ISSN: 3066-7194
Research Article
Novel Journal of Applied Sciences Research

Solving the Time-Dependent Vehicle Routing Problem with Hard Time Windows using the Saving Ant Colony Algorithm

Uri Lipowezky*

Gett Delivery, 19 HaBarzel Street, Tel Aviv, Israel

***Corresponding author:** Uri Lipowezky, Gett Delivery, 19 HaBarzel Street, Tel Aviv, Israel.

Submitted: 06 June 2025 **Accepted:** 12 June 2025 **Published:** 15 September 2025

https://doi.org/10.63620/MKNJASR.2025.1054

Citation: *Lipowezky, U. (2025). Solving the Time-Dependent Vehicle Routing Problem with Hard Time Windows using the Saving Ant Colony Algorithm. Nov Joun of Appl Sci Res, 2(5), 01-35.*

Abstract

The present study considers a method of parcel dispatching based on a solution of a capacitated vehicle routing problem with hard time windows per customer (no waiting is available) and an auto-updated static time-dependent traffic model. The static traffic model update is based on the actual duration from historical delivery data and the couriers' location data, which comes from their mobile devices' global positioning systems (GPS). The solution to the vehicle routing problem is based on a two-stage algorithm: obtaining an initial feasible (greedy) solution at the first stage and sequentially improving this initial solution at the second stage. The improvement of the initial feasible solution is based on the combination of selecting the appropriate simulated annealing (SA) temperature in the SA process and applying a saving matrix-based ant colony optimization (ACO) algorithm, which is accomplished using the Ruin and Recreate (R&R) method. This research aims to enhance existing dispatching systems by reducing the number of vehicles as the primary objective and minimizing the total route duration for the minimum number of available vehicles as the secondary objective. The application to real industrial delivery tasks shows that the proposed approach is highly effective.

Keywords: Capacitated Vehicle Routing Problem, Time-Dependent Travel Times, Time Windows, Static Traffic Model, Ant Colony Optimization, Ruin and Recreate Strategy

Introduction

The Vehicle Routing Problem (VRP) determines a set of vehicle routes originating and terminating at a single depot, such that all customers are visited exactly once, and the total demand of the customers assigned to each route does not exceed the vehicle's capacity.

Time windows are imposed for customer destinations, meaning that the vehicle is only permitted to arrive at the customer's destination within a specific time window and stay at the customer site for the duration of the customer's service time. Sometimes, when waiting is permitted (soft time window), the vehicle can arrive before the time window and wait until the early bounce of the time window (waiting before the delivery) or wait at the previous customer's site to arrive at the next customer on its early bounce of the time window (waiting after the delivery). However, in megapolises, early arrival causes a problem due to the unavailability or high cost of parking for waiting and is strictly forbidden; therefore, hard or strict time window constraints or VRPHTW are imposed [1-3].

In practical applications, the traversal times between the customers are not time-invariant but may vary due to traffic congestion [4]. In this case, we have a time-dependent vehicle routing problem with time window constraints (TDVRPTW) [69], two primary components contribute to variability in travel times. The first component is derived from hourly, daily, weekly, or seasonal deviations from the average traffic volumes. The second

component of travel time variability stems from random events, including accidents, weather conditions, or other unforeseen circumstances. No one can foresee the second one while the first one systematically occurs. It has been demonstrated that most observed delays are dependent on the first or static traffic component. This static or deterministic traffic model is widely used in the industry [5].

Additionally, the neighbor time intervals are merged into time buckets [6]. For example, the same traffic is observed on Sundays between 2:00 AM and 4:00 AM. Eventually, the traffic data is represented as a multi-layer travel time matrix, with one layer for each time bucket [7].

The literature survey presents many exact and heuristic solutions for solving VRP and its extensions. One of the approaches, Ant Colony Optimization (ACO), is a population-based metaheuristic that has been successfully applied to solving the TDVRPTW [8, 9, 3]. In this study, the ACO approach is adopted in combination with the saving principle to solve the TDVRPTW with hard time windows efficiently [10].

Problem Description

Following traditional flow-arc formulation, the time-dependent vehicle routing problem with hard time windows studied in this research can be described as follows. Let us denote $G = (V, A)$ a complete directed graph with n + 1 nodes, where the node-set $A = (v_i, v_j): i \neq j$ and $^{i,j \in V}$ is an arc set [11, 1]. The vertex set $V = \{v_0, v_1, \dots, v_n, v_{n+1}\}$ includes the depot associated with nodes v_0(start node) and v_{n+1} (return node) and the set of customers $C = \{v_1, \dots, v_n\}$, which must be visited. It is supposed that there is a fleet of K available homogeneous vehicles of capacity q_{max}, and each vehicle is located at the depot. Each vehicle starts a tour from the depot v_0 and must either return to the depot v_{n+1}, if the tour is closed, or finish with the last customer, if the tour is open (Open VRP case) [12].

Each vertex in V has an associated demand $q_i > 0$, a service time $g_i > 0$, and a service time window $[e_i, l_i]$ when the location ought to be visited. The depot also has a loading time (Pan et al., 2021) [13] $g_0 > 0$, the time needed to load commodities to a vehicle, and the scheduled time e_0, denotes the time when the commodities are available at the depot. The arrival time of a vehicle at customer $i \in C$ is denoted a_i and its departure time b_i.Each arc has an associated travel distance $d_{ij}(b_i) > 0$ and a travel time $t_{ij}(b_i) > 0$, depending on the departure time between location $i \in V$ and the destination $j \in V$. It is supposed that a courier reward is significantly higher than fuel expense, so the fastest path between two locations (v_i, v_j) is used. In this case, the travel distance also depends on the departure time. For example, suppose a depot is located in the industrial zone, and a customer is in the residential area. In this case, for delivery at 15:00, before the pick time, a path will be taken, including a highway segment with a travel distance of 20 km and a traveling time of 20 minutes. However, for the same delivery at 17:00, due to traffic congestion on the highway at the pickup time, the route will be rerouted through city streets, resulting in a 10 km travel distance and a 40-minute travel time. The cost per kilometer traveled is denoted c_d, and the cost per hour of the route duration is denoted c_t. Besides, to ensure the welfare and safety of couriers, the total trip duration per vehicle is limited by the maximum working shift CS_{max} and the maximum number of stop points on each tour N_{max} [13].

We assume that a feasible solution exists, i.e., it is always possible to serve any customer starting from the depot within its time window:

$$e_j \leq e_0 + g_0 + t_{0j}(e_0 + g_0) \leq l_j, \quad \forall j \in C, \tag{1}$$

where $t_{0j}(e_0 + g_0)$ is the travel time from the depot to customer j at the time of $e_0 + g_0$. Let us denote x_{ij}^k a binary decision variable that indicates whether vehicle $k \in K$ travels between customers i and j. The primary objective function is the minimization of the number of routes, meaning:

$$\min \sum_{k \in K} \sum_{j \in C} x_{0j}^k \tag{2}$$

$$x_{ij}^k \in \{0, 1\}, \ \forall i, j \in A, \ \forall k \in K \tag{3}$$

Since V_0 denotes the start depot and v_{n+1} the final depot, then

$$x_{i0}^k = 0 \text{ and } x_{n+1,i}^k = 0, \forall i \in V, \ \forall k \in K \tag{4}$$

Let us denote another decision variable y_i^k, which indicates the expected arrival time (ETA) for customer i served by vehicle k. The secondary objective function is to minimize the price for the minimal number of vehicles. With these notations, the secondary objective function can be written as follows:

$$\min c_d \sum_{k \in K} \sum_{i,j \in C} d_{ij}^k x_{ij}^k + c_t \sum_{k \in K} \sum_{j \in C} x_{0j}^k \left(y_{n+1}^k - y_0^k\right), \tag{5}$$

subject to:

$$\sum_{i \in C} q_i \sum_{j \in V} x_{ij}^k \leq q_{max}, \ \forall k \in K \tag{6}$$

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1, \ \forall i \in C \tag{7}$$

$$\sum_{i \in V} x_{il}^k - \sum_{j \in V} x_{lj}^k = 0, \forall l \in C, \ \forall k \in K \tag{8}$$

$$\sum_{j \in V} x_{0j}^k = 1, \text{ and } \sum_{i \in V} x_{i,n+1}^k = 1, \forall k \in K \tag{9}$$

$$e_i \sum_{j \in V} x_{ij}^k \leq y_i^k, \forall i \in C, \ \forall k \in K \tag{10}$$

$$l_i \sum_{j \in V} x_{ij}^k \geq y_i^k, \forall i \in C, \ \forall k \in K \tag{11}$$

$$y_j^k = y_i^k + g_i + t_{ij}\left(y_i^k + g_i\right), \forall i, j \in A, \ \forall k \in K \tag{12}$$

$$\sum_{j \in C} x_{0j}^k \left(y_{n+1}^k - y_0^k\right) \leq CS_{max}, \quad \forall k \in K \tag{13}$$

$$\sum_{i,j \in C} x_{ij}^k + 1 \leq N_{max} \quad \forall k \in K \tag{14}$$

The primary and secondary objectives are defined in (2) and (5). The constraints are defined as follows: vehicle capacity cannot be exceeded (6); all customers must be visited only once (7); if a vehicle arrives at a customer, it must also depart from that customer (8). Each vehicle departs from and returns to the depot only once (9); ETA must satisfy time windows early (10) and late (11) times, obey the traffic equation (12), and not violate the working regulations (13) and (14). Unlike previous formulations for TDVRPTW (Figliozzi, 2012), no waiting is allowed in this case [1].

Literature Review

To the author's knowledge, the first reference to a time-dependent travel time model belongs to [14]. The author adapted the saving algorithm to consider two periods of the planning horizon with different values of travel times. Later, the time-dependent VRP (TDVRP) without time windows (TW) was first formulated by Malandraki (1989) [68] and Malandraki and Daskin (1992) [69]. In these studies, the travel times are computed using step functions. Nearest-neighbor (greedy) heuristics for the TDTSP and the TDVRP without time windows and a branch-and-cut algorithm are proposed based on the mixed linear programming formulation for solving small-scale problems with 10–25 nodes. A study (Hill & Benton, 1992) [64] considered a node-based time-dependent vehicl routing problem (without time windows). They proposed a modeling approach in which each node was assigned a time-dependent piecewise constant speed function. Then, at each edge, the travel time duration was derived from the average speed of the incident nodes. Computational results for one vehicle and five customers were reported. The time windows and the time-dependent travel time model were first introduced by Ahn and Shin (1991) [15]. They discussed modifications to the savings, insertion, and local improvement algorithms to better deal with TDVRPTW. They reported computation time reductions in randomly generated instances as a percentage of the "unmodified" savings, insertions, and local improvements achieved by these algorithms. A study by Malandraki and Dial (1996) [17] proposed a restricted dynamic programming algorithm for a time-dependent traveling salesman problem (TSP), specifically for a fleet of one vehicle with a given scheduling time when the delivery is ready for dispatching and a constant service time at each customer. They reported on solving randomly generated problems for up to 55 customers.

An important property for time-dependent problems is the First–In–First–Out (FIFO) property [15-17]. A model with a FIFO property ensures that if a vehicle travels between two locations, a later departure cannot result in an earlier arrival at the destination. The formal definition given in is as follows:

$$If\ a_i \le b_i, then\ a_i + t_{ij}(a_i) \le b_i + t_{ij}(b_i), \qquad \forall i, j, a_i, b_i \quad (15)$$

The FIFO property is kept whenever the shortest path between two locations is selected. However, the FIFO assumption is not necessarily satisfied if the fastest route is selected between two locations for distance d_{ij} and duration t_{ij} estimations. Ahn and Shin (1991) [15] demonstrated that real traffic does not always maintain the FIFO property, as a courier prefers the fastest route over the shortest one.

Ichoua et al. (2003) [16] introduced the Ichoua-Gendreau-Potvin (IGP) traffic model, guaranteeing the FIFO property. They proposed a tabu search solution method, based on the work of Taillard et al. (1997) [22], to solve the time-dependent vehicle routing problem (VRP) with soft time windows. Instead of two objective functions, (2) and (5), they deal only with one objective function, which consists of the sum of total travel time plus penalties associated with the sum of lateness. At the same time, the early arrival, causing the waiting, is not penalized. They showed that ignoring time dependency. i.e., using VRP models with constant speed leads to poor solutions. Ichoua et al. (2003) [16] tested their method using a set of 56 problems from Marius Solomon (1987) [37] with three different traffic scenarios. Each scenario in the IGP model is described with a 3x3 time-dependent travel speed matrix, where each row corresponds to a category of arc and each column to a time interval.

Contribution, (Fleischmann et al., 2004) [18] however, pointed out that the IGP model relies on constant edge distances, which is a hypothesis suitable for road networks but not for VRP, where links between customers represent the fastest paths, which change due to traffic congestion during rush hours (Fleischmann et al., 2004) [18]. Typically, couriers get these paths online from a navigation system. The authors created a non-FIFO (non-passing condition) traffic model from Berlin travel time data and solved incapacitated TDVRP with and without time windows. A study proposed a solution to minimize the sum of costs associated with the number of vehicles, distance, duration, and lateness [19]. They used a genetic algorithm to solve the problem up to 30 stop points.

Donati et al. (2008) [9] presented an Ant Colony System (ACS) for the first time to solve the TDVRP with hard time windows (TW) constraints, allowing, however, waiting at the site before delivery. While adopting the FIFO traffic model, they employed two objective functions: fleet size (the primary objective) and total route duration (the secondary objective). The algorithm was applied to a real road network in the Padua logistics district in the Veneto province of Italy. Balseiro et al. (2008) [43] suggested improving the algorithm of Donati et al. (2008) [9] on the Solomon (1987) [37] benchmarks, utilizing aggressive insertion heuristics that rely on the minimum delay metric, which is combined with ACS to fill the gap. The authors studied dynamic traffic models with 10-minute intervals (144 time zones per day) and used TABU-search (TS) to solve the problem [20]. The algorithm was tested on the Augerat et al. (1998) [57] benchmark with different time-dependent speed models. Maden et al. (2010) [23] developed the LANTIME algorithm, which minimizes CO_2 emissions or total route duration. The algorithm is based on parallel insertion and neighborhood moving operations [21, 22]. The LANTIME algorithm was applied to the South West England truck traffic with 15-minute intervals (time band) or 672 distance-duration matrices for the entire week. The route between any two points never changes to ensure the FIFO property. Ehmke and Mattfeld (2012) [29] used Taxi drivers' GPS data (Floating Car Data (Taxi-FCD)) from Stuttgart, Germany, to create 24x7 weekly time buckets. They applied k-means clustering to distinguish between 6 kinds of traffic models and then used the LANTIME algorithm [23].

Figliozzi (2012) [1] presented an Iterative Route Construction and Improvement (IRCI) solution for hard and soft time windows. He also introduced the TD benchmarks, based on the instances of Solomon (1987) [37] and different travel time distributions, adapted to the FIFO property, over the delivery planning horizon. Additionally, some genetic algorithms have been proposed (Kumar & Panneerselvam, 2017) [67], as well as meta-heuristics [24]. More detailed information about TDVRPTW is specified in the review by Michel Gendreau et al. (2015) [4]. Recently, Teng et al. (2024) [75] successfully demonstrated the advantages of ACS for solving the dynamic VRPTW on the modified Solomon (1987) [37] benchmark.

There are various exact algorithms for small-scale tasks with a

small number of stop points (up to 40) [25-27]. Some of these works do not ensure that the FIFO property is observed. The latter study introduces the makespan secondary objective: to finish the vehicle route as early as possible, in contrast to the minimal duration route [26-28]. The following example illustrates the difference between the two objectives. Let us suppose that the traffic model is described by time buckets, such as the Vienna multi-layer travel times matrix model, and the earliest depot departure time is 7:00 AM [6]. Suppose the depot-to-customer duration is 65 minutes at 7:00 and 40 minutes at 10:00, and the customer time window is [8:00, 11:00]. In this case, the optimal makespan solution is for the depot to depart at 7:00 and for the customer to arrive at 8:05. The total duration is 1 hour and 5 minutes. The courier is free at 8:05. The optimal minimal duration solution is the depot departure at 10:00 and the customer arrival at 10:40, with a total duration of 40 minutes, but the courier is free only at 10:40. In this paper, the makespan objective, which is commonly accepted in the industry is studied [26]. Furthermore, the IGP model has not been used since, according to Mancini (2014) [5], it is a substantial simplification that does not accurately represent real urban networks.

Static Traffic Model

All researchers universally accept that there is no standard for a static traffic model; however, each city has its specific model. Kok et al. (2012) [66] described the TIGER/Line speed model for Rhode Island, Connecticut, Maryland, Massachusetts, and New Jersey. In this model, every road belongs to a category with a corresponding average normalized speed. There are three traffic time buckets: morning pick hours (6:30 – 9:30), evening pick hours (15:30-19:00), and out-of-pick hours. Additionally, each road has a degree of urbanization, which affects the relative speed drop and direction of commuter traffic towards urban areas and industrial zones. Approximately 15,000 to 38,000 nodes are used to describe the city and define the degree of urbanization, but the model does not meet the FIFO property. Simona Mancini (2014) [5] describes an average speed model in Torino, Italy, using a sixth-degree polynomial function. This function approximates the traffic data collected for working weekdays from 7:00 until 20:00 with 5-minute intervals. This model also does not guarantee that the FIFO property is respected.

The study is based on the street network of Vienna, which includes 70,775 edges [6]. The travel times are derived from Floating Car Data (FCD), which is provided by a fleet of taxis in Vienna for each 15-minute time slot. Eventually, there are 96 buckets for a 24-hour planning horizon. The travel speed is artificially adjusted to match the travel times, ensuring the FIFO property is maintained. Ehmke and Mattfeld (2012) [29] adopted the same model for traffic congestion modeling in Stuttgart, Germany, having 1,147,776 edges. They used 24x7 = 168 buckets to describe the average weekly traffic. Lombard et al. (2018) [7]describe a time-dependent traffic model for the urban area of Paris, utilizing a multi-layer travel time matrix. The data for the matrix is collected from Google Maps with a two-week accumulation interval, and the planning horizon is from 8:00 to 20:00 with time steps of two hours.

This paper adopts an edge-based, time-dependent FCD traffic model [29, 6]. The data are collected from taxis and couriers from mobile GPS devices with a resolution of one second. The traffic of two cities is modeled: Tel Aviv, Israel, with 78,221 edges, and Moscow, Russia, with 1,252,963 edges. In contrast to FCD, the model also includes traffic lights and junction edges in all possible directions with zero distance and an average waiting time, which is also included in the total travel time. These data undergo smoothing and alignment to the OpenStreetMap (OSM) graph using the Kalman filter. This alignment begins with the calculation of a priori probabilities for each junction passing, using the frequencies from the historical data collected. The rough data from the GPS devices is generally too noisy to reconstruct a valid vehicle path passing through the OSM edges and intersections. These probabilities are used to create the discrete-time Markov chain model [30]. They are further utilized for optimal matching between a courier or taxi driver's location points and OpenStreetMap (OSM) edges.

For each edge of the OSM graph, which is the segment between two junctions, the triple $(m_{it}, \sigma_{it}, n_{it})$ is collected. In this triple m_{it} is the average duration for edge *i* at timestamp t, σ_{it} it is the standard deviation of the duration for edge *i* at timestamp *t*, and n_{it} is the number of couriers that passed edge *i* at timestamp *t*. The timestamp *t* consists of the integer starting hour, related to the starting time for the edge *i* and the weekday number for the original day of the week or code of the holiday if there is a holiday on the date of *t*. Lombard et al. (2018) [7] suggested that 168-240 initial timestamps form a multi-layer travel times matrix.

Table 1: An example of time-bucket merging between single timestamps

Weekday/Hour	Monday	Tuesday	Wednesday	Thursday
7:00		Neighbor - 1		
8:00	Neighbor - 4	Tested Timestamp	Neighbor - 2	
9:00		Neighbor - 3		

If some edges lack sufficient data for a few timestamps, they are merged into integrated time buckets [6, 7]. This merging reduces the prediction error caused by the lack of data and decreases the number of time buckets the system needs to save for traffic duration prediction. The rationale for the merging is to partition the planning horizon into T time intervals with a constant duration on every edge. Let us consider four neighbors of each at timestamp, as shown in Table 1. For weekdays, these neighbors are the and timestamps, which are ±24 hours or ±1 hours different from the tested timestamp. For holidays, there is ±1 hour. For the weekend days, there are ±24 hours. To prove the statistical equivalence between the tested timestamp T_t and its neighbor timestamp T_n, the Student T-test is applied as follows.

Table 2: An example of time-bucket merging between the cluster and single timestamps

Weekday/Hour	Monday	Tuesday	Wednesday	Thursday	Friday
7:00		Neighbor - 1			
8:00	Neighbor - 6	Cluster C	Neighbor - 2		
9:00	Neighbor - 5		Neighbor - 3		
10:00		Neighbor - 4			

For each edge i, having $\min(n_{iT_t}, n_{iT_n}) \geq 7$, the absolute average difference $\Delta M = |m_{iTr} - m_{iTn}|$ and variance $V = \frac{\sigma_{iT_t}^2}{n_{iT_t}} + \frac{\sigma_{iT_n}^2}{n_{iT_n}}$ are calculated.

Using these values, the two-tailed significance of the T-test is calculated as $p = T^{-1}\left(\frac{\Delta M}{\sqrt{V}}, n_{iT_t} + n_{iT_n} - 2\right)$, where $T^{-1}(z, df)$ is the inverse Student distribution with df degrees of freedom. If $p > 0.05$, the two edges are considered homogeneous.

The ratio of homogeneous edges is calculated as $\eta = \frac{N_h}{N_t}$, where N_h is the number of homogeneous edges, and N_t is the total number of the tested edges. This ratio measures the homogeneity between two timestamps.

The timestamps, having the highest homogeneity ratios, are merged and a joint triple: $\{m_{ic}, \sigma_{ic}, n_{ic}\}$ is calculated as follows:

$$n_{ic} = n_{iT_t} + n_{iT_n}, \; m_{ic} = \frac{n_{iT_t} m_{iT_t} + n_{iT_n} m_{iT_n}}{n_{ic}}, \; \sigma_{ic} = \sqrt{\frac{S_2 - n_{ic} m_{ic}^2}{n_{ic}}},$$
$$\text{where } S_2 = n_{iT_t}\left(m_{iT_t}^2 + \sigma_{iT_t}^2\right) + n_{iT_n}\left(m_{iT_n}^2 + \sigma_{iT_n}^2\right). \quad (16)$$

The timestamp clusters are merged if the homogeneity ratio η is more than 0.75. DBSCAN clustering is processed at the next stage, and new cluster C is tested to be merged with all its four neighbors, as shown in Table 2 [31]. The process is repeated until a timestamp or cluster is merged. Figure 1 shows the final merged time buckets for weekdays in Moscow, Russia.

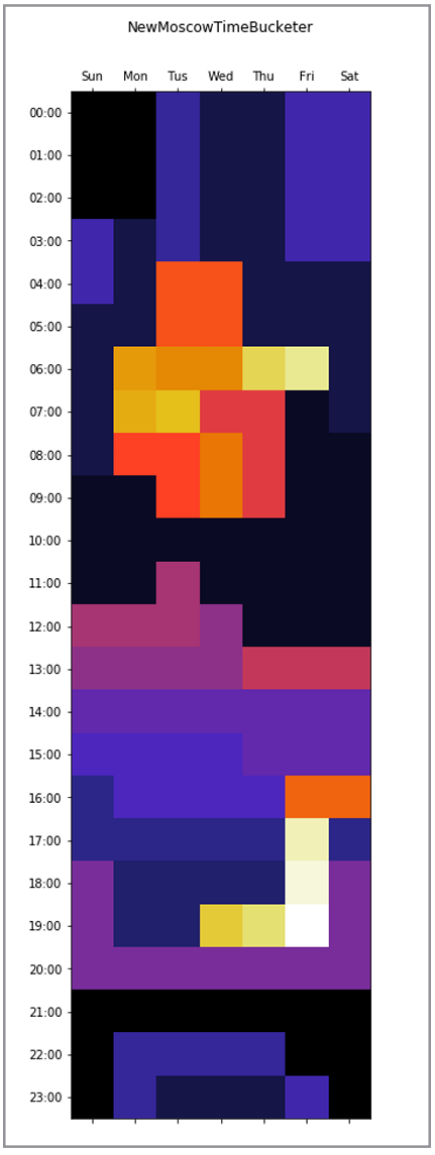


Figure 1: An example of the final merged time buckets for Moscow, Russia

The set of n+1 locations consists of a depot, and n customers form a directed graph with n+1 nodes and (n+1)*n arcs that represent travel between locations. For every arc and each time bucket, the fastest path is calculated using a time-dependent Dijkstra algorithm [6]. As a result, a multi-layer distance duration matrix A(t) is created [7]. Each element of the distance duration matrix is a pair consisting of travel duration τ_{ij}^{t} and travel distance d_{ij}^{t}, $i,j = 0,1. \ldots, n$, where t=1,…,T. Figure 2 illustrates an example of a multi-layer distance duration matrix. For all buckets, the distance d_{ij}^{t} is always greater than or equal to the shortest, time-independent, or OSM distance since traffic congestion causes one to select the route that can be longer by distance but faster by time.

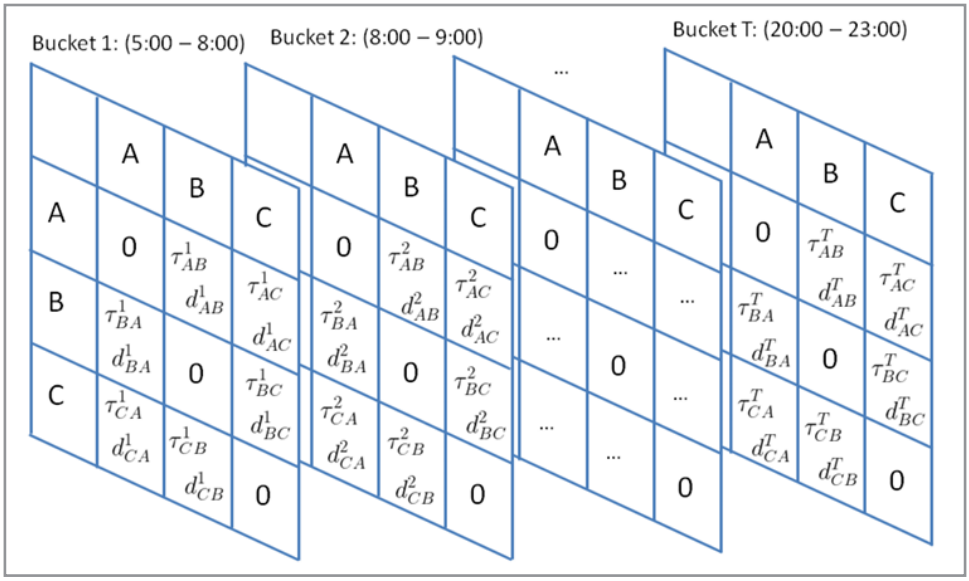


Figure 2: An example of a multi-layer distance duration matrix

Time buckets do not necessarily produce a non-decreasing stepwise function, so the FIFO property is not always satisfied. For example, if the duration between locations A and B is 20 minutes between 19:00 and 20:00 and 15 minutes between 20:00 and 23:00, then, from the modeling perspective, the vehicle departing from A at 19:59 will arrive at B at 20:19. The car, departing from A two minutes later at 20:01 will arrive at B three minutes before, at 20:16 since it will take another route, optimal for the next time bucket. A feasible solution to the TD-VRPTW with no waiting is a set of the TSP sequences with TD and TW, i.e., TDTSPTW of $m \leq n$ customer-time pairs: $\{(v_0, t_0), (v_{k1}, t_1), \ldots, (v_{km}, t_m), (v_0, t_{m+1})\}$. Where, v_0 is the depot, *k=1,…K* is the vehicle, i.e., TSP index, and $t_i \in [e_{ki}, l_{ki}]$ is the arrival time at the customer v_{ki}. For the depot $t_0 \geq e_0$, and $t_{m+1} \leq l_0$. The relation between the consequence pairs is described by the traffic equation (12):

$$t_{i+1} = t_i + g_i + \tau_{ij}\big(Bucket(t_i + g_i)\big). \quad (17)$$

Solving the VRP is known to be a combinatorial NP-hard optimization problem [32]. For the practical number of customers, typically n ≥30, the exact solution is not viable on the time scale [9]. To this end, the solution consists of two principal stages: an initial feasible or greedy solution and iterative improvement until a reasonable optimum is reached [33].

Greedy Feasible Solution for TDVRPTW

Route construction heuristics select nodes (or arcs) sequentially until a feasible solution has been created [34]. There are two principal approaches to creating greedy feasible solutions: variable neighborhood search and saving heuristics [35, 10]. Variable Neighbourhood Search (VNS) or Adaptive Large Neighbourhood Search (ALNS) is a type of first-order operator with a minimal computational load per iteration [36]. Still, it involves a significant number (up to 13) of adjustable parameters. The saving concept or Clarke-Wright (CW) concept, proposed by Clarke and Wright (1964) [10], is a kind of second-order operator, estimating a saving measure of each arc:

$$s_{ij} = d_{i0} + d_{0j} - d_{ij}\,, \quad (18)$$

where d_{ij} is the distance between customers *i* and *j*, and customer 0 denotes the depot. Thus, the values of s_{ij} express the saving of combining two customers, *i* and *j*, on one tour rather than serving them on two different tours. The saving algorithm forms the basis of most tools for solving VRPs and can be easily applied to VRPTW, as initially suggested by Solomon (1987) [37]. However, VRPTW in models like (Doerner et al., 2002) [6] has no traffic, and the riding time between the customers equals the Euclidean distance between them, so the saving criteria like C_2 do not directly apply to TDVRPTW [37, 38]. The simplified pseudo-code of a greedy algorithm, based on the saving principle, is shown in Algorithm 1. The algorithm's input is the customer list with TWs and service times, depot TW and loading time, and a multi-layer distance duration matrix. The output is a feasible TDVRPTW [37].

Algorithm 1: Greedy Saving TDVRPTW Solution

Step 1 Select the first arc $A = (u_1, u_2)$ of seed customers [34].

Step 2 Create a list $Y = C \backslash A$ of unvisited customers.

Step 3: Set partial tour $\pi_i = A$, $i = 1$, and $s_{max} = 1$.

Step 4 Until $s_{max} > 0$, repeat:

Step 4.1 Select the best head-interior customer from Y to π_i, having saving h_{max}.

Step 4.2 Select the best tail-interior customer from Y to π_i, having saving t_{max}.

Step 4.3 If $s_{max} = max(h_{max}, t_{max}) > 0$, add a new customer, $y = argmax(h_{max}, t_{max})$ to π_i, and update list $Y \leftarrow Y \backslash y$.

Step 5 Until the list of unvisited customers Y is empty, repeat:

Step 5.1 Add tour π_i to TDVRPTW, and set $i = i + 1$

Step 5.2 Select the first arc $A = (u_{1i}, u_{2i})$ of seed customers from the list Y for a new tour.

Step 5.3 Create a new tour π_i by repeating Steps 2-4.

Saving Concept for TDVRPTW

For TDVRPTW, let us modify the classical CW-saving algorithm and, instead of a single saving matrix S, produce two matrices: a saving matrix S and a time-start matrix T_0. It's supposed that the fleet is homogeneous, i.e., all vehicles are the same type [10]. The element s_{ij} of the saving matrix S means the saving from using the path $\{0, i, j, 0\}$ instead of two paths: $\{0, i, 0\}$ and $\{0, j, 0\}$. The element t_{ij}^0 of the time-start matrix T_0 means the earliest time of arriving at the starting location i on the path $\{0, i, 0\}$ from the depot (location 0) if the path is $\{0, i, j\}$. The path is not always feasible. For example, if the time window at location i is $[13:00 - 17:00]$ and at location j is $[9:00, 12:40]$, the path $\{0, i, j\}$ is infeasible, but the path $\{0, j, i\}$ is feasible, providing the travel time between (i, j), and the service time of customer i at the time bucket of 12:00 exceeds 20 minutes. If the path $\{0, i, j\}$ is infeasible $t_{ij}^0 = 0$ and $s_{ij} = 0$. Thus, if $e_i + g_i > l_j$ or $l_i + g_i < e_j$ then $t_{ij}^0 = 0$ and $s_{ij} = 0$ because of the time windows incompatibility.

If there are no feasible paths on the entire matrix, then all customers get the point-to-point solution: $\{0, i, 0\}$ and $\{0, j, 0\}$.

The elements of the matrices $S = [s_{ij}]$ and $T_0 = [t_{ij}^0]$ are calculated based on the objective function, traffic model, and constraints. For example, if vehicle capacity is ten items, customer i demands five items, and customer i demands six items, both paths $\{0, i, j\}$ and $\{0, j, i\}$ are infeasible because of the capacity violation. So, if $q_i + q_j > q_{max}$, then s_{ij},=0 and $t_{ij}^0 = 0$, where q_{max} is the vehicle capacity.

If $e_i \geq e_j$, then the earliest arrival time at location i is e_i, thus $t_{ij}^0 = e_i$. Otherwise, the earliest departure time from i to j is the root of the traffic equation or the backward travel time function:

$$t^d = e_j - \tau_{ij}(t^d) \quad (19)$$

Equation (19) is solved iteratively, and Figure 3 illustrates the delivery process [13]. The initial guess is $t_0^d = e_j - 0{:}01$ minute, i.e., the previous time bucket if it is different from the time bucket of e_j. The iterations $t_{k+1}^d = e_j - \tau_{ij}(t_k^d)$ are processing for k=0,1,2,... until $|t_{k+1}^d - t_k^d| > 0$. Sometimes, the iteration process is prone to oscillations. Let's consider an example depicted in Figure 4. There is a depot with a time window [6:00-12:00], loading time $g_0 = 0{:}30$, and customer i with TW = [7:55-9:30]. The depot-to-customer travel time is $\tau_{0i}(6{:}00 - 7{:}00)$ = 40 minutes (before traffic congestion) and $\tau_{0i}(7{:}00 - 9{:}00)$ = 70 minutes or 1:10 at the rush time. The process (19) starts with $t_0^d = e_j$ - 0:01 = 7:54, and t_1^d = 7:55 - 1:10 = 6:45. The next iteration gets t_2^d = 7:55 - 0:40 = 7:15. Thus, there is an oscillation: 6:45, 7:15, 6:45, etc. In this case, the target time e_i = 7:55 is unreachable, and there is a dead zone between 7:40 and 8:10. In the dead zone, the customer cannot be reached from the depot. The solution of (19) is the earliest departure time of the time bucket at the start time, i.e., t^d 7:00, and the arrival time at the location j is 7:00 + 1:10 = 8:10. In this case, the arriving time at the depot t_0, is $t_0 = t^d - g_0$ = 7:00 – 0:30 = 6:30.

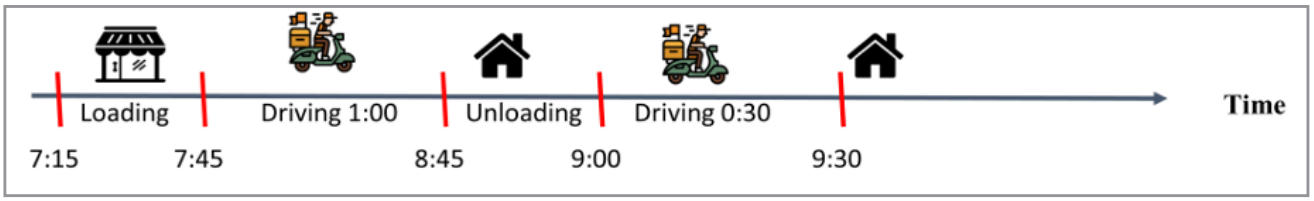


Figure 3: An illustration of the delivery process for the depot with TW = [6:00-12:00], g_0=0:30 and two customers i with TW = [8:00-10:00], service time g_i=0:15 and j with TW = [9:30-12:00], service time g_j=0:20

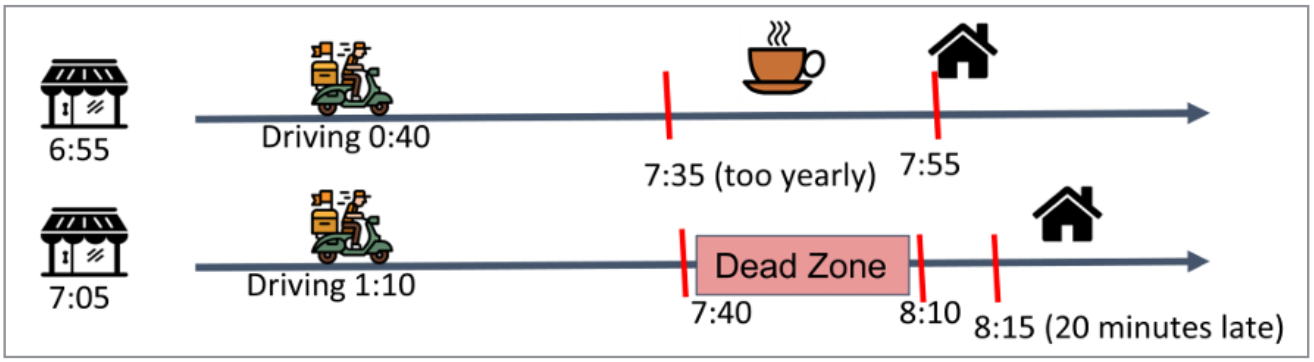


Figure 4: An illustration of the oscillations in solving equation (19)

In many cases, equation (19) has a multi-root solution. For example, let customer i has TW = [19:05-19:40] with the depot-to-customer travel time $\tau_{0i}(19{:}00-20{:}00)$ = 3 minutes (after the rush) and $\tau_{0i}(18{:}00-19{:}00)$ = 6 minutes (at the rush time). In this case, there are two roots: t^d = 19:02 and 18:59, so a minimal duration (3 minutes at 19:02) is chosen.

To know the time when the vehicle has to leave the depot to arrive at location i at t_{ij}^0 (depot departure time), the following backward time equation, similar to equation (19), has to be solved:

$$t_0^d = t_{ij}^0 - \tau_{ij}\left(t_0^d\right) \tag{20}$$

The root of (20) is clipped into the TW interval $[e_0, l_0]$. Eventually, for the route (*0,i,j,0*) there is a sequence of the arrival times: $\left(t_0^a,\ t_{ij}^0, t_j, t_0^r\right)$,where the depot arrival time is just $t_0^a = t_0^d - g_0$, where t_0^d is the depot departure time from (20), t_0^r is the depot return time and g_0 is the depot loading time, i.e., the time needed for the pickup. This time does not depend on traffic; however, the rest of the arrival times are traffic-dependent and are calculated using the forward propagation, or ready time function [13]:

$$t_{ij}^0 = t_{0+}^d \tau_{0i}\left(t_0^d\right), \qquad t_j = t_{ij}^0 + g_i + \tau_{ij}\left(t_{ij}^0 + g_i\right),$$
$$t_0^r = t_j + g_j + \tau_{j0}\left(t_j + g_j\right) \tag{21}$$

If $t_0^d, t_0^r \notin [e_0, l_0]$ or $t_{ij}^0 \notin [e_i, l_i]$ or $t_j \notin [e_j, l_j]$, then the pair is infeasible and $s_{ij} = 0$, $t_{ij}^0 = 0$.Besides, following the regulations, the tour duration should not exceed the maximum courier shift working time, i.e.:

$$t_0^r - t_0^d + g_0 \le CS_{max}, \tag{22}$$

where CS_{max} is the maximal courier working shift. If condition (22) is violated, then $s_{ij} = 0$, and$t_{ij}^0 = 0$.

The saving matrix elements are:

$$\begin{aligned} s_{ij} = {} & \tau_{0i}\left(t_{0i}^d\right) + \tau_{i0}\left(t_{0i}^d + \tau_{0i}\left(t_{0i}^d\right) + g_i\right) + \tau_{0j}\left(t_{0j}^d\right) \\ & + \tau_{j0}\left(t_{0j}^d + \tau_{0j}\left(t_{0j}^d\right) + g_j\right) - \\ & \left[\tau_{0i}\left(t_0^d\right) + \tau_{ij}\left(t_{ij}^0 + g_i\right) + \tau_{j0}\left(t_j + g_j\right)\right] \end{aligned} \tag{23}$$

Times t_{ij}^0 and t_0^d are also roots of equations (20) and (21), t_{0i}^d and t_{0j}^d are the earliest departure time for a point-to-point tour, and they are defined as a root of the backward time equation:

$$t_{0i}^d = e_i - \tau_{0i}\left(t_0^d\right) \tag{24}$$

The First Seed Arc Selection for the Greedy Algorithm

The first step is to find a pair of "seed" customers to initialize the path construction [34]. The path starts with the earliest possible delivery pair, having the maximal saving:

$$(u_1, u_2) = argmax\left(s_{ij} | (i,j) \in A\right). \tag{25}$$

Where set A consists of the earliest pairs : $(i,j) | t_{ij}^0 < l_{min}$, where l_{min} is the earliest late time window for all of the customers, i.e., l_{min}=$minl_i$. Let us denote this route, consisting of the first pair, as a set containing the path and arrival times: $\left\{(0, u_1,\ u_2, 0), \left(t_0^d(u_1),\ t_{u_1u_2}^0, t_{u_2}, t^r\right)\right\}$,where the depot departure time $t_0^d(u_1)$ is the root of equation (20). The route is feasible if the arrival times $t_{u_1u_2}^0$ and t_{u_2} are inside their time windows, $t_0^d(u_1)$ is inside the depot time window, and the total courier shift duration does not exceed the maximum courier shift working time (22). The already visited sites are collected in the visited customers list $T = \{u_1, u_2\}$.

The first seed arc selection requires the calculation of n(n-1) items of the saving matrix (complexity $O(n^2)$) and their sorting (complexity O(n(n-1)ln(n(n-1)))), therefore its overall complexity is $O\left(2n^2 ln(n)\right)$.

The Head Interior Heuristics for the Greedy Algorithm

Let us suppose that we have already created a partial solution of VRP with k ready feasible paths: $\pi_1, \pi_2, \ldots, \pi_k$,where $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d\right)$is an ordered list of the visited sites, along with the pickup time t_i^d. Besides, we still have a list of m unvisited customers: $Y = (y_1, y_2, \ldots, y_m)$ and the current feasible path $\pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d\right)$ under construction. To find the next best partial greedy solution of VRP, we would have to test $m(n_c - 1)$ push-forward insertions (Solomon, 1987) [37]. For a large number of customers, the overall time complexity following [39] is O(n³), and this becomes infeasible due to the computational load [39]. Instead, only head and tail interior insertions are processed as suggested in the original Clarke-Wright heuristic [10]. In addition to the insertions of m unvisited customers, we check the relocate heuristic with the last visited customers of each ready path π_i, subject to $n_i > 2$. Let us denote this list ST =$\left(u_{n_1}^1, u_{n_2}^2, \ldots, u_{n_k}^k\right)$ as a soft tabu list. In our traffic model, the backward update (Bräysy and Gendreau, 2005a) [39] demands solving n times the equation (20) [34]. To reduce the computational load, only the forward update is processed for the soft tabu list.

The algorithm is described in Algorithm 2. For the current path, the arrival time at each site u_i^c is known. Let us denote this value, or ready time, as t_i^c. The following site, to be visited in the current path, is selected from the list of unvisited sites Y and the soft tabu list ST (Step 1) [13]. If the tested customer y does not meet capacity, time window, or maximal shift duration constraints (Steps 2.1-2.3), the next customer from the list is tested. For each valid customer, its saving value is calculated (Step 2.6), and if this saving exceeds the initial saving s_0, a modification of the Coefficient Weighted Distance Time Heuristics (CWDTH) or level of urgency is calculated as follows [41, 42].

Let t_y be the arrival time at the customer y, which is calculated in Step 2.2, then the time left to the end of the delivery is $\Delta t_y = (l_y - t_y)$. The measure of actual urgency also depends on the planning horizon of customer y and is defined as $\Delta\mu_y = \Delta t_y (l_y - t_y)^2$. Let $M_y = max(\Delta\mu_y)$ be the maximal gap to the end of the delivery time window for all valid customers, then the weight $CWDTH(y) = M_y/\Delta\mu_y$. Among the unvisited customers $Y = (y_1, y_2, \ldots, y_m)$, a customer with the maximum weighted saving $s'_y = (s_y - s_0)CWDTH(y)$ (Step 2.7) is selected. The output of the algorithm is the updated current path $\pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c, u_h^c | t_c^d\right)$.

Algorithm 2: TDVRPTW Greedy Head Interior Insertion

Step 1 Create a soft tabu list $ST = (u_{n_1}^1, u_{n_2}^2, \ldots, u_{n_k}^k)$ and for the current path $\pi_c = (u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d)$, calculate the following parameters:

Step 1.1 Current demand $Q_c = \sum_{j=1}^{n_c} q_j$.

Step 1.2 Arrival time at the last customer t_c.

Step 1.3 On-demand duration, i.e., the sum of durations for depot-to-customer delivery to each customer separately:

$$D_0 = \sum_{i=1}^{n_c} \tau_{0u_i^c}\left(t_{0u_i^c}^d\right) + \tau_{u_i^c 0}\left(t_{0u_i^c}^d + \tau_{0u_i^c}\left(t_{0u_i^c}^d\right) + g_{u_i^c}\right) + g_{u_i^c}, \quad (26)$$

where $t_{0u_i^c}^d$ is the earliest departure time from the depot for a single-point u_i^c, which is defined as a root of equation (23).

Step 1.4 Current path duration:

$$D_c = \tau_{0u_1^c}(t_c^d) + \sum_{i=2}^{n_c} \tau_{u_{i-1}^c u_i^c}\left(t_{i-1}^c + g_{u_{i-1}^c}\right) + \tau_{u_{n_c}^c 0}\left(t_{n_c}^c + g_{u_{n_c}^c}\right) + \sum_{i=1}^{n_c} g_{u_i^c}. \quad (27)$$

Step 1.5 Current saving $s_0 = D_0 - D_c$.

Step 2 For each customer $y \in Y$ repeat the following steps:

Step 2.1 If customer y violates capacity constraints, i.e., $q_y + Q_c > q_{max}$, or $n_c + 1 > N_{max}$exceeds the maximal available amount of the stop points N_{max}, continue with the next customer.

Step 2.2 Calculate the arrival time at customer y: $t_y = t_{n_c}^c + g_{u_{n_c}^c} + \tau_{cy}\left(t_{n_c}^c + g_{u_{n_c}^c}\right)$ .If the arrival time is too late, i.e., $t_y > l_y$ or too early, i.e.,$t_y < e_y$, continue with the next customer.

Step 2.3 Calculate total delivery shift duration: $CS_y = t_y^r + g_0 - t_c^d$, where $t_y^r = t_y + g_y + \tau_{y0}(t_y + g_y)$. If $CS_y > \mathrm{CS}_{max}$, continue with the next customer; otherwise, calculate the saving value as follows.

Step 2.4 Calculate on-demand duration (sum of durations for depot-to-customer delivery to each customer separately):

$$D_y^0 = D_0 + \tau_{0y}\left(t_{0y}^d\right) + \tau_{y0}\left(t_{0y}^d + \tau_{0y}\left(t_{0y}^d\right) + g_y\right) + g_y, \quad (28)$$

where t_{0y}^d is the earliest departure time from the depot for single point y, which is defined as a root of equation (24).

Step 2.5 Calculate the path duration with customer y:

$$D_y = D_c - \tau_{u_{n_c}^c 0}\left(t_{n_c}^c + g_{u_{n_c}^c}\right) + \tau_{u_{n_c}^c y}\left(t_{n_c}^c + g_{u_{n_c}^c}\right) + \tau_{y0}\left(t_y + g_y\right), \quad (29)$$

where t_y is the arrival time at customer y, calculated in Step 2.2.

Step 2.6 Calculate the saving value $s_y = D_y^0 - D_y$.If $s_y \le s_0$, there is no reason to process customer y, in this case, continue with the next customer.

Step 2.7 Calculate $CWDTH(y) = \frac{M_y}{\Delta\mu_y}$ and (Carić et al., 2007) [41] weighted time-saving :

$$s_y' = (s_y - s_0)CWDTH(y). \quad (30)$$

Step 3 Find $s_{max} = max\,(s_y')$, and $y^* = argmax(s_{max})$.

Step 4 For each customer z, associated with the path π_i from the soft tabu list z∈ST repeat the following steps:

Step 4.1 If customer z violates capacity constraints, i.e., $q_z + Q_c > q_{max}$ or $n_c + 1 > N_{max}$ exceeds the maximal available amount of the stop points N_{max}, continue with the next customer.

Step 4.2 Calculate the arrival time at customer z: $t_z = t_{n_c}^c + g_{u_{n_c}^c} + \tau_{cz}\left(t_{n_c}^c + g_{u_{n_c}^c}\right)$. If the arrival time is too late, i.e., $t_z > l_z$ or too early, i.e., $t_z < e_z$, continue with the next customer.

Step 4.3 Calculate total delivery shift duration: $CS_z = t_z^r + g_0 - t_c^d$, where $t_z^r = t_z + g_z + \tau_{z0}(t_z + g_z)$. If $CS_z > \mathrm{CS}_{max}$, continue with the next customer; otherwise, calculate the saving value as follows.

Step 4.4 Calculate current saving s_{oz} for path π_z, corresponding to the last customer z: $s_z = D_{0i} - D_i$, where D_{0z} is calculated using (26) and D_i using (27) for path π_i

Step 4.5 Calculate on-demand duration (sum of durations for delivery to each customer separately) without customer z: $D_z^0 = D_{0z} - \tau_{0z}(t_z) - \tau_{z0}(t_z + g_z)$, where D_{0z} is the duration of the tour π_i , calculated by equation (26).

Step 4.6 Calculate the path duration up to the customer before z:

$$D_z = \tau_{0u_1^i}(t_i^d) + \sum_{j=2}^{n_i-1} \tau_{u_{j-1}^i u_j^i}\left(t_{j-1}^i + g_{u_{j-1}^i}\right) + \\ + \tau_{u_{n_i-1}^i 0}\left(t_{n_i-1}^i + g_{u_{n_i-1}^i}\right) + \sum_{j=1}^{n_i-1} g_{u_j^i} \quad (31)$$

where $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_i-1}^i, z\right)$ is the path associated with the customer z, t_z^d is the pickup time for the path π_i, and the time $t_j^i$is the arrival time at the customer u_j^i

Step 4.7 Calculate the savings lost for finishing the path π_i without customer z:

$$\Delta s_z = s_{0z} - (D_z^0 - D_z). \quad (32)$$

Step 4.8 Calculate the saving value s_z for head inserting user z into the current path π_c by applying Steps 2.1-2.6 to customer z instead of y.

Step 4.9 Calculate the actual savings concerning the savings lost and the level of urgency:

$$s_z' = (s_z - s_0 - \Delta s_z)CWDTH(z) \quad (33)$$

Step 5 If $(s_z') > s_{max}$, then the updated path π_c with the customer z is better than the previous path with the customer y. In this case, the new path $\pi_c(z)$ is the algorithm's output. Otherwise, the output is the old path $\pi_c(y)$.

Step 6 If the new saving value $s_{max} \le 0$, then path π_c cannot be updated by head interior insertion.

Suppose the algorithm selected a customer from a soft tabu list. In that case, we must calculate the saving loss for withdrawing a soft tabu customer from its path to compare with the unvisited customers (Step 4). A soft tabu flag indicates that the path π_z (associated with customer z) ought to be updated since another unvisited customer might replace the withdrawn customer z. Soft tabu testing is a type of local search and insertion operation [43].

If Algorithm 2 returns $s_{max} \le 0$, then one of the possible reasons might be the gap between the next possible earliest delivery and the already finished delivery at the current path π_c . In this case, we need to check the possibility of adjusting the pickup time to synchronize the customers' TWs. Let $\hat{Y} \subseteq Y$ be the subset of unvisited customers Y, passing the capacity con-

ditions (Step 4.1). Let $e_{\hat{Y}} = min(e_i | i \in \hat{Y})$ be the earliest delivery time for all customers $\hat{Y}$. Then, we can add the next customer to the route if the pickup time is delayed by the positive value $\Delta t = \max(e_{\hat{Y}} - t_y | e_y = e_{\hat{Y}})$ i.e., the maximal gap among the earliest delivery time customers. This new pickup time is applied to the path $\pi_c' = \{u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d + \Delta t\}$, and now, if in the time-shifted path π_c' all customers are inside their TWs: $\forall i = 1, \ldots, n_c [t_i \in [e_i, l_i]]$. Algorithm 2 is applied again to the current path π_c'. Algorithm 2 involves m+k tests, and each test involves no more than n calculations since max(m+k)=n-2, then its complexity is $O(n^2)$.

The Tail Interior Heuristics for the Greedy Algorithm

Besides head interior insertion, the tail interior insertion is also tested, as Clarke and Wright (1964) [10] suggested. The input is the same: a partial VRP with k feasible paths: $\pi_1, \pi_2, \ldots, \pi_k$, where $\pi_i = (u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d)$ is an ordered list of the visited sites, along with the pickup time t_i^d, a list of m unvisited customers: $Y = (y_1, y_2, \ldots, y_m)$, and the current path $\pi_c = (u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d)$ under construction. In the case of the tail interior heuristics, the potential customer is tested to be delivered before u_1^c. So, two backward-time equations have to be solved:

1 Arrival time at the tested customer y:

$$t_y + \tau_{yu_1^c}(t_y + g_y) = t_1, \quad (34)$$

where t_1 is the arrival time at the first customer u_1^c, calculated as $t_1 = t_c^d + \tau_{0u_1^c}(t_c^d)$. Equation (34) is solved iteratively, like equation (20), with the initial guess $t_y = t_1$

New pickup time for the current path π_c:

$$t_c^d(y) + \tau_{0y}(t_c^d(y)) = t_y \quad (35)$$

Equation (35) is also solved iteratively, with the initial guess $t_c^d(y) = t_y$.

The feasibility of the new path, starting with the customer y is tested by iterative forward propagation. Let, t_y be the arrival time at the customer y, then the arrival time at the first customer of the initial path u_1^c is

$$t_1 = t_y + g_y + \tau_{yu_1^c}(t_y + g_y), \quad (36)$$

If $t_{i+1} \notin [e_{u_{i+1}^c}, l_{u_{i+1}^c}]$,then the path is infeasible. The rest of the arrival times on the path π_c are updated as follows:

$$t_{i+1} = t_i + g_{u_i^c} + \tau_{u_i^c u_{i+1}^c}(t_i + g_{u_i^c}), i = 1, \ldots, n_c - 1 \quad (37)$$

If $t_{i+1} \notin [e_{u_{i+1}^c}, l_{u_{i+1}^c}]$, then the path is infeasible.

The process is summarized in Algorithm 3, having the same input and output as Algorithm 2.

Algorithm 3: TDVRPTW Greedy Tail Interior Insertion

Step 1 Create soft tabu list $ST = (u_{n_l}^l, u_{n_2}^2, \ldots, u_{n_k}^k)$.

Step 2 Calculate parameters Q_c, D_0 and s_0 of the current path using Step 1 of Algorithm 2.

Step 3 Calculate the arrival time at the first customer u_1^c as $t_1 = t_c^d + \tau_{0u_1^c}(t_c^d)$.

Step 4 For each customer $y \in Y$ repeat the following steps:

Step 4.1 If customer y violates capacity constraints, i.e., $q_y + Q_c > q_{max}$, or $n_c + 1 > N_{max}$ exceeds the maximal available amount of the stop points N_{max}, continue with the next customer.

Step 4.2 Calculate arrival time t_y at the customer y as the root of equation (34) and the pickup time $t_c^d(y)$ as the root of equation (35). If $t_y \notin [e_y, l_y]$, continue with the next customer.

Step 4.3 Find the arrival time at the first customer location of the initial path u_1^c, using equation (36). If this time $t_1 \notin [e_{u_1^c}, l_{u_1^c}]$, continue with the next customer.

Step 4.4 Find the arrival time at the rest of the customer locations from the initial path u_i^c, $i = 1, \ldots, n_c$, using equation (37). If one of these times is out of TW, i.e., $t_i \notin [e_{u_i^c}, l_{u_i^c}]$, continue with the next customer.

Step 4.5 Calculate total delivery shift duration:

$$CS = t_{n_c}^r + g_0 - t_c^d(y),$$

where $t_{n_c}^r = t_{n_c} + g_{u_{n_c}^c} + \tau_{u_{n_c}^c 0}(t_{n_c} + g_{u_{n_c}^c})$. (38)

If $CS > CS_{max}$, continue with the next customer.

Step 4.6 Calculate on-demand duration (sum of durations for depot-to-customer delivery to each customer separately) D_y^0 with equation (28).

Step 4.7 Calculate the path duration with the customer *y*:

$$D_y = \tau_{0y}(t_c^d(y)) + \sum_{i=1}^{n_c} [\tau_{u_i^c u_{i+1}^c}(t_{u_i^c} + g_{u_i^c}) + g_{u_i^c}] + \tau_{u_{n_c}^c 0}(t_{u_{n_c}^c} + g_{u_{n_c}^c}) \quad (39)$$

where $t_{u_i^c}$ is the arrival time at the customer u_i^c location, recursively calculated by (17) as: $t_{u_i^c} = t_{u_{i-1}^c} + \tau_{u_{i-1}^c u_i^c}(t_{u_{i-1}^c} + g_{u_{i-1}^c})$.

Step 4.8 For the new path, calculate the saving value s_y' using Steps 2.6 and 2.7 of Algorithm 2.

Step 5 Find $s_{max} = max(s_y')$, and $y^* = argmax(s_{max})$.

Step 6 For each customer *z*, associated with the path π_i from the soft tabu list $z \in ST$ repeat the following steps:

Step 6.1 If customer *z* violates capacity constraints, i.e., $q_z + Q_c > q_{max}$ or $n_c + 1 > N_{max}$ exceeds the maximal available amount of the stop points N_{max}, continue with the next customer.

Step 6.2 Calculate arrival time t_z at the customer *z* as the root of equation (34) and the pickup time $t_c^d(z)$ as the root of equation (35). If $t_z \notin [e_z, l_z]$, continue with the next customer.

Step 6.3 Apply step 4 for customer z and calculate the saving value $s_z > s_0$.

Step 6.4 Calculate the lost savings Δs_z, using Steps 4.7 and 4.8 of Algorithm 2.

Step 6.5 Calculate actual saving s_z' according to the saving loss and level of urgency (33).

Step 7 If $(s_z') > s_{max}$, then the updated path π_c with the customer z is better than the previous path with the customer y. In this case, the new path $\pi_c(z)$ is the algorithm's output. Otherwise, the output is the old path $\pi_c(y)$.

Step 8 If new saving $s_{max} \leq 0$ then path π_c cannot be updated by tail interior insertion.

There are three exceptional cases when Algorithm 3 cannot create a tail interior path.

1. All unvisited sites have to be visited after the first customer u_1^c, i.e., $\forall_y[e_y > t_1]$. In this case, instead of the tail interior path, the Solomon insertion or Push-Forward Insertion Heuristics (PFIH) at position v is processed, where $v = argmin_i min_y(t_i \geq e_y)$ according to the maximal free time (MFT) heuristic (see Algorithm 4) [37, 43].
2. All unvisited sites have to be visited before the first customer u_1^c, i.e.,$\forall_y[t_1 > l_y]$. In this case, the pickup time may be started early by $\Delta t = min_y(l_y - t_1 - \varepsilon)$, where ε is a small confidence constant, say 10 seconds, to prevent rounding errors. The time shift is applied to the path $\pi_c' = (u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d - \Delta t)$ with the new pickup time $t_c^d - \Delta t$. If in the shifted path π_c', all the customers are inside their TWs: $\forall i = 1, \ldots, n_c\,[t_i \in [e_i, l_i]]$, Algorithm 3 is applied again with the new current path π_c'. The time shift is tested for all customers y, sorted in ascending order of $l_y - t_1$ and the first feasible time shift is applied.
3. If there are valid customers and no insertion in cases 1 and 2 were found, then there is no feasible path due to the TW incompatibility. In this case, Solomon's insertion PFIH at position v is tested, where $v = argmax_i\left(l_i = min_j l_{u_j^c}\right)$, where $i,j = 1, \ldots, n_c$, i.e., maximal index, having the minimal late time TW over the current path π_c, cf. minimum delay metric [43]. In this case, Algorithm 4 is applied to the index v.

Algorithm 3 checks m+k customers, and each test requires k forward calculations; thus, the Algorithm 3 complexity is O(k(m+k)). In the worst case, it becomes O(n²).

Solomon's PFIH for the TDVRPTW Greedy Algorithm

Push-forward insertion heuristics are processed if the tail insertion fails, and instead of the first position, the new insertion position *v* is tested according to the MFT heuristic [43]. In this case, the new customer is inserted into any predefined position v between 1 and n_c of the current path π_c. The process is described by Algorithm 4, and the output of the algorithm is the updated current path $\pi_c = (u_1^c, u_2^c, \ldots, u_v^c, \ldots, u_{n_c}^c | t_c^d)$.

Algorithm 4: TDVRPTW Greedy Solomon PFIH Insertion

Step 1 Create soft tabu list $ST = (u_{n_1}^1, u_{n_2}^2, \ldots, u_{n_k}^k)$..

Step 2 Calculate parameters Q_c, D_0 and $s_0$of the current path using Step 1 of Algorithm 2.

Step 3 Using equation (34), calculate the arrival time at the first v customers $u_1^c, \ldots, u_v^c$.

Step 4 Calculate the on-demand duration (when each customer is served with a separate vehicle) up to the customer u_v^c. That is a particular case of equation (26) and is read as:

$$D_{0v} = \sum_{i=1}^{v} \tau_{0u_i^c}\left(t_{0u_i^c}^d\right) + \tau_{u_i^c 0}\left(t_{0u_i^c}^d + \tau_{0u_i^c}\left(t_{0u_i^c}^d\right) + g_{u_i^c}\right) + g_{u_i^c} \quad (40)$$

Step 5 Calculate the current path duration up to the customer u_v^c without return to the depot, which is written as:

$$D_v = \tau_{0u_1^c}(t_c^d) + \sum_{i=2}^{v}\left[\tau_{u_{i-1}^c u_i^c}\left(t_{u_{i-1}^c} + g_{u_{i-1}^c}\right) + g_{u_{i-1}^c}\right] \quad (41)$$

Step 6 For each customer $y \in Y$, repeat the following steps:

Step 6.1 Calculate arrival time t_y at the customer y as the result of the forward propagation using equation (37). If $t_y \notin [e_y, l_y]$, continue with the next customer.

Step 6.2 Find the arrival time for the rest of the customers from the initial path u_i^c, $i = v, \ldots, n_c$,using equation (36). If one of these times is out of TW, i.e., $t_i \notin \left[e_{u_i^c}, l_{u_i^c}\right]$, continue with the next customer.

Step 6.3 Calculate total delivery shift duration CS using equation (38). If CS>CS_{max} , continue with the next customer.

Step 6.4 Calculate on-demand duration (sum of durations for depot-to-customer delivery to each customer separately) D_y^0 using equation (28).

Step 6.5 Calculate the path duration with the customer y:

$$D_y = D_v + \tau_{u_v^c y}(t_v + g_v) + g_y + \tau_{y u_{v+1}^c}\left(t_y + g_y\right) + g_y + \sum_{i=v+1}^{n_c-1}\left[\tau_{u_i^c u_{i+1}^c}\left(t_{u_i^c} + g_{u_i^c}\right) + g_{u_i^c}\right] + \tau_{u_{n_c}^c 0}\left(t_{u_{n_c}^c} + g_{u_{n_c}^c}\right), \quad (42)$$

where D_v is the result of (41), $t_{u_i^c}$ is the arrival time at the customer u_i^c, recursively calculated by (17), t_v is the arrival time at the customer u_v^c, and t_y is the arrival time at customer y.

Step 6.6 For the new path, calculate the saving value s_y' using Steps 2.6 and 2.7 of Algorithm 2.

Step 7 Find $s_{max} = max\,(s_y')$, and $y^* = argmax(s_{max})$.

Step 8 For each customer z, associated with the path π_i from the soft tabu list $z \in ST$ repeat the following steps:

Step 8.1 If customer z violates capacity constraints, i.e., $q_z + Q_c > q_{max}$ or $n_c + 1 > N_{max}$ exceeds the maximal available amount of the stop points N_{max}, continue with the next customer.

Step 8.2 Calculate arrival time t_z at the customer z as the result of the forward propagation using equation (37). If $t_z \notin [e_z, l_z]$, continue with the next customer.

Step 8.3 Repeat Steps 6.2 and 6.3 for the customer *z*.

Step 8.4 Calculate on-demand duration (sum of durations for depot-to-customer delivery to each customer separately) D_z^0 using equation (28) and the path duration with the customer , using equation (42).

Step 8.5 Apply Steps 6.4-6.6 for customer z and calculate the saving value $s_z > s_0$.

Step 8.6 Calculate the lost saving Δs_z, using Steps 4.7 and 4.8 of Algorithm 2.

Step 8.7 Calculate actual saving s_z', concerning saving loss and the level of urgency (33).

Step 9 If $(s_z') > s_{max}$, then the updated path π_c with the customer z is better than the previous path with the customer y . In this case, the new path $\pi_c(z)$ is the algorithm's output. Otherwise, the output is the old path $\pi_c(y)$.

There are two exceptional cases when Algorithm 4 cannot create Solomon's PFIH interior path.

1 All unvisited sites are incompatible with TW with the arrival time at insertion index v: $\forall_y[t_y \notin [e_y, l_y]]$. In this case, the new insertion index is defined as $v' = argmin(e_i \leq t_y \leq l_y | i > v)$. If $v' = n_c$, then the PFIH interior is identical to the head interior;

otherwise, Algorithm 4 is repeated with the new insertion index v'.

2 If there are valid customers and no insertion related to case 1, then there is no feasible path due to the TW incompatibility. In this case, the new insertion index is defined as $v' = argmax_i\left(l_i = min_j\, l_{u_j^c}\right)$, , where $i,j = v+1,\ldots,n_c$ i.e., maximal index, having the minimal late time over TW on the current path π_c. If $v' = n_c$, then the PFIH interior is identical to the head interior; otherwise, Algorithm 4 is repeated with the new insertion index v'.
The complexity of Algorithm 4 is also $O(k(m+k))$, and in the worst case, it becomes $O(n^2)$.

The Current Path Growth in the TDVRPTW Greedy Feasible Solution
Let us suppose that we created a partial solution of VRP with k feasible paths: π_1, π_2, ..., π_k, where $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_i}^i | t_i^d\right)$ is the ordered list of visited sites, along with the pickup time t_i^d. Besides, we still have a list m unrouted customers: $Y = (y_1, y_2, \ldots, y_m)$, and the current path under construction: $\pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d\right)$. By applying Algorithm 2 (TDVRPTW Greedy Head Interior Insertion), we get a candidate for the updated current path: $\pi_h = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c, u_h^c | t_c^d\right)$, having the saving value s_h and by applying Algorithm 3 (TD VRPTW Greedy Tail Interior Insertion), get another candidate: $\pi_t = \left(u_t^c, u_1^c, u_2^c, \ldots, u_{n_c}^c | t_{ct}^d\right)$ with the saving value s_t. Alternatively, if insertion before u_1^c is infeasible, then Algorithm 4 (TDVRPTW Greedy Solomon's PFIH Insertion) is used, and the candidate path is $\pi_t = \left(u_1^c, u_2^c, \ldots, u_v^c, \ldots, u_{n_c}^c | t_c^d\right)$. For each candidate, let us introduce the urgency indicator κ, where $\kappa_h = 1$ if $l_h - t_h \le \theta$ and zero otherwise. For the tail interior: $\kappa_t = 1$ if $l_t - t_t \le \theta$ and zero otherwise, and for Solomon's PFIH insertion $\kappa_t = \kappa_v = 1$ if $l_v - t_v \le \theta$ and zero otherwise, where the adjustable urgency parameter θ = 100 minutes. The winning candidate is selected using the rules collected in Table 3.

Table 3: The winning candidate selection

State	Winning path
$s_h > 0$, $s_t > 0$, $\kappa_t = 0$, and $\kappa_h = 1$	π_h
$s_h > 0$, $s_t > 0$, $\kappa_t = 1$, and $\kappa_h = 0$	$0\pi_t$
max $(s_h, s_t) > 0$, and $\kappa_h = \kappa_t$	π_ζ, where ζ=argmax (s_h, s_t)
$s_h > 0$, and $s_t = 0$	π_h
sh > 0, and sh=0	π_t
max $(s_h, s_t) \le 0$	New path π_c is started

If a new customer y, which was added to the winning path, was selected from the set of un-routed customers, i.e., $y \in Y$, then y is inserted into the tabu list, and the path growth is continued with the winning path. If the new customer was selected from the soft tabu list, i.e., $y \in Z$, then the following actions of Algorithm 5 ought to be taken.

Algorithm 5: Updating the Path after Selecting a Customer from the Soft Tabu List.
Step 1 Remove the last customer z from the path π_z , associated with the customer z.
Step 2 Apply Algorithm 2 to the path π_z without Step 4, i.e., only for unvisited sites. Let π_{zh} be the resulting candidate path with the saving value s_{zh}.
Step 3 Apply Algorithm 3 to the path π_z without Step 6 (only for unvisited sites) or (if tail insertion is infeasible) Algorithm 4 without Step 8. Let π_{zt} be the resulting candidate path with the saving value s_{zt}.
Step 4 If $max(s_{zh}, s_{zt}) \le 0$ then path π_z cannot be updated, and the process goes on with the reduced path π_z without customer z.
Step 5 If $max(s_{zh}, s_{zt}) > 0$, then update path π_z, as $\pi_z = \pi_\zeta$, where $\zeta = argmax(s_h, s_t)$, add the newly inserted customer into the tabu, and repeat Steps 2-5 for the updated path π_z until $max(s_{zh}, s_{zt}) > 0$.

Starting a New Path in the TDVRPTW Greedy Feasible Solution
When the current path π_c meets the stopping criteria and cannot grow any further, a new feasible path must be initiated. Let $Y = (y_1, y_2, \ldots, y_m)$ be the list of m unrouted customers. If this list is empty, all sites have been visited, and the greedy feasible solution is ready. If m=1, then the new path is trivial: $\pi = (0, y, 0)$. If m>1, then the new path can be obtained by applying Algorithm 1 on reduced matrices $S' = \|s_{ij}\|$, and $T'_0 = \|t_{ij}\|$, where $i,j \in Y$.

Eventually, the Solution Creation Algorithm is summarized in Algorithm 6 as follows. The output of the algorithm is the list of feasible paths: $\Pi = \{\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d\right)\}$, where i=1,...,k.

Algorithm 6: The Greedy Solution Creation
Step 1 Create the first seed pair using equation (25) as described in section 5.2.
Step 2 Grow the new path π_c, using Algorithms 2-4, described in section 5.6, until the saving is positive.
Step 3 If there is more than one unvisited site after updating the tabu list, create a new path and repeat Steps 2 and 3.
Step 4 The solution is ready if the list of unvisited sites is empty or contains only one customer.

Algorithm 6 complexity in the worst case is the same as Algorithm 1, i.e., $O(n^2)$. Since the greedy solution processes algorithms 1-4 in the worst case for all n-2 customers, the overall complexity is $O(n^3 \ln(n))$.

Eventually, the saving of the greedy VRP solution is the sum of the savings of the TSP greedy solutions, i.e., $S_0 = \sum_{i=1}^{k} S(\pi_i)$. The saving of each greedy TSP $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d\right)$ is the sum of its arc savings: $S(\pi_i) = \sum_{j=1}^{n_k - 1} S(u_j^i, u_{j+1}^i)$. To prove this additive

saving property, let us consider a simplified TSP: π=*(0, A, B, C, 0)* with the saving $S(ABC) = p_{A0} + p_{0B} + p_{B0} + p_{0C} - p_{AB} - p_{BC}$, where p_{ij} is a price calculated by objective (5) for the traversal arc *(i, j)*. The saving of the TSP with only one arc (A, B) is S(AB)=$p_{A0} + p_{0B} - p_{AB}$ and the saving of the TSP with only one arc (B, C) is S(B, C)=$p_{B0} + p_{0C} - p_{BC}$. Thus, S(ABC)=S(AB)+S(B,C). Besides, the saving function is always positive S(π)>0; otherwise, Algorithms 2-4 meet the stopping criterion. Moreover, the saving function is also monotone since every customer added to the path π only increases the saving function. Following (Nemhauser et al., 1978) [70], if the greedy saving function is also submodular, then the upper bound of the global saving function is estimated as:

$$\hat{S}_{max} \leq \frac{S_0}{(1-1/e)} \approx 1.582 S_0 \, . \tag{43}$$

Let us prove the greedy saving function submodularity property, i.e., that for every new customer x:

$$S(u_1, \ldots, u_{n-1}, u_n) - S(u_1, \ldots, u_{n-1}) \geq S(u_1, \ldots, u_n, x) - S(u_1, \ldots, u_n) \tag{44}$$

Applying the adaptive saving property to inequality (44) allows us to rewrite the inequality as: $S(u_{n-1}, u_n) \geq S(u_n, x)$. Since Algorithm 1 is greedy, it always selects the customer with the maximum savings addition. Thus, customer *x* cannot be a predecessor of the customer u_n, and the savings difference decreases every time a new customer is added.

Figure 5 shows the relationship between the relative saving improvement $\frac{\hat{S}_{max}-S_0}{S_0} 100\% \leq 58.2\%$ vs. the number of customers for 4,806 industrial deliveries in the Tel Aviv area. The number of customers varies from 2 to 140, and the maximum improvement is consistently below 35%, reaching its peak for 35-55 customers.

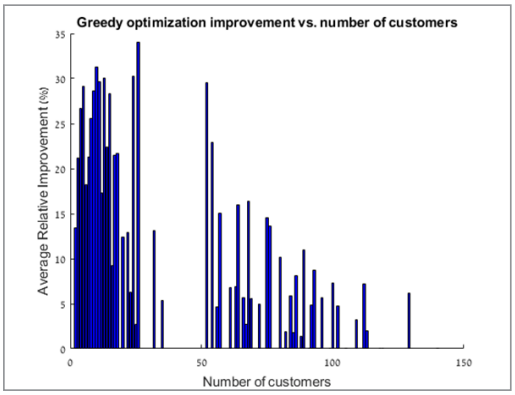


Figure 5: Relation between the relative saving improvement in percent on the number of customers, based on 4,806 deliveries in the Tel Aviv area

Improving the Greedy Feasible Solution for TDVRPTW by Ant Colony Optimization

Ant Colony Optimization (ACO) was initially introduced by Marco Dorigo et al. (1996) [8]. The idea behind the ACO is to select the subsequent head insertion of the growing current path with a probability proportional to the path distance or saving value [8, 38]. The ACO has two principal advantages: it can easily adapt to time window constraints by considering the probabilities of subsequent customer selection, and it enables parallel implementation [44]. The ACO is inspired by natural models of the foraging behavior of N ants looking for food. Studies on real ants show that despite not having a sense of sight, they can find the shortest path from the food sources to the nest [45]. Ants randomly explore their surroundings, and when they find food, they return to the nest, depositing a pheromone trail, a trace of a chemical substance that can be smelled by other ants [46].

Suppose each ants create a feasible VRP solution using the selection probabilities and constraints. At the first or the exploration stage, the likelihood of selecting the customer j after the customer i is proportional to the saving value. Following the Gibbs sampling, it can be written as:

$$p_{ij} = \frac{exp\left(-\frac{s_{max}-s_{ij}}{T}\right)}{\sum_{k \in R} exp\left(-\frac{s_{max}-s_{ik}}{T}\right)} \tag{45}$$

where R is the set of all valid customers j that are reachable from the customer i and $s_{max} = \max_{k \in R} s_{ik}$. The scale factor T is a control parameter or simulated annealing (SA) temperature [47]. The incentive behind the SA is explained in Figure 6. When the temperature is zero, we have the pure greedy solution by selecting $j = arg(s_{max})$ at any insertion. When T= ∞, we have an entirely random choice at every insertion. The locally optimal temperature is located somewhere between zero and infinity. To this end, the SA temperature measures the selection sensitivity to the saving value.

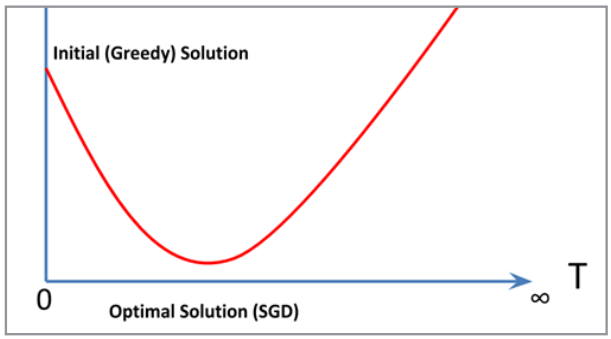


Figure 6: Duration (D) dependency on the simulated annealing temperature (T)

To select a starting value of the temperature T let us introduce the half-life of saving values (Schrimpf et al., 2000) [73]. Let us suppose that the half-life length equals 5% of the maximal saving value s_max, and the probability of an arc selection reduces by half per half-life. Thus, we have the decay equation: $exp(0.05s_{max}/T) = 2$. However, the value of s_{max} is unknown before the optimization is finished, but it can be estimated from the greedy solution using the upper bound (43) as $s_{max} = \hat{S}_{max}/(n-k)$, where n is the number of customers, and k is the number of greedy paths. After the substitution and taking the logarithm from both parts of the equation, we have $0.05\hat{S}_{max}/T(n-k) = ln2$; therefore, the initial SA temperature is

$$T = \frac{\hat{S}_{max}}{20(n-k)ln2}. \quad (46)$$

After the pure exploration stage, which is typically implemented with N_e=25-40 trials we have N_e different solutions with the saving values S_i, total durations D_i and the number of paths k_i, where i=1,2,…,N_e. Besides, there is the greedy solution with the saving value S_0, total duration D_0 and the number of paths k_0. If, at the exploration stage, the greedy solution was improved, i.e., $\min_i k_i < k_0$ or $\min_i k_i = k_0$ and $D_{i^*} < D_0$, where i* is the best solution according to objectives (2) and (5), then the initial temperature T is close to the optimal, and ACO continues with the temperature T. Otherwise, Algorithm 7 for SA temperature selection is applied. The input of the algorithm is a greedy solution S_0, D_0, k_0, N_e exploration solutions: S_i, D_i, k_i and initial temperature T.

Algorithm 7: Selection of the Working SA Temperature

Step 1 Introduce a penalty factor for the non-minimal number of paths as D_0/k_0 and calculate penalized durations for exploration solutions: $\tilde{D}_i = D_i + \frac{D_0}{k_0}(k_i - k_0)$ (47)

Step 2 Save values $T_{old} \leftarrow T$ and $\bar{D}_{old} \leftarrow \frac{1}{N_e}\sum_{i=1}^{N_e}\tilde{D}$

Step 3 Rerun another N_e trials with decreased temperature $T_{new} = \gamma T_{old}$, where decay factor γ = 0.95, and for these solutions, calculate $\bar{D}_{new}$, using (47) and Step 2, and then calculate $\delta = T_{new} - T_{old}$

Step 4 If the new best solution shows an improvement, then T_{new} is the working temperature; otherwise, save values $T_{prev} \leftarrow T_{new}$, $\bar{D}_{prev} \leftarrow \bar{D}_{new}$, and calculate the new temperature as follows. If $\delta(\bar{D}_{new} - \bar{D}_{old}) < 0$, then $T_{new} = \beta T_{prev}$, where incremental factor β = 0.01, otherwise, $T_{new} = \gamma T_{prev}$.

Step 5 Rerun another N_e trials with a new temperature T_{new}. If the best solution improves, then T_{new} is the working temperature; otherwise, apply the Newton-Raphson process: calculate the central difference central difference $\delta T = \frac{T_{new} - T_{old}}{2}$, current gradient $G_{new} = \frac{\bar{D}_{new} - \bar{D}_{prev}}{T_{new} - T_{prev}}$, previous gradient $G_{prev} = \frac{\bar{D}_{prev} - \bar{D}_{old}}{T_{prev} - T_{old}}$, and Hessian $H = \frac{G_{new} - G_{prev}}{\delta T}$. Save values $T_{old} \leftarrow T_{prev}$, $\bar{D}_{old} \leftarrow \bar{D}_{prev}$, $T_{prev} \leftarrow T_{new}$, $\bar{D}_{prev} \leftarrow \bar{D}_{new}$, and calculate the new temperature $T_{new} = T_{prev} - \frac{G_{new}}{H}$. If $|T_{new} - T_{prev}| > 5\,\delta T$, then the Newton-Raphson process becomes unstable, and the new temperature is calculated using Step 4.

Step 6 Repeat Step 5 ten times, and if improvement is not reached, then continue ACO with the initial temperature T.

Pheromone Updating Policy

The saving matrix is updated every time the ants finish all the paths (local pheromone information) to reward the most successful solutions or penalize the less successful ones. Let us represent the continuously updated saving matrix as a sum of the initial saving matrix and pheromone trace matrix: $s_{ij} + \varphi_{ij}$, where φ_{ij} is a pheromone matrix. In this case, Gibbs sampling (44) becomes Metropolis sampling:

$$p_{ij} = \frac{exp\left(-\frac{s_{max} - (s_{ij} + \varphi_{ij})}{T}\right)}{\sum_{k \in R} exp\left(-\frac{s_{max} - (s_{ik}(s_{ij} + \varphi_{ik}))}{T}\right)}. \quad (48)$$

The initial pheromone trace matrix is zero $(\varphi_{ij} = 0)$, and then it is updated after each trial (iteration) as: $\varphi_{ij}(r) = \varphi_{ij}(r) + \Delta\varphi_{ij}(r)$ where r is the incumbent solution and pairs (i,j)∈Π_r, i.e., all pairs in the solution.

The pheromone updating term $\Delta\varphi_{ij}(r)$ is calculated as follows. Using all the data collected while the working SA temperature is calculated, evaluate the following statistics:

1. Minimal penalized saving value $S_{min} = \min_i \tilde{s}_i$, where $\tilde{s}_i = s_i - (k_i - k_{min})\frac{s_{min}}{k_{min}}$, where $s_{min} = \min_i s_i$, and $k_{min} = \min_i k_i$,

2. Maximal penalized saving value $S_{max} = \max_i \tilde{s}_i$,

3. Average penalized saving value $\bar{S} = \frac{1}{N_e}\sum_{i=1}^{N_e}\tilde{s}_i$,

4. The standard deviation of the penalized saving value $\bar{\sigma} = \sqrt{\frac{1}{N_e+1}\sum_{i=1}^{N_e}(\tilde{s}_i - \bar{S})^2}$.

To reduce the noisy behavior, only λ≈ 0.2 of the solutions cause the pheromone updating. Compared to the long short-term memory (LSTM) approach proposed by Hochreiter and Schmidhuber (1997) [43], λ is the dropout factor, which is widely used for optimization process stabilization. Let us suppose that there are N ants at the colony, and the colony increases the maximal saving value twice if $\Delta\varphi_{ij} = \frac{QS_{max}}{\lambda N}$, where Q is a pheromone updating policy or an activation function in LSTM. Selection of the Parametric Rectified Linear Unit (pReLU) function (He et al., 2015) [63] gets the following expression for Q.

1. $Q = \frac{s - \bar{S} - \alpha\bar{\sigma}}{S_{max} - \bar{S} - \alpha\bar{\sigma}}$, if $s > \bar{S} + \alpha\bar{\sigma}$. In this case, the significantly successful solution is rewarded with positive Q.

2. Q=0, if $s \le \bar{S} + \alpha\bar{\sigma}$, in this case, the solution is close to the average solution or is unsuccessful and does not affect the pheromone trace.

The pheromone updating policy $Q(s)$ is shown in Figure 7. Let us suppose that the solution savings distribution is close to the Gaussian, then the parameter α of the pReLU function is α= $\Phi^{-1}(1-\lambda)$, where Φ^{-1} is inverse Gaussian cumulative distribution function; for λ=0.2, so α≈ 0.8415.

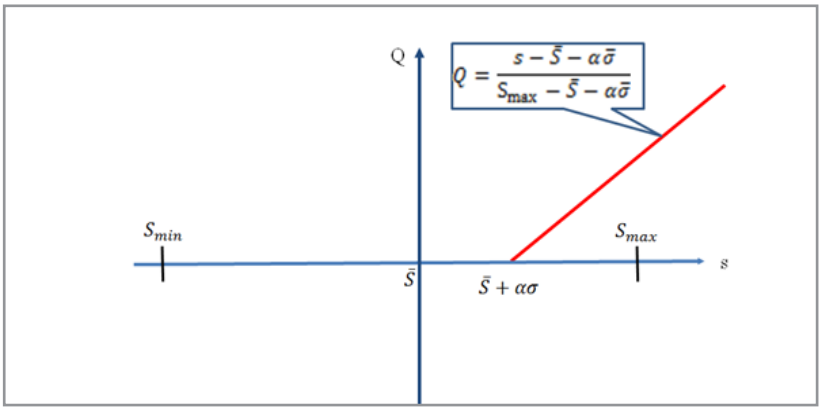


Figure 7: Pheromone updating function *(Q)* dependency on the solution saving *(s)*

The pheromone updating process can be described as LSTM training with stochastic gradient descent (SGD) [48]. Then, the training epoch is the number of trials sufficient for coverage of all possible arcs, so the epoch is:

$$E = \frac{n(n-1)}{n-k_{min}} \tag{49}$$

In equation (49) k_{min} is the number of paths in the best solution. Following this definition, the pheromone evaporation process becomes the LSTM forget gate and can be implemented as a moving average over one epoch [49].

To define the maximal number of iterations N_a, let us examine the pheromone matrix as a matrix of stochastic gradients. The convergence of the stochastic gradient descent (SGD) process takes 10-100 epochs, so $N_a \approx$ 10-100n. However, due to the lack of computational resources, typically N_a is limited to 2-3K iterations for most tasks. The number of iterations can be computed within 2-6 computational hours on a standard 4-core computer between finishing the VRP data collection and dispatching the package.

ACO Convergence to the Global Minimum

Let us apply the results of the adaptive random search study by Zhigljavsky & Žilinskas (2008) [77] to study the conversion of the ACO to a global minimum. We suppose that after $N_{min} \le N_a$ iterations, there are L_{min} solutions, having a minimal number of paths k_{min} and the total duration $D_i,\ i = 1$,..,L_{min}. If N_{min} and L_{min} are big enough (following Zhigljavsky (1991), L_{min}>200), then k minimal records durations obey the Weibull's distribution with the cumulative distribution function (c.d.f.):

$$W_\alpha(z) = \begin{cases} 0 \ if \ z < 0 \\ 1 - exp(-z^\alpha) \ if \ z \ge 0 \end{cases} \tag{50}$$

where α is an unknown tail index. The number of record durations should be big enough, but $\lim_{L_{min}\to\infty} \frac{k}{L_{min}} = 0$ So, Zhigljavsky (1991) [76] suggests choosing $k = \lfloor\sqrt{L_{min}}\rfloor$, where [.] is the ceiling function. For a given value *k*, the expected minimal duration $\hat{m}$ is estimated using one of two methods: the maximum likelihood estimator and the linear estimator.

The maximum likelihood estimator estimates $\hat{m}$ without knowing the tail index α [50]. Let us denote η_i, *i=1,..,k* the first k record durations in ascending order: $\eta_1 \le \eta_2 \ldots \le \eta_k$. Then, the lower duration bound is the root of the equation:

$$H = \frac{1}{k} + \frac{1}{\sum_{j=1}^{k-1} \beta_j(\hat{m})} - \frac{1}{\sum_{j=1}^{k-1} ln(\beta_j(\hat{m})+1)} = 0, \tag{51}$$

where $\beta_j(\hat{m}) = \frac{\eta_k - \eta_j}{\eta_j - \hat{m}}$. If equation (51) has more than one root in the open interval $(\eta_1, 0)$, the minimal root is taken. The equation (51) is solved using Newton-Raphson iterations, starting with three initial values: $\hat{m}_0 = max(0, \eta_1 - C_k(\eta_k - \eta_1))$,where $C_k = \{0.01, 0.025, 0.05\}$ as $\hat{m}_{i+1} = \hat{m}_i - \frac{H}{H'},\ i = 0,1,..,$ i=0,1,.., providing $\hat{m}_i \in [\eta_1, 0)$ until $|\hat{m}_{i+1} - \hat{m}_i| < 0.1sec$ The derivative *H'* is written as:

$$H' = \frac{\sum_{j=1}^{k-1} \frac{\beta'_j(\hat{m})}{1+\beta_j(\hat{m})}}{\left(\sum_{j=1}^{k-1} ln(\beta_j(\hat{m})+1)\right)^2} - \frac{\sum_{j=1}^{k-1} \beta'_j(\hat{m})}{\left(\sum_{j=1}^{k-1} \beta_j(\hat{m})\right)^2}, \tag{52}$$

Where: $\beta'_j(\hat{m}) = \frac{\eta_k - \eta_j}{(\eta_j - \hat{m})^2}$

If equation (51) has no valid solution, the tail index α should be estimated. There are two popular tail index estimators: the Hill (Hill, 1975) [64] and the Pickands (Pickands, 1975) [71] estimators. Following the Hill estimator,

$$\hat{\alpha}^{(H)} = \left(ln\eta_k - \frac{1}{k}\sum_{j=1}^{k} ln\eta_j\right)^{-1}. \tag{53}$$

The Pickands estimator suggests:

$$\hat{\alpha}^{(P)} = \frac{1}{ln2} ln \frac{\eta_{2k} - \eta_k}{\eta_{4k} - \eta_{2k}} \tag{54}$$

If $\hat{\alpha} = max(\hat{\alpha}^{(H)}, \hat{\alpha}^{(P)}) < 2$, it is supposed that there is not enough data for the expected minimum estimation; otherwise, the expected minimum is:

$$\hat{m} = \eta_1 - C_k(\eta_k - \eta_1) \tag{55}$$

where $C_k = \frac{\hat{\alpha} - \gamma}{\psi(k+1) + \gamma}$, where γ≈0.577 is the Euler-Mascheroni constant and ψ(.) is the digamma function. Since k is an integer, $C_k = \frac{\hat{\alpha} - \gamma}{\sum_{j=1}^{k} \frac{1}{j}}$. If the estimated minimum improves the incum-

bent solution for less than 15 seconds, the ACO process can be stopped to reduce the computational load.To find a solution, using the transition probabilities (44) and (47), Algorithms 1-5 have to be slightly modified. Derigs and Reuter (2009) [59] and Fleszar et al. (2009) [61] reported that ACO is eeasily adapted to VRPTW and TDVRPTW. .

The First Step of the TDVRPTW ACO Feasible Solution

Following the maximal free time (MFT) heuristic (Balseiro et al., 2008) [43]), the path is started with the earliest possible delivery pair, having $t_0=min(t_{ij}^0)$. The first pair is selected from set A=argmin (t_{ij}^0)(see equation 25) of the customers with the same minimal arrival time. The transition probability (48) is applied to these pairs, and a random seed pair is selected according to the given transition probability distribution. The process is described in Algorithm 8. The input of the algorithm is the saving matrix $S = \|s_{ij}\|$start time matrix $T_0 = \|t_{ij}\|$, pheromone matrix $\Phi = \|\varphi_{ij}\|$, and working SA temperature T from Algorithm 7.

Algorithm 8: Selection of the first seed pair (u_1, u_2)

Step 1 Sort all values $t_{ij} > 0$ in ascending order and put into the list T^U.

Step 2 If the list T^U is empty, i.e., no positive saving exists, then the optimization cannot be done, and the on-demand delivery is optimal. Otherwise, find the list of unique values of T^U: t^U, arranged in ascending order.

Step 3 For each $t \in t^U$ repeat the following steps until a pair of (u_1, u_2) is not found:

Step 3.1 Find set $A = \{i,j|t_{ij} = t\}$.

Step 3.2 Extract from the saving matrix S set S set S^U= $\{s_{ij}|(i,j)\in A\}$ and calculate $s_{max} = \max(S^U)$

Step 3.3 If $s_{max}\leq 0$, then continue with the next $t \in t^U$.

Step 3.4 Extract from the pheromone matrix Φ set $\Phi^U = \{\varphi_{ij}|(i,j) \in A\}$, and calculate the sum of $S_\Phi{}^U=S^U+\Phi^U$.

Step 3.5 If $\max(S_\Phi^U) \leq 0$, then continue with the next $T \in t^U$.

Step 3.6 For each $s \in S_\Phi^U$calculate probability: $p(s) = exp\left(-\frac{s_{max}-s}{T}\right)$

Step 3.7 Calculate P_S=∑p(s), sort $p(s)$ in descending order, calculate the softmax $p'(s) = \frac{p(s)}{P_S}$, , where the cumulative sum (probability distribution function) is $P(s)=\sum_0^s p'(s)$

Step 3.8 Pick a random number $0 \leq \zeta \leq 1$ and find the first s=argmin(P(s)≥ζ)

Step 3.9 Find a pair (u_1,u_2)={$(u_1,u_2)\in$A|s(u_1,u_2)=s}, having the saving value s.

Step 4 If no pair is found, the optimization cannot be done, and the on-demand delivery is optimal; otherwise, create the tabu list T={u_1,u_2} of already visited customers.

Step 5 Define the seed route as: $\{(0, u_1, u_2, 0), (t_0^d(u_1), t_{u_1u_2}^0, t_{u_2}, t^r)\}$, where the depot departure time $t_0^d(u_1)$ is the root of equation (20).

The Head Interior Heuristics for the TDVRPTW ACO Feasible Solution

The input for the head insertion heuristics for the ACO solution is the same as for the greedy solution: k ready feasible paths: $\pi_1,\pi_2,...,\pi_k$, where $\pi_i = (u_1^i, u_2^i, \dots, u_{n_k}^i|t_i^d)$ is an ordered list of the visited sites, along with the pickup time t_i^d, a list of m unvisited customers: Y = $(y_1, y_2,...,y_m)$ and the current feasible path $\pi_c = (u_1^c, u_2^c, \dots, u_{n_c}^c|t_c^d)$ under construction. Besides, we have the pheromone matrix Φ=$\|\varphi_{ij}\|$, and working SA temperature T. As in Section 5, let us denote a soft tabu list as $ST = (u_{n_1}^1, u_{n_2}^2, \dots, u_{n_k}^k)$. Algorithm 9 describes the process and returns the updated current path $\pi_c = (u_1^c, u_2^c, \dots, u_{n_c}^c, u_h^c|t_c^d)$

Algorithm 9: TDVRPTW ACO Head Interior Insertion

Step 1 For the current path, calculate the following parameters using the expressions (28)-(33) from Algorithm 2: current demand Q_c, the arrival time at the last customer t_c, total delivery shift CS_y, on-demand duration Do , calculated by equation (26), current path duration D_c, calculated by equation (27), and current saving s_0.

Step 2 If the customer y ∈Y does not violate capacity and time constraints, calculate the saving $s_y = D_y^0 - D_y$, following Steps 2.1 – 2.6 of Algorithm 2.

Step 3 For each customer z ∈ ST from the soft tabu, calculate $s_z=D_{0i}-D_i$, using Steps 4.1-4.4 of Algorithm 2 and Δs_z, using Steps 4.5-4.7 of Algorithm 2.

Step 4 For relevant customers, having $s_y > s_0$, add pheromone factor and calculate weighted time-saving [41]:

$$s_y' = (s_y - s_0 + \varphi_{cy})CWDTH(y) \quad (56)$$

Step 5 For each customer from the soft tabu list $z \in ST$ calculate the weighted time-saving concerning saving lost Δs_z, calculated by (31) and pheromone factor:

$$s_z' = (s_z - s_0 - \Delta s_z + \varphi_{cz})CWDTH(z) \quad (57)$$

Step 6 Concatenate all valid customers from Y, having $s_y > s_0$ and all valid customers from ST, having $s_z-\Delta s_z>s_0$. Get a concatenated list $s_x' = \left\{\frac{s_y'}{T}, \frac{s_z'}{T}\right\}$ and sort s_x in descending order.

Step 7 Calculate the maximal saving in the list: $s_{max}=max(s_x')$ and probability density function: $p(x) = exp(s_x' - s_{max})$

Step 8 Calculate the normalizing factor P_0=∑p(x) , and the cumulative distribution function:

$$P(s) = \sum_{x-1}^{s} \frac{p(x)}{P_0} \quad (58)$$

Step 9 Pick a random number 0≤ ξ ≤1 and find the first argument x for which P(s) exceeds the random number ξ.

Step 10 If x∈Y, then path π_c is updated along with its saving value s_x. It x∈ST, than path π_c is updated by the insertion of customer x, and customer x is removed from the source path π_z .

If Algorithm 8 did not detect any valid customers and returned with $s_{max}\leq 0$, the same actions as in Algorithm 2 are taken, and a new insertion position ν is calculated.

The Tail Interior Heuristics for the TDVRPTW ACO Feasible Solution

The input and output are identical to Algorithm 9 of TDVRPTW ACO Head Interior Insertion. The tail interior insertion algorithm is described in Algorithm 10 as follows.

Algorithm 10: TDVRPTW ACO Tail Interior Insertion

Step 1 For the current path, calculate the following parameters using the expressions (28)-(33) from Algorithm 3: current demand Q_c, the arrival time at the last customer t_c, total delivery shift CS_y, on-demand duration D_0 , calculated by equation (26), current path duration D_c, calculated by equation (27), current saving s_0, and the arrival time at the first customer u_1^c of the current path $t_1 = t_c^d + \tau_{0u_1^c}(t_c^d)$

Step 2 If the customer y∈Y does not violate capacity and time constraints, calculate saving $s_y = D_y^0 - D_y$, following Steps 4.1-4.7 of Algorithm 3.

Step 3 For relevant customers, having $s_y > s_0$, add pheromone factor and calculate weighted time-saving [41]:

$$s_y^{'} = \left(s_y - s_0 + \varphi_{yu_1^c}\right)CWDTH(y) \quad (59)$$

Step 4 If the customer z∈ST does not violate capacity and time constraints, calculate saving $s_z = D_{0i} - D_i$, along with the lost saving Δs_z, following Steps 6.1-6.4 of Algorithm 3.

Step 5 For each customer from the soft tabu list z∈ST calculate the weighted time-saving concerning saving lost Δs_z, calculated by (31) and pheromone factor:

$$s_z^{'} = \left(s_z - s_0 - \Delta s_z + \varphi_{yu_1^c}\right)CWDTH(z) \quad (60)$$

Step 6 Repeat Steps 6-10 of Algorithm 10 and get the updated current path $\pi_c = \left(u_t^c, u_1^c, u_2^c, \ldots, u_{n_c}^c | t_{ct}^d\right)$. If Algorithm 9 did not detect any valid customers and returned with $s_{max} \leq 0$, the same actions as in Algorithm 3 are taken, and a new insertion position v is calculated.

Solomon's PFIH for the TDVRPTW ACO Feasible Solution

In this case, the new customer is inserted at any position $1 < v < n_C$ of the current path π_c. The tail PFIH algorithm is described in Algorithm 11. The input of algorithms is the partial VRP with k feasible paths: $\pi_1, \pi_2, \ldots, \pi_k$, where $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{nk}^i | t_i^d\right)$, along with the list of the visited sites, with the pickup time t_i^d, a list of m unvisited customers: $Y = (y_1, y_2, \ldots, y_m)$, the current feasible path $\pi_c = \pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d\right)$ pheromone matrix $\boldsymbol{\Phi} = \|\varphi_{ij}\|$, working SA temperature T, and insertion position $1 < v < n_c$. The output is the updated current path $\pi_C = \left(u_1^c, u_2^c, \ldots, u_v^c, \ldots, u_{n_c}^c | t_c^d\right)$.

Algorithm 11: Solomon's **PFIH** for **ACO of TDVRPTW**

Step 1 Create a soft tabu list ST $= \left(u_{n_1}^1, u_{n_2}^2, \ldots, u_{n_k}^k\right)$, and calculate path parameters Q_c, D_0, and s_0 by applying Step 1 of Algorithm 9.

Step 2 Calculate the on-demand duration D_{0v} up to the customer u_v^c , using equation (40), and the current path duration up to the customer u_v^c without returning to the depot D_v , using equation (41).

Step 3 If the customer y∈Y does not violate capacity and time constraints, calculate the arrival time t_y, and saving $s_y = D_y^0 - D_y$, following Steps 6.1-6.5 of Algorithm 4.

Step 4 For relevant customers, having $s_y > s_0$, add pheromone factor and calculate weighted time-saving [41]:

$$s_y^{'} = \left(s_y - s_0 + \varphi_{u_v^c y} + \varphi_{y u_{v+1}^c} - \varphi_{u_v^c u_{v+1}^c}\right)CWDTH(y) \quad (61)$$

Step 5 If the customer z∈ST does not violate capacity and time constraints, calculate saving $s_z = D_{0i} - D_i$, along with the lost saving Δs_z, following Steps 8.1-8.6 of Algorithm 4.

Step 6 For each customer from the soft tabu list z∈ST calculate the weighted time-saving concerning saving lost Δs_z, calculated by (32) and pheromone factor:

$$s_z^{'} = \left(s_z - s_0 - \Delta s_z + \varphi_{u_v^c z} + \varphi_{z u_{v+1}^c} - \varphi_{u_v^c u_{v+1}^c}\right)CWDTH(z) \quad (62)$$

Step 7 Repeat Steps 6-10 of Algorithm 10 and get the updated current path $\pi_c = \left(u_1^c, u_2^c, \ldots, u_v^c, \ldots, u_{n_c}^c | t_c^d\right)$

If Algorithm 11 did not detect any valid customers and returned with $s_{max} \leq 0$, the same actions as in Algorithm 4 are taken, and a new insertion position v is calculated.

Starting a New Path in the TDVRPTW ACO Feasible Solution

After each insertion into the current path, the best path is selected according to the rules collected in Table 3. If the new customer was selected from the soft tabu list, some additional actions, described in Section 5.6, will be processed. When the current path growth meets the terminal criteria, and no new customer can be inserted, a new path is started. Let $Y = (y_1, y_2, \ldots, y_m)$ be the list of m unrouted customers. If this list is empty, all sites have been visited, and the greedy feasible solution is ready. If $m=1$, then the new path is trivial: $\pi = (0, y, 0)$. If m>1, then the new path can be obtained by applying Algorithm 8 on reduced matrices, $S' = \|s_{ij}\|$, $\boldsymbol{\Phi}' = \|\varphi_{ij}\|$, and $T'_0 = \|t_{ij}\|$, where i, $j \in Y$. Eventually, the ACO solution creation algorithm is summarized as follows.

Algorithm 12: TDVRPTW Solution Creation

Step 1 Create the greedy solution using Algorithms 1-6. Set the best solution to the greedy solution and initial pheromone matrix $\boldsymbol{\Phi} = \|\varphi_{ij}\|$ to zero matrix, $\varphi_{ij} = 0$. Calculate the working SA temperature T using Algorithm 7.

Step 2 Create the first seed pair using Algorithm 8.

Step 3 Grow the new path π_c , using Algorithms 9-11 until the saving is positive.

Step 4 If there is more than one unvisited site after updating the tabu list, create a new path on the reduced matrices and repeat Steps 2 and 3.

Step 5 The current solution is ready if the list of unvisited sites is empty or contains only one customer.

Step 6 If the current solution has fewer paths, i.e., $k < k_{best}$, replace the best solution with the current one, according to the primary objective (2).

Step 7 If the current solution has the same number of paths ($k=k_{best}$), but the saving exceeds the best saving ($S_0 < S_{best}$), replace the best solution with the current one, according to the secondary objective (5).

Step 8 Update the pheromone matrix $\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi} + \Delta\varphi_{ij}$, where $\Delta\varphi_{ij} = \frac{QS_{\max}}{\lambda N}$

Step 9 Estimate the minimal duration $\hat{m}$ as the root of equation (51) or using equation (55). If the potential improvement is less than 15 seconds, i.e.,$(\eta_1 - \hat{m})$ <15sec, the best solution is ready. Where the value η_1 is the sum of the durations of all paths in the best solution.

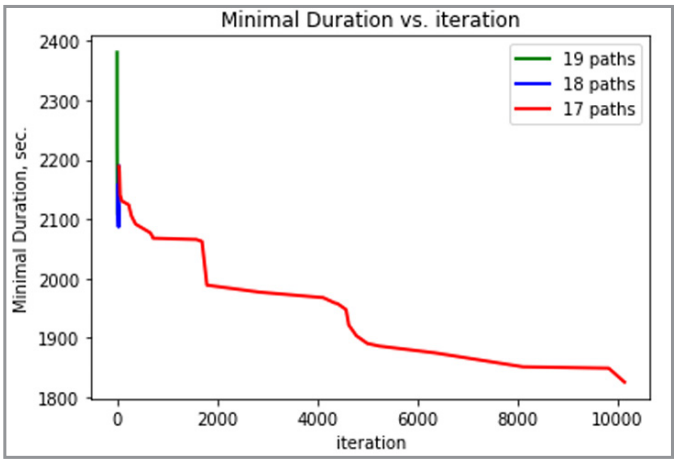


Figure 7: The ACO TDVRPTW optimization progress vs. iterations, the number of paths is the primary objective, and the total minimal makespan duration is $D = \sum_{i=1}^{k}(t_i^r - t_i^d)$

Step 10 Repeat Steps 2-9 until the convergence (Step 9) or the maximum number of iterations N_a, or computational time limit (typically 1-4 hours) is reached.

As greedy as the ACO algorithm for a single path has the same complexity, i.e.,$O(n^2)$ Since the maximal number of paths is limited by n, the overall complexity of one solution is $O(n^3\ln(n))$. Since N_a is also proportional to epoch, the overall complexity to get the best solution is $O(n^4\ln(n))$. Figure 7 illustrates the typical progression of the ACO optimization, with the number of paths as the primary objective and the minimal makespan duration as the secondary objective.

ACO Solution Acceleration for TDVRPTW Using the Riun and Recreate Strategy

After 1.5–2 epochs of the ACO algorithm's execution, there is sufficient pheromone and historical VRP statistics to accelerate the optimization process using the Ruin and Recreate (R&R) strategy (Shaw, 1998) [74]. With this data, the top ten elite solutions, which have the best saving values or minimal total duration (47), are inserted into the Tabu List. This Tabu List is also known as Short-Term Memory (STM) (Glover & Laguna, 1997) [62], component of Long Short-Term Memory (LSTM). The incumbent solution is selected from the Tabu List according to the SA temperature (Schrimpf et al., 2000) [73]. For each solution, the probability of being selected is:

$$p(s_i) = \frac{1}{C} exp\left(-\frac{s_{max}-s_i}{T}\right) \quad (63)$$

Where, s_i is the saving of solution i from the Tabu List, T is the SA temperature, s_{max}=max(s_i), and $C = \sum_i exp\left(-\frac{s_{max}-s_i}{T}\right)$.

ACO Ruin Method

At the ruin stage, 10% of the shortest tours with low string cardinality and less than 80% of the vehicle load are removed from the incumbent solution [51]. The vehicle load is defined as the ratio of the number of parcels distributed on the path to the vehicle capacity q_{max} (see inequality (6)). The rationale for this small removal rule is shown in Figure 8 [36]. The histogram of Figure 7(a) shows that the short paths are only a small fraction of the total path length distribution, and Figure 7(b) shows that the short paths are underloaded. Typically, the short paths consist of customers who are hardly incorporated into the longer paths, and their removal gives them another chance to integrate these short paths into longer ones. To reach a sufficient number of removed customers, an integer random number min(0.1n,30)≤γ≤min(0.4n,60) is selected [36].

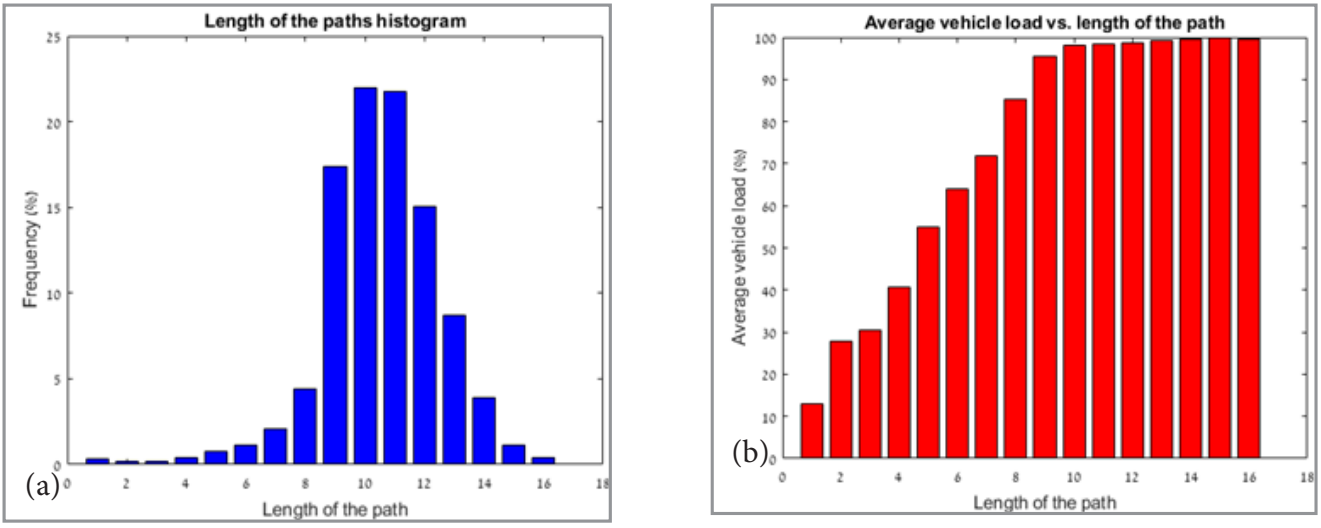


Figure 8: Histogram of the length of the tours on 950 historical solutions, limited to 16 stop points (a) and the average vehicle load's dependency on the path length (b)

If the sum of the customers in the removed top shortest paths is less than γ, the rest of the customers are removed according to the relatedness matrix $R=\|r_{ij}\|$ (Pisinger & Ropke, 2007; Shaw, 1998) [36, 74], where

$$r_{ij} = \varphi\Delta D_{ij} + \chi\left(1 - J\left(w_i, w_j\right)\right) + \\ + \psi\frac{|q_i - q_j|}{q_{max}} + \omega\left(1 - \frac{\varphi_{ij}}{max(\Phi)}\right). \quad (64)$$

In equation (64) ΔD_{ij} is the duration impact that is calculated as: $\Delta D_{ij} = \frac{\tau_{ij}\left(t_{ij}^0\right)}{\max_{i,j}\tau_{ij}\left(t_{ij}^0\right)}$, if path *(i, j)* is feasible and $\Delta D_{ij} = 1$ otherwise.

Where $\tau_{ij}\left(t_{ij}^0\right) = \tau_{ij}\left(Bucket\left(t_{ij}^0\right)\right)$ (see equation (18)) and t_{ij}^0 is the root of equation (22). Maximum $\max_{i,j}\tau_{ij}\left(t_{ij}^0\right)$ is calculated for the feasible paths only, having $t_{ij}^0 > 0$. Following the recommendation by Pisinger and Ropke (2007) [36], duration weight $\varphi = 9$.

Time impact $J(w_i,w_j)$ is the time-windows intersection over union (IoU) ratio, or Jaccard index

$$J\left(w_i, w_j\right) = \frac{w_i \cap w_j}{w_i \cup w_j} = \frac{max\left(0, min(l_i, l_j) - max(e_i, e_j)\right)}{l_i - e_i + l_j - e_j - w_i \cap w_j} \quad (65)$$

Time-windows weight following Pisinger and Ropke (2007) [36] is set to $\chi = 5$, capacity weight $\psi = 1$, and pheromone weight $\omega = 3$.

The next customer to be removed or seed customer (Christiaens & Berghe, 2020) [51] is selected using the Shaw removal algorithm on the relatedness matrix (64). The process is summarized in Algorithm 13, which receives as input the relatedness matrix $R = \|r_{ij}\|$, and the list of removed customers $D = (c_1, .., c_l)$, where $l < \gamma$. The algorithm returns seed customers c^{seed}.

Algorithm 13: Seed Customer Selection

Step 1 For all $c_i \in D$ until c^{seed} is not found, do the following steps:

Step 1.1 From matrix R, extract row $=R(c_i, j), j = 1, .., n, j \neq c_i$

Step 1.2 Sort row L in ascending order.

Step 1.3 Pick a uniformly distributed random number r ∈ [0,1] and select the customer at the position $\lfloor (n-1)r^p \rfloor$ in the sorted list L, i.e., $c = arg\left(L((n-1)r^p)\right)$, where $\lfloor . \rfloor$ denotes the integer part. The power p following Pisinger and Ropke (2007)[36] is set to p=3, so the average selected position is $(n-1)/4$ with the standard deviation of 3(n-1)/(4√7).

Step 1.4 If c∈D repeat Steps 1.1-1.3, otherwise $c^{seed} = c$, and the seed customer is found.

Step 2 If a seed customer is not found after Step 1 is passed for the entire list D, create a row flat vector $L_{row} = r_{ij}, i \neq j$, of length n(n-1), and sort it in ascending order.

Step 3 Pick a uniformly distributed random number $r \in [0,1]$ and select the pair of customers at the position $\lfloor n(n-1)r^p \rfloor$ in the sorted list L_{row}, i.e., $c_{pair} = arg\left(L_{row}(n(n-1)r^p)\right)$,, where p = n-1. Thus, the average pair position is (n-1), and the standard deviation is $\frac{(n-1)}{n\sqrt{2n-1}}$.

Step 4 Create a minimal spanning tree (MST) on matrix R using the Kruskal algorithm [52]. This step is processed only once if needed.

Step 5 Cluster removal heuristics is applied on the MST between pairs of customers from c_{pair}, and cluster c_{clust} is created [53].

Step 6 If all customers of c_{clust} is already removed, i.e., $c_{clust} \in$D, repeat Steps 3 and 5.

Step 7 If at least one of the customers in c_{clust} is not already removed, then $c^{seed} = c_{clust} \backslash D$.

If the seed customer is the last $u_{n_c}^c$ in the tour, the route $\pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d\right)$ is updated as $\pi'_c = \left(u_1^c, u_2^c, \ldots, u_{n_c-1}^c | t_c^d\right)$, and customer $u_{n_c}^c$ is added to the list of removed customers D. If the seed customer is the first customer u_1^c in the tour π_c, the route is updated as $\pi'_c = \left(u_2^c, u_3^c, \ldots, u_{n_c}^c | t_c^d - \Delta t\right)$ where Δt is the shift of the depot arrival time concerning the MFT principle, calculated with Algorithm 3. If the seed customer is located at position $1<v<n_c$ in the middle of the tour, try to insert the following customers $u_{v+1}^c, \ldots, u_{n_c}^c$ after the customer u_{v-1}^c in the same order again using Algorithm 4. If the tour is feasible, the updated tour is $\pi'_c = \left(u_1^c, u_2^c, \ldots, u_{v-1}^c, u_{v+1}^c, \ldots, u_{n_c}^c | t_c^d\right)$. If the tour becomes infeasible, string removal (Christiaens & Berghe, 2020) [52] is applied, and customers $u_{v+1}^c, \ldots, u_{n_c}^c$ are removed along with the seed customer u_v^c. If v=2 and string removal is applied, the pass π_c is completely removed so as not to leave a route with only one customer.

All removed customers are added to the list of removed customers *D*, and if $\|D\| < \gamma$,, the next customer from c^{seed} is removed. If the list c^{seed} is empty, Algorithm 13 is applied again until at least γ customers are removed.

ACO Recreation Method

The recreation method is based on the parallel regret heuristics (Potvin & Rousseau, 1993) [21]. The difference from the greedy interior insertion algorithms (Algorithms 2-4) is in the input and output. The input consists of two kinds of paths: untouched paths $\pi_i = \left(u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d\right), i = 1, \ldots, k$ and ruined paths $\pi_c = \left(u_1^c, u_2^c, \ldots, u_{n_c}^c | t_c^d\right), \ c = 1, \ldots, l$. If $l \leq 1$,the greedy interior insertion Algorithms 1-5 are processed without change. The list of unvisited customers: $Y = (y_1, y_2, \ldots, y_m)$ is the list of removed customers *D* (in this case, *Y=D*). The output is an updated list of the paths: π_i and π_c, and the list of unvisited customers *Y*. The soft tabu list *ST* is calculated for only untouched paths π_i. Algorithms 2-4 are finished by calculating the savings s_y' for unvisited customers and the savings s_z' for the soft tabu. Eventually, two candidate paths are created. There are the best routes $\pi_h^j(c)$ and $\pi_h^j(c)$, having maximal saving values $s_y' \cup s_z'$ for head and tail interior insertion and the second-best routes: $\pi_h^j(c)$ and $\pi_h^j(c)$, where j=1,…,m, and c=1,…,l. For all these routes, the maximal 2-regret heuristic is calculated as follows [36]:

$$\Delta_{max} = \max_c \max_j max\left(\Delta S_h(j,c), \Delta S_t(j,c)\right), \quad (66)$$

$$\text{where } \Delta S_h(j,c) = s'\left(\pi_h^j(c)\right) - s'\left(\pi_{nh}^j(c)\right),$$

$$\text{and } \Delta S_t(j,c) = s'\left(\pi_t^j(c)\right) - s'\left(\pi_{nt}^j(c)\right)$$

providing $s'\left(\pi_{nh}^j(c)\right) > 0$ and $s'\left(\pi_{nt}^j(c)\right) > 0$, where $s'(.)$ is the saving value. The process of greedy recreation using the parallel regret heuristics is summarized in Algorithm 14, which is finished with the solution: $\Pi = \{\pi_i = (u_1^i, u_2^i, \ldots, u_{n_k}^i | t_i^d)\}$, where $i = 1, \ldots, k$.

Algorithm 14: Greedy Recreation Using the Parallel Regret Heuristics

Step 1 While the list of unvisited customers *Y* is not empty and there are at least two ruined paths (l>1) do the following steps:

Step 1.1 For each ruined path π_c calculate savings $\{s'(\pi_{hj}(c)), s'(\pi_{tj}(c))\}$ for head and tail interior insertions.

Step 1.2 If $\left(s'\left(\pi_h^j(c)\right), s'\left(\pi_t^j(c)\right)\right) \le 0$,i.e., tour π_c cannot grow anymore, remove π_c from ruined paths and add to the untouched paths.

Step 1.3 Otherwise, calculate regret value, using the best and the second-best savings as follows:

$$\Delta_c = \begin{cases} s'(\pi_h(c)) - s'(\pi_{nh}(c)) \\ \text{if } s'(\pi_h(c)) \ge s'(\pi_t(c)) \text{and } s'(\pi_{nh}(c)) > 0 \\ s'(\pi_t(c)) - s'(\pi_{nt}(c)) \\ \text{if } s'(\pi_t(c)) > s'(\pi_h(c)) \text{ and } s'(\pi_{nt}(c)) > 0 \end{cases} \quad (67)$$

where $s'(\pi_h(c)) = \max_i s'\left(\pi_h^j(c)\right), s'(\pi_h(c)) = \max_i s'\left(\pi_h^j(c)\right)$ $s'(\pi_{nh}(c))$, and $s'(\pi_{nt}(c))$, are the second-best savings. If the winner of (67) does not have the second-best savings, $\Delta_c = \Delta_{max}$. If the urgency indicator $\kappa_t \neq \kappa_h$, the winner in (67) changes according to Table 3.

Step 1.4 Using (67), create a regret list $\Delta = \{\Delta_c, \pi_c, \pi_{nc}\}$, where π_c and π_{nc} are the best and the second-best paths. If the second-best path does not exist $\pi_{nc} = \emptyset$.

Step 1.5 Select the requested path c*,y*=argmax(Δ_c), and update the appropriate path with π_{c^*}.

Step 1.6 Remove customer *y** from the list Y and all items from the regret list Δ, having an intersection with the updated path, i.e., $\pi_c \cap \pi_{c^*} \neq \emptyset$ or $\pi_{nc} \cap \pi_{c^*} \neq \emptyset$.

Step 1.7 If the regret list is not empty, repeat Steps 1.5 and 1.6; otherwise, repeat Steps 1.1-1.6.

Step 2 If the list of unvisited customers Y is not empty but has fewer than two ruined paths, apply greedy Algorithms 2-4.

The complexity of Algorithm 14 can be roughly estimated as $O(\alpha k m n N_{itr})$, where $k_r = \alpha k$ is the number of ruined paths as part of the total number of paths k, typically, $\alpha \le 0.2$, and N_{itr} is the maximal number of iterations of Step 1 of the algorithm. Since, $N_{itr} < m$ and $k \approx \frac{n}{q_{max}}$, then the complexity is $O\left(\frac{\alpha m^2 n^2}{q_{max}}\right)$. For a large number of customers (200 or more), the number of deleted customers is a constant, so the complexity is $O(n^2)$. For a small number of customers m~n, so the complexity is $O(n^4)$.

The ACO recreation has two changes from the greedy recreation algorithm. First, savings s_y and s_z' are calculated with pheromone matrix Φ, as in Algorithms 9-11. Second, if there are more than two items on the joint list of $s_y' \cup s_z'$, the roulette wheel selection principle (58) is applied twice on the removal, and two savings s_1' and s_2', are selected. The regret value is $\Delta_c = max(s_1', s_2') - min(s_1', s_2')$. Besides, in Step 2, algorithms 9-12 are applied. The complexity of the ACO recreation algorithm is also $O(n^2)$ for a large number of customers. The R&R algorithm is typically 2-5 times faster than ACO for 200–1,433 customers. When the R&R greedy or ACO solution is finished, the Tabu List is updated if needed.

To manage the adaptation between ACO (exploration) and R&R ACO (exploitation) algorithms, let us introduce a mini-batch (segment) size ϕ [48]. Following the recommendations of (Ribiero & Laporte, 2012) [53], the segment size is 50, but for the sake of the parallel computations, it is set to 32 [53]. Since in the R&R method, only a tiny part of the customers are deleted, and a small part of the tours are ruined, the epoch (49) becomes:

$$E_r = \frac{m(m-1)}{m-k_r}. \quad (68)$$

For example, if n = 200, and $k_{min} = 20$, then the ACO epoch (49) is E = 221. If m = 30, and $k_r = 5$, then the R&R epoch is $E_r = 35$. So after E_r exploitations, all feasible arcs will be revised, and the rest of the segment trials can be used for exploration. Let us define the minimal number of explorations per segment as E_{min}=min (E_r,0.4ϕ), and the maximal number as E_{max}=max(ϕ-E_r,0.8ϕ). To define the proportion between exploration and exploitation, let us consider the results of the latest $N_t$solutions, where N_t is set to 200. These solutions can be subdivided into results of exploitation and exploration algorithms, i.e., $N_t = N_{exploration} + N_{exploitation}$. If there are no representative statistics, i.e., $N_{exploration} < N_{min}$ or $N_{exploitation} < N_{min}$, then the number of recreations or exploitations is set to E_{min} if $N_{exploitation} < N_{min}$ and to E_{max} if $N_{exploitation} < N_{min}$, where N_{min} is set to 20. If there are representative statistics, two arrays $K_{exploration}$, and $K_{exploitation}$ of the number of paths in exploitation and exploration solutions are collected. The number of recreations is a convex combination: $p_w E_{min} + (1 - p_w) E_{max}$, where p_w=Wilcoxon($K_{exploration}$,$K_{exploitation}$) is the p-value of the Wilcoxon rank sum test about the statistical equivalence of the medians of $K_{exploration}$ and $K_{exploitation}$ arrays. The statistics collected for 186 industrial deliveries in Moscow, Russia, for 179-837 customers show the following distribution of the winning solutions. A greedy solution (Algorithms 2-5) wins 13% of the solutions. ACO solution (Algorithms 8-12) wins 23% of the solutions. The greedy R&R solution is the best in 40% of the industrial tasks, and the R&R ACO solution is the best at 24%. Thus, the greedy R&R solution is the cheapest and the most effective.

Results

The schematic chart of the TDVRPTW dispatching technology used in Gett Delivery is shown in Figure 9 [54]. The chart reflects the next-day delivery technology.

For each depot, the TDVRPTW task is solved once a day, typically between 1:00 a.m. and 5:00 a.m., for the next delivery date, which usually happens between 6:00 a.m. and 11:00 p.m.

The information includes the depot's geographical coordinates (latitude and longitude), street address, and open hours (time window). The new customer orders, collected from the previous day, along with the returned parcels, are the input for the task creator. Each customer provides its street address, converted to geographical coordinates using geocoding, contact information, a list of parcels to be delivered, a time window with a 5-minute rounding, a minimum length of 30 minutes, and the service time. The TDVRPTW solver receives the traffic model and the created task. The solver is written in Go and runs on the AWS server. The load balancer selects an appropriate server of 4-32 cores, depending on the number of customers per task. The resulting tours are sent to the relevant couriers, who use a courier routing application to manage the trip. The service dashboard displays routes and other information for the customer and the courier support. The customer receives a notification about the exact dispatching time. During the route, the courier's coordinates are collected every second to update the traffic model and the service time. The experiments were conducted in two different cities: Tel Aviv, Israel, and Moscow, Russia. The challenge in these experiments lies in comparing the TDVRPTW dispatching method with traditional dispatching when the city is divided among couriers, with each courier serving a specific area only.

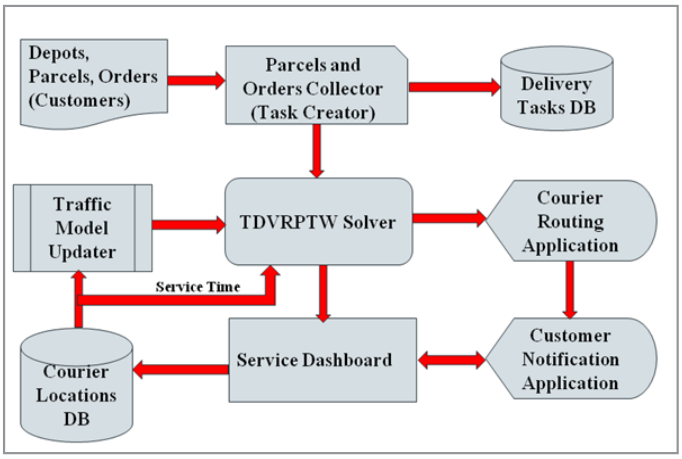


Figure 9: The TDVRPTW technology scheme of the dispatching system

Tel Aviv, Israel Metropolitan Area

The traffic model in the Tel Aviv, Israel, metropolitan area includes 78,275 edges with 45 different time buckets. Since delivery does not operate on weekends and holidays, only the weekday buckets are relevant. The 35-day test, conducted from July 29 to September 13, 2018, was selected for evaluation. The automatic dispatcher distributes, every day, up to 257 parcels (37,582 in total) between 2-155 customers (1.22 parcels per customer on average). There were 1-3 time window slices, rounding to a whole hour, and each slice lasted at least 4 hours. So, time windows do not significantly affect the routes.

An example of VRP in Figure 10 is similar to the VRP of academic benchmarks, cf. Solomon (1987) [37]. Additionally, there are some differences in dispatching within the Tel Aviv metropolitan area. First, the VRP is open, so the vehicle does not return to the depot but goes home after the last delivery. Second, there is only one objective function instead of (2) and (5). The functions are merged using a starting price for every tour, and the merged function is:

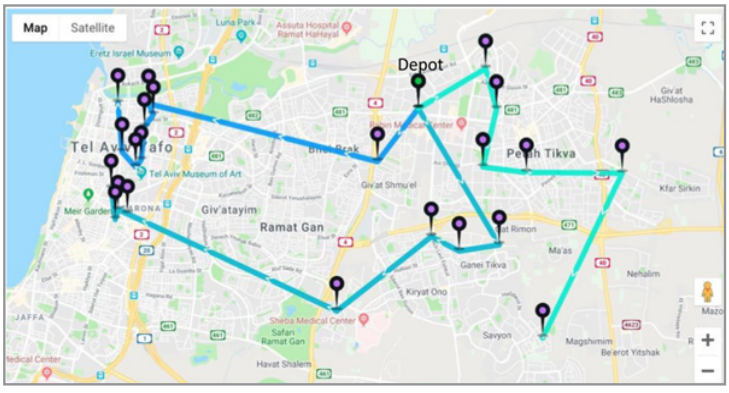


Figure 10: An example of the open VRP in the Tel Aviv metropolitan area

$$\min c_d \sum_{k\in K}\sum_{i,j\in C} d_{ij}^k x_{ij}^k + c_t \sum_{k\in K}\sum_{j\in C} x_{0j}^k \left(y_{n+1}^k - y_0^k\right) + c_{st} \sum_{k\in K}\sum_{j\in C} x_{0j}^k, \tag{69}$$

where c_{st} is a starting price. Third, the cost per kilometer c_d is not constant but is 3.3 NIS/km for the first 13 km and 2.75 NIS/km starting from the 14th km. The cost per hour c_t is 30 NIS/hour, and the start price c_{st} is 29 NIS. The service time g_i is constant for all customers and equals 10 minutes per stop point. The measure of the improvement is the relative price reduction in percent.

$$\mu_P = 100\frac{P_0 - P_{opt}}{P_0}, \tag{70}$$

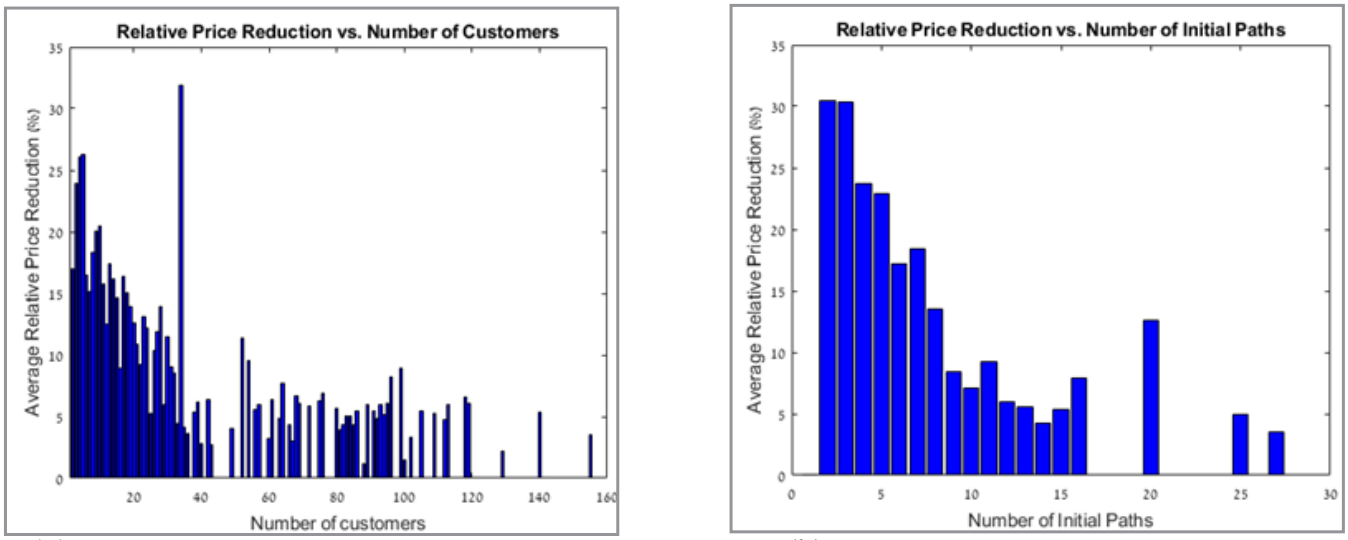
where P_0 is the price of the traditional dispatching and P_{opt} is the optimal price (69). The dependence of the average value of μ_P

from the number of customers and the initial number of paths are shown in Figure 11. The average price reduction is 19.2%. The relatively low improvement for the considerable number of customers (50 or more) is explained by using some optimization (greedy no traffic, no time windows) for these tasks in the traditional dispatching. Although the number of tours is not an objective of (69) optimization, improving the number of path reductions is also evaluated similarly to (70).

The measure of the number of tours reduction is

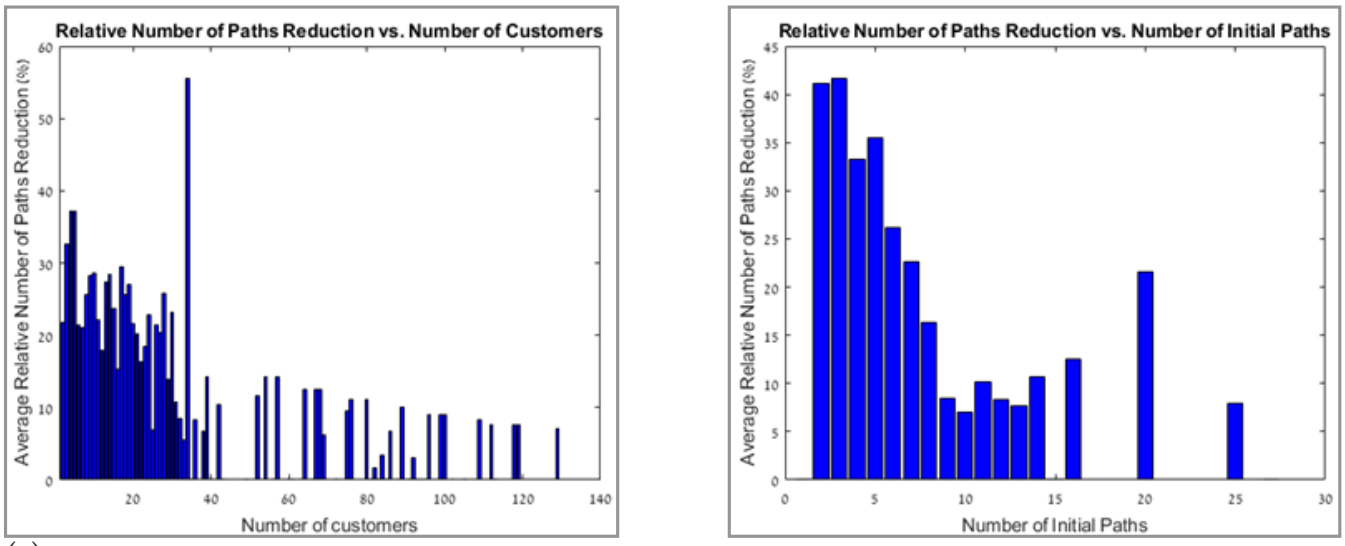
$$\mu_T = 100\frac{k_0 - k_{opt}}{k_0} \tag{71}$$

where k_0 is the number of tours of the traditional dispatching and k_{opt} is the optimal number of tours resulting from the minimization (69). The dependencies of the average value of μ_T from the number of customers and the initial number of paths are shown in Figure 12. The average price reduction is 26.2%. Although the number of tours is not an objective of the optimization, there is also a significant improvement in the number of paths.



(a) (b)

Figure 11: Dependency of the average relative price reduction (70) from the number of customers (a) and the number of the initial paths (b)



(a) (b)

Figure 12: Dependency of the average relative number of tours reduction (71) from the number of customers (a) and the number of the initial paths (b) in Tel Aviv, Israel

Moscow, Russia Metropolitan Area

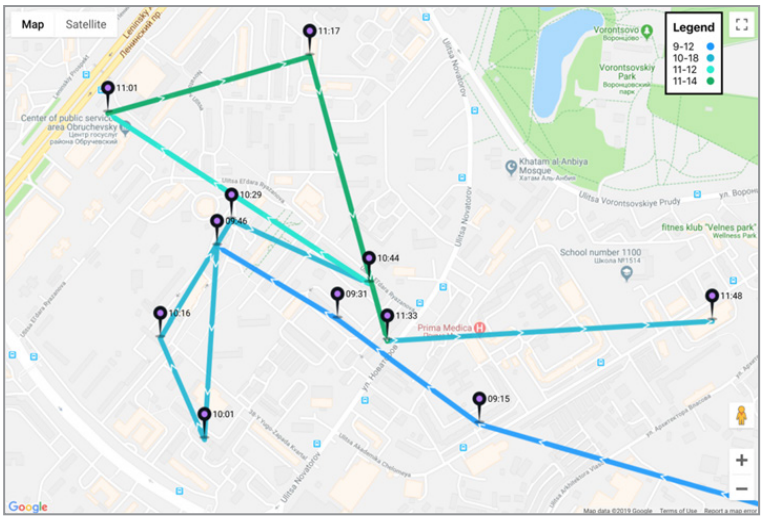


Figure 13: An example of the TDVRPTW tour in the Moscow metropolitan area: hard time windows, makespan, and traffic make the tour messy

The traffic model in the Moscow, Russia, metropolitan area includes 1,201,717 edges with 58 weekly time buckets. The primary objective function is (2), and the secondary is (5). A conventional taxi fleet is used for delivery, with a maximum capacity of 30 parcels and a service time of 12.5 minutes. The service time does not depend on the number of parcels per customer. Loading time is 30 minutes, and the maximum duration of a courier's working shift is 8 hours. The experiment spans 102 working days, from October 15, 2018, to January 19, 2020, involving three depots and 125 VRP tasks. On these VRP tasks, 12-1,022 parcels were distributed per task (27,414 parcels in total) between 12-837 customers (22,879 customers in total). During the optimization, the automatic dispatcher created 2-44 tours with 1-15 hard-constrained time window slices (5.9 on average per VRP). Hard-constrained time windows and traffic constraints make the routes messy and far from a fine academic shape. An example of a tour is shown in Figure 13.

In the traditional dispatching for the same tasks, 2-57 tours were created; however, only 12.6% of these routes were feasible, meaning every customer got parcels within the time window. The feasibility dependency from the initial paths is shown in Figure 14. The feasibility decreases with the number of paths, dropping from 20-40% for 4-8 paths to 2% for more than 40 paths. The plots in Figure 15 depict the effectiveness (71) of reducing the number of tours vs. the number of customers 15a and the number of initial paths 15b. The optimization process reduces the number of tours by an average of 18.5%. Due to the computational load, the computation time was limited to two hours. For this reason, for 500 or more customers, only one epoch was processed. As a result, the effectiveness of reducing the number of paths is slightly lower for a large number of customers and the initial paths.

The measure of improvement for the secondary objective function is the relative reduction in total VRP duration in percentage as follows:

$$\mu_D = 100\frac{D_0 - D_{opt}}{D_0}, \quad (72)$$

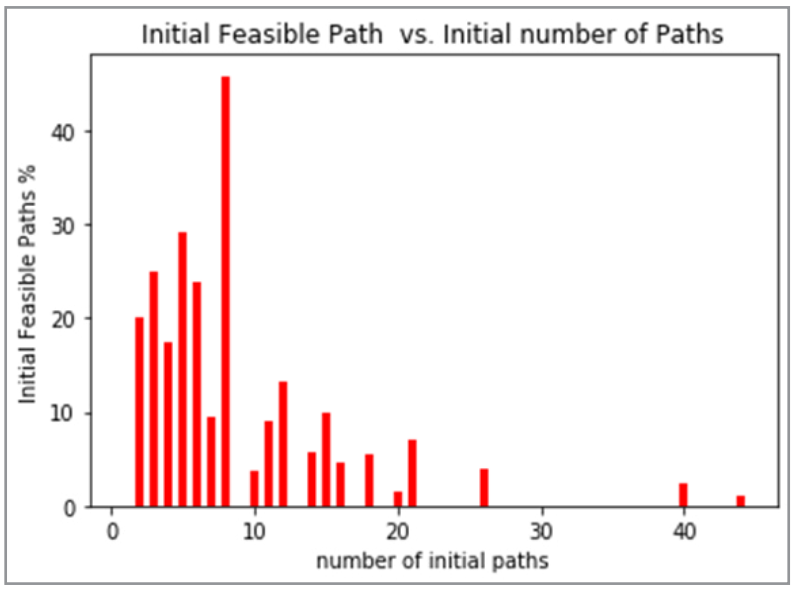
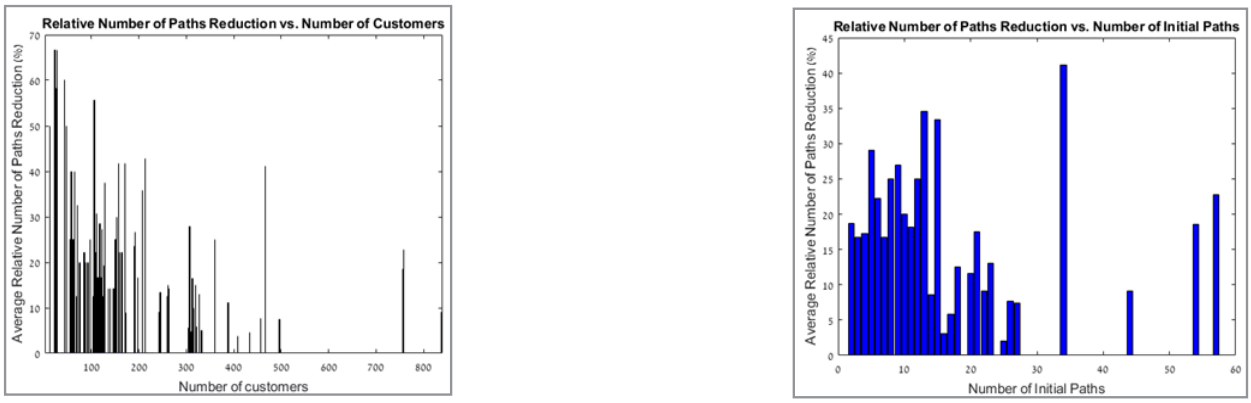


Figure 14: Dependency of the path feasibility of the traditional dispatching vs. the number of initial paths

where D_0 is the total VRP duration for the traditional dispatching and D_{opt} is the optimal total VRP duration. The plots in Figure 16 depict the average relative total VRP duration reduction (72) vs. the number of customers 16a and the number of the initial paths 16b. The optimization process reduces the total VRP duration by an average of 59.3% and is independent of the number of customers or the initial paths.

Although the total VRP distance is not an objective of the optimization, the relative distance reduction can also be measured using (72) by substituting distances instead of durations.



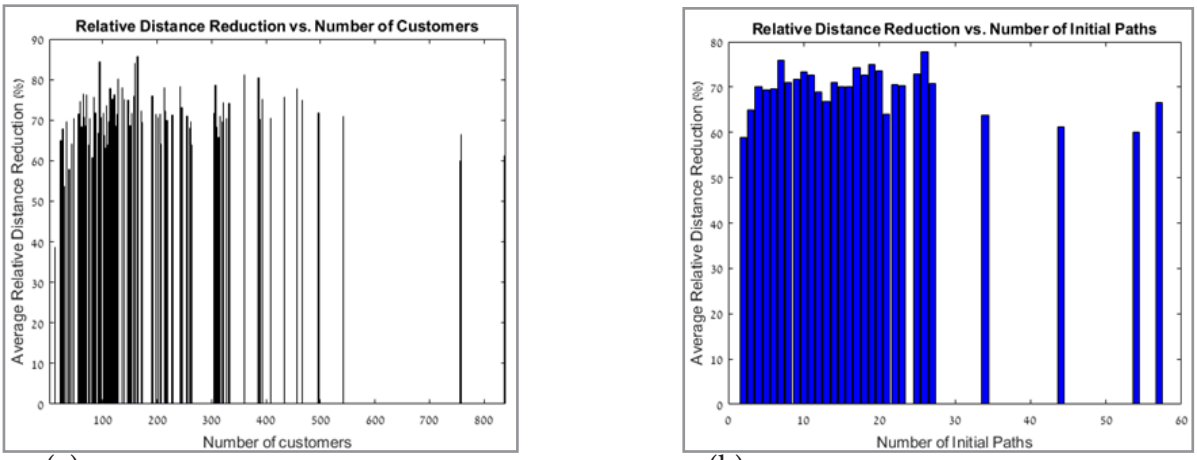
(a) (b)

Figure 15: Dependency of the average relative number of tours reduction (71) vs. the number of customers (a) and the number of the initial paths (b) in Moscow, Russia

(a) (b)

Figure 16: Dependency of the average relative total VRP duration reduction (72) vs. the number of customers (a) and the number of the initial paths (b) in Moscow, Russia

The plots in Figure 17 depict the average relative total VRP distance reduction vs. the number of customers 17a and the number of the initial paths 17b. The optimization process reduces the total VRP duration by an average of 69.7% and is independent of the number of customers or the number of initial paths.



(a) (b)

Figure 17: Dependency of the average relative total VRP distance reduction vs. the number of customers (a) and the number of the initial paths (b) in Moscow, Russia

Gehring & Homberger Benchmark

Comparing the TDVRPTW algorithms is challenging due to the varying traffic models and diverse constraints prevalent in the industry. Academic benchmarks, such as Solomon's benchmark or Gehring and Homberger's benchmark, comprise a set of 25-1000 customers, where each customer is represented as a 2D point with a given time window and service time [36, 55]. The Euclidean distance between the points is equivalent to the trip duration between the customers, and the benchmarks do not involve any traffic model. The benchmarks include randomized problems (R), clustered problems (C), and randomized-clustered problems (RC), and they are generated so that 75% of the customers, including the depot, have different time windows.

The reported best results were associated with the soft time window constraint, allowing for waiting before delivery [56]. The secondary objective function is the minimal sum of VRP distances. Following Figliozzi (2012) [1], these benchmarks represent the archetypal and ubiquitous case, where the central depot services a set of surrounding customers. To compare with the proposed industrial algorithms, let us compare the sum of VRP durations of the best algorithms given in [56]. The results for Solomon's benchmark are provided in Table 4.

Table 4: Comparative analysis with the best results on Solomon's benchmark. The first column is the instance's name, the second is the number of vehicles (tours) in the best solution, and the third is the number of vehicles found. Column "Vehicles difference" is the difference between the found and the best numbers of vehicles [56]. Column "The sum of duration best" contains the sum of VRP durations of the best solution, and the next column is the same sum for the found solution. The last column is the difference between the found and the best sum of durations.

Instance	Number of Vehicles best	Number of Vehicles found	Vehicles difference	The sum of durations best	The sum of durations found	Duration difference
R211	2	3	1	1964.63	2152.01	187.38
R210	3	3	0	2669.36	2391.77	-277.59
R110	3	3	0	2163.95	2384.15	220.2
R109	11	13	2	2228.45	2456.25	227.8
R101	19	22	3	3275.11	3542.18	267.07
R112	10	11	1	2093.97	2195.46	101.49
R107	10	12	2	2181.04	2406.34	225.3
R108	9	10	1	1516.5	1663.63	143.13
R103	13	15	2	2609.55	2809.9	200.35
R102	17	19	2	2346.74	2597.49	250.75
R105	14	14	0	2278.48	2468.464	189.98
R104	10	11	1	2212.9	2451.25	238.35
R201	4	4	0	3334.9	3018.46	-316.44
R202	3	4	1	2749.39	2772.6	23.21
R203	3	3	0	2626.21	2418.52	-207.69
R204	2	3	1	1989.14	2108.16	119.02
R205	3	3	0	2397.49	2355.95	-41.54
R206	3	3	0	2434.91	2304.88	-130.03
R207	2	3	1	1983.48	2169.93	186.45
R208	2	3	1	1858.36	1946.32	87.96
R209	3	3	0	2267.0	2385.41	118.41
R111	10	12	2	2189.54	2406.97	217.43
R106	12	13	1	2413.09	2524.47	111.38
C207	3	3	0	9660.4	9919.22	258.82
C206	3	4	1	9588.49	9816.83	228.34
C205	3	3	0	9588.88	9678.61	89.73
C204	3	4	1	9590.6	9691.03	100.43
C203	3	4	1	9601.72	9620.43	18.71
C202	3	4	1	9591.56	9666.33	74.77
C201	3	3	0	9591.56	9591.56	0
C208	3	3	0	9744.23	9641.11	-103.12
C108	10	10	0	9828.93	10017.51	188.58
C109	10	10	0	9828.93	10006.91	177.98

C104	10	10	0	10014.6	10101.57	86.97
C105	10	10	0	9828.93	9897.3	68.37
C106	10	11	1	9828.93	10070.95	242.02
C107	10	10	0	9828.93	9845.14	16.21
C101	10	10	0	9828.93	9878.06	49.13
C102	10	10	0	9828.93	10035.53	206.6
C103	10	11	1	10063.03	10158.71	95.68
RC102	12	14	2	2671.18	2795.68	124.5
RC201	4	5	1	3358.42	3222.83	-135.59
RC202	3	4	1	2683.88	2873.57	189.69
RC203	3	4	1	2670.9	2619.99	-50.91
RC204	3	3	0	2371.1	2238.71	-132.39
RC205	4	5	1	3286.82	3023.4	-263.42
RC206	3	4	1	2444.87	2586.82	141.95
RC207	3	4	1	2417.51	2477.53	60.02
RC103	11	12	1	2416.12	2534.14	118.02
RC101	14	16	2	2956.33	3005.78	49.45
RC107	11	13	2	2346.32	2523.14	176.82
RC106	11	13	2	2466.45	2536.23	69.78
RC105	13	15	2	2829.3	3010.08	180.78
RC104	10	11	1	2259.3	2337.51	78.21
RC208	3	3	0	2040.14	2137.69	97.55
RC108	10	12	2	2247.13	2347.39	100.26

Table 4 shows that soft time window constraints reduce the number of paths by 11.2% on average and the sum of duration in 72% of the instances, with an average reduction of 2.1%. Sometimes, one to two additional tours are added to avoid waiting before delivery, or the tour is extended to accommodate this. Thus, the courier spends the time on the route instead of waiting on site [57]. In real life, the courier does the same when he cannot find parking. For the clustered problems (C), the difference is less significant, with only 5% more tours and the sum of durations increasing by 1.1%. The results for the 200 customers for the Gehring and Homberger benchmark are provided in Table 5 [58].

Table 5: Comparison with the best results on the 200-customers Gehring and Homberger benchmark. The best solutions for instances R1_2_2 and RC1_2_7 are infeasible and are not included in the table.

Instance	Number of Vehicles best	Number of Vehicles found	Vehicles difference	The sum of durations best	The sum of durations found	Duration difference
R2_2_1	4	5	1	8716.66	9208.44	491.78
R1_2_7	18	19	1	8885.08	9526.84	641.76
R2_2_10	4	4	0	7936.55	6725.57	-1210.98
R1_2_10	18	18	0	8076.92	7166.44	-910.48
R1_2_3	18	20	2	9272.66	9802.95	530.29
R1_2_1	20	22	2	10556.83	10685.47	128.64
R1_2_2	18	19	1	9153.27	8750.87	-402.4
R2_2_8	4	4	0	7557.33	5999.59	-1557.74
R1_2_6	18	20	2	9386.6	9467.73	81.13
R1_2_8	18	18	0	8530.38	7951.27	-579.11
R2_2_3	4	5	1	8162.32	8550.82	388.5
R1_2_4	18	20	2	9093.99	9715.19	621.2
R2_2_2	4	5	1	8412.9	8619.92	270.02
R2_2_4	4	4	0	8069.15	7186.26	-882.89
R2_2_9	4	5	1	8441.09	7593.83	-847.26

R1_2_5	18	20	2	9330.58	9647.16	316.58
R2_2_5	4	4	0	8507.56	7895.2	-612.36
R2_2_6	4	4	0	8285.85	7919.07	-366.78
R2_2_7	4	4	0	7922.48	9084.5	1162.01
C1_2_5	20	20	0	20771.36	20913.27	141.91
C2_2_6	6	7	1	19944.46	21206.9	1262.44
C2_2_2	6	7	1	20586.91	21444.3	857.39
C1_2_10	18	19	1	20724.36	21562.58	838.33
C2_2_4	6	7	1	20724.45	21101.5	377.05
C1_2_9	18	19	1	20702.82	21277.21	574.39
C2_2_5	6	6	0	20035.82	20075.36	39.54
C1_2_4	18	18	0	21009.45	21988.21	978.76
C2_2_1	6	6	0	19931.44	19935.26	3.82
C2_2_10	6	6	0	20150.78	20464.18	313.4
C1_2_7	20	20	0	20708.35	20984.54	276.10
C1_2_3	18	20	2	20708.35	21361.59	653.24
C2_2_3	6	7	1	20600.67	20722.77	122.1
C2_2_7	6	7	1	19913.8	20443.32	529.52
C1_2_2	18	20	2	21188.54	22151.95	963.41
C1_2_1	20	20	0	20788.07	20953.51	165.44
C2_2_8	6	6	0	19895.06	20269.04	373.98
C2_2_9	6	7	1	19944.6	20704.45	759.85
C1_2_6	20	22	2	20820.75	21451.92	631.17
C1_2_8	19	21	2	20841.41	21408.68	567.27
RC2_2_7	4	5	1	7999.51	7083.55	-915.96
RC2_2_4	4	5	1	6503.51	7229.99	726.48
RC2_2_3	4	5	1	7005.62	7917.79	912.17
RC2_2_8	4	5	1	7793.43	6631.37	-1162.06
RC1_2_3	18	19	1	8036.41	7154.3	-882.11
RC1_2_5	18	20	2	7702.14	8384.31	682.17
RC2_2_2	5	6	1	9880.44	9223.38	-657.06
RC2_2_6	4	5	1	8378.88	6928.98	-1449.9
RC1_2_8	18	19	1	7202.46	6616.12	-586.34
RC1_2_9	18	19	1	7319.88	6581.99	-737.89
RC2_2_1	6	8	2	11610.36	8600.49	-3009.87
RC2_2_9	4	5	1	5604.53	6117.24	512.71
RC1_2_10	18	18	0	6510.07	6994.95	484.88
RC1_2_1	18	20	2	8206.83	8363.97	157.14
RC2_2_5	4	6	2	8425.86	7292.44	-1133.42
RC1_2_2	18	19	1	7970.14	8128.69	158.55
RC1_2_4	18	19	1	6691.86	6091.76	-600.1
RC1_2_6	18	19	1	7672.45	7725.75	53.3
RC2_2_10	4	4	0	7026.54	5868.85	-1157.69

Table 5 shows that 40 out of 58 instances require one or two additional tours to meet the hard window constraints. The number of paths increases by 7.5% on average, varying from 6% for clustered problems to 9.7% for randomized-clustered problems [59]. Although in 38 out of 58 instances, the total duration is longer for the hard-windows constraints, the average duration decreases by 0.1%. For the randomized problems, the average duration decreases by 14.6%, and for the randomized-clustered problems, it decreases by 6.2%. However, the average duration increases by 2.5% for clustered problems with the maximum sum of the durations [60-63]. The results for the 400 customers for the Gehring and Homberger benchmark are provided in Table 6.

Table 6: Comparison with the best results on the 400-customers Gehring and Homberger benchmark. The best solutions for instances R2_4_4, C1_4_2, RC1_4_8, RC2_4_2, and RC1_4_10 are infeasible and are not included in the table.

Instance	Number of Vehicles best	Number of Vehicles found	Vehicles difference	The sum of durations best	The sum of durations found	Duration difference
R1_4_1	40	42	2	25982.71	27388.14	1405.43
R2_4_3	8	11	3	20939.27	21030.41	91.14
R1_4_4	36	39	3	20516.4	21245.44	729.04
R2_4_6	8	9	1	20095.05	22082.67	1987.62
R2_4_10	8	8	0	19426.61	15481.86	-3944.75
R2_4_1	8	11	3	21844.88	22683.51	838.63
R1_4_5	36	39	3	23255.8	23059.48	-196.32
R1_4_10	36	37	1	20931.92	17686.39	-3245.53
R2_4_2	8	11	3	21258.2	23634.7	2376.5
R2_4_8	8	8	0	19579.65	15503.25	-4076.4
R2_4_9	8	9	1	20278.93	18199.83	-2079.1
R1_4_7	36	40	4	21987.45	22677.07	689.62
R2_4_5	8	9	1	20747.97	17308.45	-3439.52
R1_4_2	36	40	4	23942.06	25540.72	1598.66
R1_4_6	36	40	4	23136.36	25505.44	2369.08
R1_4_3	36	40	4	23331.03	26256.5	2925.47
R1_4_8	36	40	4	18063.94	20131.89	2067.95
R1_4_9	36	40	4	22159.22	19720.62	-2438.6
R2_4_7	8	9	1	19552.09	20214.92	662.83
C1_4_10	36	38	2	43119.26	45556.11	2436.85
C2_4_10	11	12	1	39927.68	42265.89	2338.21
C1_4_5	40	40	0	43562.05	43698.66	136.61
C2_4_8	11	12	1	40233.2	42076.54	1843.34
C1_4_9	36	38	2	43164.86	45555.07	2390.21
C1_4_6	40	44	4	43558.23	46679.35	3121.12
C2_4_3	11	14	3	40248.81	43170.79	2921.98
C2_4_7	12	14	2	40704.18	43027.38	2323.2
C2_4_4	11	12	1	40208.82	42719.96	2511.44
C2_4_5	12	14	2	40699.22	41807.24	1108.02
C2_4_1	12	12	0	40277.39	40262.46	-14.93
C2_4_3	36	39	3	43990.81	46038.42	2047.61
C2_4_2	12	14	2	41104.13	43551.93	2447.8
C1_4_4	36	37	1	43593.59	47712.64	4119.05
C1_4_1	40	40	0	43676.04	43748.43	72.39
C1_4_8	37	41	4	43477.78	45405.12	1927.34
C2_4_9	12	14	2	40215.73	44283.84	4068.11
C2_4_6	12	13	1	40306.53	44213.94	3907.41
C1_4_7	39	41	2	43517.3	44687.31	1170.01
RC1_4_3	36	37	1	17583.16	18647.71	1064.55
RC1_4_1	36	40	4	19727.44	20476.29	748.85

RC2_4_5	8	12	4	20179.13	20855.89	676.76
RC2_4_7	8	10	2	19946.12	19105.9	-840.22
RC2_4_10	8	8	0	16842.17	14361.54	-2480.63
RC1_4_9	36	37	1	18758.29	16391.05	-2367.24
RC2_4_1	11	15	4	26896.81	18923.56	-7973.25
RC1_4_2	36	40	4	19861.16	21457.07	1595.91
RC1_4_4	36	37	1	15729.24	16217.69	488.45
RC2_4_2	9	13	4	21834.57	17839.72	-3994.85
RC1_4_6	36	39	3	19879.74	19066.85	-812.89
RC1_4_5	36	40	4	18719.41	21676.15	2956.74
RC2_4_3	8	12	4	19786.69	21737.83	1951.14
RC2_4_8	8	9	1	18992.04	15912.39	-3079.65
RC1_4_7	36	38	2	19501.69	19089.34	-412.35
RC2_4_6	8	12	4	20137.76	16310.08	-3827.68
RC2_4_9	8	8	0	18313.96	15120.81	-3193.15

Table 6 shows that 48 out of 55 instances require from one to four additional tours to meet the hard window constraints. This fact can be explained by the growth in paths, which is twice the average number of 400 customers compared to 200 customers [64]. The number of paths increases by 8.8% on average, varying from 6.7% for clustered problems to 10.5% for randomized-clustered problems. Although the total duration is longer in 37 out of 55 instances due to the constraints of the hard-time window, the average duration increases by only 1.2%. For the randomized problems, the average duration decreases by 0.4%, and for the randomized-clustered problems, it decreases by 6.2%. However, the average duration increases by 4.8% for clustered problems with the maximum sum of the durations. The results for the 600 customers for the Gehring and Homberger benchmark are provided in Table 7 [65-68].

Table 7: Comparison with the best results on the 600-customers Gehring and Homberger benchmark. The best solution for the RC2_6_10 instance is infeasible and not included in the table.

Instance	Number of Vehicles best	Number of Vehicles found	Vehicles difference	The sum of durations best	The sum of durations found	Duration difference
R2_6_1	11	16	5	45380.01	46283.73	903.72
R1_6_4	54	60	6	43208.03	48989.42	5781.39
R2_6_10	11	13	2	42717.59	30847.98	-11869.61
R2_6_6	11	12	1	42089.66	45080.08	2990.42
R2_6_3	11	14	3	43300.34	46076.56	2776.22
R1_6_10	54	56	2	48165.64	37569.47	-10576.17
R1_6_1	59	66	7	57705.58	62896.45	5190.77
R1_6_3	54	60	6	52052.31	57609.73	5557.42
R1_6_7	54	56	2	51213.63	51281.85	68.22
R2_6_4	11	13	2	42832.2	39367.62	-3464.58
R1_6_5	54	60	6	52473.93	51494.84	-979.09
R2_6_7	11	12	1	42256.58	40389.94	-1866.64
R2_6_5	11	15	4	44269.2	36508.63	-7760.67
R1_6_8	54	56	2	41603.94	43082.44	1478.5
R1_6_9	54	60	6	50895.16	45861.53	-5033.63
R2_6_8	11	11	0	40749.71	32286.83	-8462.88
R2_6_9	11	16	5	44342.27	37364.23	-6978.04
R2_6_2	11	16	5	45192.93	50781.1	5588.17
R1_6_2	54	60	6	54161.14	57631.05	3469.91
R1_6_6	54	59	5	51268.35	54306.64	3038.29
C2_6_4	17	19	2	61740.41	67217.56	5477.15
C1_6_3	56	60	4	69679.84	75899.35	6219.51
C1_6_1	60	60	0	68508.46	69310.16	801.7

C2_6_4	17	21	4	62185.0	68475.88	6290.88
C1_6_9	56	58	2	67813.96	73628.19	5814.23
C2_6_2	17	22	5	62446.42	67378.06	4931.64
C1_6_2	56	62	6	69331.56	75795.6	6464.04
C1_6_7	57	61	4	69744.4	71933.41	2189.01
C2_6_9	17	21	4	61967.03	68013.95	6046.92
C2_6_6	18	20	2	62036.68	67211.48	5174.8
C2_6_8	17	19	2	61810.35	64068.77	2258.42
C1_6_5	60	60	0	68316.27	69305.71	989.44
C2_6_1	18	18	0	62198.33	62150.72	-47.61
C1_6_4	56	57	1	69126.88	75661.97	6535.09
C1_6_8	56	63	7	68542.78	73391.35	4848.57
C2_6_10	17	18	1	61564.35	64510.53	2946.18
C1_6_10	56	58	2	67890.08	73343.03	5452.95
C2_6_5	18	18	0	61732.77	64772.78	3040.01
C2_6_7	18	21	3	62468.15	69419.11	6950.96
C1_6_6	59	61	2	69946.75	75546.33	5599.58
RC2_6_4	11	12	1	38305.06	37396.49	-908.57
RC2_6_5	11	18	7	41367.43	38477.17	-2890.26
RC2_6_8	11	17	6	40982.48	33295.63	-7686.85
RC1_6_9	55	57	2	44248.88	32809.64	-11439.24
RC2_6_3	11	17	6	43092.9	42958.69	-134.21
RC1_6_1	55	60	5	45915.06	43752.08	-2162.98
RC1_6_8	55	57	2	44355.38	35539.82	-8815.56
RC2_6_7	11	17	6	40875.78	35530.43	-5345.35
RC1_6_6	55	60	5	46203.2	43994.21	-2208.99
RC1_6_2	55	59	4	46063.41	43986.12	-2077.29
RC2_6_1	14	20	6	53169.69	42100.67	-11069.02
RC1_6_10	55	56	1	42074.31	31946.69	-10127.62
RC2_6_9	11	14	3	40185.54	31319.1	-8866.44
RC2_6_6	11	18	7	42371.52	33189.89	-9181.63
RC1_6_4	55	58	3	35892.54	35075.31	-817.23
RC2_6_2	12	15	3	46684.92	43160.04	-3524.88
RC1_6_3	55	58	3	41653.53	41197.05	-456.48
RC1_6_7	55	60	5	44894.05	41640.06	-3253.99
RC1_6_5	55	60	5	44823.87	47414.94	2591.07

Table 7 shows that 54 out of 59 instances require from one to seven additional tours to meet the hard window constraints [69]. This fact can be explained by the growth of almost twice the average number of paths for 600 customers vs. 400 customers. The number of paths increases by 9.1% on average, varying from 6.3% for clustered problems to 10.9% for randomized-clustered problems. So, the number of paths growing is the same as for 400 customers [70]. Although in 31 out of 59 instances, the total duration is longer for the hard-windows constraints, the average duration decreases by 0.7%. For the randomized problems, the average duration decreases by 2.2%, and for the randomized-clustered problems, it decreases by 12%. However, the average duration increases by 6.4% for clustered problems with the maximum sum of the durations. The results for the 800 customers for the Gehring and Homberger benchmark are provided in Table 8 [71].

Table 8: Comparison with the best results on the 800-customers Gehring and Homberger benchmark. The best solutions for instances RC2_8_2 and RC1_8_9 are infeasible and are not included in the table.

Instance	Number of Vehicles best	Number of Vehicles found	Vehicles difference	The sum of durations best	The sum of durations found	Duration difference
R2_8_3	15	17	2	79888.63	81854.45	1965.82
R2_8_6	15	18	3	82801.05	75950.72	-6850.33

R2_8_5	15	16	1	78211.64	69133.05	-9078.59
R1_8_6	72	81	9	91144.99	95233.93	4088.94
R2_8_9	15	21	6	80973.53	61847.89	-19125.64
R1_8_4	72	78	6	78433.66	86261.93	7828.27
R2_8_4	15	15	0	77512.9	66386.3	-11126.6
R1_8_2	72	82	10	64471.29	74577.73	10106.44
R1_8_8	72	74	2	73708.7	77615.24	3906.54
R1_8_7	72	76	4	87412.16	88341.47	929.31
R1_8_5	72	76	4	92393.98	86622.93	-5771.05
R1_8_9	72	80	8	89154.23	77514.0	-11640.23
R1_8_10	72	74	2	87514.99	64163.49	-23351.5
R2_8_5	15	20	5	82816.71	62583.11	-20223.6
R2_8_2	15	23	8	82562.11	86704.2	4142.09
R2_8_8	15	16	1	73305.77	50102.25	-23203.52
R2_8_1	15	24	9	82350.64	86329.35	3978.71
R1_8_1	80	91	11	103848.6	112930.84	9082.24
R2_8_10	15	16	1	79369.34	53453.39	-25915.95
R1_8_3	72	80	8	90212.2	108122.71	17910.51
C1_8_9	72	80	8	96371.22	106294.4	9923.18
C1_8_2	72	84	12	100110.3	120746.53	20636.23
C1_8_7	77	83	6	98804.3	103245.91	4441.61
C2_8_1	24	24	0	84384.4	84192.57	-818.93
C2_8_10	23	24	1	84067.4	87817.96	3750.56
C1_8_8	73	80	7	98294.13	110581.76	12287.63
C2_8_4	22	27	5	83218.9	97278.57	14059.67
C1_8_3	72	79	7	98471.23	120228.28	21759.05
C2_8_8	23	25	2	83363.45	91260.45	7897.0
C1_8_4	72	75	3	101370.3	110136.43	8766.13
C1_8_5	80	80	0	97812.43	99136.32	1323.89
C2_8_5	24	27	3	84559.57	88817.33	4257.76
C2_8_7	23	29	6	85590.77	96411.73	10820.96
C1_8_6	79	87	8	99114.66	119793.71	20679.05
C2_8_9	23	28	5	83805.93	94630.04	10824.11
C2_8_3	23	28	5	84919.02	102493.41	17574.39
C1_8_6	23	27	4	84293.77	92074.98	7781.21
C1_8_1	80	80	0	97730.41	98959.41	1229.0
C2_8_2	23	31	8	84819.19	99309.66	14490.47
C1_8_10	72	79	7	96263.12	108166.27	11903.15
RC1_8_5	72	77	5	81800.44	82523.87	723.43
RC1_8_10	72	75	3	77365.26	55585.74	-21779.52
RC2_8_9	15	19	4	33332.79	45469.14	12136.35
RC2_8_8	15	21	6	73669.62	57588.65	-16080.97
RC2_8_1	18	29	11	90798.43	70708.25	-20090.18
RC2_8_5	15	27	12	75666.24	60079.97	-15586.27
RC2_8_6	15	25	10	74734.67	53947.32	-20787.35
RC1_8_3	72	75	3	72076.91	80087.78	8010.87
RC1_8_7	72	82	10	80751.65	79618.23	-1133.42
RC2_8_3	15	24	9	71067.1	70286.21	-780.89
RC2_8_10	15	17	2	71931.11	44297.74	-27633.37

RC2_8_7	15	26	11	72575.83	60053.71	-12522.12
RC1_8_8	72	78	6	77047.52	66102.54	-10944.98
RC1_8_4	72	75	3	60126.82	67067.75	6940.93
RC2_8_4	15	18	3	66802.17	58167.45	-8634.72
RC1_8_2	72	82	10	78923.57	78337.52	-586.05
RC1_8_1	72	77	5	81519.86	75494.18	-6025.68
RC1_8_6	72	81	9	81157.38	80014.12	-1143.26

Table 8 shows that 54 out of 58 instances require 1-12 additional tours to meet the hard window constraints due to a 30% increase in the average number of path growth for 800 customers compared to 600 customers [72-75]. The number of paths increases by 10.8% on average, varying from 9% for clustered problems to 13.4% for randomized-clustered problems. Therefore, the number of paths increases slightly compared to the 400-600 customer range [76]. Although in 33 out of 58 instances, the total duration is longer for the hard-windows constraints, the average duration decreases by 0.5%. For the randomized problems, the average duration decreases by 5.9%, and for the randomized-clustered problems, it decreases by 11.5%. However, the average duration increases by 10% for clustered problems with the maximum sum of the durations. The growth in the average relative number of paths and changes in the average relative total duration for various customer numbers are summarized in Tables 9 and 10.

Table 9: The growth in the relative average number of paths for different types of problems and the different numbers of customers

Problems vs. Number of customers	100	200	400	600	800
Average	11.3%	7.6%	8.6%	9.2%	10.8%
R	12.1%	7.3%	9.5%	10.4%	10.2%
C	5.0%	6.0%	6.7%	6.4%	9.0%
RC	14.5%	9.7%	10.6%	10.9%	13.4%

Table 10: The changes in the relative total duration for different types of problems and the various numbers of customers. The last row indicates the number of instances where the algorithm improves the state-of-the-art (SOTA).

Problem vs. Number of customers	100	200	400	600	800
Average	2.14%	-0.14%	1.27%	-0.67%	-0.5%
R	3.84%	-14.6%	-0.41%	-2.2%	-5.9%
C	1.07%	2.48%	4.89%	6.3%	10.05%
RC	1.9%	-6.19%	-6.23%	-12.03%	-11.47%
SOTA improvement instances	7	8	4	2	2

The tables show that the randomized-clustered problems require the maximum addition to tours for the hard time window constraints and the maximum total duration reduction. In contrast, the clustered problems demand minimal new tours but maximally increase the total duration [77].

Conclusion, Discussion, and Future Work

This paper presented the solution to a time-dependent vehicle routing problem with hard time window constraints using a saving ant colony approach. Recently, the problem has garnered increasing attention due to the growing popularity of parcel dispatching in megacities, limited parking capacity, and traffic congestion. The approach developed here is based on a real-life, auto-updated static traffic model using a multi-layer distance-duration matrix without limitation to the FIFO property. Various constraints and multiple objective functions, such as (2) and (5), are implemented using the saving ant colony optimization [78]. The fuzziness of the saving ant colony optimization allows for the addition of other diverse constraints, such as open VRP, package compatibility, and lunch break time for the courier, by minor adaptations of the proposed algorithms. Another advantage of saving ant colony optimization is its closeness to the LSTM network, which allows the use of a known technique to estimate the number of iterations and evaluate convergence to the global minimum.

Results showed that the proposed approach is feasible for real-life applications with the software written in Go for a multi-core computer. The proposed approach significantly reduces the number of vehicles and the total travel duration compared to the naïve dispatching system used in megacities such as Moscow, Russia, and Tel Aviv, Israel. The comparison to the Gehring and Homberger benchmark showed that the hard time windows constraint significantly affects the tours, and the real-life algorithms do not always provide the best results on the existing benchmarks. The proposed method demonstrates its maximum effectiveness for medium-sized tasks with 40-450 customers, which are the most popular delivery tasks in last-mile delivery problems. Figures 12 and 16 show the maximum reduction in the number of vehicles for this size of task. The existing exact

solutions are well-suited for small tasks and are typically faster than the proposed method. For a large number of customers, the technique demands significant computational resources that are not always available in city logistics. Another limitation is the adaptation of makespan to industrial delivery, so the proposed method is not optimal for academic benchmarks. Nevertheless, Table 10 shows SOTA improvement for some instances, typically of 100-400 customers.

Another future research path involves applying the developed approach to the pickup and delivery problem (TDPDPTW), the split delivery problem (TDSDVRPTW), and the VRP with backhauls (TDVRPBTW), among others, including those with time windows and traffic constraints. Moreover, using the time-dependent pheromone matrix should provide further performance improvement. The relatively high overall complexity $O(n^3\ln(n))$ is the most significant limitation for a large number of customers, necessitating the use of multi-core computers for implementation. This limitation can be partially alleviated by testing (56) not for all available customers in the head and tail interior selection but for customers with the positive pheromone trace only.

Finally, although the current paper is devoted to the static approach, in real-life applications, dynamic traffic changes and a user's unpredicted inability to receive packages occur after the vehicle is on the route. In this case, a branch-and-bound algorithm is applied to correct TDTSPTW after every change in the data, starting from the nearest stop point.

References

1. Figliozzi, M.A. (2012). The time-dependent vehicle routing problem with time windows: Benchmark problems, as efficient solution algorithm, and solution characteristics. Transportation Research Part E Logistics and Transportation Review, 48(3), 616–636.
2. Qureshi, A. G., Taniguchi, E., & Yamada, T. (2010). Exact solution for the vehicle routing problem with a semi-soft time window and its application. Procedia Social and Behavioral Science, 2(3), 5931-5943.
3. Yıldırım, U. M., & Çatay, B. (2009). An ant colony algorithm for time-dependent vehicle routing problem with time windows. In Fleischmann, B. et al. (Eds.), Operations Research Proceedings 2008, pp. 337-342. Springer, Berlin, Heidelberg.
4. Gendreau, M., Ghiani, G., & Guerriero, E. (2015). Time-dependent routing problems: A review. Computers & Operations Research 64(2), 189-197.
5. Mancini, S. (2014). Time-dependent travel speed vehicle routing and scheduling on a real road network: the case of Torino. Transportation Research Procedia, 3, 433-441, DOI:10.1016/j.trpro.2014.10.024
6. Kritzinger, S., Doerner, K., F., Hartl, R., F., Kiechle, G., Stadler, H., & Manohar, S., S. (2012). Using traffic information for time-dependent vehicle routing. Procedia - Social and Behavioral Sciences, 39, 217-229, DOI:10.1016/j.sbspro.2012.03.103
7. Lombard, A., Tamayo, S., & Fontane, F. (2018). Modeling the time-dependent VRP through open data. arXiv: 1804.07555, April 2018, DOI:10.48550/arXiv. 1804.07555.
8. Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, 26(1), pp. 29-41.
9. Donati, A., Montemanni, R., Casagrande, N., Rizzoli, A., & Gambardella, L. M. (2008). Time-dependent vehicle routing problem with a multi-ant colony system. European Journal of Operational Research, 185(3), 1174-1191
10. Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12(4), 568-581.
11. Desrochers, M., Lenstra, J. K., Savelsbergh, M. W. P., & Soumis, F. (1988). Vehicle routing with time windows: Optimization and approximation. In Golden, B. L. and Assad, A. A. (Eds.), Vehicle Routing: Methods and Studies 16, pp. 65–84. Elsevier Science Publishers B. V. (North-Holland).
12. Tarantilis, C. D., Ioannou, G., Kiranoudis, C. T., & Prastacos, G. P. (2005). Solving the open vehicle routing problem via a single parameter metaheuristics algorithm. Journal of the Operational Research Society, 56(6), 588-596.
13. Pan, B., Zhang, Z., & Lim, A. (2021). Multi-trip time-dependent vehicle routing problem with time windows. European Journal of Operational Research, 291(1), pp. 218-231.
14. Beasley, J.E. (1981). Adapting the saving algorithm for varying inter-customer travel times. Omega, 9(6), 658-659.
15. Ahn, B. H., & Shin, J. Y. (1991). Vehicle routing with time windows and time-varying congestion. Journal of the Operational Research Society, 42(5), 393-400.
16. Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. European Journal of Operational Research, 144(2), 379-396.
17. Malandraki, C., & Dial, R. B. (1996). A restricted dynamic programming heuristic algorithm for the time-dependent traveling salesman problem. European Journal of Operational Research, 90(1), 45–55.
18. Fleischmann, B., Gietz, M., & Gnutzmann, S. (2004) 'Time-varying travel times in vehicle routing. Transportation Science, 38(2), pp. 160-173.
19. Haghani, A., & Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. Computers and Operations Research, 32(11), 2959–2986.
20. Van Woensel, T., Kerbache, L., Peremans, H., & Vandaele, N. (2008). Vehicle routing with dynamic travel times: a queuing approach. European Journal of Operational Research, 186(3), 990–1007.
21. Potvin, J.-Y., & Rousseau, J.-M. (1995). An exchange heuristic for routing problems with time windows. Journal of Operations Research Society, 46(2), 1433-1446.
22. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., & Potvin, J.Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31(2), 170–186.
23. Maden, W., Eglese, R., & Black, D. (2010). Vehicle routing and scheduling with time-varying data: A case study. Journal of Operations Research Society, 61(3), 515-522.
24. Wen, L., & Eglese, R. (2015). Minimum cost VRP with time-dependent speed data and congestion charge. Computers and Operations Research, 56(C), 41-50, DOI:10.1016/j.cor.2014.10.007.
25. Arigliano, A., Ghiani, G., Grieco, A., Guerriero, E., & Plana, I. (2019). Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. Discrete Applied Mathematics, 261(3), 28-39.

26. Duc Minh, V., Hewitt, M., Boland, N., & Savelsbergh, M. (2019). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. Transportation Science, 54(3), pp. 1-18.
27. Montero, A., Mendez-Diaz, I., & Miranda-Bront, J. J. (2017). An integer programming approach for the time-dependent traveling salesmen problem with time windows. Computers and Operational Research, 88, 280-288, DOI:10.1016/j.cor.2017.06.026
28. Adamo, T., Ghiani, G., Greco, P., & Guerriero, E. (2021). Learned upper bounds for the time-dependent traveling salesman problem. arXiv: 2107.1364v1 [cs.AI] 28 Jul. 2021, DOI:10.48550/arXiv.2107.13641
29. Ehmke, J. F., & Mattfeld, D. C. (2012). Vehicle routing for attend-home delivery in city logistics. Procedia Social and Behavioral Science, 39(12), 622-632.
30. Halpern, J. (1977). Shortest route with time-dependent length of edges and limited delay possibilities in nodes. Zeitschrift für Operations Research, 21, 117–124, DOI: https://doi.org/10.1007/BF01919767
31. Esher, M., Kriegel, H-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In KDD-96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pp. 226-231. AAAI Press.
32. Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling salesman problem. Operations Research, 2, 393–410.
33. Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. European Journal of Operational Research, 231(1), pp. 1-21.
34. Bräysy, O., & Gendreau, M. (2005). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. Transportation Science, 39(1), 104-118.
35. Mladenović, N., & Hansen, P. (1997). Variable Neighbourhood Search. Computers and Operational Research, 24(11), 1097-1100.
36. Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. Computers and Operational Research, 34(8), 2403-2435.
37. Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35, 254–265, DOI: http://dx.doi.org/10.1287/opre.35.2.254.
38. Doerner, K., Gronalt, M., Hartl, R. F., Reimann, M., Strauss, C., & Stummer, M. (2002). Savings ants for the vehicle routing problem. In Cagnoni, S. et al. (Eds.), Applications of Evolutionary Computing. EvoWorkshops 2002. Lecture Notes in Computer Science, Vol. 2279, pp. 11-20. Springer, Berlin, Heidelberg.
39. Bräysy, O., & Gendreau, M. (2005a). Vehicle routing problem with time windows, Part II: Metaheuristics. Transportation Science, 39(1), 119-139.
40. Kilby, P., Prosser, P., & Shaw, P. (1997). Guided local search for the vehicle routing problems. In MIC97: Proceedings of the 2nd International Conference on Metaheuristics, pp. 1-10. Sophia-Antipolis, France, July 21-24.
41. Carić, T., Fosin, J., Galić, A., Gold, H., & Reinholz, A. (2007). Empirical analysis of two different metaheuristics for real-world vehicle routing problems. In Bartz-Beielstein, T. et al. (Eds.), Hybrid Metaheuristics, Lecture Notes in Computer Science (LNCS), Vol. 4771, pp. 31-44. Springer-Verlag, Berlin/Heidelberg, DOI:10.1007/978-3-540-75514-2_3.
42. Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. (2013). A review of dynamic vehicle routing problems. European Journal of Operational Research, 225(1), 1–11.
43. Balseiro, S.R., Loiseau, I., & Ramonet, J. (2008). An ant colony algorithm hybridized with insertion heuristics for the time-dependent vehicle routing problem with time windows. Computers and Operations Research, 38(6), 954–966.
44. Yu, B., Yang, Z-Z., & Xie, J-X. (2011). A parallel improved ant colony optimization for multi-depot vehicle routing problem. Journal of the Operational Research Society 62(1), 183-188.
45. Khoshbakht, M. Y., and Sedighpour, M. (2012). An optimization algorithm for capacitated vehicle routing problem based on ant colony system. Australian Journal of Basic and Applied Science, 5(12), 2729-2737.
46. Rizzoli, A. E., Oliverio, F., Montemanni, R., & Gambardella, L. M. (2004). Ant Colony Optimisation for vehicle routing problems: from theory to applications. Galleria Rassegna Bimestrale Di Cultura, 9(1), pp. 1-50.
47. Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research, 41(4), 421-451, DOI: https://doi.org/10.1007/BF02023004.
48. Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Montavon, G. et al. (Eds.), Neural Networks: Tricks of the Trade (pp. 437-478). Springer, Berlin, Heidelberg.
49. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-80.
50. Hall, P. (1982). On estimating the endpoint of a distribution. Annals of Statistics, 10(2), 556-568.
51. Christiaens, J., & Berghe, G. V. (2020). Slack induction by string removals for vehicle routing problems. Transportation Science, 54(2), 417-433.
52. Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society, 7(1), 48-50.
53. Ribiero, G. M., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers and Operational Research, 39(3), 728-735.
54. Gett Delivery (2022, January, 17). https://www.gett.com/il/delivery/.
55. Gehring, H., & Homberger, J. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In Miettinen, K., Mäkelä, M. and Toivanen J. (Eds.), Proceedings of EUGOGEN99 – Short Course on Evolutionary Algorithms in Engineering and Computer Science, pp. 57-64. University of Jyväskylä.
56. SINTEF (2008, February, 17) Benchmarks-Vehicle routing and traveling salesperson problems, SINTEF Applied Mathematics, Department of Optimization, Norway https://www.sintef.no/projectweb/top/vrptw/.
57. Augerat, P., Belenguer, J.M., Benavent, E., Corber, A., & Naddef, D. (1998). Separating capacity constraints in the CVRP using tabu search. European Journal of Operational Research, 106(2-3), 546-557.

58. Cordeau, J., Desaulniers, G., Desrosiers, J., Solomon, M. M., & Soumis, F. (2001). VRP with time windows. In Toth, P. and Vigo, D. (Eds.), The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications, pp. 157-193. SIAM Publishing, Philadelphia, PA.
59. Derigs, U., & Reuter, K. (2009). A simple and efficient tabu search heuristic for solving the open vehicle routing problem. Journal of the Operational Research Society, 60(12), 1658-1669.
60. Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. Computers and Industrial Engineering, 57(4), 1472–1483.
61. Fleszar, K., Osman, I. H., & Hindi, K. S. (2009). A variable neighborhood search algorithm for open vehicle routing problem. European Journal of Operational Research, 195(3), 803-809.
62. Glover, F., & Laguna, M. (1997). Tabu Search, Kluwer Academic, Boston.
63. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. in ICCV, Proceedings IEEE International Conference on Computer Vision, pp. 1026-1034. Santiago, Chile, DOI: 10.1109/ICCV.2015.123.
64. Hill, A. V., & Benton, W. C. (1992). Modeling intra-city time-dependent travel speed for vehicle scheduling problems. Journal of the Operational Research Society, 43(4), 343-351.
65. Hill, B. M. (1975), A simple general approach to inference about the tail of a distribution. Annals of Statistics, 3(5), 1163-1174.
66. Kok, A. L., Hans, E. W., & Schutten, J. M. J. (2012). Vehicle routing under time-dependent travel times: the impact of congestion avoidance. Computers and Operations Research 39(5), 910–918.
67. Kumar, S. N., & Panneerselvam, R. (2017). Development of an efficient genetic algorithm for the time-dependent vehicle routing problem with time windows. American Journal of Operations Research, 7(1), pp. 1-25.
68. Malandraki, C. (1989). Time-Dependent Vehicle Routing Problems: Formulations, Solution Algorithms and Computational Experiments, PhD Thesis, Northwestern University, Evanston, Illinois.
69. Malandraki, C., & Daskin, M. S. (1992). Time-dependent vehicle-routing problems – formulations, properties, and heuristic algorithms. Transportation Science, 26(3), 185–200.
70. Nemhauser, G. L., Wolsey, L., A. & Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions-I. Mathematical Programming, 14(1), pp. 265-294.
71. Pickands, J. (1975). Statistical inference using extreme order statistics. Annals of Statistics, 3(1), pp. 119-131, DOI: http://dx.doi.org/10.1214/aos/1176343003.
72. Potvin, J.-Y., & Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and the scheduling problem with time windows. European Journal of Operational Research, 66(3), 331–340.
73. Schrimpf, G., Schneider, G., Stamm-Wilbrandt, H., & Duek, J. (2000). Record-breaking optimization results using the ruin and recreate principle. Journal of Computational Physics, 159(2), 139-171.
74. Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In CP'98: Proceedings of the 4th International Conference on Principle and Practice of Constraint Programming, pp. 417-431. London, UK, Springer-Verlag
75. Teng, Y., Chen, J., Zhang, S., Wang, J., & Zhang, Z. (2024). Solving dynamic vehicle routing problem with time windows by ant colony system with bipartite graph matching. Egyptian Informatics Journal, 25, 1-12, DOI:10.1016/j.eij.2023.100421
76. Zhigljavsky A. (1991). Theory of global random search, Kluwer Acadademic Publishers, Dordrecht.
77. Zhigljavsky A., and Žilinskas A. (2008). Stochastic Global Optimization, 1st ed., Springer Berlin, Heidelberg.
78. SINTEF (2008, February, 17) Benchmarks-Vehicle routing and traveling salesperson problems, SINTEF Applied Mathematics, Department of Optimization, Norway https://www.sintef.no/projectweb/top/vrptw/.