CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Tabu Search for the Time-Dependent Vehicle Routing Problem with Time Windows on a Road Network

Maha Gmira
Michel Gendreau
Andrea Lodi
Jean-Yves Potvin

August 2019

CIRRELT-2019-32

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Tabu Search for the Time-Dependent Vehicle Routing Problem with Time Windows on a Road Network

Maha Gmira[1,2,3], Michel Gendreau[1,2], Andrea Lodi[1,2,3], Jean-Yves Potvin[1,4,*]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

[3] Canada Excellence Research Chair in Data Science for Real-Time Decision-Making, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

[4] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

Abstract. Travel times inside cities can vary a lot during a day and this may have a significant impact on the duration of delivery routes. Several authors in the past have suggested time-dependent variants of the most commonly encountered vehicle routing problems. In these papers, however, time-dependent variations are usually defined on a customer-based graph. Consequently, a major consequence of travel time variations in urban areas is over-looked because not only do travel times change, but also the paths used to travel from one customer to another. That is, different paths should be used at different points in time during the day. One must therefore work with a road network and consider travel time (or travel speed) variations on road segments, if realistic delivery routes are to be obtained. In this paper, we propose a solution approach based on tabu search for a time-dependent vehicle routing problem with time windows where the travel speeds are defined on the road network. A major contribution of this work is the development of techniques to evaluate the feasibility as well as the approximate cost of a solution in constant time, thus allowing our algorithm to handle instances with up to 200 nodes and 580 arcs in very reasonable computing times. We show that solutions of high quality are obtained on a set of benchmark instances, when compared with solutions obtained with an exact method.

Keywords: Vehicle routing, time windows, time-dependent travel times, road network, tabu search.

Acknowledgements. Financial support was provided by the Natural Sciences and Engineering Council of Canada (NSERC), while computing facilities were provided by Compute Canada. This support is gratefully acknowledged.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.

Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

[*] Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2019

© Gmira, Gendreau, Lodi, Potvin and CIRRELT, 2019

1 Introduction

For more than 25 years now, it has been reported that neglecting variations in travel times in cities due, for example, to congestion during peak hours in the early morning and late afternoon, can lead to inefficient or even sometimes infeasible delivery routes. Several authors in the past have suggested time-dependent versions of the most commonly encountered vehicle routing problems. In these papers, however, time-dependent variations are usually defined with respect to customer-based graphs where the nodes are customers and an arc between two customers corresponds to a fixed shortest path previously calculated in the underlying road network.

The Time-Dependent Vehicle Routing Problem with Time Windows on a Road Network ($TDVRPTW_{RN}$) is aimed at producing more realistic routes by taking into account the time of the day to compute the shortest path (in time) in a road network to travel from one customer to the next. That is, not only do we observe different travel times during the day to go from one customer to the next, but even the paths used are different. It is clear that many applications may benefit from efficient problem-solving methodologies for the $TDVRPTW_{RN}$, like home delivery services.

Considering different paths to travel from one customer to the next in the road network, as well as accounting for the time-dependent travel time on each one of those paths, make the problem much more complex (which may explain the few works reported in the literature that address this problem). In this work, we propose a metaheuristic approach based on tabu search to efficiently solve benchmark instances of the $TDVRPTW_{RN}$ on road network graphs with up to 200 nodes and 580 arcs. This is achieved first by testing the feasibility of a neighbor solution in constant time and, second, by approximately evaluating the solution, again in constant time. The computational results demonstrate that near-optimal solutions are produced with this approach.

In the following, Section 2 provides a literature review about 1) time-dependent VRPs on customer-based graphs, 2) time-dependent VRPs on customer-based multigraphs and 3) time-dependent VRPs on road network graphs. Section 3 provides a formal description of our problem. Section 4 describes the problem-solving approach for the Time-Dependent Shortest Path Problem ($TDSPP$) on a road network, while Section 5 focuses on the tabu search heuristic. Section 6 describes the techniques used to assess the feasibility and evaluate solutions in constant time. The computational experiments and results are presented in Section 7. Finally, we conclude and state future research directions in Section 8.

2 Literature review

Many papers, as pointed out in [1, 5, 6], consider that travel times between two customers are constant over the day, which is rather unrealistic in practice. This assumption hugely impacts solution quality and may result in sub-optimal or even infeasible solutions, as shown in [8, 16]. In the following, we thus review the various approaches proposed in the literature to solve VRPs with time-dependent travel times.

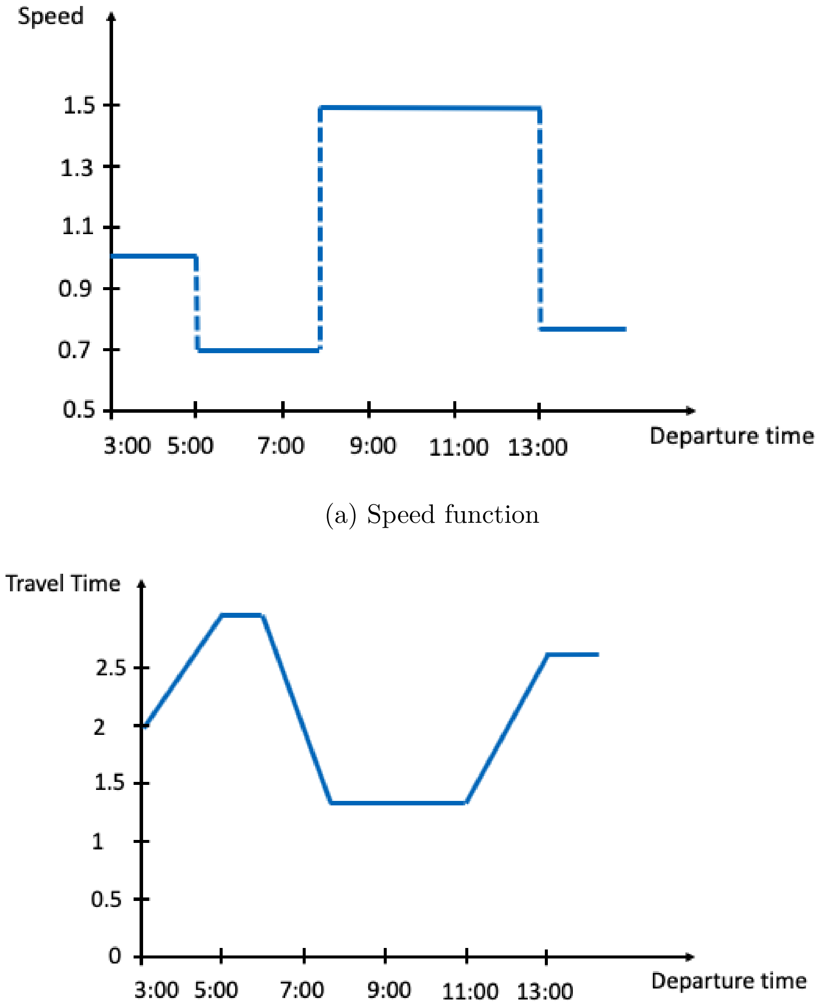
2.1 Time-Dependent VRP on customer-based graphs

In this section, we consider exact and heuristic methods for solving $TDVRPs$, where the shortest path between two customers is precomputed and does not change over time, although the time to travel along that path depends on the departure time. In this case, the problem is defined on a customer-based graph, where an arc between two customers stands for a fixed path in the underlying road network. A good overview of this class of problems can be found in [12].

2.1.1 Heuristics

The work in [21] was the first, in 1992, to deal with a $TDVRP$ without time windows. The time horizon was divided into a few intervals and the travel time on each arc was modeled as a stepwise function with a different travel time associated with each interval. Unfortunately, this model does not satisfy the First-In-First-Out (FIFO) property. That is, a vehicle can depart later than another vehicle and arrive earlier at destination, even if the same path is followed by the two vehicles. This situation occurs, for example, if one vehicle waits just a little before departing to catch a shorter travel time associated with the next interval.

The authors in [16] were the first to consider a $TDVRPTW$, in this case with soft time windows. As opposed to [21], stepwise speed functions are proposed, like the one illustrated in Figure 1(a), where the day is divided into a number of time periods and a speed is associated with each time period. This model, referred to as IGP (due to the initials of the authors), satisfies the FIFO property since a vehicle can only arrive later at destination if it departs later. For a given arc, a stepwise speed function can easily be translated into a corresponding piecewise linear travel time function, as shown in Figure 1(b). In [16], the time horizon is divided into three time periods, one each for the early morning and late afternoon peak hours and one for mid-day. Computational results are reported on instances derived from Solomon's classical VRPs with time windows (VRPTW). The benefits of time-dependency are demonstrated by comparing the reported solutions with those obtained with constant travel times.



(a) Speed function

(b) Travel time function

Figure 1 – Piecewise linear travel time function derived from stepwise speed function of an arc of length 2

In [14, 17], the authors solve a $TDVRP$ with soft time windows using a genetic algorithm. In the first work the problem is dynamic, that is, new customer nodes occur during the day and must be integrated in the solution while the current routes are being executed. The objective to be minimized includes fixed costs for the vehicles, routing costs and user inconvenience costs (lateness). In this work, time dependency is modeled with continuous travel time functions that must satisfy different assumptions to guarantee the FIFO property.

The authors in [8] address the $TDVRP$ with hard time windows using a hierarchy of two artificial ant colonies and a local search method, using the IGP time-dependency model. Their method is applied to instances derived from Solomon's VRPTW instances, as well as a real network in Italy. The time-dependent problem with soft and hard time windows is considered in [9], where a fast iterative route construction and improvement method is proposed, still based on the IGP time-dependency model.

In [25], the authors deal with a $TDVRP$ with soft time windows and stochastic travel times. A tabu search and an adaptive large neighborhood search are proposed to optimize both service efficiency and reliability.

In the context of a traffic information system, the authors in [10] provide a general framework to implement time-dependency in various algorithms for the TDVRPTW. Computational tests based on real traffic data from the city of Berlin show that using constant average travel times underestimates the total travel time by approximately 10%.

2.1.2 Exact method

It is only in 2013 that an exact method to solve a $TDVRP$ with time windows was proposed [7]. In this paper, the IGP time-dependency model is used and the objective is to minimize the total duration of the routes. A branch-and-price algorithm is developed where the master problem is a set-partitioning problem and the pricing problem is a time-dependent shortest path problem with resource constraints, which is solved by a tailored labeling algorithm. The latter is aimed at identifying new feasible routes (columns) of negative reduced cost for the master problem in a typical column generation scheme. When solving the pricing problem, a label is associated with each customer to store the service completion time, the route duration and its reduced cost. Dominance criteria are introduced to discard labels that cannot lead to profitable routes while considering, at the same time, different departure times from the depot. Heuristics are also used to quickly find routes or columns of negative reduced cost. Experiments conducted on randomly generated instances of different sizes show that instances with 25 vertices can be routinely solved to the optimum. Even a few instances with 100 vertices were successfully addressed. As far as we know, no other authors have reported an exact method for tackling $TDVRPs$ with time windows on a customer-based graph.

2.2 Time-Dependent VRP on customer-based multigraphs

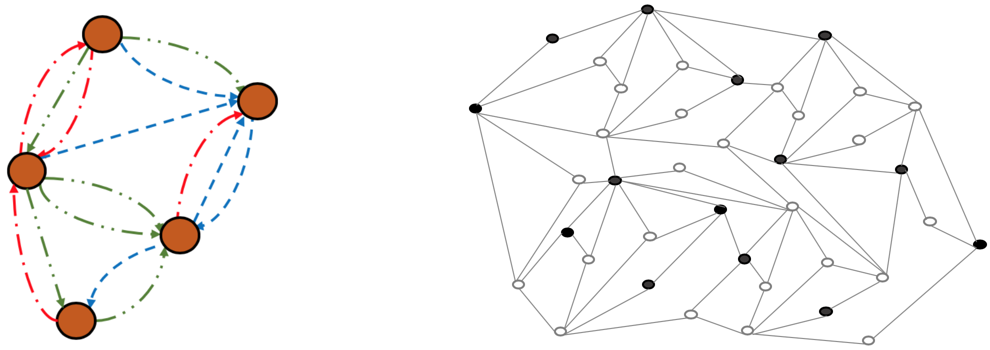
Here, we consider an intermediate representation between $TDVRPs$ defined on customer-based graphs and those defined on road networks. We have a customer-based graph, but with multiple links between two customers that represent different paths in the road network obtained with different departure times. Thus, rather than having a single fixed path between two customers, alternative paths can be used, see Figure 2(a). In this figure, the nodes represent customers and the multiple links between two customers stand for alternative paths in the underlying road network, depending on the departure time.

We are only aware of heuristic methods in this case. The authors in [23] investigate a $TDVRP$ defined on a multigraph where parallel arcs stand for different time-dependent paths obtained with speed functions, thus satisfying the FIFO property. A tabu search is proposed to solve the problem with neighborhoods based on swapping two customers and reversing subsequences of customers. In [18], a tabu search is applied to a multigraph representation to address a $TDVRP$ with heterogeneous fleet. In this work, the insertion of a customer in a route must also integrate the selection of appropriate arcs between consecutive customers, which makes the problem more difficult. This is addressed with dynamic programming, but also with heuristic methods.

A multigraph representation is used in [3] for the $TDVRPTW$. When the sequence of customers in a route is determined, a Fixed Sequence Arc Selection Problem is solved using dynamic programming, to identify the arc (path) between each pair of consecutive customers in the route that must be selected (as it is done in [11] for a dial-a-ride application). In [26], the authors consider the time-dependent alternative vehicle routing problem with time windows. Two arcs are defined between each pair of nodes: the first arc is associated with a time-dependent travel speed distribution, to be used in case of low traffic, and the second one is associated with a constant travel time for heavy congestion hours. Thus, depending on the departure time, one of the two arcs dominates the other.

2.3 Time-Dependent VRP on Road Networks

In this section, the road network is fully exploited when considering time-dependent paths between two customers. That is, no abstraction takes place by first generating a customer-based graph. Exact methods and heuristics for solving the $TDVRP_{RN}$ are reviewed in the following.



(a) Multigraph

(b) Road network graph

Figure 2 – Road Network representations

2.3.1 Heuristics

Using a Mixed-Integer Linear Programming (MILP) solver, the authors in [15] solve a $TDVRP$ with path flexibility under stochastic and deterministic conditions. The objective is to minimize the total expected cost. The authors compute, off-line, a non-exhaustive subset of shortest paths between each pair of customers by selecting different departure times and using a time-dependent Dijkstra's algorithm. Each path is associated with a decision variable in the MILP. This approach solves exactly an approximation of the problem because not all shortest paths are computed. Thus, it can be considered as a heuristic.

In [22], the authors address a time-dependent VRPTW on a real network made of two categories of arcs: main roads with time-dependent travel speeds, and small streets with constant speeds. Time-dependent travel speeds on arcs are created with high degree polynomial functions. A Greedy Randomized Adaptive Search Procedure is used to solve the problem. It enables savings of 12.5% in travel time when compared to solutions obtained with constant travel times.

2.3.2 Exact methods

To the best of our knowledge, there is no exact method for the $TDVRP_{RN}$ (without time windows). However, an exact method for the $TDVRPTW_{RN}$ is reported in [2, 4]. The authors adapt a branch-and-price algorithm previously developed for a time-independent variant. The pricing problem is a Time-Dependent Shortest Path Problem with Resource Constraints, which is addressed with a time-dependent labeling algorithm based on a bi-directional search strategy that limits the number of generated labels. To account for the road network, where an arc can be traversed several times and by more than one vehicle, the branching scheme is defined as follows: 1) select an arc (i, j) with fractional flow ϕ_{ij} and 2) generate two branches by setting either the flow upper limit to $\lfloor \phi_{ij} \rfloor$ or the flow lower limit to $\lceil \phi_{ij} \rceil$. Unfortunately, it can happen that all flows are integer, while the routing solution is fractional. In this situation, an alternative binary branching scheme is applied. In the first branch, it is checked if the flow support graph of the current fractional solution (i.e., all arcs with a positive flow) contains a solution by enumerating all feasible routes and by solving a set covering formulation. In the second branch, at least one arc not in the flow support graph is enforced. The results of the branch-and-price algorithm on the road network graph are compared to those of two customer-based graphs where the single path between two customers is based either on travel distance or travel time. The comparison shows that the solution cost on the road network provides average improvements of 1.7% and 7.3% over the min-distance graph and min-time graph, respectively.

3 Problem Description

In the following, we first define the time-dependent road network graph. Then, we introduce two problems defined on this graph, namely, the Time-Dependent Shortest Path Problem ($TDSPP$) and the $TDVRPTW_{RN}$.

3.1 Time-dependent road network

The road network is a directed graph $G = (V, E)$, where the set of vertices $V = \{0, 1, 2, ..., n\}$ corresponds to road junctions and the set of arcs E to road segments between pairs of junctions. Each arc (i, j) is associated with a distance d_{ij}, a time-dependent speed function $v_{ij} : t \to R^+$ that returns the speed at time t, and a time-dependent cost function $c_{ij} : t \to R^+$ that returns the cost of traversing arc (i, j) at time t (often, c_{ij} corresponds to the travel time). Each speed function is a stepwise function, from which a piecewise travel time function can be derived, see Figure 1.

3.2 $TDSPP$

A path p in the road network G from a source node $s \in V$ to a destination node $s' \in V$ is defined as a sequence of consecutive arcs $(i_1, i_2), (i_2, i_3), ..., (i_{j-1}, i_j)$ with $i_1 = s$ and $i_j = s'$. Alternatively, the path can be viewed as the sequence of nodes $s = i_1, i_2, i_3, ..., i_{j-1}, i_j = s'$.

The $TDSPP$ then consists in identifying a minimum-cost path p from a source node $i_1 = s$ to a destination node $i_j = s'$, given a departure time t_0. The cost of path p at time t_0, $cp_p(t_0)$, is defined recursively as follows:

$$cp_{i_1,i_2}(t_0) = c_{i_1,i_2}(t_0), \tag{1}$$

$$cp_{i_1,\ldots,i_j}(t_0) = cp_{i_1,i_2,\ldots i_{j-1}}(t_0) + c_{i_{j-1},i_j}(t_0 + tp_{i_1,\ldots,i_{j-1}}(t_0)), \tag{2}$$

where

$$tp_{i_1,i_2}(t_0) = tt_{i_1,i_2}(t_0), \tag{3}$$

$$tp_{i_1,\ldots,i_j}(t_0) = tp_{i_1,i_2,\ldots i_{j-1}}(t_0) + tt_{i_{j-1},i_j}(t_0 + tp_{i_1,\ldots,i_{j-1}}(t_0)). \tag{4}$$

Note that $tp_p(t_0)$ is the travel time of path p at departure time t_0 and $tt_{ij}(t) = \frac{d_{ij}}{v_{ij}(t)}$ is the time-dependent travel time along arc (i, j) at time t.

3.3 $TDVRPTW_{RN}$

In the $TDVRPTW_{RN}$, a set of customers $C \subset V$ and a depot (node 0) are located on the time-dependent road network G, as defined above. Each customer $i \in C$ has a demand q_i, a time window for the service start time $tw_i = [a_i, b_i]$ and a service or dwell time s_i. A vehicle cannot arrive at customer i after the upper bound b_i of the time window, but can arrive before the lower bound a_i, in which case the vehicle waits until time a_i to start the service. The set of vehicles K, each of capacity Q, is located at the depot. The time window at the depot $[a_0, b_0]$ defines the beginning and end of the time horizon. The problem is then to generate a set of feasible vehicle routes (solution), one for each vehicle, that start and end at the depot and serve all customers at minimum cost. The latter is obtained by summing the route durations (travel time + waiting time + service time), where the travel time is time-dependent. A solution is feasible if it satisfies the capacity constraints and the time windows.

Note that the travel time of a path in the road network between two customers i and j for any given departure time t, $tp_{i,j}(t)$, is a piecewise linear function derived from the speed functions of all arcs along the path.

4 Time-dependent shortest paths

A time-dependent variant of Dijktra's algorithm is used to identify the minimum cost (travel time) path between a source customer s and a target customer s' in the road network G at a given departure time t_0, see Algorithm 1. In the pseudo-code, i denotes the current vertex; t_j is the arrival time at vertex j; $flag_j$ is *True* if vertex j is permanently labeled, that is, if the best path from customer s to vertex j has been found; and $label_j$ is the label of vertex j, that is, the travel time of the best path from s to vertex j. At the beginning, $label_s = 0$ and $flag_s = True$, while $label_j = \infty$ and $flag_j = False$ for $j \neq s$. At each iteration and starting from the current node, which is customer s at the beginning, the arrival time at each non-permanently labeled successor is calculated with the IGP procedure [16]. If the new path is better than the best known one, then the label is updated. In the special case where the successor is customer s', the new path is considered only if $t_{s'}$ does not exceed the time window's upper bound of s'. Once all successors of the current vertex have been considered, the vertex in the road network with the minimum label among all those that are not permanently labeled becomes the current one and its label is made permanent. This is repeated until customer s' is permanently labeled.

Algorithm 1 Time-dependent Dijkstra's algorithm

```
Input: road network $G = (V, A)$, source $s$, target $s'$, departure time $t_0$
Output: time-dependent path of minimum travel time from $s$ to $s'$
$label_s = 0$
$flag_s = True$
for $j \in V$, $j \neq s$ do
    $label_j \leftarrow \infty; flag_j \leftarrow False$
$i \leftarrow s$
$t_i \leftarrow t_0$
while $flag_{s'} \neq True$ do
    for every successor $j$ of $i$ with $label_j = False$ do
        $t_j \leftarrow Algorithm\ 2(G, (i, j), t_i)$
        if $j = s'$ then
            if $t_j \leq b_j$ then
                $label_j \leftarrow min\{label_j, t_j\}$
        else
            $label_j \leftarrow min\{label_j, t_j\}$
    $i \leftarrow argmin\{label_j \mid j \in V$ and $flag_j = False\}$
    $flag_i \leftarrow True$
```

As previously mentioned, the path from s to the current node is extended to all non-permanently labeled successors at each iteration of the time-dependent Dijkstra's algorithm. The extension from the current node to a given successor through a particular exit arc is done with the IGP procedure, using the speed function associated with that arc, to produce the arrival time at the successor, see Algorithm 2. In the pseudo-code, *lb(period)* and *ub(period)* denote the lower and upper bounds of a time period and we assume that the associated speed is defined over the interval

$[lb(period), up(period)[$ (i.e., the speed changes at the upper bound). After identifying the time period, and thus the speed, associated with the departure time t on arc (i, j), the procedures moves from one period to the next until the distance traveled d^+ exceeds d_{ij}. At that point, the arrival time at j is obtained by adding to the lower bound of the current period, which is stored in t, the time needed to travel the remaining distance to node j.

Algorithm 2 Arrival time at a successor of a node

```
Input: road network $G = (V, A)$, arc $(i, j)$, departure time $t$
Output: arrival time at $j$
$period \leftarrow 0$
while $(t \geq ub(period))$ do $period \leftarrow period + 1$        ▷ find time period of $t$
$d^- \leftarrow 0$
$d^+ \leftarrow (ub(period) - t) \times v_{ij}(t)$
while $d^+ \leq d_{ij}$ do
    $t \leftarrow ub(period)$
    $d^- \leftarrow d^+$
    $period \leftarrow period + 1$
    $d^+ \leftarrow d^- + (ub(period) - t) \times v_{ij}(t)$
$t \leftarrow t + (d_{ij} - d^-)/v_{ij}(t)$
```

5 Problem-solving method

Our solution approach is based on the tabu search metaheuristic. First introduced by Glover [13], tabu search is a neighborhood-based metaheuristic widely used for solving vehicle routing problems [19]. First, an initial solution is generated with a simple heuristic and becomes the current solution. Then, at each iteration, the best solution in the neighborhood of the current solution is selected (even if worse than the current solution) and becomes the new current solution. This is repeated until a stopping criterion is satisfied, at which point the best solution visited during the search is returned. To avoid cycling, it is forbidden or tabu to perform certain moves that could lead back to a previously visited solution. In our case, the initial solution is produced with a greedy insertion heuristic and the neighborhood of the current solution is generated using CROSS exchanges [24], see Section 5.2. Since evaluating this neighborhood is computationally expensive, different techniques are used to assess the feasibility and approximate the value of neighborhood solutions in constant time. The main components of our tabu search will now be described in the following.

5.1 Initial solution

The initial solution is generated with a greedy insertion heuristic, where the routes are constructed sequentially. At each iteration, a customer is randomly selected and inserted at the best feasible insertion place in the current route. When a route cannot admit any new customer due to capacity or time window constraints, a new route is constructed. This is repeated until all customers are visited. The greedy heuristic exploits the techniques developed for assessing the feasibility of an insertion in constant time (see Section 6), although each insertion is evaluated exactly by propagating its impact along the route.

5.2 Neighborhood structure

The tabu search exploits a neighborhood structure based on CROSS exchanges [24]. This type of exchange is well suited for problems with time windows because it does not reverse segments of routes. The CROSS exchange consists in swapping sequences of customers of arbitrary length, up to a maximum length L, between two routes, where the number of customers in the two sequences does not need to be the same. Parameter L was set to 8, based on the work in [24]. A CROSS exchange is illustrated in Figure 3, where a sequence of n_1 customers in the first route is exchanged with a sequence of n_2 customers in the second route, by removing arcs (i_l,i_{l+1}), (j_h,j_{h+1}), (i_{l+n_1}, i_{l+n_1+1}), (j_{h+n_2}, j_{h+n_2+1}) and by adding arcs (i_l,j_{h+1}), (j_h,i_{l+1}), (i_{l+n_1}, j_{h+n_2+1}), (j_{h+n_2}, i_{l+n_1+1}). In the figure, the squares are copies of the depot that represent the start and end of each route.

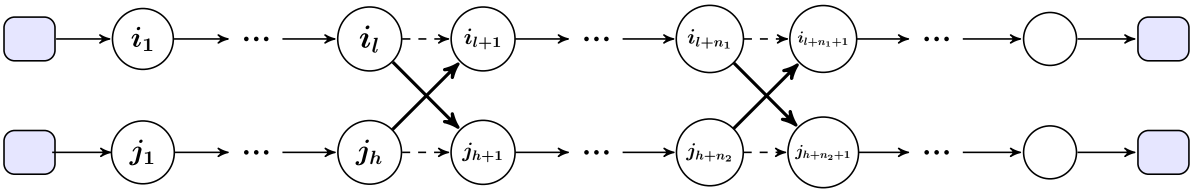


Figure 3 – CROSS exchange

The neighborhood is explored in a systematic way by considering all possible exchanges of sequences of length $n_1 = 1, 2, ..., L$ and $n_2 = 1, 2, ..., L$ for every pair of routes in the current solution. Due to the size of this neighborhood, the feasibility of a neighborhood solution as well as its approximate cost are evaluated in constant time, as discussed in Section 6.

5.3 Tabu list

When a move is performed to get from the current solution to the next one, the inverse move is declared tabu for a number of iterations, which is called the tabu tenure (tab). That is, if an exchange is performed at a given iteration ($iter$), then the exchange that leads back to the previous solution is tabu until iteration $iter + tab$.

5.4 Aspiration criterion

The classical aspiration criterion is used, where the tabu status of an exchange is overridden if it leads to a neighborhood solution that is better than the best known solution.

5.5 Diversification strategy

Diversification is used in tabu search to favor the exploration of new regions in the solution space when search stagnation is observed. The latter is detected when a number of consecutive iterations is performed without improving the best known solution. Diversification is realized by using an alternative objective, namely, minimization of the total distance, for a certain number of iterations. In such a case, the shortest paths in the road network between two customers are based on distance while the time-dependent travel times are only taken into account to guarantee feasibility (i.e., any path from customer i to customer j should reach j at or before its time window's upper bound b_j). Obviously, the solution value is also based on the total distance. After a certain number of iterations with the distance objective, the original objective (duration) is restored until stagnation occurs again. The parameter values for the number of iterations during diversification and number of consecutive iterations for detecting stagnation are discussed in the computational results, see Section 7.

6 Constant time evaluation framework

This section describes how the feasibility and approximate cost of a solution in the neighborhood of the current solution can be assessed in constant time. With regard to feasibility, we focus on time constraints, given that the vehicle load and, consequently, capacity constraints do not pose any difficulty.

It is important to note that additional information must be stored in the current solution to achieve this result. The corresponding information needs to be updated (not in constant time) when the tabu search moves from the current solution to the next one. However, this is done much less frequently (once per iteration) than evaluating all solutions in the neighborhood of the current solution.

For the sake of simplicity, we assume in this section that service or dwell time $s_i = 0$ for every customer i.

6.1 Feasibility

The goal here is to avoid propagating the impact of a move along each modified route when a CROSS exchange is applied to the current solution, due to the size of this neighborhood. The additional information that needs to be maintained in the current solution for this purpose is derived from the so-called dominant shortest-path structure, as it is explained below.

6.1.1 Dominant shortest-path structure

Before running the tabu search, we generate different shortest paths for each pair i, j of customers by applying the time-dependent Dijkstra's algorithm with different departure times from i. A piecewise linear function is then calculated and associated with each path from i to j by combining the speed functions of all arcs that constitute the path. These piecewise linear functions return the arrival time at j for any departure time from i between a_i and b_i, see Figure 4. If we gather the functions of all paths between customers i and j, it is then possible to create a so-called dominant shortest path structure for the pair i, j by considering the best path at any moment in time. For example, $path2$ in Figure 4 is the best path between $t = a_i$ and $t = t_1$. Similarly, $path3$ is the best path between $t = t_1$ and $t = t_2$, etc. At the end, we obtain the structure shown in Figure 5, where the black dots at time $t = t_1$, t_2, t_3 correspond to crosspoints where the best path changes. This structure returns the arrival time at j for any given departure time from i, using the best path at the given departure time.

This dominant structure is used to evaluate the feasibility of a CROSS exchange, as it is explained below.

6.1.2 Bounds on departure times

Given the sequence of customers in a route of the current solution, the dominant shortest path structure associated with each pair of consecutive customers in this route can be used to determine the latest departure time at each customer to maintain feasibility from that customer up to the end of the route. Starting from the end depot and moving backward, the procedure is the following.

Let $i_0, i_1, i_2, ...i_{nr}, i_{nr+1}$ be a route r in the current solution with nr customers, where i_0 and i_{nr+1} are copies of the depot at the start and end of r. We also denote d_i the departure time from customer i in route r in the current solution. To compute

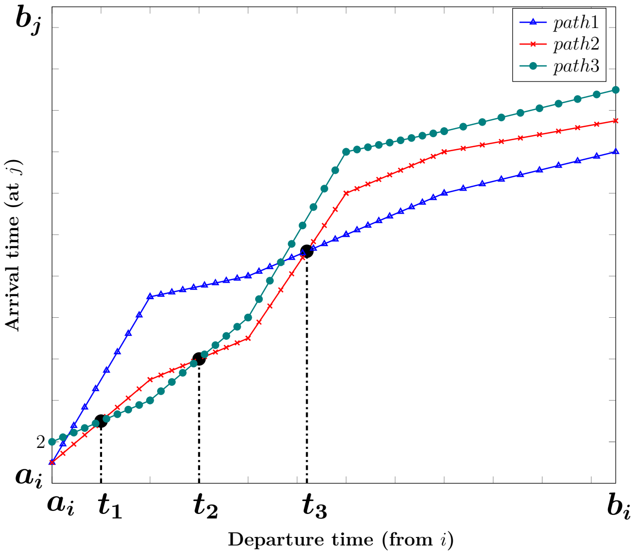


Figure 4 – Piecewise linear arrival time functions for different paths between customers i and j

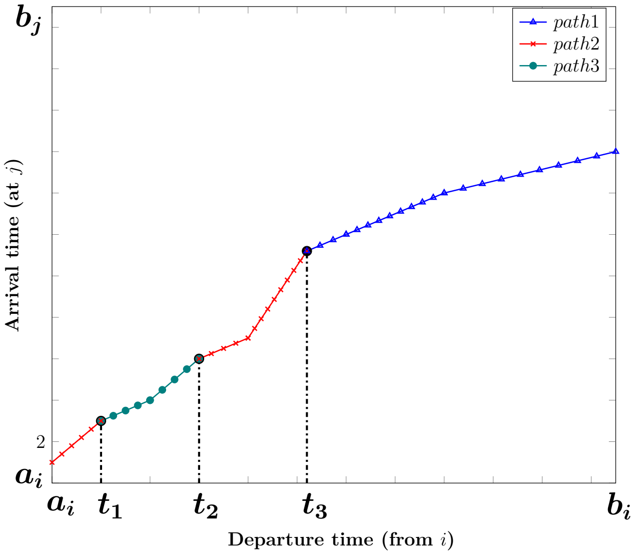


Figure 5 – Dominant shortest path structure between customers i and j

the latest departure time ld_i from each customer i in route r we start with the end depot i_{nr+1} and end of time horizon $ld_{i_{nr+1}}$. Based on the path p used in the current solution to go from customer i_{nr} to the end depot i_{nr+1}, we move backward along that path to infer $ld_{i_{nr}}$ from $ld_{i_{nr+1}}$. It should be noted that $ld_{i_{nr}}$ is reset to $b_{i_{nr}}$ when $ld_{i_{nr}} > b_{i_{nr}}$. Now, two cases can occur:

(a) If the dominant shortest path structure indicates that the best path from i_{nr} to i_{nr+1} at time $ld_{i_{nr}}$ is the same as the one used in the current solution at time d_i, then we continue moving backward, now with customer i_{nr} and time $ld_{i_{nr}}$.

(b) Otherwise, there is a better path than path p to go from i_{n_r} to i_{nr+1} at time $ld_{i_{nr}}$. In this case, we increase $ld_{i_{n_r}}$ as much as possible (i.e., without exceeding $b_{i_{nr}}$) to get as close as possible or match $ld_{i_{nr+1}}$ using the dominant shortest path structure of arc (i_{nr}, i_{nr+1}). In other words, the dominant shortest path structure allows us to identify the appropriate path and departure time at i_{nr} to reach the next node, here the end depot. Afterward, we continue moving backward with customer i_{nr} and the updated time $ld_{i_{nr}}$.

The above procedure is repeated until i_0 is reached, see Algorithm 3. From a practical point of view, the latest departure time is determined in case (b) by moving from one crosspoint to another along the dominant shortest path structure associated with customers i and j, until either b_i is reached, in which case b_i corresponds to the latest departure time at i, or the arrival time at j exceeds its latest departure time ld_j. In the second situation, ld_i is obtained by interpolating between the last and next-to-last crosspoints. This is illustrated in Figure 6, where ld_i^0 is the latest departure time at customer i obtained from ld_j by going backward through the path used in the current solution (it is assumed that this path is not $path3$, which is the best at departure time ld_i^0). We then successively move to ld_i^1 and ld_i^2 (where ld_j is exceeded) along the dominant shortest path structure. The latest departure time at customer i is finally identified by interpolating between ld_i^1 and ld_i^2.

Algorithm 3 Latest departure times

```
Input: road network $G = (V, A)$; route $i_0, i_1, ...i_{nr}, i_{nr+1}$, end of time horizon
    $ld_{i_{nr+1}}$ at end depot $i_{nr+1}$
Output: bounds on departure times at each node
for $k = nr, nr-1, ..., 0$ do
    calculate $ld_{i_k}$ from $ld_{i_{k+1}}$ by moving backward along the path $p$ used to go
        from $i_k$ to $i_{k+1}$ in current solution
$ld_{i_k} = min\{ld_{i_k}, b_{i_k}\}$
if best path from $i_k$ to $i_{k+1}$ at time $ld_{i_k}$ is not $p$ then
    find best path $p' \neq p$ and departure time $ld_{i_k}$ to reach $i_{k+1}$, using dominant
        shortest path structure
```

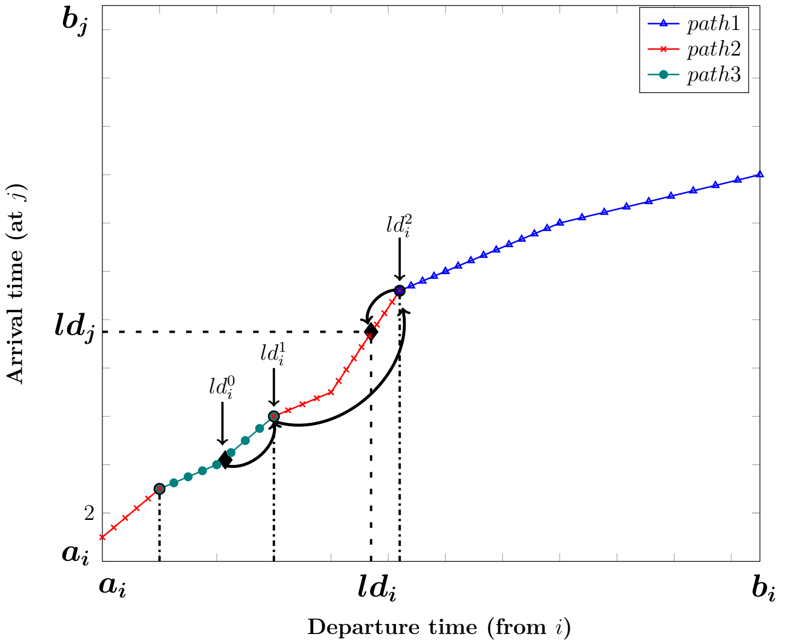


Figure 6 – Latest departure time update

6.1.3 CROSS exchange feasibility

Considering again Figure 3, where a neighborhood solution is obtained by exchanging two subsequences of customers between two routes in the current solution, the feasibility of this solution can be assessed in constant time. This is done as follows. First, the impact of replacing arc (i_l, i_{l+1}) by (i_l, j_{h+1}) in the first route is propagated from i_l to the customers in the sequence j_{h+1}, j_{h+2}, ..., j_{h+n_2} (which contains at most L customers) and then to i_{l+n_1+1}, using the dominant structure associated with each pair of consecutive customers. The arrival time at i_{l+n_1+1} is then checked against its latest departure time to determine if the new route is feasible. Thus, there is no need to propagate until the end of the route. The same procedure is repeated for the second route involved in the exchange.

6.2 Approximate cost

To evaluate a neighborhood solution in constant time, we have no choice but to use an approximation. Essentially, for each customer i in a route of the current solution, we store a penalty p_i that stands for the delay incurred in the service start time of the next customer if the departure time at i is delayed by one time unit. Once the impact of a CROSS exchange, like the one in Figure 3, is propagated up to i_{l+n_1+1} (resp. j_{h+n_2+1}), the additional cost of the two new routes in the neighborhood solution is approximated by multiplying the delay $\Delta_{i_{l+n_1+1}}$ (resp. $\Delta_{j_{h+n_2+1}}$) by the penalty $p_{i_{l+n_1+1}}$ (resp. $p_{j_{h+n_2+1}}$). That is, the additional cost associated with this neighborhood solution is $\Delta_{i_{l+n_1+1}} \times p_{i_{l+n_1+1}} + \Delta_{j_{h+n_2+1}} \times p_{j_{h+n_2+1}}$. Note that Δ_i can well be positive or negative. For example, if a longer sequence is exchanged with a smaller one, then the delay can be positive on one route and negative on the other.

To alleviate the effect of the approximation, we keep the n_{approx} best solutions in the neighborhood, based on this approximate computation. Then, each one of these n_{approx} solutions is evaluated exactly, through propagation to the end of each route modified by the CROSS exchange, and the best solution obtained is selected as the new current solution. Through preliminary experiments, n_{approx} was set to 15 (values 5, 10, 15 and 20 were tested).

7 Computational experiments

In the following, we first describe the test instances used to perform our computational study. Then, a comparison is provided between our algorithm and the exact branch-and-price method BP reported in [4]. All experiments were performed on a Dell PowerEdge R630 server with two Intel Xeon processors E5-2637V4 with 4 cores and 128GB of memory each.

7.1 Test instances

The test instances, called $NEWLET$, come from a recent work in [4]. These instances were generated by the authors using a procedure for creating sparse graphs previously reported in [20]. Three graphs are available with n = 50, 100 and 200 nodes. Three different sets of random static travel times, called basic travel times, are associated with the set of arcs in each graph, based on different levels of correlation, namely, non-correlated (NC), weakly correlated (WC) and strongly correlated (SC). This leads to $3 \times 3 = 9$ different networks. After dividing the time horizon into five periods, time dependency is accounted for by associating a speed profile with each arc in each network. This is done by multiplying the basic speed, which is derived from the basic travel time, by a multiplier in each period. It is possible to choose from three different sets of multipliers to define either congestion-free, normal or congested arcs. Then, 21 different categories of test instances are obtained by randomly selecting 16 or 33 customer nodes from the three 50-node networks ($3 \times 2 = 6$); 25, 33 or 50 customer nodes from the three 100-node networks ($3 \times 3 = 9$); and 25 or 50 customers from the three 200-node networks ($3 \times 2 = 6$). Finally, each category is duplicated by considering either narrow or wide time windows, for a total of 42 categories of instances. Since 5 different instances are available in each category, we have a total of

210 instances. The capacity of each vehicle is set to 200. The demands were produced as follows in [4]: first, routes were created with a greedy heuristic while considering only the time windows; then, the demand at each customer was generated to ensure that each one of those routes remained feasible. Finally, the service time at each customer is randomly selected in $\{1, 2\}$.

7.2 Comparison with optimal solutions

In the following, we first describe the procedure for tuning the main parameters of our algorithm. Then, we compare our solutions with the optimal ones reported in [4]. Finally, we analyze the impact of diversification.

It is important to note that the objective to be minimized in [4] is the total distance. Thus, at the road network level, minimum-distance paths are considered between two customers (or between the depot and a customer), while the time-dependent travel times are only used to guarantee path feasibility. In other words, minimum-distance feasible paths in the road network are looked for. At the route level, the set of vehicle routes must minimize the total distance. Consequently, we modified our algorithm by setting the total distance as the main objective and duration as the objective in the diversification phase.

7.2.1 Parameter tuning

Our proposed tabu search, called TS, has four main parameters, two of which are related to diversification: tabu tenure tab, maximum number of iterations it_{max}, number of consecutive iterations without improvement it_{cons} to initiate the diversification phase and number of iterations in the diversification phase it_{div}.

The following values were considered, where n_c denotes the number of customers:

- $tab = n_c/6,\ n_c/3,\ n_c/2,\ n_c$,
- $it_{max} = 5n_c,\ 10n_c,\ 15n_c$,
- $it_{cons} = n_c/2,\ n_c,\ 2n_c$,
- $it_{div} = n_c/10,\ n_c/5,\ n_c/2,\ n_c$.

Thus, we have a total of $4 \times 3 \times 3 \times 4 = 144$ possible combinations of parameter settings. An exhaustive evaluation of all those combinations was performed on a subset of 42 test instances. Precisely, one instance was randomly selected among the five instances available in each category. The parameter setting leading to the best average objective value on this subset was: $tab = n_c/3$, $it_{max} = 10n_c$, $it_{cons} = n_c$ and $it_{div} = n_c/5$.

In Table 1, we show how the average objective values evolve for each parameter when fixing the three other parameters at their best value. Each entry corresponds to the average gap in percentage between the distance $Dist(TS)$ produced by our Tabu Search and the optimum distance $Dist(Opt)$, that is:

$$Gap_O = \frac{Dist(TS) - Dist(Opt)}{Dist(Opt)} \times 100. \tag{5}$$

Considering the results for parameter it_{max}, it is clear that a sufficient number of iterations is required for the Tabu Search to converge. After $5n_c$ iterations, the best known solution is still far from the optimum. On the other hand, a total of $15n_c$ iterations does not bring any significant improvement over $10n_c$ iterations. For parameter it_{cons}, the value n_c is a good compromise between a larger number of diversification phases with shorter runs in-between ($n_c/2$) and a smaller number of diversification phases with longer runs in-between ($2n_c$). Otherwise, the value $n_c/3$ for parameter tab is clearly the best one, as is the value $n_c/5$ for parameter it_{div}.

Table 1 – Sensitivity analysis results

Parameter	$5n_c$	$10n_c$	$15n_c$
it_{max}	7.9240	0.9285	0.9282

	$n_c/6$	$n_c/3$	$n_c/2$	n_c
tab	2.449	0.929	2.139	2.331

	$n_c/2$	n_c	$2n_c$
it_{cons}	1.258	0.929	1.154

	$n_c/10$	$n_c/5$	$n_c/2$	n_c
it_{div}	2.940	0.929	1.109	1.589

7.2.2 Results on $NEWLET$ instances

Tables 2 and 3 report the results of our tabu search TS on the $NEWLET$ test instances for narrow and wide time windows, respectively. In these tables, each result was obtained on a single instance of each category, as defined by the graph size (*#nodes*, *#arcs*), number of customers (*#cust.*) and correlation type (*Corr.*). Thus, the results here are based on 42 instances only, not 5 × 42 = 210 instances, because we had to choose in each category the single instance used by the authors in [4] to report their results. The optimum was obtained on all these instances within a computation time limit of 7,200 seconds in [4], except four instances with wide time windows. Tables 2 and 3 show the CPU time in seconds of TS (TS $Time$), the CPU time in seconds of the exact branch-and-price algorithm in [4] (BP $Time$), the improvement ($Impr.$) in percentage of the final distance produced by TS ($Dist(TS)$) over the initial distance produced by the greedy insertion heuristic ($Dist(Init)$), as defined by

$$Impr = \frac{Dist(Init) - Dist(TS)}{Dist(Init)} \times 100, \tag{6}$$

and the gap in percentage between the final distance obtained by TS and the optimal distance (Gap_O), which was previously defined in equation (5). Due to the slower machine used in [4], the CPU times of TS reported in the tables should be multiplied by a scaling factor of approximately 1.6.

Table 2 – Results on *NEWLET instances* - Narrow time windows

Instance	# nodes	# arcs	# cust.	Corr.	*TS* Time(s)	*BP* Time(s)	Impr. (%)	Gap$_O$ (%)
NEWLET	50	134	16	NC	0.640	95.6	3.020	0.761
				WC	0.967	1.7	1.712	0.751
				SC	0.296	1.0	1.846	0.781
			33	NC	2.813	3.5	5.610	0.822
				WC	2.945	14.4	3.860	0.816
				SC	2.694	8.9	2.823	0.850
	100	286	25	NC	2.719	1.1	6.713	0.874
				WC	1.935	1.4	2.528	0.922
				SC	1.429	1.5	5.727	0.930
			33	NC	2.488	1.0	1.704	0.968
				WC	2.982	2.0	1.492	0.964
				SC	3.264	49.6	2.019	0.942
			50	NC	5.251	4.0	2.898	0.992
				WC	5.464	4.0	1.826	0.982
				SC	7.151	201.3	1.961	0.962
	200	580	25	NC	2.393	4.0	3.071	0.948
				WC	2.731	4.0	2.986	0.883
				SC	2.406	3.9	1.833	0.991
			50	NC	17.931	13.0	1.547	0.929
				WC	27.182	7089.2	1.848	0.967
				SC	18.274	18.7	6.918	0.908

Table 3 – Results on *NEWLET instances* - Wide time windows

Instance	# nodes	# arcs	# cust.	Corr.	*TS* Time(s)	*BP* Time(s)	Impr. (%)	Gap$_O$ (%)
NEWLET	50	134	16	NC	1.561	15.0	6.772	0.751
				WC	1.526	2.8	7.556	0.716
				SC	1.463	1.7	7.083	0.743
			33	NC	4.769	7125.2	6.952	0.652
				WC	4.871	7200	6.693	–
				SC	2.163	7200	6.606	–
	100	286	25	NC	4.737	28.2	3.712	0.693
				WC	2.130	2.4	3.792	0.857
				SC	3.916	26.2	5.820	0.862
			33	NC	7.491	2229.9	6.681	0.882
				WC	5.766	5.1	4.570	0.901
				SC	5.865	24.4	4.548	0.909
			50	NC	13.051	6.1	3.774	0.945
				WC	17.366	56.7	2.769	0.845
				SC	11.774	1532.5	5.702	0.905
	200	580	25	NC	9.079	7199.2	4.693	0.822
				WC	8.481	96.6	6.914	0.966
				SC	9.316	27.4	5.628	0.901
			50	NC	63.811	7200	4.338	–
				WC	53.245	4364.3	2.511	0.974
				SC	43.602	7200	3.853	–

Concerning the results in Tables 2 and 3, we can see that *TS* significantly improves the solutions of the greedy insertion heuristic. The average improvement is 3.04% and 5.28% on the instances with narrow and wide time windows, respectively. More importantly, the gap between the final solutions produced by *TS* and the optimum is systematically under 1% on all instances, even if our algorithm was not intended to minimize the distance. The average gap on the instances with wide time windows is 0.843%, which is slightly better than the average gap of 0.902% on the instances with narrow time windows. It should also be noted that *TS* runs faster than *BP* on 11 out of 21 instances with narrow time windows and 17 out of 21 instances with wide time windows (after multiplying *TS Time* by the scaling factor of 1.6). The average computing times in seconds of *TS* and *BP* are 5.43 and 358.27 over the instances with narrow time windows, and 13.14 and 2454.46 over the instances with wide time windows, respectively. Thus, TS is much faster on average, even after multiplying its computing times by 1.6. This is not a surprise, because exact methods are very sensitive to the instance size. Also, they can sometimes be erratic and exhibit large computing times even on instances of relatively small size.

7.2.3 Impact of diversification

Here, we examine the impact of the diversification phase on solution quality. Tables 4 and 5 show the average gap with optimal solutions (Gap_O) for TS with and without diversification on the $NEWLET$ instances. These results show that diversification helps to significantly cut the gap with respect to the optimum, sometimes by more or close to one half (see the results with 100 nodes, 286 arcs, and 25 customers in Table 4 or those with 50 nodes, 134 arcs and 33 customers in Table 5).

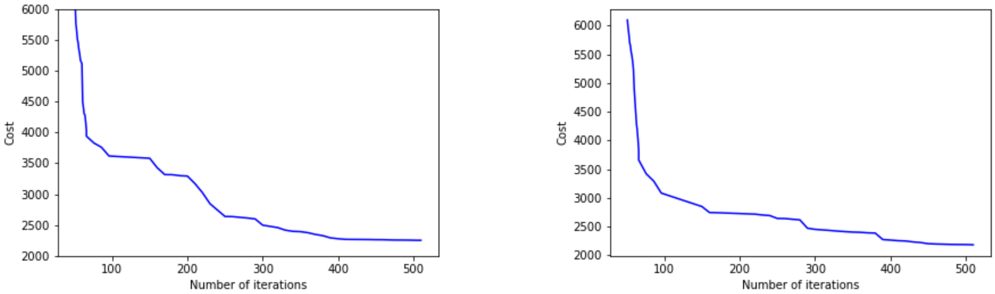
Table 4 – TS optimal gaps with and without diversification - Narrow time windows

# nodes	# arcs	# cust.	Without diversification	With diversification
50	134	16	1.2560	0.7643
		33	1.3223	0.8293
100	286	25	2.0117	0.9088
		33	1.3730	0.9578
		50	1.1363	0.9784
200	580	25	1.2749	0.9406
		50	1.2009	0.9345

Table 5 – TS optimal gaps with and without diversification - Wide time windows

# nodes	# arcs	# cust.	Without diversification	With diversification
50	134	16	1.0202	0.7366
		33	1.1461	0.6520
100	286	25	1.1030	0.8040
		33	1.2300	0.8972
		50	1.4346	0.8983
200	580	25	1.1398	0.8963
		50	1.3710	0.9740

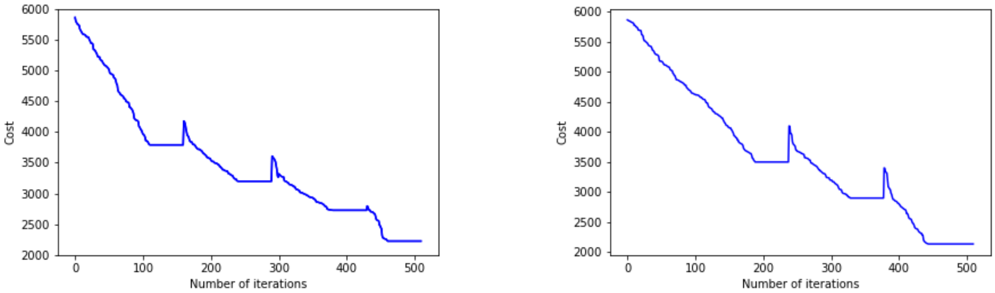
Figures 7 and 8 illustrate the evolution of the objective value (distance) on two instances with narrow and wide time windows, on a graph with 100 nodes, 286 arcs and 50 customers, while running TS without diversification and with diversification, respectively. Due to the range of the objective values, it is not possible to clearly see the final solution values and to claim that diversification is beneficial. However, this is indeed the case: on the instance with narrow time windows a total distance of 2246.8 is obtained without diversification, as compared to 2242.7 with diversification. Similarly, on the instance with wide time windows, a total distance of 2165.1 is obtained without diversification, as compared to 2154.6 with diversification.



(a) Narrow time windows

(b) Wide time windows

Figure 7 – Evolution of objective value of current solution without diversification



(a) Narrow time windows

(b) Wide time windows

Figure 8 – Evolution of objective value of current solution with diversification

7.3 Minimum duration objective

Our algorithm was primarily designed to minimize the total duration of the routes, which seems appropriate in the case of time-dependent travel times. Even if there is no alternative algorithm to compare with, we ran our Tabu Search on the 210 *NEWLET* instances with this new objective (while minimizing the distance in the diversification phase). The results are reported in Tables 6 and 7, using the format of Tables 2 and 3 in Section 7.2.2.

Table 6 – Results for *NEWLET instances* - NTW

Instance	# nodes	# arcs	# cust.	Corr.	*TS* Time(s)	Impr. (%)	Gap_T (%)	Gap_D (%)
NEWLET	50	134	16	NC	0.753	4.993	0.901	1.797
				WC	0.719	6.697	1.182	2.998
				SC	0.442	5.073	1.412	1.262
			33	NC	1.268	5.852	0.686	1.298
				WC	2.217	3.830	1.156	1.321
				SC	2.222	3.907	1.837	1.200
	100	286	25	NC	1.471	5.834	1.375	1.540
				WC	1.480	1.941	1.856	1.737
				SC	1.593	1.868	3.469	1.472
			33	NC	0.493	4.894	1.711	1.251
				WC	0.560	6.122	1.982	1.862
				SC	2.385	3.902	1.005	1.854
			50	NC	1.464	3.730	1.335	1.598
				WC	1.065	6.994	1.600	1.115
				SC	7.956	0.011	1.374	1.071
	200	580	25	NC	3.109	6.886	1.746	1.261
				WC	2.572	7.087	1.847	1.401
				SC	2.238	6.875	1.693	1.453
			50	NC	7.830	7.702	1.374	1.480
				WC	7.413	4.609	1.544	1.323
				SC	9.327	4.783	1.866	1.284

Table 7 – Results for *NEWLET instances* - WTW

Instance	# nodes	# arcs	# cust.	Corr.	TS Time(s)	Impr. (%)	Gap$_\text{T}$ (%)	Gap$_\text{D}$ (%)
NEWLET	50	134	16	NC	0.778	7.985	0.843	2.994
				WC	0.730	7.711	1.334	4.370
				SC	0.734	2.619	1.118	3.446
			33	NC	2.189	6.869	1.468	1.233
				WC	2.271	5.838	1.855	2.450
				SC	2.227	4.874	1.764	1.086
	100	286	25	NC	1.499	5.088	1.577	1.620
				WC	1.009	7.916	1.690	1.905
				SC	1.681	5.852	1.710	1.327
			33	NC	2.594	5.001	1.130	1.496
				WC	2.517	5.025	1.520	1.208
				SC	2.315	7.805	1.134	1.118
			50	NC	3.507	5.814	1.011	1.268
				WC	7.723	7.149	1.814	1.810
				SC	8.334	6.955	1.779	1.052
	200	580	25	NC	2.464	8.137	1.071	1.896
				WC	2.431	8.931	1.122	1.614
				SC	2.438	5.084	1.453	1.326
			50	NC	8.403	5.031	1.960	2.467
				WC	8.220	7.916	1.702	2.791
				SC	9.812	6.010	1.240	2.368

The two last columns (Gap_T and GAP_D) are calculated as

$$Gap_T = \frac{Time(TS_D) - Time(TS_T)}{Time(TS_T)} \times 100, \tag{7}$$

$$Gap_D = \frac{Dist(TS_T) - Dist(TS_D)}{Dist(TS_D)} \times 100, \tag{8}$$

to compare the solutions produced by TS when the distance (TS_D) or the duration (TS_T) is minimized. That is, Gap_T is the gap between the duration of the routes when the distance is minimized ($Time(TS_D)$) and when the duration is minimized ($Time(TS_T)$), while Gap_D is the gap between the total distance of the routes when the duration is minimized ($Dist(TS_T)$) and when the distance is minimized

($Time(TS_D)$). These two gaps indicate that the solutions obtained are significantly different depending if duration or distance is minimized.

8 Conclusion

This paper has proposed a tabu search heuristic, coupled with innovative techniques to evaluate neighborhood solutions in constant time, for the time-dependent vehicle routing problem with time windows on a road network. Computational experiments show that our method can identify high-quality solutions in very reasonable computation times on benchmark instances recently reported in the literature.

With regard to future work, we want to combine our tabu search with a previously developed deep learning neural network model that predicts travel speeds on the arcs of a road network, depending on a number of contextual variables. We will now apply this neural network in a real-time environment to continuously update the speed predictions based on current data. This update may, in turn, lead to a reoptimization of the planned routes using the tabu search.

References

[1] R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

[2] H. Ben Ticha. *Vehicle Routing Problems with Road-Network Information.* PhD thesis, Clermont Auvergne, 2017.

[3] H. Ben Ticha, N. Absi, D. Feillet, and A. Quilliot. Adaptive large neighborhood search for the vehicle routing problem with time windows with a multigraph representation for the road network. Technical report, Technical Report EMSE CMP–SFL 2017/7, Ecole des Mines de Saint Etienne, CMP, Gardanne, France, 2017.

[4] H. Ben Ticha, N. Absi, D. Feillet, A. Quilliot, and T. Van Woensel. A branch-and-price algorithm for the vehicle routing problem with time windows on a road network graph. Technical report, Technical Report EMSE CMP–SFL 2017/9, Ecole des Mines de Saint Etienne, CMP, Gardanne, France, 2017.

[5] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[6] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

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

[8] A.V. Donati, R. Montemanni, N. Casagrande, A.E. Rizzoli, and L.M. Gambardella. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, 185(3):1174–1191, 2008.

[9] M.A. Figliozzi. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, 48(3):616–636, 2012.

[10] B. Fleischmann, M. Gietz, and S. Gnutzmann. Time-varying travel times in vehicle routing. *Transportation Science*, 38(2):160–173, 2004.

[11] T. Garaix, C. Artigues, D. Feillet, and D. Josselin. Vehicle routing problems with alternative paths: An application to on-demand transportation. *European Journal of Operational Research*, 204(1):62–75, 2010.

[12] M. Gendreau, G. Ghiani, and E. Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189–197, 2015.

[13] F. Glover. Tabu search–Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[14] A. Haghani and S. Jung. A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, 32(11):2959–2986, 2005.

[15] Y. Huang, L. Zhao, T. Van Woensel, and J.-P. Gross. Time-dependent vehicle routing problem with path flexibility. *Transportation Research Part B: Methodological*, 95:169–195, 2017.

[16] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

[17] S. Jung and A. Haghani. Genetic algorithm for the time-dependent vehicle routing problem. *Transportation Research Record: Journal of the Transportation Research Board*, (1771):164–171, 2001.

[18] David S.W. Lai, Ozgun Caliskan Demirag, and Janny M.Y. Leung. A tabu search heuristic for the heterogeneous vehicle routing problem on a multigraph. *Transportation Research Part E: Logistics and Transportation Review*, 86:32 – 52, 2016.

[19] Gilbert Laporte, Stefan Ropke, and Thibaut Vidal. Chapter 4: Heuristics for the vehicle routing problem. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 87–116. SIAM, 2014.

[20] A.N. Letchford, S.D. Nasiri, and D.O. Theis. Compact formulations of the Steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1):83–92, 2013.

[21] C. Malandraki and M.S. Daskin. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, 1992.

[22] S. Mancini. Time dependent travel speed vehicle routing and scheduling on a real road network: the case of Torino. *Transportation Research Procedia*, 3:433–441, 2014.

[23] M. Setak, M. Habibi, H. Karimi, and M. Abedzadeh. A time-dependent vehicle routing problem in multigraph with FIFO property. *Journal of Manufacturing Systems*, 35:37 – 45, 2015.

[24] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[25] D. Taş, N. Dellaert, T. van Woensel, and T. de Kok. The time-dependent vehicle routing problem with soft time windows and stochastic travel times. *Transportation Research Part C: Emerging Technologies*, 48:66–83, 2014.

[26] H. Wang and Y. Lee. Two-stage particle swarm optimization algorithm for the time dependent alternative vehicle routing problem. *Journal of Applied & Computational Mathematics*, 3(4):1–9, 2014.