle moves, which is useful to model a vehicle subject to traffic conditions varying over the day.

Even without any objective, TSPTW is a challenging problem since determining whether there exists a feasible solution visiting all customers within their time windows is strongly NP-complete [1]. This is why

many resolution approaches were defined during the last four decades, including both exact methods that aim at producing optimal solutions and incomplete methods that quickly deliver good quality solutions. In this article, we introduce another incomplete method adapted to TSPTW and TDTSPTW. For this method, the objective is first to find a feasible solution and then to minimize the completion time of the sequence of visits (or makespan). From a global point of view, the algorithm proposed is based on a Large Neighborhood Search (LNS [2]), a metaheuristic that was shown to be efficient in various contexts [3]. Basically, LNS starts from a solution s to a given problem and iteratively updates s by searching at each step for a better solution in a large neighborhood. In LNS, the neighborhood is defined by a *destroy method* that removes some decisions from the current solution following a degree of destruction, and a *repair method* that rebuilds a full solution based either on complete search or on heuristic search.

In the LNS approach that we propose for (TD)TSPTW, a solution is simply a sequence of visits of the customers, the destroy method consists in removing a subset of customers from such a sequence, and the repair method uses dynamic programming for exploring all possible reinsertions of the customers removed. One particularity of the destroy method introduced is that it attempts to remove a maximum subset of customers from the incumbent solution under the constraint that the repair method must remain applicable with a limited complexity. The latter is measured by a parameter called the *insertion-width* of the partial solution obtained after destruction. On top of these destroy and repair mechanisms, we apply perturbations when a locally optimal solution is found, as in the Iterated Local Search metaheuristic (ILS [4]). These perturbations consist in performing 1-shift moves that randomly change the position of some customers in the sequence of visits. Restarts from empty solutions are also performed from time to time to diversify search. In the end, the algorithm proposed is called *Iterated Maximum Large Neighborhood Search (ImaxLNS)* because it combines LNS, ILS, and an effort to use a maximum destroy degree given the fixed-parameter complexity of the repair phase.

Several benchmarks are considered to show the efficiency of ImaxLNS. First, on seven TSPTW benchmarks covering 467 instances, ImaxLNS produces the best-known solution in less than a second for each instance on average. Second, on a TDTSPTW benchmark introduced by Arigliano et al. [5], ImaxLNS produces new feasible solutions and new best solutions. Third, on a TDTSPTW benchmark related to urban delivery problems [6], ImaxLNS provides 59 new best solutions over the 60 largest instances. Fourth, on a new TDTSPTW benchmark related to Earth observing satellites, it produces feasible solutions in less than a second for hard instances, which is good news since this application is the main motivation for this work.

The article is organized as follows. Section 2 provides some background on TSPTW and TDTSPTW. Section 3 formalizes the problem tackled. Section 4 describes preprocessing techniques. Section 5 introduces ImaxLNS. Sections 6 and 7 detail its destroy and repair phases. Section 8 provides experimental results, and Section 9 gives perspectives for this work. Most of the proofs are given in the supplementary material.

2. Literature review

This section provides an overview of the methods developed over the years for solving TSPTWs and TDTSPTWs. These methods are based on mathematical programming, dynamic programming, constraint programming, relaxations, local search, metaheuristics, or hybrid optimization.

TSPTW: exact methods. In the early works on TSPTW, Christofides et al. [7] defined a branch-and-bound algorithm for TSPTW-M where the bounding part uses dynamic programming and state-space relaxations. Afterwards, Baker [8] proposed another branch-and-bound method for TSPTW-M, and Ascheuer et al. [9] proposed branch-and-cut for TSPTW-TC. These early works also introduced preprocessing techniques to tighten the time windows available for visiting the customers, infer precedence constraints between some visits, or fix some decisions. On the modeling side, efforts were made to find good mathematical programming formulations [9, 10]. On this point, some authors studied integer linear programming models obtained from *time-expanded networks*, where the time window associated with each customer is discretized into a restricted set of possible visit times. Such time-expanded networks were introduced by Dash et al. [11] for a static network, and extended by Boland et al. [12] to a dynamic network refined step-by-step. These approaches were defined for TSPTW-TC but are applicable to TSPTW-M as well.

In parallel, several authors studied Dynamic Programming (DP) for TSPTW. Initially, Dumas et al. [13] used DP for TSPTW-TC. Their algorithm starts from an empty sequence of visits and explores the possible paths extending it, an extension being represented by a state (S, i, t) composed of a set of customers already visited (S), the last customer visited (i), and the time at which the visit of this last customer ends (t). These states are explored by increasing the cardinality of S while keeping only the Pareto-optimal states in terms of total traveling cost and current time. Several state elimination techniques were proposed to decrease the size of the state space, for instance by discarding states that are incompatible with mandatory precedence constraints or with the latest visit times of the customers not visited yet. Later on, Mingozzi et al. [14] proposed a bidirectional DP algorithm applicable to TSPTW-TC with additional precedence constraints. This algorithm exploits stronger state elimination techniques inspired by the relaxations defined by Christofides et al. [7]. Recently, Baldacci et al. [15] proposed even stronger state elimination methods by using mathematical programming and column generation to compute bounds on the cost required to visit the customers not visited yet.

Last, Constraint Programming (CP) was applied to TSPTW. The seminal contribution was performed for TSPTW-TT by Pesant et al. [16], who proposed a specific constraint propagation process based on the computation of minimum spanning trees. To better deal with the optimization part, Focacci et al. [17] then enhanced the CP approach proposed by Pesant et al. with the help of *cost-based domain filtering*.

TSPTW: incomplete methods. Numerous incomplete methods were studied for TSPTW. Many approaches employ a two-phase process where first insertion heuristics produce an initial solution, and then local search improves the quality of this solution, typically by using *or-opt-k* neighborhoods that reinsert a bloc of k successive customers in the sequence of visits, or *k-opt* neighborhoods that change k edges in the chain representing this sequence. Initially, Savelsbergh [1] proposed such a two-phase process for TSPTW-TT and TSPTW-M, based on or-opt-3, or-opt-2, or-opt-1, and 2-opt for the local search phase. Gendreau et al. [18] then explained how to efficiently check the feasibility of the sequences of visits at each local optimization step. After that, a two-phase algorithm using 3-opt for the local search phase was introduced by Calvo [19], and Ascheuer et al. [9] studied several heuristic techniques.

In parallel, several metaheuristics were tested, such as the tabu search algorithm defined by Carlton and Barnes [20] to minimize the makespan and then travel times, the variant of simulated annealing called compressed annealing introduced by Ohlmann and Thomas [21] for TSPTW-TT, the Ant-Colony Optimization (ACO) algorithm proposed by Favaretto et al. [22] still for TSPTW-TT, or its extension to Beam-ACO by López-Ibáñez and Blum [23] to combine ACO and the manipulation of a beam of partial solutions. The Beam-ACO and compressed annealing methods were adapted to TSPTW-M by López-Ibáñez et al. [24], showing the advantages of Beam-ACO against compressed annealing in this case. Several authors also studied the so-called General Variable Neighborhood Search (GVNS) metaheuristic, where VNS is coupled with a Variable Neighborhood Descent (VND) that explores a sequence of neighborhoods in a predefined order. To give a bit more details, for TSPTW-TT, da Silva and Urrutia [25] proposed a GVNS algorithm where the VNS part aims at finding a feasible solution based on a increasing number of 1-shift moves (reinsertion of a single customer in the sequence of visits), and the VND part aims at minimizing the transition times based on two successive neighborhoods (1-shift and 2-opt). Mladenovic et al. [26] extended this GVNS approach by using six successive neighborhoods for the VND part. Later on, Amghar et al. [27] adapted these ideas for a makespan minimization objective. Their experimental results showed that GVNS is robust and at least as good as the state-of-the-art incomplete methods for TSPTW-M, including Beam-ACO [24].

A last class of incomplete methods is *restricted dynamic programming.* The restricted DP procedure of Mingozzi et al. [14] that keeps k least-cost states at each state expansion layer belongs to this class. Another example is an algorithm defined by Balas and Simonetti [28], usable for TSPTW-TT and TSPTW-M. This algorithm starts from an initial tour and applies DP with constraints like "for every pair of customers placed at positions i and j in the current sequence of visits and such that $j \geq i + k(i)$, the visit of customer i must precede the visit of customer j", where $k(i)$ is a bound depending on i. When all $k(i)$ terms are bounded by a constant K, DP can find an optimal solution in time and space linear in the number of customers. Repeated usages of DP within an LNS procedure then progressively improve the incumbent solution. One drawback however is a so-called locality issue since each customer can only be moved around its current position due

to the $k(i)$ constants. To overcome this difficulty, Balas and Simonetti proposed to gather some contiguous customers before applying DP or to use DP in conjunction with a global interchange heuristic like k-opt.

TDTSPTW. We finish this literature review by listing techniques available for Time-Dependent TSPTWs. Globally, TDTSPTW received much less attention than TSPTW. Initially, Malandraki and Daskin [29] introduced time-dependent routing problems where travel times are modeled as step functions, together with an MILP model able to take time windows into account. Afterwards, Albiach et al. [30] tackled problems involving time-dependent travel times and costs, together with waiting costs when the vehicle arrives too early at a given customer. To minimize the total cost, they defined an exact approach that uses a problem transformation leading to an asymmetric TSP.

Next, Aguiar-Melgarejo et al. [6] applied Constraint Programming to TDTSPTWs representing urban delivery problems. They defined a new *time-dependent no-overlap constraint* and its constraint propagation rules to deal with problems involving time-dependent transition times and activities that must not overlap.

Still for exact methods, mathematical programming was considered. A first reference is the branch-and-bound algorithm of Arigliano et al. [5], usable for TDTSPTW with a makespan minimization objective and where travel times are described by a piecewise linear function. For the bounding part, the main idea is to exploit a time-independent problem where a maximum travel speed is considered, as in previous works on TDTSP by Cordeau et al. [31]. Next, Montero et al. [32] reused an Integer Linear Programming model proposed by Sun et al. [33] and defined a branch-and-cut algorithm for TDTSPTW-M. They also introduced several constructive heuristics together with local moves on the best solution found (swap move, or-exchange, arc reversal...). Recently, the idea of Boland et al. [12] to handle integer programming models obtained from dynamic time-expanded networks was adapted to TDTSPTW-M by Vu et al. [34]. The approach is shown to outperform the previous exact techniques. Moreover, it can tackle TDTSPTW-D where the objective is to minimize the total duration spent by the vehicle outside a depot.

Last, Lera-Romero et al. [35] used dynamic programming for TDTSPTW-M and TDTSPTW-D. Their algorithm incorporates many features such as efficient preprocessing rules, state expansion conditions during DP where the number of ancestors and descendants of customers in a precedence graph are analyzed, dominance relations among state labels, or bidirectional search. It also uses several relaxations for evaluating states, including the *ngL-tour* relaxation of Baldacci et al. [15] where customers can be visited several times or the *ti-tour* relaxation that exploits a time-independent problem. Extensive experiments showed that the approach is very efficient when compared to the state-of-the-art. There also exist algorithms for TDVRPTW, where a fleet of vehicles visits the customers [36], but our work is focused on single vehicle routing problems.

Contributions. Compared to previous works, the ImaxLNS algorithm we propose has the following features.

- First, contrarily to the best state-of-the-art methods for TDTSPTW, the approach proposed is able to

handle non-linear time-dependent transition functions. This is important for the observation satellite application that motivates this work, where transition times are obtained from a black-box space toolkit. It can also be relevant to exploit transition functions defined from machine learning models.

- Second, as existing incomplete methods, ImaxLNS progressively improves an incumbent solution through local modifications, and similarly to GVNS for TSPTW, it exploits the idea of using an increasing number of random 1-shift moves to diversify search at some steps. But ImaxLNS uses a new neighborhood that is not or-opt-k, k-opt, or an arbitrary combination of these. Instead, the unique neighborhood considered consists in reinserting k customers in a partial solution through dynamic programming. Another originality is that the size of this large neighborhood (the value of k) is adapted at each LNS step depending on the structure of the precedence graph of the problem, and the position of tens of customers can be optimized in a single step on some instances involving tight time windows. Moreover, apart from the constructive heuristics and local moves proposed by Montero et al. [32], the work on TDTSPTW is mainly focused on complete search, hence ImaxLNS is an original contribution on this point (even if incomplete methods were defined for extensions of TDTSPTW [37]).

- Third, with regards to dynamic programming approaches that are coupled with state elimination and state-space relaxation, ImaxLNS uses DP only as a local reoptimization procedure applied as many times as possible per second. As shown later, ImaxLNS guarantees that each local application of DP has a complexity that is linear in the number of customers, contrarily to the global DP procedure defined by Lera-Romero et al. [35] that can consume an exponential CPU time or memory size when no search limit is imposed. For state elimination, ImaxLNS does not solve any linear program, but as in the work of Lera-Romero et al., it exploits the counts of ancestors and descendants of each customer to reduce the set of states that need to be expanded. One difference however is that ImaxLNS also exploits these counts beforehand, to maximize the size of the neighborhood that DP can explore in linear time. Next, the ngL-tour state-space relaxation [15] mentioned before exploits a mandatory chain of precedences between customers. For ImaxLNS, the counterpart of this chain is a partial solution that helps reducing the complexity of DP.

- Fourth, as in the DP algorithm introduced by Balas and Simonetti [28] for TSPTW, ImaxLNS intensively exploits the precedence constraints of the problem together with side constraints to get a "local" DP procedure that is applicable in linear time. The main difference is that our side constraints correspond to a partial solution that is built step-by-step by a destroy phase, and not to maximum distances between pairs of customers. One strong advantage of this strategy is that ImaxLNS does not suffer from the locality issue mentioned by Balas and Simonetti and does not need to be combined with other global interchange neighborhoods. Last, ImaxLNS is defined for both TSPTW and TDTSPTW.

3. Problem definition

This section formally defines the problem tackled, namely a TDTSPTW possibly involving additional precedence constraints. In this problem, we consider:

- a set of customers numbered from 1 to N, with for each customer $i \in [1..N]$ a time window $[Start(i), End(i)]$ during which the visit of i can start;
- two fictitious customers numbered 0 and $N+1$ that respectively represent the depot from which the sequence of visits must start and the depot at which it must end (not necessarily the same depots);
- a time window $[0, H]$ usable for performing the visits; by convention, times windows associated with customers 0 and $N+1$ are $[Start(0), End(0)] = [Start(N+1), End(N+1)] = [0, H]$;
- a set of precedence constraints $\mathcal{P} \subseteq [0..N] \times [1..N+1]$; each pair $(i, j) \in \mathcal{P}$ expresses that customer i must be visited before customer j; the precedence graph G induced by $\mathcal{P}$ contains one arc $i \to j$ per pair $(i, j) \in \mathcal{P}$; this graph must be acyclic, and we assume that customer 0 precedes every customer in $[1..N+1]$ and customer $N+1$ follows every customer in $[0..N]$, either directly or by transitivity;
- a transition time function tt such that for any pair of distinct customers $(i, j) \in [0..N] \times [1..N+1]$ and any time $\tau \in [Start(i), +\infty[$, quantity $tt(i, j, \tau)$ gives the transition time required between the start of the service of customer i and the start of the service of customer j, if the service of customer i starts at time τ and j is visited just after i; this transition time covers both the service time for i and the travel time from i to j; function tt usually satisfies the FIFO property expressing that the earlier a transition starts, the earlier it ends, that is $(\tau \leq \tau') \to (\tau + tt(i, j, \tau) \leq \tau' + tt(i, j, \tau'))$; the FIFO property provides some guarantees but is not mandatory to apply the algorithm proposed in this article.

Transition time function tt may also satisfy the triangular inequality, meaning that for three distinct customers i, j, k and for every time $\tau \geq Start(i)$, we have $tt(i, k, \tau) \leq tt(j, k, \max(\tau + tt(i, j, \tau), Start(j)))$. For the sake of some pruning rules detailed later, we introduce a function $\overrightarrow{tt}$, called the *forward path transition time* function, such that for any pair of distinct customers $(i, j) \in [0..N] \times [1..N+1]$ and any time $\tau \in [Start(i), +\infty[$, quantity $\overrightarrow{tt}(i, j, \tau)$ is assumed to give a lower bound on the transition time required between a visit of customer i at time τ and a visit of customer j, possibly with some other customers visited between i and j. When tt satisfies the triangular inequality, it suffices to take $\overrightarrow{tt} = tt$. Otherwise, $\overrightarrow{tt}(i, j, \tau)$ can be computed from tt by solving a time-dependent shortest-path problem between i and j. However, as such lower bounds need to be computed many times per second during ImaxLNS, we assume that the modeler can define a lower bound function $\overrightarrow{tt}$ that is easier to compute. For instance, for TDTSPTW instances involving a vehicle subject to time-dependent speeds, we can derive $\overrightarrow{tt}$ from the maximum (constant) vehicle speed. Assuming that $\overrightarrow{tt}$ returns an actual lower bound on path transition times brings some guarantees

on the consistency of pruning rules used thereafter, but is not mandatory to apply the algorithm proposed. Formally, we only require that $\overrightarrow{tt} \leq tt$ holds, and when no lower bound function is specified, we use $\overrightarrow{tt} = tt$ even if tt might not satisfy the triangular inequality. Note that checking the satisfaction of the triangular equality is not possible for a black-box transition function tt defined over a continuous time domain.

Solutions and partial solutions. A *solution* is a sequence of visits that starts from the initial depot, visits all customers in $[1..N]$ while respecting the precedence constraints, and returns to the final depot. More formally, a solution is a sequence $\sigma = [\sigma_0, \sigma_1, \ldots, \sigma_N, \sigma_{N+1}]$ that corresponds to a topological order of the precedence graph, meaning that if there is a path from customer i to customer j in this graph then i must appear before j in σ. This implies that σ starts with $\sigma_0 = 0$ and ends with $\sigma_{N+1} = N + 1$, and that $[\sigma_1, \ldots, \sigma_N]$ is a permutation of $[1..N]$. Solutions are represented as doubly linked lists and for a given sequence of customer visits σ, we denote by $Prev(\sigma, i)$ and $Next(\sigma, i)$ the predecessor and successor of customer i in σ. We also denote by $pos(\sigma, i)$ the position of customer i in σ ($pos(\sigma, i) = k$ equivalent to $\sigma_k = i$).

Last, a *partial solution* is a sequence $\beta = [\beta_0, \ldots, \beta_{m+1}]$ such that $\beta_0 = 0$, $\beta_{m+1} = N+1$, and $[\beta_1, \ldots, \beta_m]$ is a permutation of a subset of $[1..N]$ that satisfies the precedence constraints, meaning that if two customers i and j belong to β and there is a path from customer i to customer j in the precedence graph, then i must appear before j in β. A *completion* of β is a solution σ such that β is a subsequence of σ.

Visit times and makespan. For every solution σ, the visit time $\tau(\sigma, p)$ associated with the pth customer in σ corresponds to $\tau(\sigma, 0) = 0$ at position 0 and is recursively defined by $\tau(\sigma, p) = \max(Start(\sigma_p), \tau(\sigma, p-1) + tt(\sigma_{p-1}, \sigma_p, \tau(\sigma, p-1)))$ for every $p \in [1..N+1]$. We assume here that the vehicle can arrive earlier at a customer location and wait for the window start time. The *makespan* of solution σ, denoted by $\tau(\sigma)$, is the arrival time at the final depot, *i.e.* $\tau(\sigma) = \tau(\sigma, N+1)$.

Feasibility and tardiness. A solution σ is *feasible* if and only if it visits every customer during its time window, *i.e.* $\forall p \in [0..N+1]$, $\tau(\sigma, p) \leq End(\sigma_p)$. The degree of infeasibility of σ is measured by its *cumulated tardiness*, which evaluates by how much the window end times are exceeded. The cumulated tardiness $\delta(\sigma, p)$ for the pth customer in σ corresponds to $\delta(\sigma, 0) = 0$ at position 0 and is recursively defined by $\delta(\sigma, p) = \delta(\sigma, p-1) + \max(0, \tau(\sigma, p) - End(\sigma_p))$ for every $p \in [1..N+1]$. The cumulated tardiness of σ, referred to as $\delta(\sigma)$, corresponds to the tardiness at the last step, *i.e.* $\delta(\sigma) = \delta(\sigma, N+1)$. A solution is feasible if and only if its cumulated tardiness $\delta(\sigma)$ is null.

Total transition cost. The total transition cost $\rho(\sigma)$ of a solution σ corresponds to the sum of the transition times for the vehicle. More precisely, the total transition cost for customer σ_p, referred to as $\rho(\sigma, p)$, corresponds to $\rho(\sigma, 0) = 0$ at position 0 and is recursively defined by $\rho(\sigma, p) = \rho(\sigma, p-1) + tt(\sigma_{p-1}, \sigma_p, \tau(\sigma, p-1))$ for every $p \in [1..N+1]$. The total transition cost $\rho(\sigma)$ of σ is the final cost, *i.e.* $\rho(\sigma) = \rho(\sigma, N+1)$.

Global objective. Our first objective is to find a feasible solution and our second objective is to minimize the makespan. When searching for a makespan-optimal solution, it can be useful to temporarily work on solutions having a positive tardiness. It is also useful to minimize the total transition time, the rationale being that among two solutions having the same makespan, the one having the lowest total transition time is preferred since it is more likely to allow some visits to be moved forward (globally more idle periods between the visits). Therefore, our objective during search is to compute a solution σ such that the triple

$$Eval(\sigma) = (\delta(\sigma), \tau(\sigma), \rho(\sigma)) \tag{1}$$

is lexicographically minimal. This triple is called the *tmc-evaluation* ("tardiness-makespan-cost" evaluation) of σ. For convenience, we introduce a step evaluation function that computes the tmc-evaluation at step p from the tmc-evaluation at step $p-1$. More precisely, by denoting as $Eval(\sigma, p) = (\delta(\sigma, p), \tau(\sigma, p), \rho(\sigma, p))$ the triple formed by the tardiness, makespan, and transition cost values at position p, we have:

$$Eval(\sigma, p) = StepEval(Eval(\sigma, p-1), \sigma_{p-1}, \sigma_p) \tag{2}$$

where $StepEval((d, t, r), i, j)$ corresponds to the triple (d', t', r') defined by $t' = \max(Start(j), t + tt(i, j, t))$, $d' = d + \max(0, t' - End(j))$, and $r' = r + tt(i, j, t)$.

4. Preprocessing

As in existing works on (TD)TSPTW, we introduce several preprocessing techniques. The latter lead to an equivalent problem if transition function tt satisfies the FIFO property and if $\overrightarrow{tt}$ provides actual lower bounds on path transition times. Otherwise, the problem obtained after the preprocessing phase might be more constrained than the initial one.

Precedence constraints. To avoid considering infeasible solutions, the algorithm first infers precedence constraints from the time windows available. On this point, given two distinct customers $i, j \in [1..N]$ such that $Start(i) + \overrightarrow{tt}(i, j, Start(i)) > End(j)$, precedence (j, i) can be added to the set of precedence constraints $\mathcal{P}$ of the problem. Indeed, in this case, even when starting a transition from i to j at the earliest possible time, the vehicle cannot arrive on time to visit customer j according to $\overrightarrow{tt}$. Such a preprocessing is not new [32], however we proceed as follows to avoid testing the previous constraints for all pairs of customers.

- First, we derive a minimal set of precedence constraints that are valid whatever the content of the transition function. To do this, customers in $[1..N]$ are ordered by increasing window start times, which gives an ordered sequence $[i_1, \ldots, i_N]$. Customers are then considered from i_1 to i_N. For each customer i_u, we compute the smallest index $v > u$ such that $Start(i_v) > End(i_u)$ (i_v necessarily visited after i_u) and the largest index $w \geq v$ such that $Start(i_w) \leq End(i_v)$ (i_w not necessarily visited after

i_v). We then add precedence constraint $(i_u, i_{v'})$ to $\mathcal{P}$ for every index $v' \in [v, w]$. For all indices $w' > w$, customer $i_{w'}$ must also necessarily follow i_u, but there is no need to add precedence constraint $(i_u, i_{w'})$ to $\mathcal{P}$ since this constraint will be implied by transitivity at the end of the process.

- Second, we check condition $Start(i) + \overrightarrow{tt}(i, j, Start(i)) > End(j)$ only for the pairs of distinct customers $(i, j) \in [1..N] \times [1..N]$ such that i is neither an ancestor nor a descendant of j in the current precedence graph. Precedence (j, i) is added to $\mathcal{P}$ if the previous inequality holds.

- Third, for all customers i that have no ancestor (resp. no descendant) in the current precedence graph, we add precedence constraint $(0, i)$ (resp. $(i, N + 1)$) to $\mathcal{P}$.

To boost the algorithm, we also compute the *transitive reduction* of the precedence graph, that is we remove all precedence constraints entailed by transitivity to get the sets of immediate predecessors and successors of each customer i, denoted by $Pred_0(i)$ and $Succ_0(i)$ respectively. From this, by traversing the precedence graph in a topological order (resp. reverse topological order), the sets of mandatory ancestors and descendants of every customer i, denoted by $Anc_0(i)$ and $Desc_0(i)$, are recursively obtained by:

$$Anc_0(i) \leftarrow Pred_0(i) \cup \left(\cup_{j \in Pred_0(i)} Anc_0(j)\right) \tag{3}$$

$$Desc_0(i) \leftarrow Succ_0(i) \cup \left(\cup_{j \in Succ_0(i)} Desc_0(j)\right) \tag{4}$$

Makespan lower bound. The algorithm initially computes a lower bound τ^{LB} on the optimal makespan value. To do this, it considers the customers one by one in a topological order of the precedence graph. When considering customer $i \in [0..N + 1]$, the earliest time $\tau^{LB}(i)$ at which i can be visited is computed as:

$$\tau^{LB}(i) = \max(Start(i), \max_{j \in Pred_0(i)} (\tau^{LB}(j) + \overrightarrow{tt}(j, i, \tau^{LB}(j)))) \tag{5}$$

and the makespan lower bound is then $\tau^{LB} = \tau^{LB}(N + 1)$. If this lower bound is strictly greater than the horizon end ($\tau^{LB} > H$), then the problem has no solution according to $\overrightarrow{tt}$.

Initial greedy search. During the preprocessing phase, the algorithm also computes an initial sequence of visits σ by inserting customers one by one based on what we call the *tardiness-makespan-cost heuristic*. This heuristic starts from an empty sequence and adds at each step a non-visited customer whose mandatory predecessors are already visited. Among the candidate customers, a preference is given first to the customers whose visit leads to the highest cumulated tardiness (to limit this tardiness as much as possible), then to the customers that can be visited at the earliest possible time (earliest visit first heuristic), and then to the customers whose visit leads to the lowest cumulated transition cost (nearest visit heuristic). If the solution σ obtained satisfies $\delta(\sigma) = 0$ and $\tau(\sigma) = \tau^{LB}$, the algorithm directly returns that σ is makespan-optimal.

Finally, the preprocessing phase detects whether some customers can be harmlessly visited before the others. More precisely, given the solution $\sigma = [\sigma_0, \ldots, \sigma_{N+1}]$ produced by the tardiness-makespan-cost

heuristic, if an index $p \in [1..N]$ is such that σ is feasible until position p (i.e., $\tau(\sigma, q) \leq End(\sigma_q)$ for every $q \in [1..p]$) and all customers placed at a position $q > p$ can be visited at their earliest possible time (i.e., $\tau(\sigma, p) + tt(\sigma_p, \sigma_q, \tau(\sigma, p)) \leq Start(\sigma_q)$), then all customers in $[\sigma_1, \ldots, \sigma_p]$ can be removed from the makespan minimization problem since there exists a way to visit them without any impact on the other customers.

5. Iterated Maximum Large Neighborhood Search: general description

We now describe the algorithm proposed to search for a feasible solution minimizing the makespan (and the transition times as a side effect). As mentioned before, this algorithm is called ImaxLNS for *Iterated Maximum Large Neighborhood Search*. It exploits destroy and repair operations to locally optimize the current solution, together with perturbations and restarts to diversify search.

5.1. Preliminary: insertion-width of a partial solution

In ImaxLNS, each destroy operation removes a set of customers S from the current solution σ. Such a removal leads to a partial solution β. The repair phase is responsible for exploring all possible reinsertions of the customers in S and for returning a completion of β that is evaluated as the best one. To set the degree of destruction employed to get β, ImaxLNS exploits a parameter called the *insertion-width*, the rationale being that as shown later, the complexity of the repair phase is only linear in the number of customers when the insertion-width of β is bounded.

Basically, the insertion-width of β measures the maximum number of removed customers that can occupy each position in the sequence of visits, given the precedence graph of the problem. More formally, let us denote by $\mathcal{R}(\beta)$ the set of customers removed to get a partial solution β, that is $\mathcal{R}(\beta) = \{i \in [1..N] \,|\, i \notin \beta\}$. The set of mandatory precedences $\mathcal{P}$ of the problem, including those added at the preprocessing step, defines a precedence graph $G = ([0..N+1], \mathcal{P})$ over the set of customers. For a partial solution $\beta = [\beta_0, \ldots, \beta_{m+1}]$, graph G can be extended to get a new graph $G(\beta) = ([0..N+1], \mathcal{P} \cup \{(\beta_k, \beta_{k+1}) \,|\, k \in [1..m-1]\}$ where a precedence arc $\beta_k \rightarrow \beta_{k+1}$ is added for every pair of successive customers in β. Adding precedence arcs $\beta_0 \rightarrow \beta_1$ and $\beta_m \rightarrow \beta_{m+1}$ is useless since they are already covered by $\mathcal{P}$.

In graph $G(\beta)$, let us denote by $Anc(\beta, i)$ and $Desc(\beta, i)$ the sets of ancestors and descendants of customer i. The set of positions that can be occupied by i over all possible completions of β is then $[P_{min}(\beta, i)..P_{max}(\beta, i)]$ where $P_{min}(\beta, i) = |Anc(\beta, i)|$ and $P_{max}(\beta, i) = N + 1 - |Desc(\beta, i)|$, the rationale being that i is necessarily visited after all its ancestors and before all its descendants. From this, the set of *removed customers* that can be visited at position $p \in [0..N+1]$ is:

$$\mathcal{R}(\beta, p) = \{i \in \mathcal{R}(\beta) \,|\, p \in [P_{min}(\beta, i)..P_{max}(\beta, i)]\} \tag{6}$$

Then, the insertion-width of β at position p, referred to as $W(\beta, p)$, is defined by:

$$W(\beta, p) = |\mathcal{R}(\beta, p)| \tag{7}$$

that is it corresponds to the number of removed customers that can be visited at position p. Finally, the insertion-width $W(\beta)$ of β is the maximum insertion-width obtained over all positions:

$$W(\beta) = \max_{p \in [0..N+1]} W(\beta, p) \tag{8}$$

When $\beta = [0, N+1]$ (empty partial solution), we speak of the *insertion-width of the problem*. To illustrate these definitions, let us consider a problem involving $N = 8$ customers and a set of precedence constraints $\mathcal{P}$ given in Figure 1a. The extended graph $G(\beta)$ associated with partial solution $\beta = [0, 2, 4, 5, 9]$ is provided in Figure 1b, where two precedence constraints not already implied by transitivity are added (dashed arrows). For this example, Figure 2 gives the minimum and maximum positions of each customer and the insertion-width of each position. In this case, the insertion-width of β is $W(\beta) = 4$, meaning that even if 5 customers do not belong to β, at most 4 removed customers can occupy a given position.

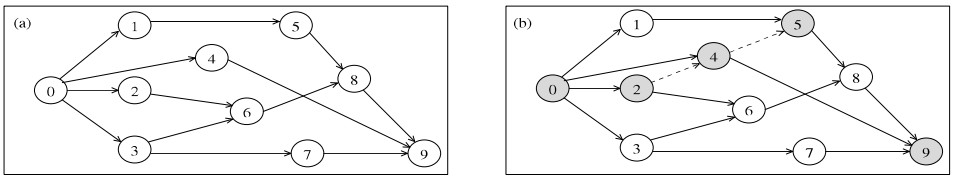

Figure 1: Initial precedence graph G (a) and precedence graph $G(\beta)$ obtained for partial solution $\beta = [0, 2, 4, 5, 9]$ (b)

i (customer)	$\lvert Anc(\beta, i)\rvert$	$\lvert Desc(\beta, i)\rvert$	$[P_{min}(\beta, i)..P_{max}(\beta, i)]$
0	0	9	[0..0]
1	1	3	[1..6]
2	1	5	[1..4]
3	1	4	[1..5]
4	2	3	[2..6]
5	4	2	[4..7]
6	3	2	[3..7]
7	2	1	[2..8]
8	7	1	[7..8]
9	9	0	[9..9]

p (position)	$\mathcal{R}(\beta, p)$	$W(\beta, p)$
0	$\emptyset$	0
1	$\{1, 3\}$	2
2	$\{1, 3, 7\}$	3
3	$\{1, 3, 6, 7\}$	4
4	$\{1, 3, 6, 7\}$	4
5	$\{1, 3, 6, 7\}$	4
6	$\{1, 6, 7\}$	3
7	$\{6, 7, 8\}$	3
8	$\{7, 8\}$	2
9	$\emptyset$	0

Figure 2: Possible positions for the customers involved in the example of Figure 1b, and insertion-width of each position

5.2. ImaxLNS: an example

Table 1 gives a possible trace of ImaxLNS on the problem seen before. Initially, at step 1, a solution is built from the tardiness-makespan-cost heuristic defined in Section 4. This solution leads to tmc-evaluation $(12, 64, 51)$, meaning that it is not feasible (tardiness equal to 12), its makespan equals 64, and the total transition time is 51. Then, ImaxLNS performs successive destroy and repair operations (steps 2-5). Each

destroy and repair step consists in removing a subset of customers S from the current solution and reinserting these customers to try and minimize the tmc-evaluation of the resulting solution. For instance, at step 2, the destroy phase removes customers in $S = \{1, 3, 6, 7, 8\}$, which leads to a partial solution $\beta = [0, 2, 4, 5, 9]$, and the repair phase reinserts these customers into β to produce solution $[0, \mathbf{3}, 2, \mathbf{6}, \mathbf{1}, \mathbf{7}, 4, 5, \mathbf{8}, 9]$.

To try and detect local optima, ImaxLNS maintains, for each customer i, the number of times $rm[i]$ this customer has been reinserted without improving the tmc-evaluation (column nRemovals). For instance, after step 5, we have $rm[1] = 2$, $rm[2] = 1$, etc. If each counter $rm[i]$ has a value greater than or equal to a parameter of ImaxLNS referred to as *Rmin*, then ImaxLNS considers that a local optimum has been reached. In this case, it applies a perturbation to the current solution. On the example, we use $Rmin = 1$ and at step 6, the perturbation applied is a 1-shift move that changes the place of one customer in the sequence of visits (customer 7). After that, destroy and repair operations are performed again to optimize the current solution (steps 7-9). On the example, ImaxLNS is attracted by the same local optimum as before. At step 10, another perturbation is performed, but this time using a larger magnitude (1-shift moves for two customers). At steps 11 to 13, the destroy and repair operations are attracted by another local optimum whose tmc-evaluation is $(0, 61, 58)$. From this, if the number of perturbations applied at the last perturbation step is equal to a parameter of ImaxLNS referred to as *Kmax*, a restart from a new initial solution is performed. On the example, we use $Kmax = 2$, hence a restart occurs at step 14. The search process goes on until there is no CPU time left or a feasible solution whose completion time equals the makespan lower bound is found.

step	update	current solution (σ)	$Eval(\sigma)$	nRemovals (rm)	nPerturbs (k)
1	initialization	$[0, 2, 3, 1, 6, 7, 4, 5, 8, 9]$	(12,64,51)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 0$
2	destroy & repair $\{1, 3, 6, 7, 8\}$	$[0, \mathbf{3}, 2, \mathbf{6}, \mathbf{1}, \mathbf{7}, 4, 5, \mathbf{8}, 9]$	(5,57,50)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 0$
3	destroy & repair $\{2, 4, 5, 6\}$	$[0, 3, \mathbf{4}, \mathbf{2}, 1, \mathbf{5}, 7, \mathbf{6}, 8, 9]$	(0,59,55)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 0$
4	destroy & repair $\{1, 5, 6, 7\}$	$[0, 3, 4, 2, \mathbf{1}, \mathbf{5}, \mathbf{7}, \mathbf{6}, 8, 9]$	(0,59,55)	$[1, 0, 0, 0, 1, 1, 1, 0]$	$k = 0$
5	destroy & repair $\{1, 2, 3, 4, 8\}$	$[0, \mathbf{1}, 5, \mathbf{4}, \mathbf{2}, \mathbf{3}, 7, 6, \mathbf{8}, 9]$	(0,59,55)	$[2, 1, 1, 1, 1, 1, 1, 1]$	$k = 0$
6	perturbation (1-shift: $\{7\}$)	$[0, 1, 5, 4, 2, 3, 6, 8, \mathbf{7}, 9]$	(0,65,65)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 1$
7	destroy & repair $\{2, 5, 6, 7\}$	$[0, 1, \mathbf{5}, 4, \mathbf{2}, 3, \mathbf{7}, \mathbf{6}, 8, 9]$	(0,59,55)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 1$
8	destroy & repair $\{1, 3, 4, 8\}$	$[0, \mathbf{1}, \mathbf{4}, 5, 2, \mathbf{3}, 7, 6, \mathbf{8}, 9]$	(0,59,55)	$[1, 0, 1, 1, 0, 0, 0, 1]$	$k = 1$
9	destroy & repair $\{2, 5, 6, 7\}$	$[0, 1, \mathbf{5}, 4, \mathbf{2}, 3, \mathbf{7}, \mathbf{6}, 8, 9]$	(0,59,55)	$[1, 1, 1, 1, 1, 1, 1, 1]$	$k = 1$
10	perturbation (1-shift: $\{1, 5\}$)	$[0, 4, 2, 3, 7, 6, \mathbf{1}, \mathbf{5}, 8, 9]$	(0,62,60)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 2$
11	destroy & repair $\{2, 3, 4, 6\}$	$[0, \mathbf{2}, \mathbf{4}, \mathbf{3}, 7, 1, \mathbf{6}, 5, 8, 9]$	(0,61,58)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 2$
12	destroy & repair $\{2, 3, 4, 7\}$	$[0, \mathbf{2}, \mathbf{4}, \mathbf{3}, \mathbf{7}, 1, 6, 5, 8, 9]$	(0,61,58)	$[0, 1, 1, 1, 0, 0, 1, 0]$	$k = 2$
13	destroy & repair $\{1, 5, 6, 7, 8\}$	$[0, 2, 4, 3, \mathbf{7}, \mathbf{1}, \mathbf{6}, \mathbf{5}, \mathbf{8}, 9]$	(0,61,58)	$[1, 1, 1, 1, 1, 1, 2, 1]$	$k = 2$
14	restart	$[\mathbf{0}, \mathbf{3}, \mathbf{7}, \mathbf{1}, \mathbf{5}, \mathbf{4}, \mathbf{2}, \mathbf{6}, \mathbf{8}, \mathbf{9}]$	(18,75,75)	$[0, 0, 0, 0, 0, 0, 0, 0]$	$k = 0$

Table 1: A possible trace of ImaxLNS for the example of Figure 1a

5.3. *ImaxLNS: main function*

The main function of ImaxLNS is given in Algorithm 1. During search, the algorithm maintains the best solution found so far (σ^*) and the best solution found since the last perturbation or restart (σ_r^*). As in the example, ImaxLNS uses the following steps.

- At Line 1, an initial solution σ is obtained from the tardiness-makespan-cost heuristic defined in Section 4. Then, the algorithm searches for solutions while there is some CPU time left and while the best solution found is not feasible or not as good as the makespan lower bound τ^{LB} (Line 4). If the goal is just to find a feasible solution, the condition on τ^{LB} can be skipped.

- During the while loop, the algorithm records the number of times each customer i has been reinserted in the sequence of visits without any improvement (counters $rm[i]$). These counters are set to 0 initially (Line 2). They are incremented after each destroy step for all customers that are not involved in partial solution β (Line 9), and reset each time an improvement is obtained with regards to σ_r^* (Line 11). They are also reset after each perturbation (Line 15) and after each restart (Line 19).

- At each destroy phase (Lines 6-7), the algorithm selects a *pivot customer* i such that counter $rm[i]$ is lower than parameter *Rmin* given in the input. In the destroy function that produces a partial solution β, the algorithm will remove customer i from σ, as well as other customers as long as the insertion-width of β is not greater than the *maximum insertion-width* parameter *Wmax*.

- During each repair phase (Line 8), the algorithm explores all possible ways to reinsert the removed customers into partial solution β and chooses the best one according to the tardiness-makespan-cost evaluations. The exploration of the set of reinsertion alternatives is based on dynamic programming.

- When a local optimum is reached, if the current solution σ is feasible and the magnitude k of the last perturbation phase is less than parameter *Kmax* given in the input, then magnitude k is incremented and k random 1-shift moves are performed by calling function *perturbOneShift* (Lines 13-14), to try and escape from the current local optimum. Each call to *perturbOneShift* randomly selects a customer σ_j in σ and a 1-shift direction (forward or backward). If a forward direction is chosen, *perturbOneShift* considers all positions for σ_j such that σ_j is placed before all its mandatory descendants in precedence graph G. Using a uniform probability distribution, it randomly chooses one of the positions that preserves the feasibility of the current solution. Backward 1-shift moves are handled in a similar way.

- Otherwise, if a local optimum is reached and either the current solution is not feasible or the magnitude of the last perturbation equals *Kmax*, then ImaxLNS restarts from a random solution and resets the magnitude of the last perturbation (Lines 17-18). To get a random solution, the algorithm starts from an empty sequence σ and adds at each step, at the end of σ, an unvisited customer whose mandatory

predecessors are already visited. This unvisited customer is chosen using a uniform distribution, independently of the time windows. Said differently, the restart phase randomly selects a topological order of precedence graph G. The new solution obtained (Line 18) is not necessarily feasible.

- At any step, if a new best solution σ is found, it is recorded and the magnitude of the last perturbation is reset to intensify search around σ (Line 20). Finally, the best solution found σ^* is returned (Line 21).

Algorithm 1 defines the global search scheme used by ImaxLNS. To get an efficient approach, one key point is the way the destroy and repair functions are implemented, which is detailed in the two next sections.

Algorithm 1: ImaxLNS($Tmax, \tau^{LB}, Rmin, Wmax, Kmax$)

Input: $Tmax$: max CPU time; τ^{LB}: makespan lower bound; $Rmin$: minimum number of removals for each customer; $Wmax$: maximum insertion-width; $Kmax$: maximum number of 1-shifts

Output: The solution minimizing the tardiness-makespan-cost vector among all solutions explored

1 $\sigma \leftarrow InitSolution()$
2 $\sigma_r^* \leftarrow \sigma$; **foreach** $i \in [1..N]$ **do** $rm[i] \leftarrow 0$
3 $\sigma^* \leftarrow \sigma$; $k \leftarrow 0$
4 **while** $(time() < Tmax) \wedge (\delta(\sigma^*) > 0 \vee \tau(\sigma^*) > \tau^{LB})$ **do**
5 **if** $\exists i \in [1..N] \,|\, rm[i] < Rmin$ **then** `// destroy and repair`
6 select $i \in [1..N]$ such that $rm[i] < Rmin$
7 $\beta \leftarrow destroy(\sigma, i, Wmax)$
8 $\sigma \leftarrow repair(\beta)$
9 **foreach** $i \in [1..N] \,|\, i \notin \beta$ **do** $rm[i] \leftarrow rm[i] + 1$
10 **if** $Eval(\sigma) < Eval(\sigma_r^*)$ **then**
11 $\sigma_r^* \leftarrow \sigma$; **foreach** $i \in [1..N]$ **do** $rm[i] \leftarrow 0$
12 **else if** $(\delta(\sigma) = 0) \wedge (k < Kmax)$ **then** `// perturbation`
13 $k \leftarrow k + 1$
14 **foreach** $l \in [1..k]$ **do** $\sigma \leftarrow perturbOneShift(\sigma)$
15 $\sigma_r^* \leftarrow \sigma$; **foreach** $i \in [1..N]$ **do** $rm[i] \leftarrow 0$
16 **else** `// restart`
17 $k \leftarrow 0$
18 $\sigma \leftarrow randomSolution()$
19 $\sigma_r^* \leftarrow \sigma$; **foreach** $i \in [1..N]$ **do** $rm[i] \leftarrow 0$
20 **if** $Eval(\sigma) < Eval(\sigma^*)$ **then** $\sigma^* \leftarrow \sigma$; $k \leftarrow 0$
21 **return** σ^*

6. Destroy procedure

This section details the destroy procedure of ImaxLNS. As mentioned before, this procedure starts from a solution σ visiting all customers and removes a maximum set of customers from σ under the constraint that the insertion-width of the partial solution obtained must not exceed a maximum value $Wmax$.

6.1. Main function: iterative customer removal attempts

Algorithm 2 details the main function of the destroy procedure. The latter manipulates a current partial solution β resulting from the customer removals made so far. For each position $p \in [1..N]$, it maintains quantity $W(\beta, p)$. For each customer i, it maintains quantities $Anc(\beta, i)$, $Desc(\beta, i)$, $P_{min}(\beta, i)$, $P_{max}(\beta, i)$, and two other quantities referred to as $lastAnc(\beta, i)$ and $firstDesc(\beta, i)$ that will be detailed in Section 6.3. As there is a unique partial solution β manipulated at each step, the algorithm actually maintains $W(p)$, $Anc(i)$, $Desc(i)$, $P_{min}(i)$, $P_{max}(i)$, $lastAnc(i)$, and $firstDesc(i)$, without any β parameter. These quantities are initialized at Lines 1-4: at the beginning, the insertion-width of every position is null, the set of ancestors (resp. descendants) of each customer is made of all its preceding (resp. following) customers in σ, and the only possible position for customer σ_k is position k. Next, the destroy procedure removes the *pivot customer* given in the input by calling function *remove* at Line 5. The latter updates data structures $W(.)$, $Anc(.)$, $Desc(.)$, $P_{min}(.)$, $P_{max}(.)$, $lastAnc(.)$, and $firstDesc(.)$ to take into account the removal of i, and returns value *true* if and only if i can be removed without exceeding the maximum insertion-width $Wmax$. At Line 5, as $Wmax \geq 1$, the removal of a single customer is necessarily accepted and the initial partial solution to consider is $\beta = \sigma \setminus \{i\}$ (Line 6).

After that, the algorithm selects at each step a random customer r still not considered yet and tries to remove r from β by calling function *remove* again (Lines 10-11). If this function returns *true*, then customer r can be safely removed from β with the guarantee that the insertion-width of the resulting partial solution does not exceed $Wmax$, and all changes made by function *remove* on the data structures are committed (Line 13). Otherwise, customer r is kept in the current partial solution and all changes made on the data structures manipulated are undone by calling the *rollback* function that restores the state obtained after the last call to *commit* (Line 15). In the while loop, the iterative removal process is applied until all customer removals have been tried. Finally, the partial solution obtained is returned (Line 16).

6.2. Remove *function: attempt to remove a single customer*

We now detail the *remove* function that is responsible for updating the insertion-width of each position following the removal of a customer r, and for detecting insertion-width overloads with regards to parameter $Wmax$. This function corresponds to Algorithm 3. Initially, the insertion-width of every position p in interval $[P_{min}(r)..P_{max}(r)]$ is incremented since a possible position for r before the removal of r from β is still a possible position for r after this removal. Doing so, we simply take into account the change of status of

Algorithm 2: DESTROY($\sigma, i, Wmax$)

Input: σ: initial solution; i: a pivot customer; $Wmax \geq 1$: maximum insertion-width

Output: A partial solution β such that $W(\beta) \leq Wmax$

1 **foreach** $p \in [1..N]$ **do** $W(p) \leftarrow 0$
2 **foreach** $k = 0$ *to* $N+1$ **do**
3 $\quad Anc(\sigma_k) \leftarrow \{\sigma_0, \ldots, \sigma_{k-1}\}$; $P_{min}(\sigma_k) \leftarrow k$; $lastAnc(\sigma_k) \leftarrow nil$
4 $\quad Desc(\sigma_k) \leftarrow \{\sigma_{k+1}, \ldots, \sigma_{N+1}\}$; $P_{max}(\sigma_k) \leftarrow k$; $firstDesc(\sigma_k) \leftarrow nil$
5 $remove(\sigma, \beta, i, Wmax)$
6 $\beta \leftarrow \sigma \setminus \{i\}$
7 $commit(W(.), Anc(.), Desc(.), P_{min}(.), P_{max}(.), lastAnc(.), firstDesc(.))$
8 $NotRemoved \leftarrow [1..N] \setminus \{i\}$
9 **while** $NotRemoved \neq \emptyset$ **do**
10 $\quad$ pick r from $NotRemoved$; $NotRemoved \leftarrow NotRemoved \setminus \{r\}$
11 $\quad$ **if** $remove(\sigma, \beta, r, Wmax)$ **then**
12 $\quad\quad \beta \leftarrow \beta \setminus \{r\}$
13 $\quad\quad commit(W(.), Anc(.), Desc(.), P_{min}(.), P_{max}(.), lastAnc(.), firstDesc(.))$
14 $\quad$ **else**
15 $\quad\quad rollback(W(.), Anc(.), Desc(.), P_{min}(.), P_{max}(.), lastAnc(.), firstDesc(.))$
16 **return** β

r from "non-removed" to "removed". If one position $p \in [P_{min}(r)..P_{max}(r)]$ is already full ($W(p) = Wmax$), then the algorithm directly returns that removing r is forbidden (Line 2).

After this initial step, the algorithm updates the insertion-width of the different positions by taking into account that the removal of customer r from β can change the number of ancestors for customers following r in initial solution σ (Lines 3-11) and the number of descendants for customers preceding r in σ (Lines 12-20). For the first point, the customers placed after r in σ (r included) are considered one by one in the order given by σ, the underlying idea being that σ is necessarily a topological order of graph $G(\beta)$ since β is a subsequence of σ. During such a forward traversal, the set of ancestors of each customer i is updated by calling function *updateAncestors* detailed later in Section 6.3. This function simulates the impact of the removal of r on set $Anc(i)$. From this, the new minimum position of i is computed (Line 6). If it differs from the previous one, the insertion-width of every new possible position for i is incremented and insertion-width overloads are detected (Line 9). Moreover, for the sake of incremental computations in function *updateAncestors*, customer i is added to set *changedAnc* that records all customers whose set of ancestors has changed. The backward traversal of σ to update the descendants of the customers preceding r

in σ is similar. In this second case, a call to function *updateDescendants* detailed in Section 6.3 updates the descendants of the customer i given in the input, the insertion-width of every new possible position for i is incremented (Lines 17-18), and if needed i is added to set *changedDesc* that records all customers whose set of descendants has changed.

Algorithm 3: REMOVE($\sigma, \beta, r, Wmax$)

Input: σ: a solution; β: a partial solution obtained from σ; r: a customer to remove from β; $Wmax \geq 1$: maximum insertion-width

Output: A Boolean indicating whether customer r can be removed from β according to $Wmax$

1 **foreach** $p \in [P_{min}(r)..P_{max}(r)]$ **do**
2 | **if** $W(p) < Wmax$ **then** $W(p) \leftarrow W(p) + 1$ **else return** *false*
3 $changedAnc \leftarrow \emptyset$
4 **for** $i \leftarrow r$; $i \neq N + 1$; $i \leftarrow Next(\sigma, i)$ **do**
5 | $updateAncestors(i, \sigma, \beta, r, changedAnc)$
6 | $newP_{min} \leftarrow |Anc(i)|$
7 | **if** $newP_{min} \neq P_{min}(i)$ **then**
8 | | **foreach** $p \in [newP_{min}..P_{min}(i) - 1]$ **do**
9 | | | **if** $W(p) < Wmax$ **then** $W(p) \leftarrow W(p) + 1$ **else return** *false*
10 | | $P_{min}(i) \leftarrow newP_{min}$
11 | | $changedAnc \leftarrow changedAnc \cup \{i\}$
12 $changedDesc \leftarrow \emptyset$
13 **for** $i \leftarrow r$; $i \neq 0$; $i \leftarrow Prev(\sigma, i)$ **do**
14 | $updateDescendants(i, \sigma, \beta, r, changedDesc)$
15 | $newP_{max} \leftarrow N + 1 - |Desc(i)|$
16 | **if** $newP_{max} \neq P_{max}(i)$ **then**
17 | | **foreach** $p \in [P_{max}(i) + 1..newP_{max}]$ **do**
18 | | | **if** $W(p) < Wmax$ **then** $W(p) \leftarrow W(p) + 1$ **else return** *false*
19 | | $P_{max}(i) \leftarrow newP_{max}$
20 | | $changedDesc \leftarrow changedDesc \cup \{i\}$
21 **return** *true*

6.3. Incremental update of ancestors and descendants

The last components of the destroy function are the *updateAncestors* and *updateDescendants* procedures used in Algorithm 3. The two procedures being similar, we focus here on the *updateAncestors* function. The naive implementation would consist in recursively computing formulas $Anc(\beta, i) \leftarrow Pred(\beta, i) \cup$

$\left(\cup_{j \in Pred(\beta,i)} Anc(\beta, j)\right)$ where $Pred(\beta, i)$ denotes the set of predecessors of i in graph $G(\beta)$. Each formula of this form requires a number of union operations that is linear in the number of predecessors of i in $G(\beta)$. Instead, we propose an incremental method for computing $Anc(\beta, i)$ by using at most two union operations. Obviously, $Anc(\beta, i)$ contains $Anc_0(i)$, the set of mandatory ancestors of i computed at the preprocessing phase. Then, if i belongs to β, it can be shown that $Anc(\beta, i)$ simply contains the customers in $Anc_0(i)$, the predecessor of i in β, and all ancestors of this predecessor, that is we can use formula:

$$\text{if } i \in \beta, \text{ then } Anc(\beta, i) \leftarrow Anc_0(i) \cup \{Prev(\beta, i)\} \cup Anc(\beta, Prev(\beta, i)) \tag{9}$$

For instance, in Figure 1b, the predecessor of customer 5 in β is customer 4, and we can compute $Anc(\beta, 5)$ by $Anc(\beta, 5) \leftarrow Anc_0(5) \cup \{4\} \cup Anc(\beta, 4) = \{0, 1\} \cup \{4\} \cup \{0, 2\} = \{0, 1, 2, 4\}$. Otherwise, for a customer i that does not belong to β, it can be shown that it suffices to add to $Anc_0(i)$ the ancestors of the ancestor of i that appears at the rightmost position in β, referred to as $lastAnc(\beta, i)$. For instance, in Figure 1b again, the ancestor of customer 8 that belongs to β and that is placed at the rightmost position in β is customer $lastAnc(\beta, 8) = 5$, and we can compute $Anc(\beta, 8)$ by $Anc(\beta, 8) \leftarrow Anc_0(8) \cup Anc(\beta, 5) = \{0, 1, 2, 3, 5, 6\} \cup \{0, 1, 2, 4\} = \{0, 1, 2, 3, 4, 5, 6\}$. Formally, we can use formula:

$$\text{if } i \notin \beta, \text{ then } Anc(\beta, i) \leftarrow Anc_0(i) \cup Anc(\beta, lastAnc(\beta, i)) \tag{10}$$

where the *last ancestors* for the removed customers are recursively obtained by:

$$lastAnc(\beta, i) \quad \leftarrow \quad argmax_{k \in \{j \in Pred_0(i) \,|\, j \in \beta\} \cup \{lastAnc(\beta, j) \,|\, j \in Pred_0(i), j \notin \beta\}} \, pos(\sigma, k) \tag{11}$$

The previous equation means that to compute the last ancestor of i, we look on the one hand at all mandatory predecessors of i that still belong to β, and on the other hand at all last ancestors of the predecessors of i that do not belong to β anymore. To get the customer that has the rightmost position in β it suffices to keep the customer that has the rightmost position in σ, since β is a subsequence of σ.

Equations 9-11 are directly reused in function *updateAncestors* given in Algorithm 4. In this function, as before, parameter β is removed from quantities $Anc(\beta, i)$ and $lastAnc(\beta, i)$. As in Equations 9-10, two cases are distinguished to simulate the impact of the removal of customer r on the set of ancestors of customer i.

- If i belongs to the partial solution after the removal of r (test at Line 1), then following Equation 9, $Anc(i)$ is revised either if the predecessor of i in β changes after the removal of r (which occurs only if $Prev(\beta, i) = r$ as tested at Line 2, and in this case the new predecessor of i will be $Prev(\beta, r)$), or if the set of ancestors of this predecessor has changed (Lines 4-5).

- Otherwise, the algorithm determines whether the last ancestor of i needs to be recomputed after the removal of r (check at Line 7). If yes, Equation 11 is applied and the set of ancestors of i is updated based on Equation 10 (Lines 8-9). Otherwise, if the last ancestor of i is unchanged, then the set of ancestors of i is updated only if the set of ancestors of $lastAnc(i)$ has changed (Lines 10-11).

Algorithm 4: updateAncestors$(i, \sigma, \beta, r, changedAnc)$

Input: i: customer to analyze; σ: a solution; β: a partial solution obtained from σ; r: the customer to remove; $changedAnc$: customers whose ancestors change if r is removed

1 **if** $(i \in \beta) \wedge (i \neq r)$ **then**
2 **if** $Prev(\beta, i) = r$ **then**
3 $Anc(i) \leftarrow Anc_0(i) \cup \{Prev(\beta, r)\} \cup Anc(Prev(\beta, r))$
4 **else if** $Prev(\beta, i) \in changedAnc$ **then**
5 $Anc(i) \leftarrow Anc_0(i) \cup \{Prev(\beta, i)\} \cup Anc(Prev(\beta, i))$
6 **else**
7 **if** $(i = r) \vee (lastAnc(i) = r)$ **then**
8 $lastAnc(i) \leftarrow argmax_{k \in \{j \in Pred_0(i) \mid j \in \beta\} \cup \{lastAnc(j) \mid j \in Pred_0(i), j \notin \beta\}} pos(\sigma, k)$
9 $Anc(i) \leftarrow Anc_0(i) \cup Anc(lastAnc(i))$
10 **else if** $lastAnc(i) \in changedAnc$ **then**
11 $Anc(i) \leftarrow Anc_0(i) \cup Anc(lastAnc(i))$

The *updateDescendants* function detailed in Algorithm 5 is similar. The main difference is that for each customer i that is not in β after the removal of r, the function computes the descendant in $Desc_0(i)$ that appears at the leftmost position in β, referred to as $firstDesc(\beta, i)$. The counterpart of Equation 11 is then:

$$firstDesc(\beta, i) \quad \leftarrow \quad argmin_{k \in \{j \in Succ_0(i) \mid j \in \beta\} \cup \{firstDesc(\beta, j) \mid j \in Succ_0(i), j \notin \beta\}} pos(\sigma, k) \tag{12}$$

and the recursive equations used to compute the descendants of every customer i are:

$$Desc(\beta, i) \quad \leftarrow \quad Desc_0(i) \cup \{Next(\beta, i)\} \cup Desc(\beta, Next(\beta, i)) \text{ if } i \in \beta \tag{13}$$

$$Desc_0(i) \cup Desc(\beta, firstDesc(\beta, i)) \text{ otherwise.} \tag{14}$$

In practice, the incremental computations detailed before are a key point in ImaxLNS as they allow to increase the number of destroy operations that can be performed per second.

6.4. Complexity of the destroy procedure

In the following, we assume that the sets of predecessors, successors, ancestors, and descendants of every customer are represented as bitsets. For example, the jth bit of $Anc_0(i)$ is equal to 1 if and only if customer j is a mandatory ancestor of i in precedence graph G.

Proposition 1. *The worst-case time complexity of the destroy procedure is* $O(N^3)$.

Proof. For each removal attempt, updating the last ancestor and the first descendant of *all* customers has a complexity $O(|\mathcal{P}|)$ coming from the *argmax* at Line 8 of Algorithm 4 and the *argmin* at Line 8 of Algorithm 5.

Algorithm 5: updateDescendants($i, \sigma, \beta, r, changedDesc$)

Input: i: customer to analyze; σ: a solution; β: a partial solution obtained from σ; r: the customer to remove; $changedDesc$: customers whose descendants change if r is removed

if $(i \in \beta) \wedge (i \neq r)$ **then**
  if $Next(\beta, i) = r$ **then**
    $Desc(i) \leftarrow Desc_0(i) \cup \{Next(\beta, r)\} \cup Desc(Next(\beta, r))$
  else if $Next(\beta, i) \in changedDesc$ **then**
    $Desc(i) \leftarrow Desc_0(i) \cup \{Next(\beta, i)\} \cup Desc(Next(\beta, i))$
else
  if $(i = r) \vee (firstDesc(i) = r)$ **then**
    $firstDesc(i) \leftarrow argmin_{k \in \{j \in Succ_0(i) \mid j \in \beta\} \cup \{firstDesc(j) \mid j \in Succ_0(i), j \notin \beta\}} pos(\sigma, k)$
    $Desc(i) \leftarrow Desc_0(i) \cup Desc(firstDesc(i))$
  else if $firstDesc(i) \in changedDesc$ **then**
    $Desc(i) \leftarrow Desc_0(i) \cup Desc(firstDesc(i))$

From this, for each removal attempt, recomputing the sets of ancestors and descendants of all customers has a time complexity $O(N^2)$ (for each customer, at most two union operations between bitsets of size $N + 2$). Over all removal attempts, the insertion-width of each position is incremented at most $Wmax$ times, therefore managing all insertion-width increments over all removal attempts has a time complexity $O(Wmax \cdot N)$. From the previous observations, testing the successive removals of all customers has a time complexity $O(N \cdot (|\mathcal{P}| + N^2) + Wmax \cdot N)$, which can be transformed into $O(N^3)$. □

With regards to Proposition 1, let us stress that first $O(N^3)$ is only a worst-case time complexity since some sets of ancestors and descendants will not need to be updated for some removal attempts, and second, one of the N factors comes from a single bitset union operation. The latter is indeed linear in N, but based on a 64-bit words encoding, it is very fast in practice even for problems containing hundreds of customers. Additionally, if the transitive reduction of the precedence graph is sparse (number of arcs linear in N), term $N \cdot |\mathcal{P}|$ is only quadratic in N. On these points, the experiments confirm that the time consumed by the destroy phase is low compared to the time consumed by the repair phase.

7. Repair procedure

This section defines the repair procedure that produces an optimized (complete) solution σ starting from the partial solution β returned by the destroy phase. This procedure uses a dynamic programming approach. Section 7.1 describes the dynamic programming equations to be computed, Section 7.2 discusses the rep-

resentation of states during dynamic programming, Section 7.3 formalizes the repair algorithm, Section 7.4 discusses implementation details, Section 7.5 gives linear complexity results, and Section 7.6 introduces state elimination techniques. As before, we remove parameter β from the notations manipulated since we work here with a unique partial solution. Therefore, the set of customers removed for β is denoted by $\mathcal{R}$ instead of $\mathcal{R}(\beta)$, and the set of removed customers that can occupy position p is denoted by $\mathcal{R}(p)$ instead of $\mathcal{R}(\beta, p)$.

7.1. Dynamic programming equations: formulation and properties

The repair procedure explores so-called *visit states* defined as triples (p, S, i) where $p \in [0..N+1]$ stands for a position in the sequence of visits, S represents a set of customers visited until position p (position p included), and i denotes the last customer visited. Initially, the set of visit states reachable at position 0 is $\mathcal{S}(0) = \{(0, \{0\}, 0)\}$ (a unique visit state corresponding to the visit of customer 0). Then, from a visit state (p, S, j) obtained at position p, a customer i can be visited at position $p+1$ if and only if it is not visited yet in S and all its ancestors in $G(\beta)$ are already visited in S. Formally, the set of visit states reachable at a position $p+1 \in [1..N+1]$ is recursively defined by:

$$\mathcal{S}(p+1) = \{(p+1, S \cup \{i\}, i) \,|\, \exists (p, S, j) \in \mathcal{S}(p) \text{ s.t. } i \in [0..N+1] \setminus S \text{ and } Anc(i) \subseteq S\} \tag{15}$$

The objective of the repair procedure is to compute the best way to reach visit state $(N+1, [0..N+1], N+1)$ that visits all customers and ends with customer $N+1$. To do this, an evaluation is associated with each visit state (p, S, i). This evaluation corresponds to a cost vector $\nu(p, S, i) \in \mathbb{R}^3$ called the tardiness-makespan-cost evaluation (or *tmc-evaluation*) of (p, S, i). It is a triple $(\delta(p, S, i), \tau(p, S, i), \rho(p, S, i))$ where $\delta(p, S, i)$, $\tau(p, S, i)$, and $\rho(p, S, i)$ respectively stand for a tardiness value, a completion time, and a cumulated transition cost. The evaluation of the initial visit state is $\nu(0, \{0\}, 0) = (0, 0, 0)$, and the evaluation of a reachable visit state $(p+1, S \cup \{i\}, i)$ is recursively defined by:

$$\nu(p+1, S \cup \{i\}, i) = \underset{(p,S,j) \in \mathcal{S}(p) \text{ s.t. } i \in [0..N+1] \setminus S \text{ and } Anc(i) \subseteq S}{minLex} StepEval(\nu(p, S, j), j, i) \tag{16}$$

The previous recursive equation means that $\nu(p+1, S \cup \{i\}, i)$ is the "best" evaluation obtained when performing a transition from a possible previous visit state (p, S, j), using the *StepEval* function introduced in Section 3. The best evaluation corresponds to the best cost vector when using a lexicographic ordering (minimum tardiness, then minimum makespan, and finally minimum transition cost). Note that Equation 16 keeps a unique evaluation for each visit state instead of recording all Pareto-optimal evaluations in terms of tardiness and current time.

During this process, a *parent customer* $\pi(p+1, S \cup \{i\}, i)$ can be recorded to memorize the customer to visit just before i to obtain cost vector $\nu(p+1, S \cup \{i\}, i)$. This parent customer is chosen so that $\nu(p+1, S \cup \{i\}, i) = StepEval(\nu(p, S, j), j, i)$ for $j = \pi(p+1, S \cup \{i\}, i)$. This allows to extract a sequence of customer visits leading to each visit state, as expressed in the following definition.

Definition 1. (path leading to a visit state) *The path leading to visit state* (p, S, i)*, referred to as* $Path(p, S, i)$*, is the sequence of customers recursively defined by:*

$$Path(p, S, i) = \begin{cases} [0] \text{ if } p = 0 \\ Path(p-1, S \setminus \{i\}, \pi(p, S, i)) \cdot [i] \text{ otherwise} \end{cases} \quad (17)$$

The repair procedure detailed in the following efficiently computes the sequence of customer visits corresponding to $Path(N+1, [0..N+1], N+1)$. Proposition 2 shows that this sequence is indeed a solution that completes the partial solution β obtained after the destroy phase, and Proposition 3 gives makespan-optimality guarantees.

Proposition 2. *The path* $Path(N+1, [0..N+1], N+1)$ *leading to visit state* $(N+1, [0..N+1], N+1)$ *corresponds to a solution (feasible or not) that is a completion of* β*.*

Proposition 3. *Let us assume that the transition function satisfies the FIFO property. If there exists a completion* σ *of partial solution* β *such that* $\delta(\sigma) = 0$ *(null tardiness), then solution* $\sigma' = Path(N+1, [0..N+1], N+1)$ *is feasible and makespan-optimal among the completions of* β*.*

As a result, the dynamic programming equations allow to compute a makespan-optimal repair as soon as there exists a feasible completion of β. Conversely, if the cumulated tardiness of $Path(N+1, [0..N+1], N+1)$ is not null, it is possible to infer that there is no feasible solution completing β. In the general case, when there does not exist a feasible completion of β, the repair process must just be seen as a heuristic procedure since it offers no guarantee to return a solution having a minimum tardiness. Indeed, let us consider the counterexample given in Figure 3, where there are only two possible sequences of visits $\sigma_1 = [0, 1, 2, 3, 4, 5]$ and $\sigma_2 = [0, 2, 1, 3, 4, 5]$. The figure gives the time window associated with each customer, and the paths representing σ_1 and σ_2 are labeled by the transition times (no time-dependency here). If the goal is to minimize the total tardiness then the best solution is σ_1 since $\delta(\sigma_1) = 1$ and $\delta(\sigma_2) = 2$. However, in this case, the dynamic programming equations applied from partial solution $\beta = [0, 5]$ will return sequence σ_2. The reason for this is that among the two solution prefixes $\sigma'_1 = [0, 1, 2, 3]$ and $\sigma'_2 = [0, 2, 1, 3]$ that lead to the same visit state $(3, \{0, 1, 2, 3\}, 3)$, the second one is preferred because even if its completion time is higher, it has a lower tardiness ($\delta(\sigma'_2) = 0$ whereas $\delta(\sigma'_1) = 1$). Said differently, the repair procedure tries to solve tardiness issues as early as possible. As shown in the experiments, this approach is robust to get a first feasible solution. Note that when the FIFO property is not satisfied, the dynamic programming approach is still applicable but the formal guarantees given in Proposition 3 do not hold anymore.

7.2. Compact representation of visit states

To efficiently compute the dynamic programming equations, it is first possible to represent visit states in a much more compact way. Indeed, for a state (p, S, i) obtained at position p, it can be shown that it is

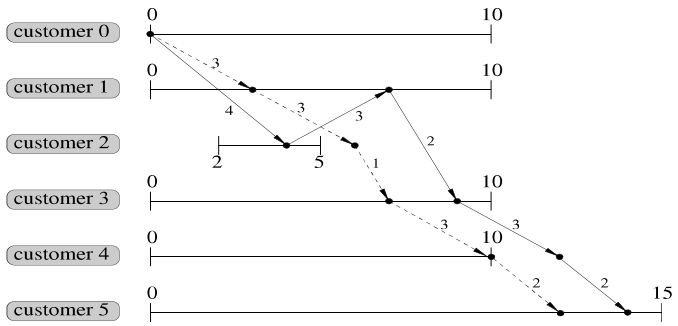

Figure 3: Example involving $N = 4$ customers, with precedence constraints $1 \to 3$, $2 \to 3$, and $3 \to 4$

useless to represent in S the removed customers that are necessarily visited strictly before position p, *i.e.* the customers in $\mathcal{R}_<(p) = \{i \in \mathcal{R} \mid P_{max}(i) < p\}$. For the example of Figure 1b, these sets are given in column $\mathcal{R}_<(p)$ in Table 2. Additionally, it is useless to explicitly represent in S the visited customers belonging to β, the rationale being that these customers can be directly inferred from p and the size of $S \cap \mathcal{R}(p)$. To illustrate these points, on the example introduced before where $\beta = [0, 2, 4, 5, 9]$ and with the precedence graph $G(\beta)$ given in Figure 1b, state $(p, S, i) = (7, \{0, 1, 2, 3, 4, 5, 6, 8\}, 8)$ can be compactly represented as $(p, R, i) = (7, \{6, 8\}, 8)$, since (1) the algorithm will guarantee that all customers in $\mathcal{R}_<(7) = \{1, 3\}$ are visited before position 7, and (2) as 4 removed customers are visited over the 8 positions available from position 0 to position 7 (customers in $\{6, 8\} \cup \{1, 3\}$), there are necessarily 4 non-removed customers visited over these positions, and as the algorithm will respect the precedence constraints defined by partial solution $\beta = [0, 2, 4, 5, 9]$, these 4 customers can only be customers 0, 2, 4, and 5. As a result, from the restricted set $R = \{6, 8\}$, we can easily infer that the full set of visited customers is $S = \{0, 1, 2, 3, 4, 5, 6, 8\}$ as in the extended representation. In the end, for a visit state (p, S, i), only the customers in $R = S \cap \mathcal{R}(p)$ need to be represented. Proposition 4 formalizes this property.

Proposition 4. *For every reachable visit state (p, S, i), if R denotes the restriction of S to $\mathcal{R}(p)$ ($R = S \cap \mathcal{R}(p)$), then we have $S = R \cup \mathcal{R}_<(p) \cup \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$. This implies that (p, S, i) can be represented by the so-called compact visit state (p, R, i). Moreover, the last customer visited in (p, S, i) always satisfies condition $i \in R \cup \{\beta_{p-|R|-|\mathcal{R}_<(p)|}\}$, that is either it belongs to R or it is a specific customer in β.*

Last, Proposition 5 shows that given a visit state (p, S, i), its compact visit state (p, R, i) suffices to quickly reconstruct the compact visit state associated with the state $(p - 1, S \setminus \{i\}, \pi(p, S, i))$ to visit at the previous position, hence manipulating compact visit states induces no information loss on this aspect. For this, the proposition uses, for each position p, set $\mathcal{R}_{max}(p)$ that corresponds to the set of removed customers whose maximum position is equal to p ($\mathcal{R}_{max}(p) = \{i \in \mathcal{R} \mid P_{max}(i) = p\}$). The values of these sets for the same example as before are given in column $\mathcal{R}_{max}(p)$ in Table 2.

p	$\mathcal{R}(p)$	$\mathcal{R}_{min}(p)$	$\mathcal{R}_{max}(p)$	$\mathcal{R}_{<}(p)$	$T_{\mathcal{R}}(p)$
0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$[-1,-1,-1,-1]$
1	$\{1,3\}$	$\{1,3\}$	$\emptyset$	$\emptyset$	$[\ 1,\ 3,-1,-1]$
2	$\{1,3,7\}$	$\{7\}$	$\emptyset$	$\emptyset$	$[\ 1,\ 3,\ 7,-1]$
3	$\{1,3,6,7\}$	$\{6\}$	$\emptyset$	$\emptyset$	$[\ 1,\ 3,\ 7,\ 6]$
4	$\{1,3,6,7\}$	$\emptyset$	$\emptyset$	$\emptyset$	$[\ 1,\ 3,\ 7,\ 6]$
5	$\{1,3,6,7\}$	$\emptyset$	$\{3\}$	$\emptyset$	$[\ 1,\ 3,\ 7,\ 6]$
6	$\{1,6,7\}$	$\emptyset$	$\{1\}$	$\{3\}$	$[\ 1,-1,\ 7,\ 6]$
7	$\{6,7,8\}$	$\{8\}$	$\{6\}$	$\{1,3\}$	$[\ 8,-1,\ 7,\ 6]$
8	$\{7,8\}$	$\emptyset$	$\{7,8\}$	$\{1,3,6\}$	$[\ 8,-1,\ 7,-1]$
9	$\emptyset$	$\emptyset$	$\emptyset$	$\{1,3,6,7,8\}$	$[-1,-1,-1,-1]$

Table 2: Sets of removed customers associated with each position, for the example in Figure 1b where $\mathcal{R} = \{1,3,6,7,8\}$

Proposition 5. *Let (p,S,i) be a visit state reached at position $p \in [1..N+1]$ and let (p,R,i) be its compact version. Then, the compact visit state associated with $(p-1, S\setminus\{i\}, \pi(p,S,i))$ is $(p-1, R', \pi(p,S,i))$ where*

$$R' = \begin{cases} (R\setminus\{i\}) \cup \mathcal{R}_{max}(p-1) \text{ if } i \in \mathcal{R}(p) \\ R \cup \mathcal{R}_{max}(p-1) \text{ otherwise} \end{cases} \tag{18}$$

This means that in the compact parent visit state $(p-1, R', \pi(p,S,i))$, customer i is not visited yet in R' and all removed customers for which position $p-1$ is the last possible one are added to R'.

7.3. Definition of the repair procedure

The repair procedure is given in Algorithm 6. It exploits compact visit states (p,R,i) instead of extended visit states (p,S,i), and instead of manipulating cost vectors $\nu(p,S,i) \in \mathbb{R}^3$ and parent customers $\pi(p,S,i)$ applied to extended visit states, it manipulates cost vectors $\nu(p,R,i) \in \mathbb{R}^3$ and parent customers $\pi(p,R,i)$ applied to compact visit states. The two cases can anyway always be distinguished since customer 0 always belong to S while it never belongs to R. For complexity reasons, the repair algorithm also maintains, for each position $p \in [0..N+1]$, a data structure $\Omega(p)$ that contains all sets R such that there exists a visit state (p,R,j) reachable at position p.

In Algorithm 6, the (unique) initial visit state is $(p,R,i) = (0,\emptyset,0)$, its evaluation is $\nu(0,\emptyset,0) = (0,0,0)$, and the unique set R such that there is a reachable visit state $(0,R,i)$ is $R = \emptyset$ (Lines 1-2). All visit state evaluations at positions $p > 0$ are initialized to (∞,∞,∞), meaning that no visit state has been reached so far (Line 3). The algorithm also computes, for each customer i and each position p that can be occupied by i, set $AncR(i,p) \leftarrow Anc(i) \cap \mathcal{R}(p)$ that contains the removed customers that are ancestors of i and can occupy position p (Line 4).

After that, the algorithm expands the visit states (p, R, j) by increasing position (Lines 5-23). For each possible set $R \in \Omega(p)$, it first computes the last and next customers visited in β according to R (Line 8). The formulas used are justified by Proposition 4 seen before. Algorithm 6 also computes set $R' = R \setminus \mathcal{R}_{max}(p)$ that contains all customers in R that must be kept in the compact representation of visit states at position $p+1$. Then, each expansion phase adds the visit of one new customer i such that all ancestors of i are already visited. Two kinds of new customer visits are analyzed at position $p+1$.

- Case 1: visit of a removed customer (Lines 10-16).

  In this case, Line 10 considers a customer $i \in \mathcal{R}(p+1) \setminus R'$ (*i.e.*, a customer that can occupy position $p+1$ and that is not visited yet) such that $\beta next \notin Anc(i)$ (*i.e.*, the next non-removed customer to visit is not an ancestor of i) and $AncR(i, p+1) \subseteq R'$ (*i.e.*, all ancestors of i that are removed customers are already visited). As shown in the supplementary material, these conditions are equivalent to "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" where S is the set of customers already visited given p and R.

  For each possible next visit of a customer i, the algorithm considers all possible previous visit states, with one visit state per customer j such that $\nu(p, R, j) \neq (\infty, \infty, \infty)$ (Line 11). It then uses function *StepEval* introduced in Section 3 to compute the tmc-evaluation v obtained when performing transition $j \rightarrow i$ from state (p, R, j). The visit state obtained is $(p+1, R' \cup \{i\}, i)$. If the resulting tmc-evaluation v is lexicographically smaller than the current tmc-evaluation of $(p+1, R' \cup \{i\}, i)$, the evaluation and the parent of this visit state are updated (Lines 14-15).

- Case 2: visit of a non-removed customer (Lines 17-23).

  In this case, the unique customer to consider is $i = \beta next$ (*i.e.*, the next non-removed customer to visit), and condition $AncR(\beta next, p+1) \subseteq R'$ ensures that all removed customers that are ancestors of $\beta next$ and candidates for occupying position $p+1$ are already visited (Line 17). As shown in the supplementary material, these conditions are again equivalent to "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" where S is the set of customers already visited given p and R.

  If customer $i = \beta next$ can be the next customer visited, the algorithm considers all possible previous visit states, with one visit state per customer j such that $\nu(p, R, j) \neq (\infty, \infty, \infty)$ (Line 18). Again, it uses function *StepEval* to compute the tmc-evaluation v obtained when transition $j \rightarrow \beta next$ is performed from state (p, R, j). Such a transition reaches visit state $(p+1, R', \beta next)$. If the resulting tmc-evaluation v is lexicographically smaller than the current tmc-evaluation of $(p+1, R', \beta next)$, the evaluation and the parent of this visit state are updated (Lines 21-22).

Once the visit states are built for all positions, a sequence of visits is extracted from the recorded parent customers thanks to the *reconstructPath* function given in Algorithm 7. The latter reuses the results provided in Proposition 5 to perform a backward traversal of the visit states.

Algorithm 6: REPAIR(β)

Input: β: a partial solution

Output: A solution σ that is completion of β

$\nu(0, \emptyset, 0) \leftarrow (0, 0, 0)$

$\Omega(0) \leftarrow \{\emptyset\}$

foreach $p \in [1..N+1]$, $R \subseteq \mathcal{R}(p)$, $i \in R \cup \{\beta_{p-|R|-|\mathcal{R}_<(p)|}\}$ **do** $\nu(p, R, i) \leftarrow (\infty, \infty, \infty)$

foreach $i \in [1..N+1]$, $p \in [P_{min}(i)..P_{max}(i)]$ **do** $AncR(i, p) \leftarrow Anc(i) \cap \mathcal{R}(p)$

foreach $p = 0$ *to* N **do**

  $\Omega(p+1) \leftarrow \emptyset$

  foreach $R \in \Omega(p)$ **do**

    $(\beta last, \beta next) \leftarrow (\beta_{p-|R|-|\mathcal{R}_<(p)|}, \beta_{p-|R|-|\mathcal{R}_<(p)|+1})$

    $R' \leftarrow R \setminus \mathcal{R}_{max}(p)$

    `/* case 1: visit a removed customer */`

    foreach $i \in \mathcal{R}(p+1) \setminus R'$ *s.t.* $\beta next \notin Anc(i)$ *and* $AncR(i, p+1) \subseteq R'$ **do**

      foreach $j \in R \cup \{\beta last\}$ *s.t.* $\nu(p, R, j) \neq (\infty, \infty, \infty)$ **do**

        $v \leftarrow StepEval(\nu(p, R, j), j, i)$

        if $v < \nu(p+1, R' \cup \{i\}, i))$ *(+ possible state pruning, see Section 7.6)* **then**

          $\nu(p+1, R' \cup \{i\}, i) \leftarrow v$

          $\pi(p+1, R' \cup \{i\}, i) \leftarrow j$

          $\Omega(p+1) \leftarrow \Omega(p+1) \cup \{R' \cup \{i\}\}$

    `/* case 2: progress in the visits of non-removed customers */`

    if $AncR(\beta next, p+1) \subseteq R'$ **then**

      foreach $j \in R \cup \{\beta last\}$ *s.t.* $\nu(p, R, j) \neq (\infty, \infty, \infty)$ **do**

        $v \leftarrow StepEval(\nu(p, R, j), j, \beta next)$

        if $v < \nu(p+1, R', \beta next))$ *(+ possible state pruning, see Section 7.6)* **then**

          $\nu(p+1, R', \beta next) \leftarrow v$

          $\pi(p+1, R', \beta next) \leftarrow j$

          $\Omega(p+1) \leftarrow \Omega(p+1) \cup \{R'\}$

return *reconstructPath*(π)

7.4. Implementation details

Bitset representations. For each position p, all along the iterations of ImaxLNS, set $\mathcal{R}(p)$ (the set of removed customers that can occupy position p) contains at most $Wmax$ customers due to the constraints imposed at the destroy phase. This allows us to represent $\mathcal{R}(p)$ using a table $T_{\mathcal{R}}(p)$ of size $Wmax$. For instance, as

Algorithm 7: reconstructPath(π)

Input: π: parent customers computed during the expansion phase of visit states

Output: A solution σ

```
1 σ ← ∅
2 (R, i) ← (∅, N + 1)
3 foreach p = N + 1 to 1 do
4 |   σ ← [i] · σ
5 |   j ← π(p, R, i)
6 |   R ← if (i ∈ R(p)) then (R \ {i}) ∪ R_max(p − 1) else R ∪ R_max(p − 1)
7 |   i ← j
8 return [0] · σ
```

illustrated in Table 2 where $Wmax = 4$, set $\mathcal{R}(p) = \{1, 3, 7\}$ at position $p = 2$ is represented by $T_{\mathcal{R}}(p) = [1, 3, 7, -1]$. The kth element of $T_{\mathcal{R}}(p)$ is referred to as $T_{\mathcal{R}}(p, k)$. From this, any state R included in $\mathcal{R}(p)$ can be represented as a bitset $\mathcal{B}(R)$ of size $Wmax$ where the kth bit of $\mathcal{B}(R)$ is set to 1 if and only if R contains customer $T_{\mathcal{R}}(p, k)$. For instance, if $T_{\mathcal{R}}(p) = [1, 3, 7, 6]$ such as at position $p = 5$ in the example of Table 2, set $R = \{1, 6\}$ corresponds to $\mathcal{B}(R) = 1001$. For two sets X, Y included in $\mathcal{R}(p)$ for a given position p, operations $X \cup Y$, $X \cap Y$, $X \setminus Y$ can then be performed as bitset operations in time linear in $Wmax$ (bitwise operations "$\mathcal{B}(X) \,|\, \mathcal{B}(Y)$", "$\mathcal{B}(X) \,\&\, \mathcal{B}(Y)$", and "$\mathcal{B}(X) \,\&\, !\mathcal{B}(Y)$" respectively). This entails that set $R \setminus \mathcal{R}_{max}(p)$ used at Line 9 of Algorithm 6 can be computed in time $O(Wmax)$, since $\mathcal{R}_{max}(p) \subseteq \mathcal{R}(p)$ and $R \subseteq \mathcal{R}(p)$ always hold at that line. Similarly, set $\mathcal{R}(p+1) \setminus R'$ can be computed in time $O(Wmax)$ at Line 10 since $R' \subseteq \mathcal{R}(p+1)$ always holds at that line. In our implementation, any bitset $\mathcal{B}(R)$ is actually represented as a single 32-bit integer since the value of $Wmax$ is always lower than 32 in our settings (CPU time and memory consumption are prohibitive otherwise).

Moreover, in a visit state (p, R, i), integer i (the last visited customer) is represented by an index $k \in [0..Wmax]$: if i is a removed customer, this index is the unique integer $k \in [0..Wmax - 1]$ such that $T_{\mathcal{R}}(p, k) = i$; otherwise (case $i = \beta_{p-|R|-|\mathcal{R}_<(p)|}$), this index takes value $k = Wmax$.

Another trick is that if every customer i always has the same index in all tables $T_{\mathcal{R}}(p)$ associated with the positions p it can occupy, then a set R that is included both in $\mathcal{R}(p)$ and $\mathcal{R}(p+1)$ has the exact same bitset representation at positions p and $p+1$. For instance, at position 6 where $T_{\mathcal{R}}(6) = [1, -1, 7, 6]$, set $\{1, 6\}$ corresponds to bitset 1001, as at position 5 where $T_{\mathcal{R}}(5) = [1, 3, 7, 6]$. This makes it easier to handle sets of customers from one position to the next. The good news is that it is always possible to define such a unique index $Windex(i) \in [0..Wmax-1]$ for each removed customer i. For this, we start from $T_{\mathcal{R}}(p)$ tables filled with value -1 and traverse the successive positions p. For each position p, a unique W-index is allocated to each

customer i that is "activated" at position p, that is to each customer in set $\mathcal{R}_{min}(p) = \{i \in \mathcal{R} \mid P_{min}(i) = p\}$. This index is then booked for i in the $T_{\mathcal{R}}(p')$ tables at all positions $p' \in [P_{min}(i)..P_{max}(i)]$. By construction, it can be easily shown that when considering customer i, finding a free W-index is always possible since the insertion-width of β is bounded by $Wmax$. For instance, at position 1 in Table 2, where $\mathcal{R}_{min}(1) = \{1, 3\}$, we first book index $Windex(1) = 0$ for customer 1 and index $Windex(3) = 1$ for customer 3. When considering position 2, we book index $Windex(7) = 2$ for customer 7, and when considering position 3, we book index $Windex(6) = 3$ for customer 6. At position 7 where $\mathcal{R}_{min}(7) = \{8\}$, the free W-index chosen for customer 8 is $Windex(8) = 0$, since index 0 was booked by customer 1 only until position 6. In the general case, it can be shown that such a process allows us to both define a unique W-index for each customer i and to compute the $T_{\mathcal{R}}(p)$ tables with a time and space complexity $O(N \cdot Wmax)$.

Last, note that for the repair algorithm to work, it suffices to store the $\pi(p, R, i)$ values and not the whole set of $\nu(p, R, i)$ values. Indeed, at each step, we only need two copies of data structures $\nu(p, R, i)$, namely one copy for the evaluation of the states at the current position and another one for the evaluation of the states at the next position.

7.5. Complexity of the repair procedure

Proposition 6. *If each call to transition function tt has a time and space complexity $O(1)$, then the repair procedure has a time complexity $O(2^{Wmax} \cdot Wmax^2 \cdot N)$ and a space complexity $O(2^{Wmax} \cdot Wmax \cdot N)$.*

Proof. At Line 4, all sets $AncR(i, p)$ can be computed in time $O(N \cdot Wmax^2)$. Indeed, for each position p, there are (1) at most $Wmax$ *removed customers* that can occupy position p, (2) at most $Wmax + 1$ *non-removed customers* that can occupy position p, since every customer of β that can be visited at position p has the form $\beta_{p-|R|-|\mathcal{R}_<(p)|}$, where the only variable component is the cardinality of R that is always between 0 and $Wmax$. Moreover, for each pair (i, p), the intersection at Line 4 can be computed in time $O(Wmax)$ since $|\mathcal{R}(p)| \leq Wmax$ and since after the destroy phase, set $Anc(i)$ is already available as a bitset.

Next, for each position $p \in [0..N]$, the number of states $R \subseteq \mathcal{R}(p)$ involved in visit states (p, R, i) is bounded by 2^{Wmax} since $|\mathcal{R}(p)| \leq Wmax$. For each pair (p, R) involved in a visit state, at most $Wmax + 1$ next customers need to be considered (at most $Wmax$ possible values for i at Line 10, plus value $i = \beta next$ in the second case analyzed). For each candidate next customer i, the "ancestors-checks" at Lines 10 and 17 are feasible in time $O(Wmax)$ using bitset representations. Then, at most $Wmax+1$ values of j are analyzed at Lines 11 and 18. As a result, the overall time complexity of Lines 7 to 23 is $O(2^{Wmax} \cdot Wmax^2)$. Over all positions, this leads to a time complexity $O(2^{Wmax} \cdot Wmax^2 \cdot N)$. Last, the path reconstruction phase is feasible in time $O(Wmax \cdot N)$ since operations "$(R \setminus \{i\}) \cup \mathcal{R}_{max}(p-1)$" and "$R \cup \mathcal{R}_{max}(p-1)$" used at Line 6 in Algorithm 7 can be performed in time $O(Wmax)$ thanks to bitset representations again. The space complexity comes from the data structures related to visit states (p, R, i) (at most $2^{Wmax} \cdot (Wmax + 1) \cdot N$ visit states over all positions $p \in [1..N]$). □

As a result, the repair process has a complexity that is only linear in N given a fixed $Wmax$ value. In comparison, let us recall that the standard Held-Karp procedure for TSP has a time complexity $O(2^N \cdot N^2)$ and a space complexity $O(2^N \cdot N)$ [38, 39]. The complexity result given before allows us to identify a new polynomial class for (TD)TSPTW.

Proposition 7. *When the transition function satisfies the FIFO property and can be computed in polynomial time, determining whether there exists a feasible solution to (TD)TSPTWs whose insertion-width is bounded by a fixed constant W is polynomial. Moreover, if there exists a feasible solution, finding a makespan-optimal solution is polynomial as well, simply by applying the repair procedure from partial solution $\beta = [0, N+1]$.*

7.6. State elimination techniques

One last standard ingredient of a dynamic programming algorithm for (TD)TSPTW is a state elimination technique. In ImaxLNS, when the tardiness of the solution σ considered before the destroy phase is null, we force the repair process to explore only solutions that have a null tardiness. For this, we use a backward propagation step that computes, before the repair process, a latest visit time for each customer in partial solution β. To do this, there is a need to dispose of a backward transition time function $\overleftarrow{tt}$ such that for any pair of distinct customers $(i,j) \in [0..N] \times [1..N+1]$ and any time $\tau \in]-\infty, End(j)]$, quantity $\overleftarrow{tt}(i,j,\tau)$ gives a lower bound on the duration required to perform a transition or a chain of transitions from i to j and reach customer j before time τ (ideally, $\overleftarrow{tt}(i,j,\tau) = \min\{\gamma \in \mathbb{R}^+ \mid \overrightarrow{tt}(i,j,\tau-\gamma) \leq \gamma\}$). When transition function tt satisfies the triangular inequality and is not time-dependent ($tt(i,j,\tau) = tt(i,j)$), it suffices to use $\overleftarrow{tt}(i,j,\tau) = tt(i,j)$. For TDTSPTW in general, $\overleftarrow{tt}(i,j,\tau)$ can be computed by an iterative method that (1) starts from an interval $[\gamma_1, \gamma_2]$ within which $\overleftarrow{tt}(i,j,\tau)$ should be looked for, and (2) considers at each step a new value $\gamma_3 \in [\gamma_1, \gamma_2]$ obtained by dichotomy or linear interpolation, to converge to $\overleftarrow{tt}(i,j,\tau)$ up to a given precision. We actually used such an iterative technique for one of the benchmark involving a black-box time-dependent transition function. Specific implementations of $\overleftarrow{tt}$ can also be defined for some TDTSPTW benchmarks. For instance, for benchmarks involving time-dependent vehicle speeds, $\overleftarrow{tt}$ can be obtained from a relaxed time-independent model making a maximum speed assumption.

Then, assume that the current solution σ considered before the destroy phase has a null tardiness ($\delta(\sigma) = 0$). Based on function $\overleftarrow{tt}$, we traverse partial solution $\beta = [\beta_0, \ldots, \beta_{m+1}]$ in a backward fashion and compute, for each customer β_k, a latest visit start time $lt(\beta_k)$. For the end customer $\beta_{m+1} = N+1$, we use $lt(\beta_{m+1}) = \tau(\sigma)$ (*i.e.* the makespan of σ). For any customer β_k that is not the last one in β, we use:

$$lt(\beta_k) = \max(m, m') \text{ where } \begin{array}{l} m = \min(End(\beta_k), lt(\beta_{k+1}) - \overleftarrow{tt}(\beta_k, \beta_{k+1}, lt(\beta_{k+1}))) \\ m' = \tau(\sigma, pos(\sigma, \beta_k)) \end{array} \quad (19)$$

Term m corresponds to the latest visit time of β_k according to function $\overleftarrow{tt}$. Term m' corresponds the current visit time of β_k in the solution σ considered before the repair phase. Taking into account both m and m'

allows to guarantee that the latest visit time computed for β_k is never more restrictive than the feasible visit time available in solution σ. This is useful to increase the robustness of the algorithm with regards to precision issues in the computation of $\overleftarrow{tt}$, and more generally with regards to cases where $\overleftarrow{tt}$ does not provide true lower bounds on transition times. If $\overleftarrow{tt}$ can return results in time $O(1)$, then the latest visit start times for all customers in β can be computed in time $O(N)$.

After that, during the repair process, when a state evaluation $v = (\delta, \tau, \rho)$ is computed after a transition $j \rightarrow i$ from a state (p, R, j), this evaluation is discarded at Lines 13 and 20 of Algorithm 6 if it does not allow to reach the next customer in β on time, that is if condition $\tau + \overrightarrow{tt}(i, \beta next, \tau) > lt(\beta next)$ holds. Extended state-elimination techniques do exist for TDTSPTW, however the first experiments performed by using such techniques within ImaxLNS were not conclusive. This is why lightweight pruning rules are used instead.

One last remark is that even if $\overleftarrow{tt}$ does not provide true lower bounds on transition times, the repair algorithm can still be used. The only impact is that it might produce suboptimal results in this case.

8. Experiments

This section presents the results obtained on several benchmarks: (1) standard TSPTW benchmarks, (2) the TDTSPTW benchmark defined by Arigliano et al. [5], (3) the TDTSPTW benchmark defined by Aguiar-Melgarejo et al. [6], and (4) a new TDTSPTW benchmark related to the management of Earth observation satellites.

8.1. Common evaluation methodology

Execution environment. ImaxLNS is implemented in C++ and tested on an Intel Xeon processor E5-2660-v3 (2.60 GHz, 25MB cache) with 65GB of RAM. For each instance of each benchmark, 5 runs are performed to obtain both best and mean results over the 5 runs. Each run is executed in a single thread. The maximum CPU time for each run is set to 5 seconds for preliminary experiments that help determining a good value for parameter *Wmax*, and to 1 or 2 minutes for the hard TDTSPTW benchmarks.

Algorithmic settings. As explained in Section 5, ImaxLNS has three parameters: (1) *Wmax*, the maximum insertion-width allowed, (2) *Rmin*, the minimum number of times each customer must be removed and reinserted before considering that a local optimum is reached, and (3) *Kmax*, the maximum number of 1-shift moves performed during the perturbation phases. For all the experiments presented thereafter, we use values $Rmin = 3$ and $Kmax = 8$ which provide good results, and we show the impact of parameter *Wmax* for $Wmax \in [1..10]$. Moreover, during the preprocessing phase, dynamic programming is systematically applied from an empty partial solution $\beta = [0, N + 1]$ when the insertion-width of the problem is less than 12. In this case, the solution found is returned without any further search.

Metrics. For each instance solved by 5 runs, we analyze several metrics for ImaxLNS, namely **#nsr**: over the 5 runs, the number of times ImaxLNS does not find a feasible solution; **BG**: the best relative gap in percent obtained over the 5 runs; the relative gap g for a run is $g = 100 \cdot (b - BK)/BK$ where b denotes the best solution found by the run and BK denotes the best-known solution for the instance; **Gm** and **Gs**: over the 5 runs, the mean value and standard deviation of the relative gap, in percent; **Tm** and **Ts**: over the 5 runs, the mean value and standard deviation of the time required to get the best solution found for a given run, in seconds; **#it/s**: over the 5 runs, the mean number of destroy-repair iterations per second.

At the level of each benchmark (covering several instances), we analyze the following metrics: **#nsr**: the number of runs for which ImaxLNS does not find a feasible solution; **#BG>0**: the number of instances for which BG is strictly positive, or equivalently the number of instances for which none of the 5 runs finds the best-known solution; **#Gm>0**: the number of instances for which Gm is strictly positive, or equivalently the number of instances for which at least one run did not find the best-known solution; max(**Gm**) and $\overline{\textbf{Gm}}$: the maximum value and the average value obtained for Gm over all instances; max(**Tm**) and $\overline{\textbf{Tm}}$: the maximum value and the average value obtained for Tm over all instances; **#spp**: the number of instances solved during the preprocessing phase of ImaxLNS.

Moreover, for each instance of each benchmark, we analyze the average number of customers removed at each destroy phase. The objective is to show that on some instances, many customers can be reinserted even when using small $Wmax$ values like $Wmax = 4$ or 5. In this case, applying *maximum* LNS is more efficient than removing a fixed number of customers at each iteration independently of the problem structure. Last, for a good value of $Wmax$ ($Wmax = 4$ or 5), we compare ImaxLNS with state-of-the-art methods.

8.2. Results on standard TSPTW benchmarks

Description of the benchmarks. We consider seven standard TSPTW benchmarks referred to as *Langevin* [10], *AFG* [40], *Dumas* [13], *SolomonPotvinBengio* [41], *SolomonPesant* [16], *GendreauDumasExtended* [18], and *OhlmannThomas* [21]. These benchmarks can be found at `http://lopez-ibanez.eu/tsptw-instances`. They cover 467 instances containing from 3 to 231 customers and involving time windows that are more of less tight. For this benchmark, the transition function tt is not time-dependent and we use $\overrightarrow{tt} = \overleftarrow{tt} = tt$.

Impact of Wmax. Table 3 shows the impact of parameter $Wmax$ for the different benchmarks. In this table, each run has a maximum duration of 5 seconds. Overall, 5 runs are performed for each of the 467 instances, which leads to 2335 runs per value of $Wmax$. Over these 2335 runs, each method *always* finds a feasible solution (#nsr = 0 for each instance). All values of $Wmax$ give quite good results, but the best option is to choose a value that is neither too low nor too high in order to be able to both consider large neighborhoods and perform many local moves (see lines #it/s). In the end, value $Wmax = 4$ is a good trade-off. It allows to find the best-known solution for each of the 2335 runs, therefore leading to a null gap all the time. Moreover,

with $Wmax = 4$, ImaxLNS reaches the best-known solutions very quickly, in less than 0.06 second on average for each benchmark. Note that all instances of the Langevin benchmark are solved during the preprocessing phase of ImaxLNS since they have a very low insertion-width.

benchmark	inst	metric	Wmax 1	2	3	4	5	6	7	8	9	10
Langevin	70	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Ts}}$(s)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		#it/s	-	-	-	-	-	-	-	-	-	-
AFG	50	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.09	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02
		$\overline{\text{Ts}}$(s)	0.38	0.12	0.02	0.02	0.01	0.02	0.01	0.01	0.04	0.05
		#it/s	45646.2	25015.5	19821.3	11100.6	9065.6	5848.3	3604.2	2211.8	1373.3	887.3
Dumas	135	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.02	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.01	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Ts}}$(s)	0.02	0.00	0.24	0.01	0.01	0.01	0.01	0.01	0.01	0.01
		#it/s	20976.2	13293.1	9460.1	6247.5	4354.9	3145.5	2149.1	1516.2	1192.8	904.7
Solomon Potvin Bengio	30	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02
		$\overline{\text{Gs}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.10
		$\overline{\text{Tm}}$(s)	0.16	0.05	0.07	0.06	0.07	0.12	0.17	0.18	0.22	0.30
		$\overline{\text{Ts}}$(s)	0.55	0.10	0.21	0.18	0.21	0.37	0.55	0.51	0.64	0.82
		#it/s	71178.1	41833.8	26144.3	16693.7	9829.3	5655.6	3396.2	1952.7	1313.7	806.8
Solomon Pesant	27	$\overline{\text{Gm}}$(%)	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01
		$\overline{\text{Gs}}$(%)	0.04	0.00	0.00	0.00	0.00	0.02	0.02	0.03	0.09	0.07
		$\overline{\text{Tm}}$(s)	0.16	0.09	0.08	0.06	0.07	0.07	0.13	0.16	0.10	0.24
		$\overline{\text{Ts}}$(s)	0.47	0.46	0.33	0.24	0.26	0.29	0.49	0.57	0.30	0.76
		#it/s	68108.6	39944.2	24597.9	16063.1	9290.4	5543.7	3456.6	2204.8	1607.1	1079.6
Gendreau Dumas Extended	130	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.04	0.01	0.01	0.00	0.00	0.00	0.00	0.01	0.01	0.01
		$\overline{\text{Ts}}$(s)	0.22	0.02	0.02	0.03	0.01	0.01	0.01	0.01	0.01	0.02
		#it/s	24590.5	14948.2	9237.7	6290.2	4063.0	2657.8	1710.8	1118.6	709.9	453.6
Ohlmann Thomas	25	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.19	0.01	0.01	0.01	0.01	0.02	0.02	0.03	0.05	0.07
		$\overline{\text{Ts}}$(s)	0.44	0.01	0.01	0.01	0.01	0.01	0.02	0.03	0.04	0.07
		#it/s	8637.1	5335.9	3547.6	2430.1	1574.5	1017.8	641.4	390.5	238.8	145.6

Table 3: TSPTW benchmarks: impact of $Wmax$ on the performance of ImaxLNS (max CPU time = 5 seconds per run, 5 runs per instance); the number of instances in each benchmark is given in column *inst*

Number of customers removed at each destroy phase. Figure 4 gives the mean number of customers removed at each destroy phase (y-axis) for each instance of each benchmark (one item on the x-axis per instance that is not solved during the preprocessing phase), when using $Wmax = 4$. These mean numbers, represented by

the blue rectangles, are averaged over all LNS iterations of all runs performed for the corresponding instance. The figure also shows the total number of customers in each instance (unfilled rectangles). For example, for the first instance of the SolomonPesant benchmark, 8 customers among 25 are reinserted at each LNS step on average. Globally, for the AFG and Dumas benchmarks, the mean number of customers removed at each step is quite high. For the SolomonPotvinBengio and SolomonPesant instances, from 4 to 16 customers are removed on average at each destroy phase. For the GendreauDumasExtended benchmark, more than 20 customers are sometimes removed and reinserted in a single iteration. For the OhlmannThomas benchmark, around 20 customers are removed on average at each step. Concerning these results, it is worth mentioning that the preprocessing phase sometimes detects that some customers can be harmlessly visited before the others (see Section 4). These customers are counted in the set of customers removed at each step, which is why some peeks appear in the histograms of Figure 4, such as for one instance of the OhlmannThomas benchmark where the preprocessing phase fixes the position of about 180 customers out of 200.

Comparison with the state-of-the-art. ImaxLNS with $Wmax = 4$ is compared with one of the best state-of-the-art methods, namely the recent GVNS algorithm of Amghar et al. [27] which is claimed to be robust and at least as good as the state-of-the-art incomplete search techniques for TSPTW-M. For the comparison, we used the raw results obtained by Amghar et al. on an Intel i7-3770 processor (3.40 GHz, 8 MB cache) and with a time limit of 24 seconds per run. These raw results include the gap statistics Gm and Gs for each instance over 5 runs, as well as the time statistics Tm and Ts given by Amghar et al. with a precision of one second. According to `https://www.cpubenchmark.net/compare`, for single-threaded computations, the performance of the processor they used for GVNS is approximately 1.5 times faster than the processor we use for ImaxLNS, however we do not apply any scaling and directly report their raw results.

The detailed comparison between ImaxLNS and GVNS for each instance of each benchmark is available in the supplementary material. Table 4 gives a global view of this comparison. It shows that for each benchmark, ImaxLNS gives both the smallest mean gaps and the smallest CPU times on average, and as shown in the supplementary material, this trend is valid at the level of each instance too. Moreover, contrarily to GVNS, ImaxLNS reaches the best-known solution at each run (see columns #Gm>0). It finds these best-known solutions with a mean computation time Tm equal to approximately 1 second in the worst case, while on some instances, GVNS requires a mean time greater than 10 seconds to find its best solution (see columns max(Tm)). Last, ImaxLNS solves several instances directly during the preprocessing phase (see column #spp), either because the insertion-width of the instance is low enough (less than 12 in our settings), or because the initial greedy search based on the tardiness-makespan-cost heuristic provides a solution whose makespan is equal to the makespan lower bound. For example, among the 30 instances of the SolomonPotvinBengio benchmark, 11 are solved during the preprocessing phase. ImaxLNS sometimes terminates during search, when a solution whose makespan is equal to the makespan lower bound is produced.

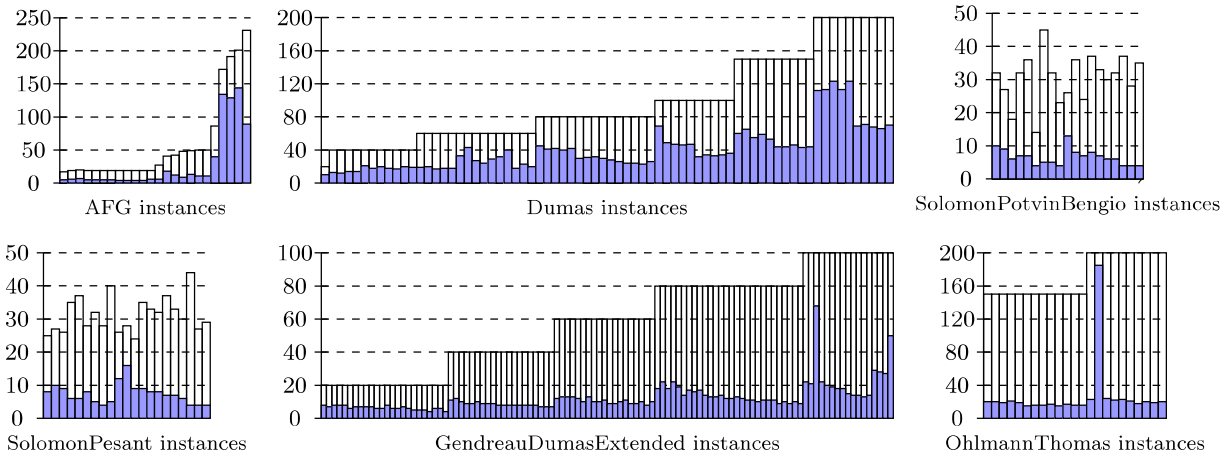

Figure 4: For standard TSPTW benchmarks and for ImaxLNS with $Wmax = 4$, mean number of customers removed at each destroy phase (y-axis) for each instance (x-axis) that is not directly solved during the preprocessing phase

benchmark	GVNS (24 sec. per run)					ImaxLNS (5 sec. per run, $Wmax = 4$)						
	#Gm>0	max(Gm)	$\overline{\text{Gm}}$	max(Tm)	$\overline{\text{Tm}}$	#nsr	#Gm>0	max(Gm)	$\overline{\text{Gm}}$	max(Tm)	$\overline{\text{Tm}}$	#spp
Langevin	**0**	**0.00**	**0.00**	**0**	**0.00**	**0**	**0**	**0.00**	**0.00**	**0.00**	**0.00**	70
AFG	0	0.00	0.00	11	0.24	**0**	**0**	**0.00**	**0.00**	**0.07**	**0.01**	26
Dumas	1	0.06	0.00	1	0.11	**0**	**0**	**0.00**	**0.00**	**0.01**	**0.00**	63
SolomonPotvinBengio	2	0.11	0.01	8	0.57	**0**	**0**	**0.00**	**0.00**	**0.60**	**0.06**	11
SolomonPesant	3	2.65	0.11	11	0.56	**0**	**0**	**0.00**	**0.00**	**1.20**	**0.06**	6
GendreauDumasExt.	5	0.23	0.01	2	0.31	**0**	**0**	**0.00**	**0.00**	**0.01**	**0.00**	22
OhlmannThomas	3	0.29	0.02	19	4.68	**0**	**0**	**0.00**	**0.00**	**0.02**	**0.01**	2

Table 4: Global comparison between GVNS and ImaxLNS on the TSPTW benchmarks (for ImaxLNS, max CPU time = 5 seconds per run, 5 runs per instance, and $Wmax = 4$)

8.3. Arigliano et al. [5] benchmark

Description. In this benchmark, a vehicle must visit locations in an environment decomposed into zones that have different congestion levels. The congestion level of each zone varies along time, which leads to time-dependent transitions between the locations. For this benchmark, we define functions $\overrightarrow{tt}$ and $\overleftarrow{tt}$ from a time-independent model using a maximum speed assumption among congestion zones. The instances differ in the number of locations $N \in \{15, 20, 30, 40\}$ to visit, the level of congestion, the traffic pattern, and the sizes of the time windows through a parameter $\beta \in \{0.00, 0.25, 0.50, 1.00\}$ (the higher β, the smaller the time windows; this parameter must not be mistaken for the notation β used before for partial solutions). In what follows, we do not consider the small instances involving $N = 15$ locations. We aggregate the results obtained for each of the remaining 12 combinations of N and β, as in the work of Lera-Romero et al. [35].

Impact of Wmax. To study the impact of $Wmax$, we consider the instances containing $N = 40$ locations, and for each value of β, we randomly select 60 instances among the 300 available ones. To be able to analyze

the gap for each value of $Wmax$, among these 60 instances for each β value, we only keep the instances that we identified as solved to optimality by Lera-Romero et al. [35]. Doing so, we end up with 45, 57, 60, and 60 instances for $\beta = 0.00, 0.25, 0.50, 1.00$ respectively. The mean gaps obtained for different values of $Wmax$ are given in Table 5, for a CPU time of 5 seconds per run. All runs of ImaxLNS manage to find a feasible solution for every instance selected, and all instances with $\beta = 1.00$ are solved during the preprocessing phase, their widths being less than 12 (between 4 and 9). For the other values of β, there is again a tradeoff to be made between choosing a small value for $Wmax$ to increase the number of LNS iterations per second, and a high value to get a larger neighborhood. In the following, we select value $Wmax = 4$. With this value, the mean gap after 5 seconds is less than 0.5% on average, and this gap is smaller for the high β values that reduce the size of the time windows. Note that the gaps are evaluated with a precision 10^{-2} on the makespan as the best values provided by Lera-Romero et al. [35] have that precision.

β	inst	metric	Wmax									
			1	2	3	4	5	6	7	8	9	10
0.00	300	$\overline{\text{Gm}}$(%)	1.67	0.72	0.49	0.44	0.49	0.59	0.81	1.17	1.82	2.63
		$\overline{\text{Gs}}$(%)	0.77	0.58	0.51	0.51	0.60	0.68	0.88	1.29	1.71	2.26
		$\overline{\text{Tm}}$(s)	2.41	2.34	2.29	2.43	2.50	2.51	2.59	2.71	2.99	3.42
		$\overline{\text{Ts}}$(s)	1.45	1.46	1.40	1.37	1.41	1.48	1.38	1.40	1.31	1.15
		#it/s	39249.7	20913.1	10726.6	5198.5	2373.0	1052.9	454.9	187.9	81.7	35.3
0.25	300	$\overline{\text{Gm}}$(%)	1.27	0.49	0.38	0.27	0.33	0.38	0.48	0.75	1.05	1.45
		$\overline{\text{Gs}}$(%)	1.00	0.58	0.71	0.45	0.64	0.85	0.83	1.10	1.38	1.61
		$\overline{\text{Tm}}$(s)	2.56	2.41	2.25	2.29	2.16	2.30	2.47	2.48	2.68	2.63
		$\overline{\text{Ts}}$(s)	1.48	1.40	1.44	1.41	1.42	1.42	1.50	1.50	1.47	1.34
		#it/s	32693.9	18360.5	10374.7	5580.4	2829.6	1389.6	663.5	308.3	146.0	68.9
0.50	300	$\overline{\text{Gm}}$(%)	0.41	0.12	0.07	0.05	0.04	0.04	0.07	0.11	0.17	0.25
		$\overline{\text{Gs}}$(%)	0.52	0.23	0.14	0.13	0.11	0.11	0.22	0.32	0.37	0.49
		$\overline{\text{Tm}}$(s)	2.34	1.57	1.39	1.37	1.51	1.41	1.72	1.84	1.96	2.28
		$\overline{\text{Ts}}$(s)	1.45	1.38	1.29	1.38	1.36	1.26	1.34	1.35	1.41	1.42
		#it/s	31277.0	17878.9	10399.8	5856.5	3182.3	1700.8	891.1	460.5	242.0	125.3
1.00	300	$\overline{\text{Gm}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Tm}}$(s)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Ts}}$(s)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		#it/s	-	-	-	-	-	-	-	-	-	-

Table 5: Arigliano et al. [5] benchmark: impact of $Wmax$ on the performance of ImaxLNS, for instances of size 40 and for different values of β (max CPU time = 5 seconds per run, 5 runs per instance)

Number of customers removed at each destroy phase. Figure 5 shows the number of customers removed at each destroy phase for all instances of size 40 selected for the preliminary experiments and that are not solved during the preprocessing phase. Globally, when using $Wmax = 4$, the mean number of customers removed is always equal to 4 for $\beta = 0.00$, between 4 and 7 for $\beta = 0.25$, and between 7 and 11 for $\beta = 0.50$.

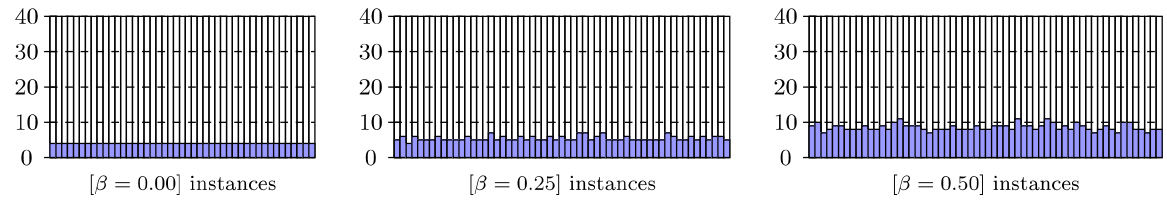

Figure 5: For the Arigliano et al. [5] benchmark and for ImaxLNS with $Wmax = 4$: mean number of customers removed at each destroy phase (y-axis) for each instance (x-axis) that is not directly solved during the preprocessing phase

Comparison with the state-of-the-art. Table 6 compares ImaxLNS using $Wmax = 4$ with the state-of-the-art exact dynamic programming algorithm recently proposed by Lera-Romero et al. [35] and referred to as DP-LR is the following. DP-LR was shown to outperform the previous approach by Arigliano et al. [5]. For DP-LR, column **opt** indicates the number of instances solved to optimality by Lera-Romero et al. on an Intel Core i7-8700 (3.20GHz, 12MB cache), with 32GB of RAM and a maximum CPU time of 1 hour. According to `https://www.cpubenchmark.net/compare`, for single-threaded computations, this processor is almost 1.5 faster than the processor we use for ImaxLNS, however we present the raw results of the two methods on the two different processors without any scaling. The best solutions obtained by DP-LR are retrieved from `https://github.com/gleraromero/tdtsptw/blob/master/tdtsptw-optima.ods` (version: August 2022) and the gaps presented in this section are computed relatively to the best solution found by DP-LR or ImaxLNS for each instance. As DP-LR performs a unique run for each instance, Table 6 indicates the number of instances for which DP-LR finds no solution (column #nsi), the number of instances for which its relative gap is strictly positive (column #G>0, which counts the number of instances for which DP-LR finds either no solution or a suboptimal solution), the maximum and average values of this gap over all instances for which a solution is found (max(G) and $\overline{\text{G}}$), and the total computation time required by DP-LR on average over the instances solved to optimality ($\overline{T^{opt}}$).

For ImaxLNS, the time limit is set to 1 minute per run. Table 6 shows that ImaxLNS leads to mean gap values that are small (less than 0.11% on average after one minute). It also shows that the number of instances for which none of the 5 runs of ImaxLNS finds the best solution is null for the instances of size $N \in \{20, 30\}$, and small for the instances of size 40 (see column #BG>0). Moreover, contrarily to DP-LR, all runs of ImaxLNS find a feasible solution (see column #nsr compared to column #nsi). Another result that is not explicitly mentioned in the table is that compared to DP-LR, ImaxLNS produces 50 new feasible solutions and 13 new best solutions for configuration $[N = 40, \beta = 0.00]$, with a quite large gap for the new best solutions (see column max(G) for DP-LR). These numbers are consistent with the fact that for this configuration, DP-LR solves to optimality all 300 instances but 63. Similarly, for configuration $[N = 40, \beta = 0.25]$, ImaxLNS produces 9 new feasible solutions and 4 new best solutions when compared to DP-LR, still with a quite large gap for the new best solutions. Again, these numbers are consistent with

the fact that for this configuration, DP-LR solves to optimality all 300 instances but 13. All new solution values found by ImaxLNS for these specific $63 + 13$ instances are available in the supplementary material. These new solution values have been obtained from 5 runs of 5 minutes of ImaxLNS, to try and improve the best values produced. One last comment is that the mean CPU time consumed by ImaxLNS to find its best solution is small on average (from a few seconds to 20 seconds), even on benchmarks where the average time required by DP-LR to solve the instances to optimality is high. To conclude, we can say that DP-LR is very good at solving numerous instances to optimality, while ImaxLNS always manages to produce optimal or near-optimal solutions very quickly and can be considered as very robust since it never fails to produce a feasible solution and never returns a solution having a very large gap, even after a few seconds of CPU time. The two approaches could be combined in the sense that ImaxLNS could provide good upper bounds that might help DP-LR. Finally, ImaxLNS has a very low memory consumption.

β	N	#inst	DP Lera-Romero et. al [35] (1 hour)						ImaxLNS (5 runs per instance, 1min per run, $Wmax = 4$)						
			opt	#nsi	#G>0	max(G)	$\overline{\text{G}}$	$\overline{\text{T}^{opt}}$	#nsr	#BG>0	max(Gm)	$\overline{\text{Gm}}$	max(Tm)	$\overline{\text{Tm}}$	#spp
0.00	20	300	300	0	0	0.00	0.00	45.64	0	0	0.00	0.00	3.97	0.13	0
	30	300	300	0	0	0.00	0.00	488.56	0	0	1.56	0.01	26.72	3.09	0
	40	300	237	50	63	134.75	5.48	2119.87	0	62	2.12	0.11	51.42	20.30	0
0.25	20	300	300	0	0	0.00	0.00	20.92	0	0	0.00	0.00	7.50	0.11	0
	30	300	300	0	0	0.00	0.00	299.96	0	0	0.55	0.00	30.56	2.37	0
	40	300	287	9	13	91.23	1.15	1654.10	0	41	0.94	0.05	47.20	14.73	0
0.50	20	300	300	0	0	0.00	0.00	6.32	0	0	0.00	0.00	0.28	0.03	0
	30	300	300	0	0	0.00	0.00	115.67	0	0	0.05	0.00	22.29	0.47	0
	40	300	300	0	0	0.00	0.00	619.69	0	11	0.96	0.01	45.63	5.40	0
1.00	20	300	300	0	0	0.00	0.00	0.01	0	0	0.00	0.00	0.00	0.00	300
	30	300	300	0	0	0.00	0.00	0.01	0	0	0.00	0.00	0.00	0.00	300
	40	300	300	0	0	0.00	0.00	0.10	0	0	0.00	0.00	0.00	0.00	300

Table 6: Global comparison between the dynamic programmaing algorithm of Lera-Romero et al. [35] and ImaxLNS on the Arigliano et al. [5] benchmark (configuration: $Wmax = 4$, maximum CPU time = 1 minute per run, 5 runs per instance)

8.4. Aguiar-Melgarejo et al. [6] benchmark

Description. This benchmark is related to an urban delivery problem [6]. The corresponding instances can be found at `http://perso.citi-lab.fr/csolnon/TDTSP.html`. They contain 10, 20, 30, 50, or 100 customers. We do not consider instances of sizes 10 and 20 in the following. As this benchmark considers an urban environment where traffic conditions vary over the day, the travel time between two customers is time-dependent. In the benchmark, time is discretized using a 6-minute time-step and a travel time matrix specifies the transition duration required between any pair of customers at each time-step. This travel time matrix does not necessarily satisfy the FIFO assumption, but it can be easily transformed to recover a FIFO travel time function [6]. For the experiments, we use $\overrightarrow{tt} = tt$ even if the triangular inequality is sometimes violated by tt, and we derive $\overleftarrow{tt}$ from the direct transition times specified by tt. In the benchmark, there are

3 possible distance matrices referred to as M00, M10, and M20 and for each problem size in $\{30, 50, 100\}$, there are 20 customer selection scenarios, which leads to 60 instances per problem size. One last remark is that as customer 0 is contained in the set of customers for each instance, we actually have $N \in \{29, 49, 99\}$.

The instances can contain precedence constraints, and one difficulty is that several time windows are sometimes available for visiting a given customer. To overcome this difficulty, the presence of several time windows $[Start(i,k), End(i,k)]$ for a given customer i is managed through the transition time function. More precisely, if the transition time from customer j to customer i equals x when starting the transition at time τ and if $\tau + x$ is located strictly between two windows $[Start(i,k), End(i,k)]$ and $[Start(i,k+1), End(i,k+1)]$, then we use $tt(i,j,\tau) = Start(i,k+1) - \tau$ to express that the vehicle must wait for the beginning of time window $[Start(i,k+1), End(i,k+1)]$ in this case. In any other case, we use $tt(i,j,\tau) = x$.

Impact of Wmax. Table 7 gives results obtained with a maximum CPU time of 5 seconds per run. In this table, the gap values are computed relatively to the best solutions found by ImaxLNS over all our experiments. These values are always at least as good as the best solution values provided on the benchmark website. As smaller average gaps are obtained with $Wmax = 5$, we choose this value for the rest of the experiments. Globally, this value is a good trade-off between the speed of each local move and the size of the neighborhood explored, whereas value $Wmax = 10$ leads to a very small number of iterations per second, and value $Wmax = 1$ sometimes has difficulty to find feasible solutions due to its restricted neighborhood. We also observed that with a CPU time of 5 seconds per run, the best solution found by ImaxLNS with $Wmax = 5$ over each instance is always at least as good as the best-known solution. Note that none of the instances is solved during the preprocessing phase.

size	inst	metric	Wmax									
			1	2	3	4	5	6	7	8	9	10
30	60	$\overline{\text{Gm}}$(%)	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		$\overline{\text{Gs}}$(%)	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.03
		$\overline{\text{Tm}}$(s)	0.47	0.22	0.10	0.11	0.12	0.12	0.17	0.24	0.30	0.36
		$\overline{\text{Ts}}$(s)	0.84	0.52	0.27	0.29	0.40	0.38	0.36	0.48	0.59	0.62
		#it/s	73030.2	41538.0	24542.3	13855.2	7537.9	3973.5	2044.6	1058.3	545.3	281.3
50	60	$\overline{\text{Gm}}$(%)	0.47	0.19	0.14	0.14	0.11	0.14	0.15	0.23	0.31	0.36
		$\overline{\text{Gs}}$(%)	0.77	0.46	0.38	0.42	0.36	0.43	0.43	0.58	0.80	0.73
		$\overline{\text{Tm}}$(s)	1.46	1.14	1.01	0.84	0.97	0.89	0.99	1.21	1.18	1.27
		$\overline{\text{Ts}}$(s)	1.42	1.35	1.33	1.19	1.27	1.20	1.25	1.38	1.39	1.30
		#it/s	39290.9	22915.0	13966.3	8058.5	4458.8	2234.1	1200.5	590.6	295.5	146.7
100	60	$\overline{\text{Gm}}$(%)	6.70	3.53	2.79	2.40	2.25	2.42	2.60	2.60	3.30	3.70
		$\overline{\text{Gs}}$(%)	3.84	2.31	1.97	1.74	1.81	1.85	2.19	2.12	3.05	3.02
		$\overline{\text{Tm}}$(s)	1.97	2.01	2.07	2.18	1.85	2.09	2.18	2.43	3.06	3.77
		$\overline{\text{Ts}}$(s)	1.65	1.57	1.57	1.62	1.56	1.49	1.43	1.32	1.39	1.10
		#it/s	18360.7	10103.7	6111.6	3674.5	2005.0	1132.1	595.0	322.4	166.3	85.5

Table 7: Aguiar-Melgarejo et al. [6] benchmark: impact of parameter *Wmax* on the performance of ImaxLNS (max CPU time = 5 seconds per run, 5 runs per instance)

Number of customer removals. Figure 6 shows the mean number of customers removed at each destroy phase. For each problem size, the x-axis contains one item for each of the 60 problem instances, and each blue rectangle represents the mean number of customers removed at each step when using $Wmax = 5$. This number is between 5 and 7 for the instances of size 30, between 6 and 8 for the instances of size 50, and between 7 and 8 for the instances of size 100. These results are consistent with the fact that many customers have large time windows in this benchmark.

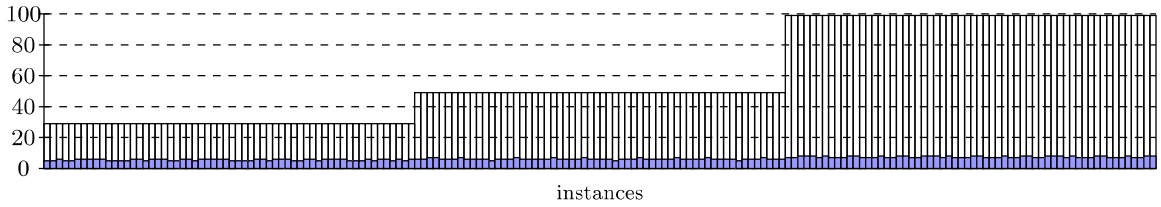

Figure 6: Aguiar-Melgarejo et al. [6] benchmark: for ImaxLNS with $Wmax = 5$, mean number of customers removed at each destroy phase (y-axis) for each instance (x-axis) of size 30, 50, or 100 (5 seconds per run, 5 runs per instance)

Comparison with the state-of-the-art. The solutions found by ImaxLNS in 5 runs of 2 minutes are compared with the best solutions provided on the website of the benchmark and obtained by Aguiar-Melgarejo et al. on processors Intel Xeon X5570 (2.93GHz, 8MB cache), with 20GB of RAM and a maximum CPU time of 2 hours ([6], page 93). Their solving technique, referred to as *CP-tdNoOverlap* in the following, uses an implementation of a specific time-dependent no-overlap constraint in the IBM ILOG CpOptimizer constraint programming engine. The detailed results for all the instances are provided in the supplementary material. Table 8 gives a global view of these results. It shows the number of instances for which no solution is found by CP-tdNoOverlap in 2 hours (column #nsi), as well as statistics concerning the gap G in percent with regards to the best solutions found during our experiments on ImaxLNS. These statistics include the number of instances for which the gap is positive (#G>0), the maximum gap value over all the instances (max(G)), and the average gap value over all the instances ($\overline{G}$). Several comments can be made on these results. First, ImaxLNS always manages to find a feasible solution while CP-tdNoOverlap has issues with 6 of the 60 instances of size 100. Second, with $Wmax = 5$, ImaxLNS reaches the best solutions many times and leads to very small gaps for each instance, whereas this gap is sometimes greater than 18% for CP-tdNoOverlap. Third, the instances of size 100 are much harder than the instances of size 30 or 50 given the dispersion of the CPU time required to get the best solution. In the end, for this benchmark, the 2-minute runs of ImaxLNS provide 6 new best solutions over the 60 instances of size 30, 29 new best solutions over the 60 instances of size 50, and 59 new feasible or best solutions over the 60 instances of size 100. This allows us to conclude that from the point of view of the quality of the solutions produced, ImaxLNS outperforms CP-tdNoOverlap, even if CP-tdNoOverlap can prove solution optimality for the small instances.

size	matrix	inst	CP-tdNoOverlap (2 hours)				ImaxLNS (5 sec. per run, $Wmax = 4$)						
			#nsi	#G>0	max(G)	$\overline{\text{G}}$	#nsr	#BG>0	max(Gm)	$\overline{\text{Gm}}$	max(Tm)	$\overline{\text{Tm}}$	#spp
30	M00	20	0	3	1.99	0.23	0	0	0.00	0.00	2.79	0.25	0
	M10	20	0	2	2.13	0.18	0	0	0.00	0.00	0.24	0.06	0
	M20	20	0	1	0.07	0.00	0	0	0.00	0.00	0.20	0.06	0
50	M00	20	0	8	2.04	0.31	0	0	0.13	0.01	58.53	6.70	0
	M10	20	0	9	5.59	0.64	0	0	0.14	0.01	62.58	5.77	0
	M20	20	0	12	7.95	1.05	0	0	1.17	0.06	82.54	8.59	0
100	M00	20	4	20	15.17	5.18	0	15	1.82	0.46	97.11	53.96	0
	M10	20	1	19	18.81	4.47	0	15	1.82	0.63	92.80	55.80	0
	M20	20	1	20	18.97	5.87	0	17	2.08	0.92	80.22	52.79	0

Table 8: Global comparison between CP-tdNoOverlap and ImaxLNS on the TDTSPTW benchmark defined by Aguiar-Melgarejo et al. [6], (configuration: $Wmax = 5$, maximum CPU time = 2 minutes per run, 5 runs per instance)

8.5. Earth observing satellite benchmark

Description. In this last benchmark, a low Earth orbit satellite moving around the Earth must take pictures of targets located at the Earth surface (see Figure 7). To observe a target, the satellite must be pointed to the corresponding ground area. The relationship with TDTSPTW is that each target i can be seen as a customer that can be visited only when the satellite overflies i, and the transition time between two customer visits corresponds to the time required by the satellite to move from one target pointing to the next [42, 43]. This transition is time-dependent because from the satellite point of view, the position of a given ground target varies along time, since the satellite is moving on its orbit around the Earth and the ground targets are moving due to the rotation of the Earth on itself. In the instances generated,[1] we consider three different satellite altitudes (500km, 700km, 800km). This leads to time windows for observing the targets that are more or less tight (the higher the altitude, the larger the time windows). The size of each time window also depends on a maximum observation angle $\alpha_i \in \{15°, 30°, 45°\}$ associated with each target i. From a general point of view, the time-dependent transition function between observations is not linear and requires calls to an external space mechatronics library. Such calls are fully compatible with ImaxLNS, but to define public instances without providing this external library, we precompute the transition time from each target i to each target j every 15 seconds and use a piecewise linear interpolation from the points obtained.

For this benchmark, the goal is not to minimize the makespan but to find a feasible sequence of visits of the targets. To evaluate ImaxLNS, hard instances are generated as follows. We consider one pass of the satellite over an area containing 100 targets located between 20 and 60 degrees of latitude. These targets are randomly ordered and added one by one to the current plan using ImaxLNS with a maximum CPU time of 10 seconds *per target insertion*. If the insertion of target i succeeds (possibility to get a feasible solution), then i is kept in the current plan, otherwise it is discarded. At the end, we obtain a set of selected targets

[1] The instances will be made public in case of acceptance.

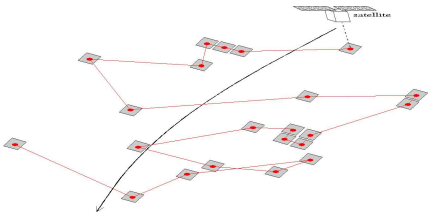

Figure 7: Satellite benchmark: example of successive images performed over a set of targets (depicted in grey)

which are seen as the customers of a TDTSPTW, and by construction the latter admits a feasible solution. Using this process, we generated 36 instances for each of the 3 satellite altitudes. Each of these 108 instances is then solved by ImaxLNS using a lower computation time (5 seconds max per run). For the experiments, we use $\overrightarrow{tt} = tt$, even if the transition time function may violate the triangular inequality.

Impact of Wmax. Table 9 details the impact of parameter *Wmax* on the efficiency of ImaxLNS. The number of runs for which no feasible solution is found (lines #nsr) shows that setting $Wmax = 5$ is a good trade-off. With this value, the mean time required to get a feasible solution is small on average: 0.03 second for the 500km instances, 0.52 second for the 700km instances, and 0.59 second for the 800km instances. Moreover, as shown in the supplementary material, the mean computation time over 5 runs is always less than 0.37 second for each 500km instance, 3 seconds for each 700km instance, and 3.5 seconds for each 800km instance. The 800km instances are the hardest ones because at this altitude, the size of the largest time windows is between 4 and 5 minutes, whereas for the 500km instances, it is between 2 and 3 minutes and the insertion-width is smaller. During the experiments, no instance was solved by the preprocessing phase of ImaxLNS.

altitude	inst	metric	Wmax									
			1	2	3	4	5	6	7	8	9	10
500km	36	#nsr	15/180	0/180	0/180	0/180	0/180	0/180	0/180	0/180	0/180	0/180
		$\overline{\text{Tm}}$(s)	0.48	0.25	0.08	0.03	0.03	0.02	0.02	0.03	0.03	0.03
		$\overline{\text{Ts}}$(s)	0.82	0.64	0.26	0.08	0.08	0.03	0.03	0.04	0.04	0.05
		#it/s	32666.5	14446.9	8767.1	5427.7	3165.7	2112.6	1099.9	717.9	520.6	398.6
700km	36	#nsr	70/180	24/180	10/180	3/180	2/180	3/180	9/180	9/180	23/180	19/180
		$\overline{\text{Tm}}$(s)	1.09	0.83	0.50	0.53	0.52	0.56	0.62	0.72	0.84	0.99
		$\overline{\text{Ts}}$(s)	1.02	1.08	0.79	0.75	0.83	0.87	0.85	0.89	1.02	1.14
		#it/s	53296.5	18904.2	9048.2	4447.9	2381.0	1334.0	845.6	503.6	396.5	214.0
800km	36	#nsr	60/180	25/180	16/180	11/180	3/180	5/180	2/180	3/180	7/180	17/180
		$\overline{\text{Tm}}$(s)	0.82	0.69	0.58	0.58	0.59	0.49	0.58	0.66	0.68	0.82
		$\overline{\text{Ts}}$(s)	0.85	1.04	0.98	0.98	0.97	0.79	0.94	0.94	0.90	0.99
		#it/s	57433.5	20627.0	10792.7	5399.4	2549.5	1443.1	752.4	431.3	279.6	204.8

Table 9: Satellite benchmark: ImaxLNS results for maximum insertion-width *Wmax* in interval [1..10] and for three different altitudes (maximum CPU time = 5 seconds per run, 5 runs per instance)

Number of customer removals. Figure 8 shows that using maximum LNS instead of LNS with a fixed number of customer removals pays off for this benchmark. Indeed, with $Wmax = 5$, the average number of customers

removed at each destroy step is between 17 and 45 for the 500km instances, between 12 and 37 for the 700km instances, and between 9 and 34 for the 800km instances. No state-of-the-art result is available on this new satellite benchmark, hence we do not include a comparison between ImaxLNS and another method.

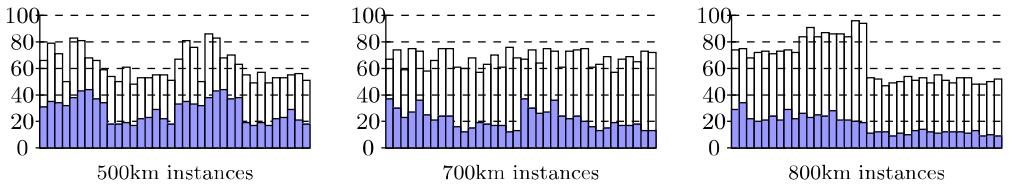

Figure 8: Satellite benchmark: for ImaxLNS with $Wmax = 5$, mean number of customers removed at each destroy phase (y-axis) for each instance (x-axis), for the three different satellite altitudes (5 seconds per run, 5 runs per instance)

9. Conclusion and perspectives

This article introduced ImaxLNS, a new algorithm for finding feasible solutions or minimizing the makespan for TSPTW and TDTSPTW, even if the transition time function is non-linear. This algorithm uses iterative destroy and repair operations over a current sequence of visits, together with perturbations and restarts to diversify search. As detailed in the pseudo-codes, the destroy phase removes customers from the current sequence of visits as long as a parameter called the insertion-width is less than a value $Wmax$, and the repair phase uses dynamic programming to explore in linear time the possible reinsertions of the customers removed. A specific effort was also put on algorithmic optimizations (incremental management of the precedence graph, compact visit states, local bitset representations containing $Wmax$ bits, etc.), and this effort allowed us to increase the number of destroy-repair operations performed per second, in conjunction with the intrinsic capabilities offered by the neighborhood proposed. Several properties of ImaxLNS were established, including makespan-optimality guarantees for the repair method and polynomial complexity results for TDTSPTWs having a bounded insertion-width. The experiments performed on seven TSPTW benchmarks covering 467 instances and three TDTSPTW benchmarks covering $3600 + 180 + 76$ instances showed the efficiency and robustness of ImaxLNS, which managed first to produce new feasible solutions and best solutions, and second to reproduce many best-known solutions within short computation times.

Several perspectives could be explored. First, other objective functions could be studied, like the duration objective. Second, the approach could be tested on TDTSPs, without time window constraints. In this case, the destroy phase would be simpler since it would suffice to remove $Wmax$ customers at each step. Third, the repair phase of ImaxLNS could compute additional precedence constraints and rebuild a full solution around a partial solution defined by a longest path in the customer precedence graph, as in the ngL-tour relaxation [15]. Fourth, Adaptive LNS could be tested to automatically adapt parameters $Wmax$, $Rmin$, and $Kmax$ during search. Last, ImaxLNS could produce good upper bounds for complete search methods.

Acknowledgement

This work has been performed with the support of the French government, in the context of the BPI PSPC project "LiChIE" of the "Programme d'Invertissements d'Avenir".

References

[1] M. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1985) 285–305.

[2] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: 4th International Conference on Principles and Practice of Constraint Programming, 1998, pp. 417–431.

[3] D. Pisinger, S. Ropke, Large neighborhood search, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, 2019, pp. 99–127.

[4] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search: Framework and applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, edition 3, Springer, 2019, pp. 129–168.

[5] A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, I. Plana, Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm, Discrete Applied Mathematics 261 (2019) 28–39.

[6] P. Aguiar-Melgarejo, P. Laborie, C. Solnon, A time-dependent no-overlap constraint: Application to delivery problems, in: 12th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, 2015, pp. 1–17.

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

[8] E. Baker, An exact algorithm for the time-constrained traveling salesman problem, Operations Research 31 (5) (1983) 938–945.

[9] N. Ascheuer, M. Fischetti, M. Grötschel, Solving asymmetric travelling salesman problem with time windows by branch-and-cut, Mathematical Programming 90 (2001) 475–506.

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

[11] S. Dash, O. Gunluk, A. Lodi, A. Tramontani, A time bucket formulation for the traveling salesman problem with time windows, INFORMS Journal on Computing 24 (2012) 132–147.

[12] N. Boland, M. Hewitt, D. M. Vu, , M. Savelsbergh, Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks, in: 14th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, 2017, pp. 254–262.

[13] Y. Dumas, J. Desrosiers, E. Gelinas, M. M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, Operations Research 43 (2) (1995) 367–371.

[14] A. Mingozzi, L. Bianco, S. Ricciardelli, Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints, Operations Research 45 (3) (1997) 365–377.

[15] R. Baldacci, A. Mingozzi, R. Roberti, New state-space relaxations for solving the traveling salesman problem with time windows, INFORMS Journal on Computing 24 (3) (2012) 356–371.

[16] G. Pesant, M. Gendreau, J.-Y. Potvin, J.-M. Rousseau, An exact constraint logic programming algorithm for the traveling salesman problem with time windows, Transportation Science 32 (1998) 12–29.

[17] F. Focacci, A. Lodi, M. Milano, A hybrid exact algorithm for the TSPTW, INFORMS Journal on Computing 14 (4) (2002) 403–417.

[18] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, Operations Research 46 (1998) 330–335.

[19] R. W. Calvo, A new heuristic for the traveling salesman problem with time windows, Transportation Science 34 (1) (2000) 113–124.

[20] W. B. Carlton, J. W. Barnes, Solving the traveling-salesman problem with time windows using tabu search, IIE Transactions 28 (8) (1996) 617–629.

[21] J. W. Ohlmann, B. W. Thomas, A compressed-annealing heuristic for the traveling salesman problem with time windows, INFORMS Journal on Computing 19 (1) (2007) 80–90.

[22] D. Favaretto, E. Moretti, P. Pellegrini, An ant colony system approach for variants of the traveling salesman problem with time windows, Journal of Information and Optimization Sciences 27 (1) (2006) 35–54.

[23] M. López-Ibáñez, C. Blum, Beam-ACO for the travelling salesman problem with time windows, Computers & Operations Research 37 (9) (2010) 1570–1583.

[24] M. López-Ibáñez, C. Blum, J. W. Ohlmann, B. W. Thomas, The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization, Applied Soft Computing 13 (9) (2013) 3806–3815.

[25] R. F. da Silva, S. Urrutia, A General VNS heuristic for the traveling salesman problem with time windows, Discrete Optimization 7 (4) (2010) 203–211.

[26] N. Mladenovic, R. Todosijević, D. Urosevic, An efficient general variable neighborhood search for large travelling salesman problem with time windows, Yugoslav Journal of Operations Research (YUJOR) 23 (1) (2013) 19–30.

[27] K. Amghar, J.-F. Cordeau, B. Gendron, A general variable neighborhood search heuristic for the traveling salesman problem with time windows under completion time minimization, Tech. rep., CIRRELT-2019-29 (2019).

[28] E. Balas, N. Simonetti, Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study, INFORMS Journal on Computing 13 (1) (2001) 56–75.

[29] C. Malandraki, M. S. Daskin, Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms, Transportation Science 26 (1992) 185–200.

[30] J. Albiach, J. M. Sanchis, D. Soler, An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation, European Journal of Operational Research 189 (3) (2008) 789–802.

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

[32] A. Montero, I. Méndez-Díaz, J. J. Miranda-Bront, An integer programming approach for the time-dependent traveling salesman problem with time windows, Computers & Operations Research 88 (2017) 280–289.

[33] P. Sun, S. Dabia, L. Veelenturf, T. Van Woensel, The time-dependent profitable pickup and delivery traveling salesman problem with time windows, Tech. rep., Eindhoven University of Technology (2015).

[34] D. M. Vu, M. Hewitt, N. Boland, M. Savelsbergh, Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows, Transportation Science 54 (3) (2020) 703–720.

[35] G. Lera-Romero, J. J. Miranda-Bront, F. J. Soulignac, Dynamic programming for the time-dependent traveling salesman problem with time windows, URL `https://optimization-online.org/2020/01/7558/` (2020).

[36] S. Dabia, S. Röpke, T. Van Woensel, T. de Kok, Branch and price for the time-dependent vehicle routing problem with time windows, Transportation Science 47 (3) (2013) 380–396.

[37] P. Sun, L. P. Veelenturf, M. Hewitt, T. Van Woensel, Adaptive large neighborhood search for the time-dependent profitable pickup and delivery problem with time windows, Transportation Research Part E: Logistics and Transportation Review 138 (2020) 101942.

[38] M. Held, R. M. Karp, A dynamic programming approach to sequencing problems, Journal of the Society for Industrial and Applied Mathematics 10 (1) (1962) 196–210.

[39] R. Bellman, Dynamic programming treatment of the travelling salesman problem, Journal of the ACM 9 (1) (1962) 61–63.

[40] N. Ascheuer, Hamiltonian path problems in the on-line optimization of flexible manufacturing systems, Ph.D. thesis, Technische Universität Berlin, Berlin, Germany (1995).

[41] J.-Y. Potvin, S. Bengio, The vehicle routing problem with time windows part II: genetic search, INFORMS Journal on Computing 8 (1996) 165–172.

[42] X. Liu, G. Laporte, Y. Chen, R. He, An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time, Computers & Operations Research 86 (2017) 41–53.

[43] G. Peng, R. Dewil, C. Verbeeck, A. Gunawan, L. Xing, P. Vansteenwegen, Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times, Computers & Operations Research 111 (2019) 84–98.

ImaxLNS for TSPTW and TDTSPTW: supplementary material

C. Pralet

1 Experiments: detailed results

1.1 TSPTW benchmarks

Remarks:

- The semantics of columns **N**, **Gm**, **Gs**, **Tm**, and **Ts** is given in the main article.
- Column **BF** gives the value of the best solution found by each method over 5 runs.
- Column **W** corresponds to the insertion-width of each instance.
- The results of GVNS are directly taken from the article of Amghar et al. [1]. In these results, the best-known solution was improved for some instances, hence Amghar et al. provided some results with a negative gap. For these instances, as ImaxLNS finds the same new solution values, we do not mention the gap and standard deviation for GVNS in this case (cells filled with "-", e.g. for instance rc202.4 in the SolomonPotvinBengio benchmark). Note that the GVNS results were obtained by Amghar et al. on another processor. For single-threaded computations, the processor used for GVNS is approximately 1.5 times faster than the processor used for ImaxLNS.
- The termination of ImaxLNS is indicated by superscript ^{t} in column BF. Termination occurs for instances whose insertion-width is low and for instances where ImaxLNS manages to produce a solution whose makespan is equal to the makespan lower bound.

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
n20w20	20	**370.4**	**370.4**	0.00	0.00	0	0	^{t}**370.4**	0.00	0.00	0.00	0.00	2.6
n20w40	20	**342.8**	**342.8**	0.00	0.00	0	0	^{t}**342.8**	0.00	0.00	0.00	0.00	2.6
n20w60	20	**362.0**	**362.0**	0.00	0.00	0	0	^{t}**362.0**	0.00	0.00	0.00	0.00	3.8
n20w80	20	**363.4**	**363.4**	0.00	0.00	0	0	^{t}**363.4**	0.00	0.00	0.00	0.00	7.0
n20w100	20	**331.6**	**331.6**	0.00	0.00	0	0	**331.6**	0.00	0.00	0.00	0.00	8.8
n40w20	40	**521.2**	**521.2**	0.00	0.00	0	0	^{t}**521.2**	0.00	0.00	0.00	0.00	3.4
n40w40	40	**512.2**	**512.2**	0.00	0.00	0	0	^{t}**512.2**	0.00	0.00	0.00	0.00	6.2
n40w60	40	**481.4**	**481.4**	0.00	0.00	0	0	**481.4**	0.00	0.00	0.00	0.00	13.2
n40w80	40	**486.6**	**486.6**	0.00	0.00	0	0	**486.6**	0.00	0.00	0.00	0.00	13.6
n40w100	40	**463.0**	**463.0**	0.00	0.00	0	0	**463.0**	0.00	0.00	0.00	0.00	16.4
n60w20	60	**626.8**	**626.8**	0.00	0.00	0	0	^{t}**626.8**	0.00	0.00	0.00	0.00	6.8
n60w40	60	**654.4**	**654.4**	0.00	0.00	0	0	^{t}**654.4**	0.00	0.00	0.00	0.00	10.2
n60w60	60	**672.8**	**672.8**	0.00	0.00	0	0	**672.8**	0.00	0.00	0.00	0.00	16.4
n60w80	60	**628.2**	**628.2**	0.00	0.00	0	0	**628.2**	0.00	0.00	0.00	0.00	16.6
n60w100	60	**620.2**	620.6	0.06	0.00	0	0	**620.2**	0.00	0.00	0.00	0.00	25.8
n80w20	80	**748.2**	**748.2**	0.00	0.00	0	0	^{t}**748.2**	0.00	0.00	0.00	0.00	5.6
n80w40	80	**725.6**	**725.6**	-	-	0	0	**725.6**	0.00	0.00	0.00	0.00	14.0
n80w60	80	**712.6**	**712.6**	0.00	0.00	0	0	**712.6**	0.00	0.00	0.00	0.00	19.0
n80w80	80	**714.6**	**714.6**	-	-	0	0	**714.6**	0.00	0.00	0.00	0.00	25.6
n100w20	100	**823.0**	**823.0**	0.00	0.00	0	0	^{t}**823.0**	0.00	0.00	0.00	0.00	9.4
n100w40	100	**821.0**	**821.0**	0.00	0.00	0	0	**821.0**	0.00	0.00	0.00	0.00	15.6
n100w60	100	**817.2**	**817.2**	0.00	0.00	0	0	**817.2**	0.00	0.00	0.01	0.01	22.0
n150w20	150	**978.4**	**978.4**	0.00	0.00	0	0	^{t}**978.4**	0.00	0.00	0.00	0.00	9.8
n150w40	150	**990.4**	**990.4**	0.00	0.00	0	0	^{t}**990.4**	0.00	0.00	0.01	0.00	20.8
n150w60	150	**988.6**	**988.6**	0.00	0.00	1	0	**988.6**	0.00	0.00	0.01	0.00	27.2
n200w20	200	**1137.8**	**1137.8**	0.00	0.00	1	1	**1137.8**	0.00	0.00	0.00	0.00	13.6
n200w40	200	**1156.0**	**1156.0**	0.00	0.00	1	1	**1156.0**	0.00	0.00	0.01	0.01	23.0

Table 1: Results for the Dumas instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
rbg010a	10	**3840**	**3840**	0.00	0.00	0	0	^{t}**3840**	0.00	0.00	0.00	0.00	1
rbg016a	16	**2596**	**2596**	0.00	0.00	0	0	^{t}**2596**	0.00	0.00	0.00	0.00	3
rbg016b	16	**2094**	**2094**	0.00	0.00	0	0	^{t}**2094**	0.00	0.00	0.00	0.00	4
rbg017.2	15	**2351**	**2351**	0.00	0.00	0	0	^{t}**2351**	0.00	0.00	0.00	0.00	5
rbg017a	17	**4296**	**4296**	0.00	0.00	0	0	**4296**	0.00	0.00	0.00	0.00	16
rbg017	15	**2351**	**2351**	0.00	0.00	0	0	^{t}**2351**	0.00	0.00	0.00	0.00	2
rbg019a	19	**2694**	**2694**	0.00	0.00	0	0	^{t}**2694**	0.00	0.00	0.00	0.00	3
rbg019b	19	**3840**	**3840**	0.00	0.00	0	0	^{t}**3840**	0.00	0.00	0.00	0.00	4
rbg019c	19	**4536**	**4536**	0.00	0.00	0	0	**4536**	0.00	0.00	0.00	0.00	18
rbg019d	19	**3479**	**3479**	0.00	0.00	0	0	^{t}**3479**	0.00	0.00	0.00	0.00	5
rbg020a	20	**4689**	**4689**	0.00	0.00	0	0	**4689**	0.00	0.00	0.00	0.00	18
rbg021.2	19	**4528**	**4528**	0.00	0.00	0	0	**4528**	0.00	0.00	0.00	0.00	18
rbg021.3	19	**4528**	**4528**	0.00	0.00	0	0	**4528**	0.00	0.00	0.02	0.01	18
rbg021.4	19	**4525**	**4525**	0.00	0.00	0	0	**4525**	0.00	0.00	0.01	0.01	18
rbg021.5	19	**4516**	**4516**	0.00	0.00	0	0	**4516**	0.00	0.00	0.01	0.01	18
rbg021.6	19	**4492**	**4492**	0.00	0.00	0	0	**4492**	0.00	0.00	0.04	0.03	19
rbg021.7	19	**4481**	**4481**	0.00	0.00	0	0	**4481**	0.00	0.00	0.03	0.02	19
rbg021.8	19	**4481**	**4481**	0.00	0.00	0	0	**4481**	0.00	0.00	0.03	0.03	19
rbg021.9	19	**4481**	**4481**	0.00	0.00	0	0	**4481**	0.00	0.00	0.03	0.03	19
rbg021	19	**4536**	**4536**	0.00	0.00	0	0	**4536**	0.00	0.00	0.00	0.00	18
rbg027a	27	**5093**	**5093**	0.00	0.00	0	0	**5093**	0.00	0.00	0.01	0.01	26
rbg031a	31	**3498**	**3498**	0.00	0.00	0	0	^{t}**3498**	0.00	0.00	0.00	0.00	3
rbg033a	33	**3757**	**3757**	0.00	0.00	0	0	^{t}**3757**	0.00	0.00	0.00	0.00	5
rbg034a	34	**3314**	**3314**	0.00	0.00	0	0	^{t}**3314**	0.00	0.00	0.00	0.00	5
rbg035a.2	35	**3325**	**3325**	0.00	0.00	0	0	^{t}**3325**	0.00	0.00	0.00	0.00	4
rbg035a	35	**3388**	**3388**	0.00	0.00	0	0	^{t}**3388**	0.00	0.00	0.00	0.00	8
rbg038a	38	**5699**	**5699**	0.00	0.00	0	0	^{t}**5699**	0.00	0.00	0.00	0.00	8
rbg040a	40	**5679**	**5679**	0.00	0.00	0	0	^{t}**5679**	0.00	0.00	0.00	0.00	4
rbg041a	41	**3793**	**3793**	0.00	0.00	0	0	^{t}**3793**	0.00	0.00	0.00	0.00	15
rbg042a	42	**3260**	**3260**	0.00	0.00	1	1	**3260**	0.00	0.00	0.01	0.01	20
rbg048a	48	**9799**	**9799**	0.00	0.00	0	0	^{t}**9799**	0.00	0.00	0.00	0.00	37
rbg049a	49	**13257**	**13257**	0.00	0.00	0	0	^{t}**13257**	0.00	0.00	0.00	0.00	31
rbg050a	50	**12050**	**12050**	0.00	0.00	0	0	^{t}**12050**	0.00	0.00	0.00	0.00	4
rbg050b	50	**11957**	**11957**	0.00	0.00	0	0	**11957**	0.00	0.00	0.00	0.01	32
rbg050c	50	**10985**	**10985**	0.00	0.00	0	0	**10985**	0.00	0.00	0.00	0.00	40
rbg055a	55	**6929**	**6929**	0.00	0.00	0	0	^{t}**6929**	0.00	0.00	0.00	0.00	5
rbg067a	67	**10331**	**10331**	0.00	0.00	0	0	^{t}**10331**	0.00	0.00	0.00	0.00	5
rbg086a	86	**16899**	**16899**	0.00	0.00	0	0	^{t}**16899**	0.00	0.00	0.00	0.00	15
rbg092a	92	**12501**	**12501**	0.00	0.00	0	0	^{t}**12501**	0.00	0.00	0.00	0.00	3
rbg125a	125	**14214**	**14214**	0.00	0.00	0	0	^{t}**14214**	0.00	0.00	0.00	0.00	8
rbg132.2	130	**18524**	**18524**	0.00	0.00	0	0	^{t}**18524**	0.00	0.00	0.00	0.00	5
rbg132	130	**18524**	**18524**	0.00	0.00	0	0	^{t}**18524**	0.00	0.00	0.00	0.00	5
rbg152.3	150	**17455**	**17455**	0.00	0.00	0	0	^{t}**17455**	0.00	0.00	0.00	0.00	11
rbg152	150	**17455**	**17455**	0.00	0.00	0	0	^{t}**17455**	0.00	0.00	0.00	0.00	6
rbg172a	172	**17783**	**17783**	0.00	0.00	11	13	**17783**	0.00	0.00	0.07	0.08	20
rbg193.2	191	**21401**	**21401**	0.00	0.00	0	0	^{t}**21401**	0.00	0.00	0.00	0.00	5
rbg193	191	**21401**	**21401**	0.00	0.00	0	0	^{t}**21401**	0.00	0.00	0.00	0.00	21
rbg201a	201	**21380**	**21380**	0.00	0.00	0	0	^{t}**21380**	0.00	0.00	0.00	0.00	20
rbg233.2	231	**26143**	**26143**	0.00	0.00	0	0	^{t}**26143**	0.00	0.00	0.00	0.00	4
rbg233	231	**26143**	**26143**	0.00	0.00	0	0	^{t}**26143**	0.00	0.00	0.00	0.00	21

Table 2: Results for the AFG instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
N20ft30	19	**730.78**	**730.78**	0.00	0	0	0	^{t}**730.78**	0.00	0.00	0.00	0.00	2.1
N20ft40	19	**730.00**	**730.00**	0.00	0	0	0	^{t}**730.00**	0.00	0.00	0.00	0.00	2.9
N40ft20	39	**999.20**	**999.20**	0.00	0	0	0	^{t}**999.20**	0.00	0.00	0.00	0.00	1.7
N40ft40	39	**996.21**	**996.21**	0.00	0	0	0	^{t}**996.21**	0.00	0.00	0.00	0.00	2.5
N60ft20	59	**1248.88**	**1248.88**	0.00	0	0	0	^{t}**1248.88**	0.00	0.00	0.00	0.00	3.8
N60ft30	59	**1247.31**	**1247.31**	0.00	0	0	0	^{t}**1247.31**	0.00	0.00	0.00	0.00	4.2
N60ft40	59	**1244.73**	**1244.73**	0.00	0	0	0	^{t}**1244.73**	0.00	0.00	0.00	0.00	6.1

Table 3: Results for the Langevin instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
rc201.1	19	**592.06**	**592.06**	0.00	0.00	0	0	^{t}**592.06**	0.00	0.00	0.00	0.00	7
rc201.2	25	**860.17**	**860.17**	0.00	0.00	0	0	^{t}**860.17**	0.00	0.00	0.00	0.00	6
rc201.3	31	**853.71**	**853.71**	0.00	0.00	0	0	^{t}**853.71**	0.00	0.00	0.00	0.00	2
rc201.4	25	**889.18**	**889.18**	0.00	0.00	0	0	^{t}**889.18**	0.00	0.00	0.00	0.00	8
rc202.1	32	**850.48**	**850.48**	0.00	0.00	0	0	**850.48**	0.00	0.00	0.00	0.00	24
rc202.2	13	**338.52**	**338.52**	0.00	0.00	0	0	^{t}**338.52**	0.00	0.00	0.00	0.00	13
rc202.3	28	**894.10**	**894.10**	0.00	0.00	0	0	^{t}**894.10**	0.00	0.00	0.00	0.00	12
rc202.4	27	**853.71**	**853.71**	-	-	0	0	**853.71**	0.00	0.00	0.00	0.00	21
rc203.1	18	**488.42**	**488.42**	0.00	0.00	0	0	**488.42**	0.00	0.00	0.00	0.00	18
rc203.2	32	**853.71**	**853.71**	0.00	0.00	0	0	**853.71**	0.00	0.00	0.01	0.00	32
rc203.3	36	**921.44**	**921.44**	0.00	0.00	0	0	**921.44**	0.00	0.00	0.01	0.00	36
rc203.4	14	**338.52**	**338.52**	-	-	0	0	^{t}**338.52**	0.00	0.00	0.00	0.00	14
rc204.1	45	**917.83**	**917.83**	0.11	0.28	8	14	**917.83**	0.00	0.00	0.31	0.27	45
rc204.2	32	**690.06**	**690.06**	-	-	0	0	^{t}**690.06**	0.00	0.00	0.08	0.08	32
rc204.3	23	**455.03**	**455.03**	0.00	0.00	0	0	**455.03**	0.00	0.00	0.01	0.01	23
rc205.1	13	**417.81**	**417.81**	0.00	0.00	0	0	^{t}**417.81**	0.00	0.00	0.00	0.00	7
rc205.2	26	**820.19**	**820.19**	0.00	0.00	0	0	**820.19**	0.00	0.00	0.00	0.00	16
rc205.3	34	**950.05**	**950.05**	0.00	0.00	0	0	^{t}**950.05**	0.00	0.00	0.00	0.00	3
rc205.4	27	**837.71**	**837.71**	0.00	0.00	0	0	^{t}**837.71**	0.00	0.00	0.00	0.00	9
rc206.1	3	**117.85**	**117.85**	0.00	0.00	0	0	^{t}**117.85**	0.00	0.00	0.00	0.00	3
rc206.2	36	**870.49**	**870.49**	0.00	0.00	4	9	**870.49**	0.00	0.00	0.05	0.03	23
rc206.3	24	**650.59**	**650.59**	0.00	0.00	0	0	**650.59**	0.00	0.00	0.00	0.00	18
rc206.4	37	**911.98**	**911.98**	-	-	4	8	**911.98**	0.00	0.00	0.05	0.03	23
rc207.1	33	**804.67**	**804.67**	0.00	0.00	1	1	**804.67**	0.00	0.00	0.11	0.04	28
rc207.2	30	**713.90**	**713.90**	0.00	0.00	0	0	**713.90**	0.00	0.00	0.02	0.01	28
rc207.3	32	**745.77**	**745.77**	0.00	0.00	0	0	**745.77**	0.00	0.00	0.19	0.14	32
rc207.4	5	**133.14**	**133.14**	0.00	0.00	0	0	^{t}**133.14**	0.00	0.00	0.00	0.00	5
rc208.1	37	**810.70**	**810.70**	0.04	0.01	0	0	**810.70**	0.00	0.00	0.60	0.32	37
rc208.2	28	**579.51**	**579.51**	0.00	0.00	0	0	**579.51**	0.00	0.00	0.03	0.02	28
rc208.3	35	**686.80**	**686.80**	0.00	0.00	0	0	**686.80**	0.00	0.00	0.42	0.41	35

Table 4: Results for the SolomonPotvinBengio instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
rc201.0	25	**853.71**	**853.71**	0.00	0.00	0	0	^{t}**853.71**	0.00	0.00	0.00	0.00	2
rc201.1	28	**850.48**	**850.48**	0.00	0.00	0	0	^{t}**850.48**	0.00	0.00	0.00	0.00	8
rc201.2	28	**883.97**	**883.97**	0.00	0.00	0	0	^{t}**883.97**	0.00	0.00	0.00	0.00	6
rc201.3	19	**722.43**	**722.43**	0.00	0.00	0	0	^{t}**722.43**	0.00	0.00	0.00	0.00	4
rc202.0	25	**850.48**	**850.48**	0.00	0.00	0	0	**850.48**	0.00	0.00	0.00	0.00	22
rc202.1	22	**702.28**	**702.28**	0.00	0.00	0	0	^{t}**702.28**	0.00	0.00	0.00	0.00	3
rc202.2	27	**853.71**	**853.71**	0.00	0.00	0	0	**853.71**	0.00	0.00	0.00	0.00	19
rc202.3	26	**883.97**	**883.97**	0.00	0.00	0	0	**883.97**	0.00	0.00	0.00	0.00	20
rc203.0	35	**870.52**	**870.52**	0.00	0.00	0	0	**870.52**	0.00	0.00	0.00	0.00	35
rc203.1	37	**850.48**	**850.48**	0.00	0.00	0	0	**850.48**	0.00	0.00	0.00	0.00	37
rc203.2	28	**853.71**	**853.71**	0.00	0.00	0	0	**853.71**	0.00	0.00	0.00	0.00	24
rc204.0	32	**839.24**	**839.24**	0.00	0.00	0	0	^{t}**839.24**	0.00	0.00	0.00	0.00	32
rc204.1	28	**492.60**	**492.60**	0.00	0.00	0	0	**492.60**	0.00	0.00	0.02	0.01	28
rc204.2	40	**870.52**	**870.52**	2.65	2.24	0	1	**870.52**	0.00	0.00	0.12	0.05	40
rc205.0	26	**834.62**	**834.62**	0.00	0.00	0	0	^{t}**834.62**	0.00	0.00	0.00	0.00	14
rc205.1	22	**899.24**	**899.24**	0.00	0.00	0	0	^{t}**899.24**	0.00	0.00	0.00	0.00	2
rc205.2	28	**908.79**	**908.79**	0.00	0.00	0	0	**908.79**	0.00	0.00	0.00	0.00	14
rc205.3	24	**684.21**	**684.21**	0.31	0.81	0	0	**684.21**	0.00	0.00	0.00	0.00	16
rc206.0	35	**893.21**	**893.21**	0.00	0.00	0	0	**893.21**	0.00	0.00	0.02	0.01	21
rc206.1	33	**756.45**	**756.45**	0.00	0.00	1	2	**756.45**	0.00	0.00	0.00	0.00	24
rc206.2	32	**776.19**	**776.19**	0.00	0.00	3	8	**776.19**	0.00	0.00	0.04	0.01	23
rc207.0	37	**847.63**	**847.63**	0.00	0.00	0	0	**847.63**	0.00	0.00	0.02	0.01	33
rc207.1	33	**785.37**	**785.37**	0.00	0.00	0	0	**785.37**	0.00	0.00	0.12	0.07	29
rc207.2	30	**650.80**	**650.80**	0.00	0.00	0	0	**650.80**	0.00	0.00	0.08	0.03	30
rc208.0	44	**836.04**	**836.04**	0.04	0.09	11	8	**836.04**	0.00	0.00	1.20	0.45	44
rc208.1	27	**615.51**	**615.51**	0.00	0.00	0	0	**615.51**	0.00	0.00	0.04	0.03	27
rc208.2	29	**596.21**	**596.21**	0.00	0.00	0	0	**596.21**	0.00	0.00	0.03	0.02	29

Table 5: Results for the SolomonPesant instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
n20w120	20	**319.6**	**319.6**	0.00	0.00	0	0	**319.6**	0.00	0.00	0.00	0.00	15.0
n20w140	20	**286.2**	**286.2**	0.00	0.00	0	0	**286.2**	0.00	0.00	0.00	0.00	16.2
n20w160	20	**311.4**	**311.4**	0.00	0.00	0	0	**311.4**	0.00	0.00	0.00	0.00	16.6
n20w180	20	**311.2**	**311.2**	0.00	0.00	0	0	**311.2**	0.00	0.00	0.00	0.00	14.8
n20w200	20	**281.8**	**281.8**	0.00	0.00	0	0	**281.8**	0.00	0.00	0.00	0.00	19.0
n40w120	40	**470.6**	**470.6**	0.00	0.00	0	0	**470.6**	0.00	0.00	0.00	0.00	19.6
n40w140	40	**458.2**	**458.2**	0.00	0.00	0	0	**458.2**	0.00	0.00	0.00	0.00	25.6
n40w160	40	**426.8**	**426.8**	0.00	0.00	0	0	**426.8**	0.00	0.00	0.00	0.00	30.4
n40w180	40	**427.4**	**427.4**	0.00	0.00	0	0	**427.4**	0.00	0.00	0.00	0.00	25.8
n40w200	40	**412.0**	**412.0**	0.00	0.00	0	0	**412.0**	0.00	0.00	0.00	0.00	22.6
n60w120	60	**573.8**	**573.8**	0.05	0.08	1	1	**573.8**	0.00	0.00	0.00	0.00	15.8
n60w140	60	**600.0**	**600.0**	0.00	0.00	0	0	^{t}**600.0**	0.00	0.00	0.00	0.00	27.0
n60w160	60	**619.6**	**619.6**	0.00	0.00	0	0	**619.6**	0.00	0.00	0.00	0.00	30.6
n60w180	60	**576.0**	**576.0**	0.00	0.00	0	0	**576.0**	0.00	0.00	0.00	0.00	41.4
n60w200	60	**570.2**	**570.2**	0.02	0.02	0	1	**570.2**	0.00	0.00	0.00	0.00	37.6
n80w100	80	**711.2**	**711.2**	0.00	0.00	0	0	**711.2**	0.00	0.00	0.01	0.00	31.4
n80w120	80	**697.4**	**697.4**	0.01	0.01	1	1	**697.4**	0.00	0.00	0.00	0.00	28.6
n80w140	80	**672.8**	**672.8**	0.00	0.00	1	1	**672.8**	0.00	0.00	0.00	0.00	33.2
n80w160	80	**653.6**	**653.6**	0.23	0.24	2	2	**653.6**	0.00	0.00	0.00	0.00	45.8
n80w180	80	**656.4**	**656.4**	0.05	0.09	1	1	**656.4**	0.00	0.00	0.00	0.00	49.0
n80w200	80	**646.2**	**646.2**	0.00	0.00	2	3	**646.2**	0.00	0.00	0.00	0.00	56.6
n100w80	100	**805.8**	**805.8**	0.00	0.00	0	0	**805.8**	0.00	0.00	0.00	0.00	22.0
n100w100	100	**795.8**	**795.8**	0.00	0.00	0	0	**795.8**	0.00	0.00	0.00	0.00	32.6
n100w120	100	**895.4**	**895.4**	0.00	0.00	0	0	**895.4**	0.00	0.00	0.00	0.00	23.6
n100w140	100	**906.4**	**906.4**	0.00	0.00	0	0	**906.4**	0.00	0.00	0.00	0.00	18.8
n100w160	100	**865.0**	**865.0**	0.00	0.00	0	0	**865.0**	0.00	0.00	0.00	0.00	31.8

Table 6: Results for the GendreauDumasExtended instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

instance	N	BK	GVNS					ImaxLNS, $Wmax = 4$					
			BF	Gm	Gs	Tm	Ts	BF	Gm	Gs	Tm	Ts	W
n150w120.001	150	**972**	**972**	0.00	0.00	3	3	**972**	0.00	0.00	0.00	0.00	51
n150w120.002	150	**917**	**917**	0.00	0.00	1	0	^{t}**917**	0.00	0.00	0.01	0.00	53
n150w120.003	150	**909**	910	0.29	0.28	11	11	**909**	0.00	0.00	0.01	0.00	54
n150w120.004	150	**925**	**925**	0.00	0.00	2	2	**925**	0.00	0.00	0.01	0.00	54
n150w120.005	150	**907**	**907**	0.00	0.00	2	1	^{t}**907**	0.00	0.00	0.00	0.00	6
n150w140.001	150	**1008**	**1008**	0.00	0.00	1	0	^{t}**1008**	0.00	0.00	0.00	0.00	69
n150w140.002	150	**1020**	**1020**	0.00	0.00	3	3	^{t}**1020**	0.00	0.00	0.00	0.00	7
n150w140.003	150	**844**	**844**	0.00	0.00	1	0	^{t}**844**	0.00	0.00	0.00	0.00	61
n150w140.004	150	**898**	**898**	0.00	0.00	1	1	**898**	0.00	0.00	0.01	0.00	66
n150w140.005	150	**926**	**926**	0.00	0.00	1	0	**926**	0.00	0.00	0.01	0.01	64
n150w160.001	150	**959**	**959**	0.00	0.00	1	0	^{t}**959**	0.00	0.00	0.02	0.01	69
n150w160.002	150	**890**	**890**	0.03	0.05	18	15	**890**	0.00	0.00	0.02	0.01	70
n150w160.003	150	**934**	**934**	0.00	0.00	1	0	^{t}**934**	0.00	0.00	0.01	0.00	62
n150w160.004	150	**912**	**912**	0.00	0.00	1	0	**912**	0.00	0.00	0.01	0.01	71
n150w160.005	150	**920**	**920**	0.00	0.00	1	0	**920**	0.00	0.00	0.00	0.00	61
n200w120.001	200	**1089**	**1089**	0.28	1.04	13	11	**1089**	0.00	0.00	0.01	0.00	68
n200w120.002	200	**1072**	**1072**	0.00	0.00	2	1	**1072**	0.00	0.00	0.00	0.00	17
n200w120.003	200	**1128**	**1128**	0.00	0.00	7	6	^{t}**1128**	0.00	0.00	0.00	0.00	62
n200w120.004	200	**1072**	**1072**	0.00	0.00	3	1	^{t}**1072**	0.00	0.00	0.02	0.01	60
n200w120.005	200	**1073**	**1073**	0.00	0.00	2	0	**1073**	0.00	0.00	0.01	0.01	65
n200w140.001	200	**1138**	**1138**	0.00	0.00	19	17	^{t}**1138**	0.00	0.00	0.02	0.00	71
n200w140.002	200	**1087**	**1087**	0.00	0.00	3	0	**1087**	0.00	0.00	0.00	0.00	71
n200w140.003	200	**1083**	**1083**	0.00	0.00	5	3	**1083**	0.00	0.00	0.02	0.00	75
n200w140.004	200	**1100**	**1100**	0.00	0.00	10	8	^{t}**1100**	0.00	0.00	0.01	0.00	79
n200w140.005	200	**1121**	**1121**	0.00	0.00	5	2	**1121**	0.00	0.00	0.01	0.00	68

Table 7: Results for the OhlmannThomas instances (for GVNS: 24 seconds per run, 5 runs per instance; for ImaxLNS: 5 seconds per run, 5 runs per instance)

1.2 Arigliano et al. [2] benchmark

The new feasible solutions and the new best solutions found compared to Lera-Romero et al. [3] are given in the tables below.

40_70_A_0_A4	686.89
40_70_A_0_A7	700.59
40_70_A_0_A9	688.47
40_70_A_0_B2	750.87
40_70_A_0_B5	957.27
40_70_A_0_B6	790.03
40_70_A_0_B7	816.17
40_70_A_0_C1	915.46
40_70_A_0_C3	846.09
40_70_A_0_C4	973.48
40_70_A_0_C8	890.03
40_70_A_0_C9	898.24
40_70_B_0_A7	749.49
40_70_B_0_B5	999.18
40_70_B_0_B6	864.74
40_70_B_0_B7	898.36
40_70_B_0_C4	992.61
40_70_B_0_C10	922.96
40_80_A_0_A7	696.74
40_80_A_0_A9	677.41
40_80_A_0_B5	927.82
40_80_A_0_B6	784.58
40_80_A_0_B7	800.48
40_80_A_0_C1	895.32
40_80_A_0_C3	842.49
40_80_A_0_C4	955.81
40_80_A_0_C9	882.44
40_80_B_0_A7	720.84
40_80_B_0_B5	935.77
40_80_B_0_B6	824.06
40_80_B_0_C1	886.26
40_80_B_0_C3	825.96
40_80_B_0_C4	963.76
40_80_B_0_C10	880.93
40_90_A_0_A7	694.65
40_90_A_0_B5	904.03
40_90_A_0_C1	881.49
40_90_A_0_C3	854.18

40_90_A_0_C4	943.52
40_90_B_0_A7	704.10
40_90_B_0_B5	893.61
40_90_B_0_B7	810.52
40_90_B_0_C3	811.87
40_90_B_0_C4	936.80
40_95_A_0_A7	690.42
40_95_A_0_B5	888.59
40_95_A_0_B6	773.16
40_95_A_0_C4	930.05
40_95_A_0_C9	819.25
40_95_B_0_A7	695.29
40_95_B_0_B5	883.67
40_95_B_0_C1	851.13
40_95_B_0_C3	799.70
40_95_B_0_C4	928.07
40_95_B_0_C10	817.43
40_98_A_0_A7	687.60
40_98_A_0_C4	923.82
40_98_A_0_C9	809.15
40_98_B_0_A7	689.75
40_98_B_0_B5	873.25
40_98_B_0_C4	921.87
40_98_B_0_C9	810.63
40_98_B_0_C10	812.89
40_70_A_25_B7	929.99
40_70_B_25_B5	1011.14
40_70_B_25_B7	966.51
40_80_A_25_B7	914.45
40_80_B_25_B7	919.23
40_80_B_25_C3	987.64
40_90_A_25_B7	897.75
40_90_B_25_B7	895.82
40_90_B_25_C3	971.79
40_95_A_25_B7	885.60
40_95_B_25_B7	887.27
40_98_A_25_B7	876.92
40_98_B_25_B7	880.73

Table 8: New feasible solutions and best solutions for the Arigliano et al. [2] benchmark

1.3 Aguiar-Melgarejo et al. [4] benchmark

Tables 9 to 11 give the detailed results obtained on all instances of sizes 30, 50,100 and for distance matrices M00, M10, M20. For the *CP-tdNoOverlap* method of Aguiar-Melgarejo et al. [4] (that has a maximum CPU time of 2 hours), the table reports the best solutions found for each instance (column BF). It also gives the relative gap G in percent with regards to the best solutions found during all our experiments on ImaxLNS (column ImaxLNS-all). For each instance, the best solution found by ImaxLNS with $Wmax = 5$ over 5 runs of 2 minutes is given in column BF. The semantics of columns Gm, Gs, Tm, and Ts is given in the main article.

matrix	instance	CP-tdNoOverlap		ImaxLNS, $Wmax = 5$						ImaxLNS-all
		BF	G	BF	Gm	Gs	Tm	Ts	W	BF
M00	01	18493	1.18	**18278**	0.00	0.00	0.29	0.25	29	**18278**
	02	**17111**	0.00	**17111**	0.00	0.00	0.00	0.00	29	**17111**
	03	**18148**	0.00	**18148**	0.00	0.00	0.08	0.05	29	**18148**
	04	**20335**	0.00	^{t}**20335**	0.00	0.00	0.05	0.02	29	**20335**
	05	15482	1.51	**15251**	0.00	0.00	0.61	0.77	29	**15251**
	06	**18635**	0.00	**18635**	0.00	0.00	0.06	0.03	29	**18635**
	07	**17252**	0.00	**17252**	0.00	0.00	0.00	0.00	29	**17252**
	08	**20342**	0.00	**20342**	0.00	0.00	0.03	0.01	29	**20342**
	09	**16305**	0.00	**16305**	0.00	0.00	0.08	0.06	29	**16305**
	10	**17367**	0.00	**17367**	0.00	0.00	0.04	0.02	29	**17367**
	11	**17575**	0.00	**17575**	0.00	0.00	0.01	0.01	29	**17575**
	12	**17909**	0.00	**17909**	0.00	0.00	0.02	0.01	29	**17909**
	13	**20522**	0.00	**20522**	0.00	0.00	2.79	0.91	29	**20522**
	14	**18982**	0.00	**18982**	0.00	0.00	0.08	0.04	29	**18982**
	15	18573	1.99	**18211**	0.00	0.00	0.20	0.15	29	**18211**
	16	**15570**	0.00	**15570**	0.00	0.00	0.00	0.00	29	**15570**
	17	**18506**	0.00	**18506**	0.00	0.00	0.35	0.08	29	**18506**
	18	**22418**	0.00	**22418**	0.00	0.00	0.26	0.19	29	**22418**
	19	**25157**	0.00	^{t}**25157**	0.00	0.00	0.00	0.00	29	**25157**
	20	**22132**	0.00	^{t}**22132**	0.00	0.00	0.01	0.00	29	**22132**
M10	01	**18157**	0.00	**18157**	0.00	0.00	0.03	0.01	29	**18157**
	02	**16933**	0.00	**16933**	0.00	0.00	0.08	0.05	29	**16933**
	03	**17770**	0.00	**17770**	0.00	0.00	0.01	0.01	29	**17770**
	04	**20147**	0.00	^{t}**20147**	0.00	0.00	0.03	0.03	29	**20147**
	05	15597	1.39	**15383**	0.00	0.00	0.13	0.10	29	**15383**
	06	**18203**	0.00	**18203**	0.00	0.00	0.00	0.00	29	**18203**
	07	**17175**	0.00	**17175**	0.00	0.00	0.00	0.00	29	**17175**
	08	**20120**	0.00	**20120**	0.00	0.00	0.03	0.03	29	**20120**
	09	**16236**	0.00	**16236**	0.00	0.00	0.11	0.05	29	**16236**
	10	**17302**	0.00	**17302**	0.00	0.00	0.01	0.01	29	**17302**
	11	**17609**	0.00	**17609**	0.00	0.00	0.00	0.00	29	**17609**
	12	**17722**	0.00	**17722**	0.00	0.00	0.02	0.00	29	**17722**
	13	**19862**	0.00	**19862**	0.00	0.00	0.15	0.12	29	**19862**
	14	**18664**	0.00	**18664**	0.00	0.00	0.19	0.20	29	**18664**
	15	18394	2.13	**18010**	0.00	0.00	0.21	0.13	29	**18010**
	16	**15590**	0.00	**15590**	0.00	0.00	0.02	0.01	29	**15590**
	17	**18264**	0.00	**18264**	0.00	0.00	0.03	0.04	29	**18264**
	18	**21561**	0.00	**21561**	0.00	0.00	0.24	0.21	29	**21561**
	19	**24895**	0.00	^{t}**24895**	0.00	0.00	0.00	0.00	29	**24895**
	20	**21996**	0.00	^{t}**21996**	0.00	0.00	0.01	0.00	29	**21996**
M20	01	**18109**	0.00	**18109**	0.00	0.00	0.00	0.00	29	**18109**
	02	**16634**	0.00	**16634**	0.00	0.00	0.00	0.00	29	**16634**
	03	**17349**	0.00	**17349**	0.00	0.00	0.13	0.08	29	**17349**
	04	**20024**	0.00	^{t}**20024**	0.00	0.00	0.05	0.02	29	**20024**
	05	**15271**	0.00	**15271**	0.00	0.00	0.14	0.08	29	**15271**
	06	**17776**	0.00	**17776**	0.00	0.00	0.01	0.01	29	**17776**
	07	**16971**	0.00	**16971**	0.00	0.00	0.00	0.00	29	**16971**
	08	**19975**	0.00	**19975**	0.00	0.00	0.01	0.01	29	**19975**
	09	**16046**	0.00	**16046**	0.00	0.00	0.18	0.13	29	**16046**
	10	**16957**	0.00	**16957**	0.00	0.00	0.01	0.01	29	**16957**
	11	**17516**	0.00	**17516**	0.00	0.00	0.07	0.06	29	**17516**
	12	**17624**	0.00	**17624**	0.00	0.00	0.05	0.04	29	**17624**
	13	**19572**	0.00	**19572**	0.00	0.00	0.01	0.01	29	**19572**
	14	**18062**	0.00	**18062**	0.00	0.00	0.20	0.09	29	**18062**
	15	**17845**	0.00	**17845**	0.00	0.00	0.15	0.06	29	**17845**
	16	**15612**	0.00	**15612**	0.00	0.00	0.01	0.00	29	**15612**
	17	18055	0.07	**18042**	0.00	0.00	0.19	0.16	29	**18042**
	18	**21222**	0.00	**21222**	0.00	0.00	0.01	0.01	29	**21222**
	19	**24814**	0.00	^{t}**24814**	0.00	0.00	0.00	0.00	29	**24814**
	20	**21861**	0.00	^{t}**21861**	0.00	0.00	0.00	0.00	29	**21861**

Table 9: Detailed results on the TDTSPTW benchmark defined by Aguiar-Melgarejo et al. [4], for the instances of size 30 and transition matrices M00, M10, M20 (configuration for ImaxLNS: $Wmax = 5$, maximum CPU time = 2 minutes per run, 5 runs per instance)

matrix	instance	CP-tdNoOverlap		ImaxLNS, $Wmax = 5$						ImaxLNS-all
		BF	G	BF	Gm	Gs	Tm	Ts	W	BF
M00	01	23493	0.08	**23474**	0.00	0.00	2.80	3.07	49	**23474**
	02	**24824**	0.00	**24824**	0.00	0.00	1.14	0.32	49	**24824**
	03	**24379**	0.00	**24379**	0.00	0.00	0.05	0.06	49	**24379**
	04	26687	0.96	**26433**	0.00	0.00	9.27	9.19	49	**26433**
	05	**21911**	0.00	**21911**	0.00	0.00	0.03	0.02	49	**21911**
	06	**23056**	0.00	**23056**	0.00	0.00	0.02	0.01	49	**23056**
	07	23882	0.23	**23827**	0.00	0.00	1.22	0.34	49	**23827**
	08	**23945**	0.00	**23945**	0.00	0.00	0.03	0.02	49	**23945**
	09	**23171**	0.00	**23171**	0.00	0.00	0.06	0.06	49	**23171**
	10	24136	0.99	**23900**	0.00	0.00	2.95	2.03	49	**23900**
	11	**25034**	0.00	**25034**	0.00	0.00	0.78	0.78	49	**25034**
	12	**25541**	0.00	**25541**	0.00	0.00	10.62	8.34	49	**25541**
	13	**24885**	0.00	**24885**	0.00	0.00	3.37	2.25	49	**24885**
	14	27658	1.53	**27241**	0.13	0.13	32.11	33.99	49	**27241**
	15	**23394**	0.00	**23394**	0.00	0.00	0.02	0.02	49	**23394**
	16	25122	0.29	**25050**	0.00	0.00	0.17	0.12	49	**25050**
	17	25327	0.04	**25317**	0.00	0.00	6.33	5.71	49	**25317**
	18	**22725**	0.00	**22725**	0.00	0.00	0.03	0.02	49	**22725**
	19	26810	2.04	**26274**	0.02	0.02	58.53	29.08	49	**26274**
	20	**24501**	0.00	**24501**	0.00	0.00	4.49	2.00	49	**24501**
M10	01	**22795**	0.00	**22795**	0.00	0.00	1.27	0.98	49	**22795**
	02	**23858**	0.00	**23858**	0.00	0.00	0.57	0.41	49	**23858**
	03	**22192**	0.00	**22192**	0.00	0.00	0.02	0.01	49	**22192**
	04	25709	1.43	**25347**	0.00	0.00	8.83	5.07	49	**25347**
	05	22959	5.59	**21744**	0.00	0.00	0.90	0.65	49	**21744**
	06	22470	0.24	**22416**	0.00	0.00	0.85	0.46	49	**22416**
	07	**23358**	0.00	**23358**	0.00	0.00	4.23	2.75	49	**23358**
	08	**23244**	0.00	**23244**	0.00	0.00	0.18	0.09	49	**23244**
	09	22539	0.09	**22518**	0.00	0.00	0.28	0.13	49	**22518**
	10	**23158**	0.00	**23158**	0.00	0.00	0.25	0.46	49	**23158**
	11	**24308**	0.00	**24308**	0.00	0.00	1.69	1.17	49	**24308**
	12	**24714**	0.00	**24714**	0.00	0.00	0.05	0.04	49	**24714**
	13	**23861**	0.00	**23861**	0.00	0.00	0.05	0.02	49	**23861**
	14	27158	3.55	**26226**	0.14	0.14	32.07	29.47	49	**26226**
	15	**22987**	0.00	**22987**	0.00	0.00	0.07	0.06	49	**22987**
	16	**24111**	0.00	**24111**	0.00	0.00	0.11	0.06	49	**24111**
	17	24243	0.21	**24191**	0.00	0.00	0.09	0.07	49	**24191**
	18	22273	0.76	**22106**	0.00	0.00	0.22	0.14	49	**22106**
	19	25829	0.98	**25579**	0.00	0.00	62.58	23.87	49	**25579**
	20	23417	0.02	**23413**	0.00	0.00	1.18	0.72	49	**23413**
M20	01	22832	1.53	**22487**	0.00	0.00	1.69	1.01	49	**22487**
	02	23314	0.05	**23303**	0.00	0.00	0.96	0.69	49	**23303**
	03	**21730**	0.00	**21730**	0.00	0.00	0.03	0.02	49	**21730**
	04	25284	3.72	**24378**	0.00	0.00	2.89	1.50	49	**24378**
	05	23275	7.95	**21560**	0.00	0.00	4.17	1.08	49	**21560**
	06	22123	1.05	**21894**	0.00	0.00	3.01	1.89	49	**21894**
	07	22890	0.39	**22800**	0.00	0.00	0.97	0.62	49	**22800**
	08	**22608**	0.00	**22608**	0.00	0.00	0.15	0.09	49	**22608**
	09	22301	0.11	**22276**	0.00	0.00	1.36	0.43	49	**22276**
	10	**22670**	0.00	**22670**	0.00	0.00	0.00	0.00	49	**22670**
	11	**23778**	0.00	**23778**	0.00	0.00	0.31	0.46	49	**23778**
	12	24162	0.05	**24149**	0.00	0.00	0.28	0.19	49	**24149**
	13	23256	0.47	**23148**	0.00	0.00	18.00	14.58	49	**23148**
	14	26306	3.58	**25396**	0.00	0.00	53.15	27.64	49	**25396**
	15	**22575**	0.00	**22575**	0.00	0.00	0.30	0.33	49	**22575**
	16	23679	0.16	**23640**	0.00	0.00	0.50	0.66	49	**23640**
	17	**23495**	0.00	**23495**	0.00	0.00	0.05	0.04	49	**23495**
	18	**22000**	0.00	**22000**	0.00	0.00	0.01	0.00	49	**22000**
	19	25289	1.98	**24798**	1.17	0.54	82.54	8.84	49	**24798**
	20	**23032**	0.00	**23032**	0.00	0.00	1.51	1.10	49	**23032**

Table 10: Detailed results on the TDTSPTW benchmark defined by Aguiar-Melgarejo et al. [4], for the instances of size 50 and transition matrices M00, M10, M20 (configuration for ImaxLNS: $Wmax = 5$, maximum CPU time = 2 minutes per run, 5 runs per instance)

matrix	instance	CP-tdNoOverlap		ImaxLNS, $Wmax = 5$						ImaxLNS-all
		BF	G	BF	Gm	Gs	Tm	Ts	W	BF
M00	01	44741	7.77	41554	0.50	0.50	68.52	42.67	99	**41515**
	02	-	-	37945	0.52	0.50	23.81	10.97	99	**37917**
	03	45308	10.69	41204	1.07	0.68	86.96	31.07	99	**40931**
	04	50124	8.80	46123	0.41	0.41	59.43	37.73	99	**46070**
	05	41815	3.42	40438	0.56	0.51	45.92	38.20	99	**40434**
	06	40051	2.02	39290	0.11	0.11	73.05	41.88	99	**39259**
	07	47074	15.17	**40875**	0.02	0.02	38.79	29.16	99	**40875**
	08	-	-	44261	0.30	0.30	63.61	29.90	99	**44194**
	09	-	-	**39405**	0.18	0.18	78.30	41.60	99	**39405**
	10	45161	4.44	43291	0.31	0.31	63.29	27.04	99	**43240**
	11	45656	3.04	44646	1.20	0.85	90.34	19.25	99	**44311**
	12	45499	1.59	44815	0.28	0.28	97.11	26.03	99	**44788**
	13	43734	1.02	**43291**	0.15	0.15	61.34	10.25	99	**43291**
	14	40888	1.42	40477	0.68	0.68	23.75	34.39	99	**40314**
	15	45146	5.71	42724	0.36	0.43	41.83	42.94	99	**42708**
	16	49901	14.49	43768	1.82	1.04	39.32	35.96	99	**43585**
	17	45122	1.18	**44594**	0.00	0.00	6.37	6.61	99	**44594**
	18	44802	1.63	44127	0.13	0.13	49.29	25.92	99	**44083**
	19	44638	0.56	44619	0.52	0.52	19.99	12.73	99	**44390**
	20	-	-	**42454**	0.00	0.00	48.12	37.09	99	**42454**
M10	01	43005	8.23	39969	0.79	0.72	77.24	24.82	99	**39736**
	02	38464	4.89	36824	0.91	0.58	45.06	31.42	99	**36672**
	03	46961	18.81	39571	0.27	0.27	65.70	40.48	99	**39527**
	04	50691	12.98	45023	0.43	0.43	92.80	31.82	99	**44868**
	05	41662	5.73	39584	0.57	0.57	37.67	17.00	99	**39405**
	06	39133	0.79	**38826**	0.00	0.00	8.06	4.11	99	**38826**
	07	**40493**	0.00	**40493**	0.00	0.00	23.20	13.05	99	**40493**
	08	47085	9.94	42917	0.65	0.60	56.05	28.25	99	**42828**
	09	-	-	38656	0.87	0.78	82.41	28.34	99	**38363**
	10	42963	1.34	42579	0.55	0.55	58.31	28.86	99	**42397**
	11	44187	4.91	42234	1.82	0.90	57.53	36.31	99	**42118**
	12	44249	1.83	**43455**	0.23	0.23	77.06	35.46	99	**43455**
	13	41741	2.09	41203	1.09	0.69	37.89	21.08	99	**40885**
	14	39623	0.89	**39272**	0.38	0.38	73.30	40.79	99	**39272**
	15	43086	5.58	41175	1.57	0.87	83.79	20.03	99	**40807**
	16	41802	1.54	41508	1.06	0.48	89.99	30.58	99	**41166**
	17	45037	1.84	44309	0.48	0.48	49.92	32.94	99	**44223**
	18	44499	2.80	43343	0.76	0.76	47.64	29.15	99	**43289**
	19	44603	0.50	**44383**	0.00	0.00	3.40	1.78	99	**44383**
	20	41699	0.19	41623	0.08	0.08	49.00	22.21	99	**41621**
M20	01	40905	6.85	38316	0.77	0.62	67.71	25.61	99	**38282**
	02	36200	1.86	35547	0.39	0.39	54.09	29.87	99	**35538**
	03	46286	18.97	39525	1.78	0.78	58.79	45.40	99	**38905**
	04	48187	10.44	43679	0.41	0.41	67.90	33.09	99	**43631**
	05	40333	4.66	**38536**	0.00	0.00	23.90	9.31	99	**38536**
	06	38950	2.17	38647	1.87	0.56	42.13	45.28	99	**38121**
	07	43886	9.57	40102	0.34	0.34	50.79	19.14	99	**40053**
	08	45176	8.38	41787	1.13	0.96	78.27	40.60	99	**41683**
	09	38530	3.37	37278	0.37	0.37	38.28	22.78	99	**37275**
	10	44197	8.71	41261	1.85	0.66	9.99	15.04	99	**40656**
	11	42466	4.78	40544	0.76	0.61	76.13	38.20	99	**40530**
	12	42414	2.44	41571	0.62	0.62	40.58	26.94	99	**41405**
	13	39955	2.98	39107	2.08	1.05	24.30	17.67	99	**38800**
	14	39198	1.11	**38769**	0.51	0.51	32.83	16.52	99	**38769**
	15	40351	3.11	39380	0.97	0.87	77.37	29.84	99	**39133**
	16	40780	2.61	40075	1.11	0.70	45.05	13.86	99	**39744**
	17	45107	5.85	42643	0.60	0.53	67.63	35.86	99	**42615**
	18	43910	4.37	**42072**	0.45	0.49	64.97	36.96	99	**42072**
	19	47371	9.36	43796	1.65	0.60	80.22	39.50	99	**43316**
	20	-	-	40817	0.66	0.66	54.89	36.47	99	**40712**

Table 11: Detailed results on the TDTSPTW benchmark defined by Aguiar-Melgarejo et al. [4], for the instances of size 100 and transition matrices M00, M10, M20 (configuration for ImaxLNS: $Wmax = 5$, maximum CPU time = 2 minutes per run, 5 runs per instance)

1.4 Earth observing satellites benchmark

Column W corresponds to the insertion-width of each instance, and the semantics of the other columns is detailed in the main article.

inst	N	ImaxLNS, $Wmax = 5$			
		#nsr	Tm	Ts	W
s01	66	0/5	0.06	0.04	21
s02	79	0/5	0.01	0.01	22
s03	71	0/5	0.02	0.01	19
s04	50	0/5	0.00	0.00	16
s05	83	0/5	0.01	0.00	21
s06	81	0/5	0.05	0.03	17
s07	68	0/5	0.00	0.00	17
s08	66	0/5	0.02	0.01	18
s09	59	0/5	0.02	0.01	18
s10	54	0/5	0.01	0.01	20
s11	50	0/5	0.05	0.04	22
s12	61	0/5	0.03	0.01	25
s13	48	0/5	0.01	0.01	23
s14	53	0/5	0.02	0.02	19
s15	53	0/5	0.01	0.00	21
s16	55	0/5	0.01	0.00	14
s17	55	0/5	0.01	0.00	21
s18	51	0/5	0.03	0.01	21
s19	67	0/5	0.12	0.10	22
s20	81	0/5	0.01	0.00	22
s21	76	0/5	0.07	0.05	19
s22	50	0/5	0.00	0.00	16
s23	86	0/5	0.03	0.02	23
s24	83	0/5	0.05	0.02	17
s25	68	0/5	0.00	0.00	17
s26	70	0/5	0.01	0.00	18
s27	63	0/5	0.05	0.04	18
s28	55	0/5	0.03	0.02	20
s29	49	0/5	0.03	0.04	22
s30	57	0/5	0.37	0.22	24
s31	49	0/5	0.02	0.00	23
s32	53	0/5	0.02	0.02	19
s33	53	0/5	0.01	0.00	21
s34	55	0/5	0.01	0.00	14
s35	56	0/5	0.01	0.01	21
s36	51	0/5	0.03	0.01	21

(a) 500km

inst	N	ImaxLNS, $Wmax = 5$			
		#nsr	Tm	Ts	W
s01	67	0/5	0.02	0.01	30
s02	74	0/5	0.11	0.07	45
s03	59	0/5	0.25	0.21	38
s04	75	0/5	0.02	0.01	49
s05	73	0/5	0.07	0.04	35
s06	58	0/5	0.13	0.10	34
s07	66	0/5	0.20	0.05	42
s08	75	0/5	0.39	0.10	48
s09	75	0/5	1.36	1.18	49
s10	61	0/5	0.06	0.05	30
s11	60	0/5	0.01	0.00	31
s12	68	0/5	0.07	0.04	31
s13	57	0/5	0.02	0.01	26
s14	63	0/5	0.33	0.14	31
s15	70	0/5	0.29	0.12	31
s16	61	0/5	0.37	0.33	28
s17	76	0/5	1.63	1.08	47
s18	68	0/5	0.14	0.07	41
s19	67	0/5	0.01	0.01	30
s20	74	0/5	0.11	0.07	45
s21	64	0/5	0.32	0.24	39
s22	75	0/5	0.02	0.01	49
s23	73	0/5	0.07	0.04	35
s24	61	0/5	0.54	0.30	35
s25	73	0/5	0.67	0.50	45
s26	74	0/5	0.71	0.30	47
s27	75	1/5	1.35	1.83	50
s28	61	0/5	0.06	0.05	30
s29	63	0/5	0.03	0.02	31
s30	69	1/5	2.99	2.01	31
s31	57	0/5	0.02	0.01	26
s32	67	0/5	1.49	0.51	31
s33	69	0/5	1.24	1.27	30
s34	65	0/5	1.28	0.28	31
s35	73	0/5	2.22	0.91	43
s36	72	0/5	1.60	0.57	42

(b) 700km

inst	N	ImaxLNS, $Wmax = 5$			
		#nsr	Tm	Ts	W
s01	74	0/5	0.00	0.00	32
s02	75	0/5	0.04	0.04	30
s03	68	0/5	0.26	0.17	29
s04	72	0/5	0.00	0.00	42
s05	73	0/5	0.00	0.00	36
s06	71	0/5	0.01	0.00	35
s07	73	0/5	0.01	0.00	39
s08	74	0/5	0.02	0.00	47
s09	72	0/5	0.00	0.00	36
s10	84	0/5	1.17	0.81	40
s11	91	0/5	0.33	0.20	41
s12	84	1/5	3.27	1.33	36
s13	87	1/5	2.49	1.47	39
s14	86	0/5	2.84	1.49	38
s15	86	0/5	1.09	0.58	39
s16	84	1/5	2.84	1.77	43
s17	96	0/5	0.52	0.41	51
s18	94	0/5	2.07	0.91	53
s19	53	0/5	0.25	0.18	37
s20	52	0/5	0.10	0.06	38
s21	47	0/5	0.16	0.11	32
s22	49	0/5	0.05	0.05	41
s23	50	0/5	0.38	0.15	36
s24	54	0/5	1.40	1.13	43
s25	51	0/5	0.65	0.30	29
s26	53	0/5	0.38	0.31	32
s27	49	0/5	0.11	0.07	32
s28	55	0/5	1.77	1.13	39
s29	51	0/5	0.20	0.08	31
s30	49	0/5	0.18	0.26	29
s31	53	0/5	0.06	0.04	35
s32	53	0/5	0.03	0.01	39
s33	48	0/5	0.16	0.10	29
s34	48	0/5	0.11	0.08	36
s35	50	0/5	0.22	0.15	39
s36	52	0/5	0.46	0.27	38

(c) 800km

Table 12: Satellite TDTSPTW benchmark: detailed results for ImaxLNS with $Wmax = 5$ and for the three different altitudes (maximum CPU time = 5 seconds per run, 5 runs per instance)

2 Proofs

2.1 Recursive definition of the ancestors and descendants

Lemma 1. *By considering the customers in a topological order of precedence graph $G(\beta)$, quantities $Anc(\beta, i)$ can be recursively computed by:*

$$Anc(\beta,i) \leftarrow Anc_0(i) \cup \{Prev(\beta,i)\} \cup Anc(\beta, Prev(\beta,i)) \text{ if } i \in \beta \quad (1)$$

$$Anc_0(i) \cup Anc(\beta, lastAnc(\beta,i)) \text{ otherwise} \quad (2)$$

where $Anc_0(i)$ is the set of mandatory ancestors of i in the initial precedence graph G.

Proof. Let us first consider a customer $i \in \beta$. By definition of graph $G(\beta)$, it is easy to see that inclusion $Anc_0(i) \cup \{Prev(\beta,i)\} \cup Anc(\beta, Prev(\beta,i)) \subseteq Anc(\beta,i)$ holds. For the reverse inclusion, let us consider an ancestor j of i that is not in $Anc_0(i)$. In this case, there necessarily exists a path from j to i containing one customer k belonging to β and as $G(\beta)$ is acyclic, customer k must appear before i is β. This allows us to infer that either $k = Prev(\beta,i)$ or there exists a path from k to $Prev(\beta,i)$. In the second case, we obtain that j is necessarily an ancestor of $Prev(\beta,i)$, which proves the reverse inclusion.

Let us now consider a customer $i \notin \beta$. By definition of graph $G(\beta)$, it is easy to see that inclusion $Anc_0(i) \cup Anc(\beta, lastAnc(\beta,i)) \subseteq Anc(\beta,i)$ holds. For the reverse inclusion, let us consider an ancestor j of i that is not in $Anc_0(i)$. In this case, there necessarily exists a path from j to i containing one customer k belonging to β and as $G(\beta)$ is acyclic, customer k must either be equal to $lastAnc(\beta,i)$ or appear before $lastAnc(\beta,i)$ in β. In both cases, we can conclude that j is an ancestor of $lastAnc(\beta,i)$ in $G(\beta)$, which proves the reverse inclusion. □

Lemma 2. *By considering the customers in a reverse topological order of precedence graph $G(\beta)$, quantities $Desc(\beta, i)$ can be recursively defined by:*

$$Desc(\beta, i) \leftarrow Desc_0(i) \cup \{Next(\beta, i)\} \cup Desc(\beta, Next(\beta, i)) \text{ if } i \in \beta \quad (3)$$

$$Desc_0(i) \cup Desc(\beta, firstDesc(\beta, i)) \text{ otherwise} \quad (4)$$

where $Desc_0(i)$ is the set of mandatory descendants of i in the initial precedence graph G.

Proof. Similar to the proof of Lemma 1. □

2.2 Complexity of the destroy procedure

Proposition 1. *The worst-case time complexity of the destroy procedure is $O(N^3)$.*

Proof. Proof available in the main article. □

2.3 Properties of the dynamic programming equations

In the following, the set of solutions is referred to as Σ. As shown in Lemma 3 below, the path leading to a visit state satisfies some good properties with regards to the solution prefixes introduced in Definition 1.

Definition 1. (solution prefixes) *Let $\sigma = [\sigma_0, \ldots, \sigma_{N+1}]$ be a solution completing the partial solution β obtained after the destroy phase. The prefix of σ at a position $p \in [0..N+1]$, denoted by $\sigma[0 \rightarrow p]$, is the sequence $\sigma[0 \rightarrow p] = [\sigma_0, \ldots, \sigma_p]$. The set of possible solution prefixes at position p is $\Sigma(p) = \{\sigma[0 \rightarrow p] \,|\, \sigma \in \Sigma\}$ where Σ denotes the set of solutions that are completions of β.*

Lemma 3. *In the definition of the dynamic programming equations, after processing position $p \in [0..N+1]$, the path $Path(p, S, i)$ associated with a reachable state (p, S, i) is a solution prefix in $\Sigma(p)$ that visits all customers in S. Moreover, $\nu(p, S, i)$ corresponds to the tmc-evaluation of this prefix, that is $\nu(p, S, i) = Eval(Path(p, S, i))$ where $Eval()$ is defined as in Section 3 of the article.*

Proof. The proposition holds at position $p = 0$: indeed, for the unique initial visit state $(0, \{0\}, 0)$, path $Path(0, \{0\}, 0) = [0]$ is a solution prefix in $\Sigma(0)$ and we have $\nu(0, \{0\}, 0) = (0, 0, 0) = Eval([0])$. Let us now assume that the proposition holds at position $p \in [0..N]$. Let us consider a reachable visit state $(p+1, S \cup \{i\}, i)$ at position $p+1$ and its associated parent customer $j^* = \pi(p+1, S \cup \{i\}, i)$. The assumption that the proposition holds at position p implies that sequence $\sigma^* = Path(p, S, j^*)$ belongs to $\Sigma(p)$, and $\nu(p, S, j^*) = Eval(\sigma^*)$. Moreover, we can write $Path(p+1, S \cup \{i\}, i) = \sigma^* \cdot [i]$ according to the definition of a path leading to a visit state. As $(p+1, S \cup \{i\}, i)$ is a reachable visit state, i is not in S and all its ancestors belong to S, therefore $\sigma^* \cdot [i]$ is a solution prefix in $\Sigma(p+1)$. Moreover, by definition of parent customers, we have $\nu(p+1, S \cup \{i\}, i) = StepEval(\nu(p, S, j^*), j^*, i)$. As a result, we obtain $\nu(p+1, S \cup \{i\}, i) = StepEval(Eval(\sigma^*), j^*, i) = Eval(\sigma^* \cdot [i]) = Eval(Path(p+1, S \cup \{i\}, i))$. This proves that the proposition holds at position $p+1$. □

Proposition 2. *The path $Path(N+1, [0..N+1], N+1)$ leading to visit state $(N+1, [0..N+1], N+1)$ corresponds to a solution (feasible or not) that is a completion of β.*

Proof. Direct consequence of Lemma 3. □

Proposition 3. *Let us assume that the transition function satisfies the FIFO property. If there exists a completion σ of partial solution β such that $\delta(\sigma) = 0$ (null tardiness), then solution*

$$\sigma' = Path(N+1, [0..N+1], N+1)$$

is feasible and makespan-optimal among the completions of β.

Proof. Let $\mathcal{C}(\sigma)$ denote the set of customers visited by a sequence σ. Let us show that for every position $p \in [0..N+1]$, if there exists a solution prefix $\sigma \in \Sigma(p)$ such that $\delta(\sigma) = 0$, then solution prefix $Path(p, \mathcal{C}(\sigma), \sigma_p)$ is feasible and has an optimal makespan among the prefixes leading to visit state $(p, \mathcal{C}(\sigma), \sigma_p)$.

The proposition holds for $p = 0$ since the unique solution prefix $[0]$ associated with position 0 is both feasible and makespan-optimal. Assume that the proposition holds at position $p \in [0..N]$ and consider a solution prefix $\sigma \cdot [i] \in \Sigma(p+1)$ such that $\delta(\sigma \cdot [i]) = 0$. The visit states associated with $\sigma \cdot [i]$ and σ are $(p+1, \mathcal{C}(\sigma) \cup \{i\}, i)$ and $(p, \mathcal{C}(\sigma), \sigma_p)$ respectively. As $\delta(\sigma \cdot [i]) = 0$, we also have

$\delta(\sigma) = 0$. The assumption that the proposition holds at position p then entails that $Path(p, \mathcal{C}(\sigma), \sigma_p)$ is feasible and makespan-optimal among the solution prefixes whose visit state is $(p, \mathcal{C}(\sigma), \sigma_p)$. Thanks to Lemma 3, the feasibility and makespan-optimality of $Path(p, \mathcal{C}(\sigma), \sigma_p)$ imply that $\delta(p, \mathcal{C}(\sigma), \sigma_p) = 0$ and $\tau(p, \mathcal{C}(\sigma), \sigma_p) \leq \tau(\sigma)$ hold.

From this, let us prove that $\delta(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) = 0$ and $\tau(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) \leq \tau(\sigma \cdot [i])$. First, the dynamic programming equations entail the following lexicographic inequality:

$$\nu(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) \leq \mathit{StepEval}(\nu(p, \mathcal{C}(\sigma), \sigma_p), \sigma_p, i) \tag{5}$$

By defining $x = \tau(p, \mathcal{C}(\sigma), \sigma_p) + tt(\sigma_p, i, \tau(p, \mathcal{C}(\sigma), \sigma_p))$, Equation 5 allows us to write:

$$\begin{array}{rcl}
\delta(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) & \leq & \delta(p, \mathcal{C}(\sigma), \sigma_p) + \max(0, x - \mathit{End}(i)) \text{ (definition of the } \mathit{StepEval} \text{ function)} \\
& \leq & \delta(p, \mathcal{C}(\sigma), \sigma_p) + \max(0, \tau(\sigma) + tt(\sigma_p, i, \tau(\sigma)) - \mathit{End}(i)) \text{ (FIFO assumption)} \\
& \leq & \delta(\sigma) + \max(0, \tau(\sigma) + tt(\sigma_p, i, \tau(\sigma)) - \mathit{End}(i)) \text{ (since } \delta(p, \mathcal{C}(\sigma), \sigma_p) = 0 = \delta(\sigma)) \\
& \leq & \delta(\sigma \cdot [i]) \text{ (definition of the cumulated tardiness)}
\end{array}$$

This entails that $\delta(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) = 0$, therefore Equation 5 implies the following inequalities:

$$\tau(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) \leq \max(\mathit{Start}(i), x) \leq \max(\mathit{Start}(i), \tau(\sigma) + tt(\sigma_p, i, \tau(\sigma))) \leq \tau(\sigma \cdot [i])$$

To sum up, we obtain both $\delta(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) = 0$ and $\tau(p+1, \mathcal{C}(\sigma) \cup \{i\}, i) \leq \tau(\sigma \cdot [i])$, hence $Path(p+1, \mathcal{C}(\sigma) \cup \{i\}, i)$ is feasible and makespan-optimal among the solution prefixes $\sigma \cdot [i]$ leading to visit state $(p+1, \mathcal{C}(\sigma) \cup \{i\}, i)$. This means that the proposition holds at position $p+1$. The results concerning $Path(N+1, [0..N+1], N+1)$ are entailed by the satisfaction of the proposition at position $N+1$. □

2.4 From extended visit states to compact visit states

Proposition 4. *For every reachable visit state (p, S, i), if R denotes the restriction of S to $\mathcal{R}(p)$ ($R = S \cap \mathcal{R}(p)$), then we have $S = R \cup \mathcal{R}_<(p) \cup \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$. Moreover, the last customer visited in (p, S, i) always satisfies condition $i \in R \cup \{\beta_{p-|R|-|\mathcal{R}_<(p)|}\}$.*

Proof. Set S can be partitioned as $S = (S \cap \mathcal{R}) \cup_{\neq} (S \setminus \mathcal{R})$, by considering the removed customers on one side and the non-removed ones on the other side. It is then possible to show that $S \cap \mathcal{R} = (S \cap \mathcal{R}(p)) \cup (S \cap \mathcal{R}_<(p))$, since all removed customers visited in S must be visitable either at position p or strictly before position p. It can also be shown that $\mathcal{R}_<(p) \subseteq S$ holds for every reachable visit state (p, S, i), therefore we obtain $S \cap \mathcal{R} = R \cup \mathcal{R}_<(p)$. From this, we know that $S \setminus \mathcal{R}$ contains $p + 1 - |\mathcal{R}_<(p)| - |R|$ visits of non-removed customers, and as the latter are necessarily visited in the order specified by partial solution β, we have $S \setminus \mathcal{R} = \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$, therefore we obtain $S = R \cup \mathcal{R}_<(p) \cup \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$.

Second, if the last customer i visited in (p, S, i) is a removed customer, then it must belong to R (it cannot belong to $\mathcal{R}_<(p)$ since it is visited at position p). Otherwise, the last visited customer can only be customer $\beta_{p-|R|-|\mathcal{R}_<(p)|}$ to respect the precedence constraints induced by β. □

Proposition 5. *Let (p, S, i) be a visit state reached at position $p \in [1..N+1]$ and let (p, R, i) be its compact version. Then, the compact visit state associated with $(p-1, S \setminus \{i\}, \pi(p, S, i))$ is $(p-1, R', \pi(p, S, i))$ where*

$$R' = \begin{cases} (R \setminus \{i\}) \cup \mathcal{R}_{max}(p-1) & \text{if } i \in \mathcal{R}(p) \\ R \cup \mathcal{R}_{max}(p-1) & \text{otherwise} \end{cases} \tag{6}$$

Proof. The compact visit state associated with $(p-1, S \setminus \{i\}, \pi(p, S, i))$ is $(p-1, R', \pi(p, S, i))$ where $R' = (S \setminus \{i\}) \cap \mathcal{R}(p-1)$. As $S = R \cup \mathcal{R}_<(p) \cup \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$, this is equivalent to $R' = ((R \cup \mathcal{R}_<(p)) \setminus \{i\}) \cap \mathcal{R}(p-1)$. As $R \cap \mathcal{R}_<(p) = \emptyset$ and $i \in R \cup \{\beta_{p-|R|-|\mathcal{R}_<(p)|}\}$ (see Proposition 4), this is equivalent to $R' = ((R \setminus \{i\}) \cup \mathcal{R}_<(p)) \cap \mathcal{R}(p-1)$. Then, it is possible to show that we have on one hand $R \setminus \{i\} \subseteq \mathcal{R}(p-1)$ (because $R \subseteq \mathcal{R}(p)$ and all customers in $R \setminus \{i\}$ are visited at a position $p' \leq p-1$), and on the other hand $\mathcal{R}_<(p) \cap \mathcal{R}(p-1) = \mathcal{R}_{max}(p-1)$. As a result, we obtain $R' = (R \setminus \{i\}) \cup \mathcal{R}_{max}(p-1)$. □

2.5 Complexity of the repair procedure

Proposition 6. *If each call to transition function tt has a time and space complexity $O(1)$, then the repair procedure has a time complexity $O(2^{Wmax} \cdot Wmax^2 \cdot N)$ and a space complexity $O(2^{Wmax} \cdot Wmax \cdot N)$.*

Proof. Proof available in the main article. □

Proposition 7. *When the transition function satisfies the FIFO property and can be computed in polynomial time, determining whether there exists a feasible solution to (TD)TSPTWs whose insertion-width is bounded by a fixed constant W is polynomial. Moreover, if there exists a feasible solution, finding a makespan-optimal solution is polynomial as well, simply by applying the repair procedure from partial solution $\beta = [0, N+1]$.*

Proof. Direct consequence of Propositions 3 and 6. □

2.6 Path expansion conditions from a visit state

Lemmas 4 and 5 given below show how the conditions required to visit one more customer given an extended visit state (p, S, i) can be transformed into simpler conditions depending only on the basic features of the corresponding compact visit state (p, R, i). These conditions are directly reused in the pseudo-code of the repair algorithm.

Lemma 4. *Let (p, S, j) be a visit state at position $p \in [0..N]$ and let (p, R, j) be its compact version ($R = S \cap \mathcal{R}(p)$). Let R' be the set defined by $R' = R \setminus \mathcal{R}_{max}(p)$, where all customers in R for which position p is the last possible one are discarded. Then, for every removed customer $i \in \mathcal{R}$, conditions "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" are equivalent to the conjunction of the three following conditions:*

- *$i \in \mathcal{R}(p+1) \setminus R'$ (*i.e.*, i is a candidate for occupying position $p+1$ and is not visited yet in R');*
- *$\beta_{p-|R|-|\mathcal{R}_<(p)|+1} \notin Anc(i)$ (*i.e.*, the next non-removed customer to visit is not an ancestor of i);*
- *$Anc(i) \cap \mathcal{R}(p+1) \subseteq R'$ (*i.e.*, all ancestors of i that are candidates at position $p+1$ are already visited).*

When these conditions hold, visit state $(p+1, S \cup \{i\}, i)$ corresponds to compact visit state $(p+1, R' \cup \{i\}, i)$.

Proof. Let us assume that the conditions "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" hold. In this case, as all ancestors of i are included in S, we have $P_{min}(i) \leq |S| = p+1$. Moreover, by construction, no customer in $S \cup \{i\}$ can be a descendant of i, therefore $|Desc(i)| \leq N + 2 - (|S| + 1)$, which leads to $P_{max}(i) = N + 1 - |Desc(i)| \geq p+1$. The two inequalities $P_{min}(i) \leq p+1$ and $P_{max}(i) \geq p+1$ allow us to write $i \in \mathcal{R}(p+1)$.

From the previous discussion, conditions "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" can be rewritten as "$i \in \mathcal{R}(p+1) \setminus S$ and $Anc(i) \subseteq S$". Thanks to equality $S = R \cup \mathcal{R}_<(p) \cup \{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\}$ given in Proposition 4, condition $i \in \mathcal{R}(p+1) \setminus S$ can be replaced by $i \in \mathcal{R}(p+1) \setminus (R \cup \mathcal{R}_<(p))$. As $\mathcal{R}(p+1) \cap \mathcal{R}_<(p) = \emptyset$, this is equivalent to $i \in \mathcal{R}(p+1) \setminus R$. As $\mathcal{R}(p+1) \cap \mathcal{R}_{max}(p) = \emptyset$, this is equivalent to $i \in \mathcal{R}(p+1) \setminus R'$.

Condition "$Anc(i) \subseteq S$" can be split into two conditions, namely (1) "every ancestor of i that is a non-removed customer is contained in S", and (2) "every ancestor of i that is a removed customer is contained in S". As $\{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_<(p)|}\} \subseteq S$ holds and as customer $\beta_{p-|R|-|\mathcal{R}_<(p)|+1}$ is an ancestor of all remaining non-removed customers, condition 1 is equivalent to $\beta_{p-|R|-|\mathcal{R}_<(p)|+1} \notin Anc(i)$. Condition 2 can be formally stated as $Anc(i) \cap \mathcal{R} \subseteq S$, or equivalently $Anc(i) \cap \mathcal{R} \subseteq R \cup \mathcal{R}_<(p)$. As i is candidate for occupying position $p+1$, it can be shown that every ancestor j of i satisfies $P_{min}(j) \leq p$, and after a few steps we can obtain that every removed ancestor of i necessarily belongs to $\mathcal{R}_<(p) \cup \mathcal{R}_{max}(p) \cup \mathcal{R}(p+1)$. As a result, condition 2 can be replaced by $(Anc(i) \cap \mathcal{R}_<(p)) \cup (Anc(i) \cap \mathcal{R}_{max}(p)) \cup (Anc(i) \cap \mathcal{R}(p+1)) \subseteq R \cup \mathcal{R}_<(p)$. Inclusion $Anc(i) \cap \mathcal{R}_<(p) \subseteq \mathcal{R}_<(p)$ is always true, and it is possible to show that set $Anc(i) \cap \mathcal{R}_{max}(p)$ is included in R. As a result, condition 2 is equivalent to $Anc(i) \cap \mathcal{R}(p+1) \subseteq R \cup \mathcal{R}_<(p)$. As $\mathcal{R}(p+1) \cap \mathcal{R}_<(p) = \emptyset$, we obtain condition $Anc(i) \cap \mathcal{R}(p+1) \subseteq R$. As $\mathcal{R}(p+1) \cap \mathcal{R}_{max}(p) = \emptyset$, this is equivalent to $Anc(i) \cap \mathcal{R}(p+1) \subseteq R \setminus \mathcal{R}_{max}(p)$, or in other words to the third condition given in Lemma 4.

We now study the compact visit state associated with $(p+1, S \cup \{i\}, i)$. Set $\mathcal{R}(p+1)$ can be decomposed as $\mathcal{R}(p+1) = (\mathcal{R}(p) \setminus \mathcal{R}_{max}(p)) \cup \mathcal{R}_{min}(p+1)$, since the candidates for occupying position $p+1$ are either candidates at position p for which p is not the last possible position, or removed customers for which position $p+1$ is the first possible one. From this, we can write $S \cap \mathcal{R}(p+1) =$

$((S \cap \mathcal{R}(p)) \setminus \mathcal{R}_{max}(p)) \cup (S \cap \mathcal{R}_{min}(p+1))$. As $S \cap \mathcal{R}_{min}(p+1) = \emptyset$ (since $S \cap \mathcal{R} = R \cup \mathcal{R}_{<}(p)$ and $R \subseteq \mathcal{R}(p)$), we obtain $S \cap \mathcal{R}(p+1) = R \setminus \mathcal{R}_{max}(p)$. Therefore, as $i \in \mathcal{R}(p+1)$, we can write $(S \cup \{i\}) \cap \mathcal{R}(p+1) = (R \setminus \mathcal{R}_{max}(p)) \cup \{i\} = R' \cup \{i\}$. □

Lemma 5. *Let (p, S, j) be a visit state at position $p \in [0..N]$ and let (p, R, j) be its compact version ($R = S \cap \mathcal{R}(p)$). Let R' be the set defined by $R' = R \setminus \mathcal{R}_{max}(p)$, where all customers in R for which position p is the last possible one are discarded. Then, for every non-removed customer $i \in [1..N+1]$, conditions "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" are equivalent to the conjunction of two conditions:*

- $i = \beta_{p-|R|-|\mathcal{R}_{<}(p)|+1}$ *(*i.e.*, i is the next non-removed customer to visit);*
- $Anc(i) \cap \mathcal{R}(p+1) \subseteq R'$ *(*i.e.*, all ancestors of i that are candidates at position $p+1$ are already visited).*

When these conditions hold, visit state $(p+1, S \cup \{i\}, i)$ corresponds to compact visit state $(p+1, R', i)$.

Proof. If i is not a removed customer, conditions "$i \in [0..N+1] \setminus S$ and $Anc(i) \subseteq S$" can be reformulated as the conjunction of the three following conditions: (1) $i \in \beta$ and $i \notin S$, (2) all ancestors of i that are non-removed customers belong to S, and (3) all ancestors of i that are removed customers belong to S, that is $Anc(i) \cap \mathcal{R} \subseteq S$. Conditions 1 and 2 together are equivalent to $i = \beta_{p-|R|-|\mathcal{R}_{<}(p)|+1}$ since non-removed customers are totally ordered and the set of non-removed customers visited given S is $\{\beta_0, \ldots, \beta_{p-|R|-|\mathcal{R}_{<}(p)|}\}$, according to Proposition 4. Condition 3 can be transformed into $Anc(i) \cap \mathcal{R}(p+1) \subseteq R'$, as in the proof of Lemma 4. Moreover, still as in the proof of the previous proposition, we can write $S \cap \mathcal{R}(p+1) = R \setminus \mathcal{R}_{max}(p)$. As i is not a removed customer, we obtain $(S \cup \{i\}) \cap \mathcal{R}(p+1) = R \setminus \mathcal{R}_{max}(p) = R'$ in this case. □

References

[1] K. Amghar, J.-F. Cordeau, B. Gendron, A general variable neighborhood search heuristic for the traveling salesman problem with time windows under completion time minimization, Tech. rep., CIRRELT-2019-29 (2019).

[2] A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, I. Plana, Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm, Discrete Applied Mathematics 261 (2019) 28–39.

[3] G. Lera-Romero, J. J. Miranda-Bront, F. J. Soulignac, Dynamic programming for the time-dependent traveling salesman problem with time windows, URL `https://optimization-online.org/2020/01/7558/` (2020).

[4] P. Aguiar-Melgarejo, P. Laborie, C. Solnon, A time-dependent no-overlap constraint: Application to delivery problems, in: 12th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, 2015, pp. 1–17.