SEQUOIA: SEQuence-variable-based Optimization In Action for the Traveling Salesman Problem with Time Windows

Augustin Delecluse
TRAIL, ICTEAM, UCLouvain, Belgium

Pierre Schaus
ICTEAM, UCLouvain, Belgium

Pascal Van Hentenryck
Georgia Institute of Technology, USA

Abstract

The Traveling Salesman Problem with Time Windows (TSPTW) is a Vehicle Routing Problem (VRP) variant imposing time-window constraints on customer visits. The objective is to serve all customers within specified time windows while minimizing the total travel distance. Finding even a feasible solution for this problem is NP-hard. State-of-the-art local search methods typically begin with a circuit that visits all customers, possibly violating time windows, before iteratively reducing these violations through local search moves, such as shifting visits. In contrast, this paper proposes to satisfy time-window constraints while relaxing the requirement to visit all customers. It relies on constraint programming, and a model based on sequence variables for neighborhood exploration. Flexible removal and insertion-based heuristics are exploited in a large neighborhood search setting. This approach consistently yields feasible solutions in under one second for popular TSPTW benchmarks, and can be readily adapted to other routing problems.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, TSPTW, Vehicle Routing, Sequence Variables, Insertion Variables

Funding *Augustin Delecluse*: This work was supported by Service Public de Wallonie Recherche under grant n°2010235 – ARIAC by DIGITALWALLONIA4.A

1 Introduction

In numerous discrete optimization problems such as the Traveling Salesman Problem (TSP), identifying a feasible solution is often relatively straightforward, while optimizing it is the real difficulty. When time-windows constraints must be satisfied by the salesman, even finding a feasible solution becomes an NP-complete problem [20]. In the realm of discrete optimization problems with hard-to-solve feasibility constraints, a prevailing methodology involves initially procuring a feasible solution prior to engaging in further optimization, while ensuring the satisfaction of constraints. This approach facilitates the anytime proposal of feasible solutions to users. It is also noteworthy that, for some problems, the tactics employed in obtaining a feasible solution may diverge significantly from those utilized in solution improvement. For instance, one could apply a problem-specific heuristic to find a feasible solution and subsequently optimize it using a distinct strategy, such as large neighborhood search in conjunction with an exact solver such as in [7].

Frequently, the identification of a feasible solution necessitates the application of problem-specific heuristics. For the TSPTW, some of them rely on repeated insertions and path optimization [6], others solving an assignment problem [2]. A popular and effective procedure to find a feasible solution for the TSPTW rapidly is the one introduced in [3]. It is a

local search procedure applied to a complete permutation vector of all the visits, which incrementally reduces the violation of time window constraints by implementing simple shift moves, thereby relocating individual visits to alternative positions within the sequence. During stagnation, random perturbations (shaking) are introduced implementing the so-called variable neighborhood search meta-heuristic [14]. More recently, Iterated Maximum Large Neighborhood Search [19] was introduced, another method relying on the reduction of time window violations. Combining Large Neighborhood Search (LNS), Iterated Local Search, efficient preprocessing techniques and perturbation phases, this technique consistently provides feasible solutions in under one second on several TSPTW datasets.

One key factor contributing to the success of local search is the diversification of neighborhoods, which helps the search process escape local minima and converge to a globally optimal solution. However, designing intricate neighborhoods or blending these moves necessitates considerable development effort. To overcome this challenge, researchers and practitioners sometimes turn to hybrid methods, such as the LNS, which combines local search with exact methods like Constraint Programming (CP) [22]. CP is a flexible approach for modeling and solving vehicle routing problems [9]. Recently, Insertion-based Sequence Variables (SV) have been introduced for solving routing problems with CP [4, 25]. The SV methodology is particularly well suited for designing relaxation techniques based on reinsertions within the LNS framework. Unlike the conventional successor model used in CP, SVs enable the straightforward representation of optional visits within partial routes.

In this study, our primary contribution is the development of an LNS procedure that utilizes SVs to identify feasible solutions for the TSPTW. We strategically assemble existing algorithmic components and ideas to create a cohesive and efficient method for TSPTW initialization, ultimately pushing the boundaries of state-of-the-art performance of this application. Our proposed approach is described as SEQUOIA - Sequence Variable based optimization in action for the TSPTW. To achieve this, we optimize a relaxed version of the feasibility problem, where time window constraints are preserved as hard constraints, and the objective is to maximize the number of customer visits within the tour. The optimization process continues until all customers have been visited. By solving this relaxed problem optimally and ensuring all nodes are visited, we obtain a feasible solution to the original TSPTW.

The paper is organized as follows. The notations used throughout the papers are introduced in section 2. Section 3 presents the model used to find feasible solutions. Its performances are analyzed in section 4.

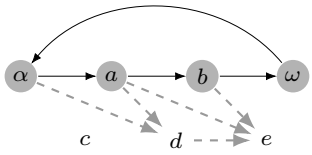
2 Modeling TSPTW With Sequence Variables

The Sequence Variable used in our approach is the one introduced in [4, 25]. We reintroduce some of their notations for the sake of completeness.

Given a set of nodes $\mathcal{X}$, an SV, written Sq, is a variable representing all possible permutations over $\mathcal{X}$. More precisely, an SV partitions $\mathcal{X}$ into three disjoint sets: nodes visited by the sequence (S), nodes that may be visited by the sequence (P) and nodes excluded from the sequence (E). The internal sequence defines an ordering over the elements from S, written $\overrightarrow{S}$. For every possible node $x \in \mathcal{X}$, the variable also represents a set of insertions I^x. This set is empty for nodes that are not possible: $I^x = \emptyset \iff x \notin P$. For a possible node $x \in P$, each element from its set of insertions I^x corresponds to a valid predecessor candidate $i \in P \cup S$. In other terms, I^x defines for a node x the set of nodes after which x may be inserted during the search.

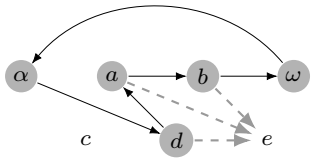
The domain of an SV is defined as $Sq = \langle \overrightarrow{S}, I, P, E \rangle$. An SV can change its state by excluding a possible node $x \in P$, moving it to the excluded set E. Alternatively, a possible node $x \in P$ can also be added to the internal sequence $\overrightarrow{S}$. In this case, the node x is inserted into the sequence Sq after one of its predecessors candidates $I^x \cap S$. This insertion operation is written $insert(Sq, p, x)$. A representation of an SV and an insertion over it can be found on Figure 1, adapted from [4]. Used in CP approaches, constraints can remove invalid insertions and exclude nodes from the sequence.

Finally, given two elements $a, b \in \overrightarrow{S}$, the notation $a \overset{\overrightarrow{S}}{\prec} b$ indicates that b follows a and $a \xrightarrow{\overrightarrow{S}} b$ that $b \in S$ directly follows a. The operation $succ(Sq, a)$ for an element $a \in S$ allows to retrieve its successor $b \in S | a \xrightarrow{\overrightarrow{S}} b$ in the internal sequence $\overrightarrow{S}$.



node x	I^x
a	ϕ
b	ϕ
c	ϕ
d	$\{\alpha, a\}$
e	$\{a, b, d\}$

$\overrightarrow{S} = [\alpha, a, b, \omega], P = \{d, e\}, E = \{c\}$



node x	I^x
a	ϕ
b	ϕ
c	ϕ
d	ϕ
e	$\{a, b, d\}$

$\overrightarrow{S} = [\alpha, d, a, b, \omega], P = \{e\}, E = \{c\}$

Figure 1 Representation of a Sequence Variable. The left part shows a sequence ordering as well as the insertions I^x (dashed lines) for nodes $x \in P$. Below is a table showing the insertions for the nodes (omitting nodes α and ω), the internal sequence $\overrightarrow{S}$ and possible P and excluded set E. The right part shows a modification after an insertion $insert(Sq, \alpha, d)$ where node d has been inserted after node α, changing its status.

The TSPTW model with a set $\mathcal{X}$ of nodes to visit, relies on the following variables:

- A Sequence Variable $Sq = \langle \overrightarrow{S}, I, P, E \rangle$ representing the sequential path taken by the salesman;
- An integer variable $Time_i$, represents the time of the visit for every node $i \in \mathcal{X}$. The initial domain of $Time_i$ is set to $[e_i \dots l_i]$, that is the time window for each node;
- An integer variable $Member$ represents the length of the sequence, that is the number of nodes visited. This is also the objective to maximize in our relaxation of the feasibility problem.

Two constraints are enforced on the problem:

TransitionTimes captures the temporal dimension of the problem [4]. It enforces that every node visited by the sequence Sq respects its time window. Its signature is given in (1):

$$\texttt{TransitionTimes}(Sq, [Time], [duration], [[dist]], SumTransition) \equiv$$
$$\left\{ \overrightarrow{S} \in D(Sq) \;\middle|\; \begin{array}{c} \forall i, j \in \overrightarrow{S}, i \overset{\overrightarrow{S}}{\prec} j \implies Time_i + duration_i + dist_{i,j} \leq Time_j \\ SumTransition = \sum_{i,j \in \overrightarrow{S} \mid i \to j} dist_{i,j} \end{array} \right\} \quad (1)$$

Where $Time$ are variables corresponding to the time window of the nodes, $duration$ to their service time, $dist$ to the distance between them and $SumTransition$ to the total traveled distance. Both $duration$ and $SumTransition$ are set to zero and unused in our model, respectively. The filtering for this constraint consists in removing insertions that would violate the time windows, possibly excluding nodes from the sequence if it is proven that they cannot be visited on time.

NMember binds $Member$ to the number of nodes visited by the sequence Sq. This constraint sets the lower bound of $Member$ to the number of nodes effectively visited $|S|$ in Sq and its upper bound to the number of nodes that are not excluded $|S| + |P|$.

3 Large Neighborhood Search

We strive to identify a feasible path for the Traveling Salesman Problem with Time Windows (TSPTW). We convert the satisfaction problem into an optimization problem by relaxing the constraint of visiting all nodes, while satisfying the time window constraints for the ones being visited. Initially, we employ a greedy approach to construct a circuit that attempts to visit as many customers as possible within their time windows, using a regret-based heuristic. If some nodes remain not visited in this first step, we then proceed to the second step, which involves a Large Neighborhood Search (LNS) method. Starting from the partial path obtained in the first step, the LNS aims to maximize the number of visited nodes (denoted as $Member$), until a path encompassing every node is established.

The two steps depend on a cost insertion heuristics c of a node $x \in P$ between two consecutive nodes p and q in the sequence, prioritizing smaller detours as per Equation (3) and increased remaining time-window flexibility, also referred to as slack, as per Equation (4). A similar cost definition is utilized in [8, 4].

$$c(p, x, q) = detour(p, x, q) - slack(p, x, q) \tag{2}$$

$$detour(p, x, q) = dist_{p,x} + dist_{x,q} - dist_{p,q} \tag{3}$$

$$slack(p, x, q) = \lceil Time_q \rceil - \lfloor Time_p \rfloor - dist_{p,x} - dist_{x,q} \tag{4}$$

3.1 Step1: Greedy Search

The objective is to build a circuit in a greedy manner that aims to visit the maximum number of customers within their respective time windows. A regret-based heuristic is employed, continuing with insertions in the sequence until no further additions can be made. This results in a partial sequence of visits, which ideally encompasses a majority of the nodes.

Given a node $x \in P$ and its insertions I^x within the sequence, the regret of x is defined as the difference between the two smallest costs c (i.e., best alternatives) based on its insertions. The regret is commonly used to guide problem solving [26, 18]. The regret calculation for our problem is described in Equation (5).

$$regret(Sq, x, c) = c(p_2(x), x, succ(Sq, p_2(x))) - c(p_1(x), x, succ(Sq, p_1(x))) \tag{5}$$

$$p_1(x) = \operatorname*{argmin}_{p \in I^x \cap S} c(p, x, succ(Sq, p)) \tag{6}$$

$$p_2(x) = \operatorname*{argmin}_{p \in I^x \cap S \setminus \{p_1(x)\}} c(p, x, succ(Sq, p)) \tag{7}$$

If no valid predecessor can be derived from either (6) or (7), the corresponding term in (5) is replaced by a large constant. The regret is used to expand a path, one node at a time, until it can no longer be extended. The complete algorithm is given in the recursive GreedySearch. As the TSPTW formulation imposes a node $x' \in \mathcal{X}$ to start from and return to, the algorithm is initially invoked with a sequence containing x' and a duplicate of x', that are set as starting and ending depots.

Procedure GreedySearch(Sq, c)

Input: Sequence $Sq = \langle \overrightarrow{S}, I, P, E \rangle$, cost function c
Output: Extended Sequence Sq
$x \leftarrow \text{argmax}_{p \in P}\, regret(Sq, p, c)$
if $x = None$ **or** $I^x \cap S = \emptyset$ **then**
  return Sq
$pred \leftarrow \text{argmin}_{p \in I^x \cap S}\, c(p, x, succ(Sq, p))$
$insert(Sq, pred, x)$
return GreedySearch(Sq, c)

3.2 Step2: Augmenting the Number of Visits With LNS

The path found by the greedy search in section 3.1 may fail to visit all the nodes. If so, a second procedure relying on an LNS is used. It consists of a succession of relaxations and search tree explorations.

Initially, a non-visited node $x \in E$ is selected at random, with the aim of inserting it into the sequence during the next LNS iteration. To facilitate this, several additional nodes in the sequence are removed based on their similarity to the chosen node. These nodes are identified by adapting the relatedness function proposed by Shaw in [21]. The relatedness $\mathcal{R}(i, j)$ between a pair of nodes $i, j \in \mathcal{X}$ is determined by calculating the difference in distance between the nodes and the difference between their time windows. Adapted from vehicle routing problems [21], this measure is outlined in Equation 8 where ϕ and ξ are constant values.

$$\frac{1}{\mathcal{R}(i,j)} = \phi \frac{dist_{i,j}}{d_{max}} + \xi \left(\left| \frac{e_i - e_j}{e_{max}} \right| + \left| \frac{l_i - l_j}{l_{max}} \right| \right) \tag{8}$$

The values relative to the distances, start time and end time are normalized by the maximum distance d_{max}, maximum start time e_{max} and maximum end time l_{max}, respectively. This approach enables us to maintain values below 1, preventing an excessive focus on one aspect of the problem, such as when time windows are very large. The larger the relatedness, the more similar the nodes.

After the nodes have been relaxed, a tree depth first-search is performed with CP, aiming to extend the partial sequence in order to get a solution visiting more nodes. The branching consists in selecting a free node $x \in P$ and creating one branch for each possible insertion in the sequence, sorted according to the cost heuristic c. The free-node selection is a first-fail decision selecting node x having the least possible insertions $I^x \cap S$, breaking ties randomly.

An important ingredient of the LNS to be successful is the number of nodes to relax. We reuse the scheme proposed in [8] for the Dial-A-Ride problem that consists in gradually augmenting the number of relaxed nodes. The complete LNS is presented in Algorithm 1. It

stops when all nodes from the problem have been visited, or when a time limit is reached. In the algorithm, multiple constants are present. The parameters n_{min} and n_{max} dictate the range of neighborhood sizes, commencing with n_{min} and progressively expanding until they reach $n_{max} - \delta$ in the outer loop. The parameter δ regulates the exploration of a variation window starting at i and n_{iter} controls the number of intensification iterations.

Algorithm 1 Large Neighboorhod Search

Input : Sequence Sq, number of visited nodes $Member$, first path $initSol$
Output : Solution $bestSol$ visiting as many nodes as possible

$bestSol \leftarrow initSol$
for $i \in \{n_{min} \dots (n_{max} - \delta)\}$ **do**
  if $i = n_{max} - \delta$ **then**
    $i \leftarrow n_{min}$
  for $j \in \{0 \dots (\delta - 1)\}$ **do**
    for $k \in \{1 \dots n_{iter}\}$ **do**
      select a node x not visited in $bestSol$, randomly.
      relax $i + j$ nodes similar to x in $bestSol$ according to (8)
      $sol \leftarrow CP.dfsOptimise(Member)$
      if $Member$ has been improved **then**
        $bestSol \leftarrow sol$
      if time limit **or** feasible path is constructed **then**
        return $bestSol$
return $bestSol$

4 Experiments

To assess the performances of SEQUOIA, we run experiments on the benchmark from [12]. It covers six datasets, where the number of nodes in the instances varies up to 232:

AFG includes 50 instances, proposed in [1].
Dumas contains 135 instances, introduced in [5].
GendreauDumasExtended has 130 instances, presented in [6]. They were constructed based on the instances from the Dumas dataset [5].
OhlmannThomas inludes 25 instances, proposed in [15].
SolomonPesant contains 27 instances [16], derived from a vehicle routing problem [24].
SolomonPotvinBengio has 30 instances [17], also derived from [24] but different from the ones belonging to the SolomonPesant dataset.

The experiments reported in this section were conducted on a computer with two Intel®Xeon®CPU E5-2687W and 128 GB of RAM. The implementation was done in Java using the MiniCP solver [13]. We first analyze the greedy search before diving into the LNS.

4.1 Greedy Search

Figure 2 shows how many nodes are visited after the first step, that is the regret-based greedy search. The smallest percentage of visited nodes 62.5% was found on an instance with 80 nodes. On 93 instances, the greedy search was able to find a feasible path, preventing the need to use an LNS. Moreover, we can observe that the difficulty of finding a feasible path changes between datasets and cannot be only expressed through the number of nodes.

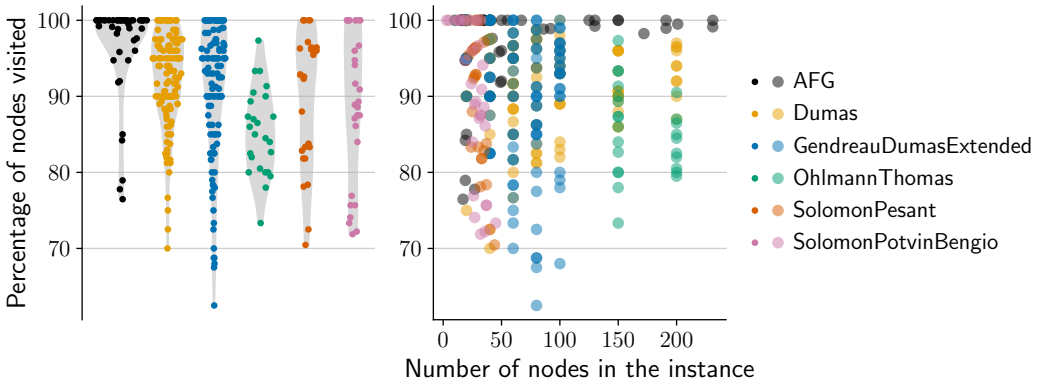


Figure 2 Percentage of nodes visited in the path constructed by the greedy search. The left figure shows the observed percentage as a SinaPlot [23], representing individual observations over the instances as dots, as well as density estimates. The instances datasets are highlighted by colors.

4.2 Feasible Path

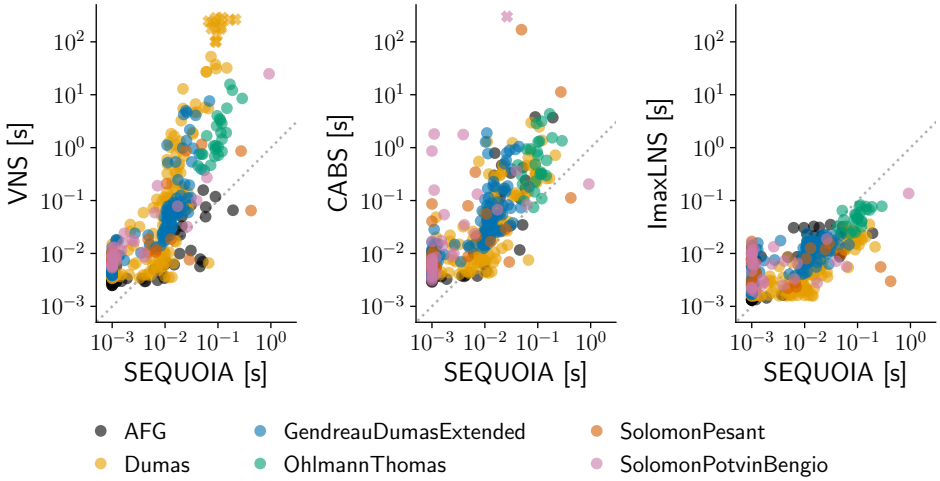


Figure 3 Comparison of the execution time between our approach (SEQUOIA) and VNS (left), CABS (middle) and ImaxLNS (right). Each dot corresponds to the mean run time on 100 experiments to find a feasible solution on one given instance. The instances datasets are highlighted by colors. The diagonal indicates that the two approaches need an equal amount of time. A dot above the diagonal means that SEQUOIA was faster. A cross indicates that a least 1 out of the 100 runs resulted in a timeout, assigning the corresponding execution time of the run to the value of the timeout.

The parameters used in the experiments, in equation 8, were set to $\phi = 9$ and $\xi = 3$. Moreover, the values introduced in Algorithm 1 were $n_{min} = 5$, $\delta = 5$, $n_{iter} = 3$, $n_{max} = \max(n/2 - \delta, n - \delta)$. The chosen parameter values have proven to be effective in practice, although a comprehensive set of experiments to identify the optimal values has not been conducted.

We evaluated SEQUOIA against two local search approches: VNS [3] and ImaxLNS [19], as well as against a dynamic programming approach - CABS [11, 27]. Figure 3 illustrates

the discrepancies in execution time among the techniques, using 100 runs per instance to account for the inherent randomness in the algorithms and to take less into account the noise induced by eventual background processes running on the machine. A timeout was set to 3 minutes. When comparing to ImaxLNS, SEQUOIA proves to be slightly slower on the majority of instances. Compared to VNS and CABS, in the majority of cases, our approach demonstrates superior speed. Even in instances where our method lags, the average execution time remains under one second, while the 2 last mentioned techniques can take several minutes or even experience a timeout in some instances.

It is worth noting that, although SEQUOIA is not always the fastest option, it relies on a general purpose CP solver, and can thus be utilized in other scenarios involving more constraints. This can for instance be used for online routing problems, such as on-demand transportation, where new requests arrive over time. In such scenarios, adding a new detour into the travel is easily done when using SEQUOIA, relying on the maximization of visited nodes, compared to other approaches minimizing overtime penalties.

4.3 Conclusion

We have introduced the SEQUOIA approach for efficiently identifying feasible solutions to the TSPTW. This method employs a regret-based greedy heuristic to initially determine a satisfactory partial path that adheres to the time-window constraints. The partial path is incrementally enhanced through LNS, incorporating crucial existing components for relaxation, branching heuristics, and an adaptive LNS scheme. By addressing a relaxation of the problem, our approach quickly finds solutions that outperform the best existing methods.

The proposed technique can be readily adapted to multi-vehicle problems such as in [10]. In fact, Algorithm 1 is unaware of the problem's constraints or the vehicles associated with relaxed nodes.

For future research, we aim to apply this methodology to online vehicle routing problems, such as on-demand transportation. In these scenarios, new transportation requests (i.e., problem nodes) may emerge over time. The goal is to visit nodes as they appear and rapidly adjust the traveler's path accordingly. Since our LNS is based on node insertions, it can efficiently assess whether a newly introduced node can be incorporated into the vehicle's route.

References

1 Norbert Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, 1996.

2 Roberto Wolfler Calvo. A new heuristic for the traveling salesman problem with time windows. *Transportation Science*, 34(1):113–124, 2000.

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

4 Augustin Delecluse, Pierre Schaus, and Pascal Van Hentenryck. Sequence variables for routing problems. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

5 Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

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

7 Xavier Gillard and Pierre Schaus. Large neighborhood search with decision diagrams. In *International Joint Conference on Artificial Intelligence*, 2022.

8 Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 400–413. Springer, 2011.

9 Philip Kilby and Paul Shaw. Vehicle routing. In *Handbook of Constraint Programming*, volume 2, pages 801–836. Elsevier, 2006.

10 Antoon WJ Kolen, AHG Rinnooy Kan, and Harry WJM Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2):266–273, 1987.

11 Ryo Kuroiwa and J Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. 2023.

12 Manuel López-Ibáñez. Instances for the TSPTW, Sep 2020. [Online; accessed 15. Feb. 2022]. URL: `https://lopez-ibanez.eu/tsptw-instances`.

13 L. Michel, P. Schaus, and P. Van Hentenryck. Minicp: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021. `doi:10.1007/s12532-020-00190-7`.

14 Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

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

16 Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1):12–29, 1998.

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

18 Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

19 Cédric Pralet. Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version. *Computers & Operations Research*, 150:106078, 2023.

20 Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

21 Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 46, 1997.

22 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98: 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings 4*, pages 417–431. Springer, 1998.

23 Nikos Sidiropoulos, Sina Hadi Sohi, Thomas Lin Pedersen, Bo Torben Porse, Ole Winther, Nicolas Rapin, and Frederik Otzen Bagger. Sinaplot: an enhanced chart for simple and truthful representation of single observations over multiple classes. *Journal of Computational and Graphical Statistics*, 27(3):673–676, 2018.

24 Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

25 Charles Thomas, Roger Kameugne, and Pierre Schaus. Insertion sequence variables for hybrid routing and scheduling problems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 457–474. Springer, 2020.

26 Frank A Tillman and Thomas M Cain. An upperbound algorithm for the single and multiple terminal delivery problem. *Management Science*, 18(11):664–682, 1972.

27 Weixiong Zhang. Complete anytime beam search. In *AAAI/IAAI*, pages 425–430, 1998.