Research School for Operations Management and Logistics

The Time-Dependent Pro_table Pickup and Delivery Traveling Salesman Problem with Time Windows

Peng Sun
Said Dabia
Lucas P. Veelenturf
Tom Van Woensel

Beta Working Paper series 490

BETA publicatie	WP 490 (working paper)
ISBN	
ISSN	
NUR	804
Eindhoven	November 2015

The Time-Dependent Profitable Pickup and Delivery Traveling Salesman Problem with Time Windows

Peng Sun
School of Industrial Engineering,
Operations, Planning, Accounting and Control (OPAC),
Eindhoven University of Technology,
Eindhoven, 5600 MB, The Netherlands, p.sun@tue.nl,

Said Dabia
VU University Amsterdam,
Amsterdam, 1081 HV, The Netherlands, s.dabia@vu.nl,

Lucas P. Veelenturf, Tom Van Woensel
School of Industrial Engineering,
Operations, Planning, Accounting and Control (OPAC),
Eindhoven University of Technology,
Eindhoven, 5600 MB, The Netherlands, {l.p.veelenturf@tue.nl,t.v.woensel@tue.nl}

In this paper we present the time-dependent profitable pickup and delivery traveling salesman problem with time windows (TDPPDTSPTW). The problem consists of determining a tour with a departure time at a depot which maximizes the difference between the collected profit and the total tour duration. Travel times are considered to be time-dependent, i.e. the travel time between two nodes depends on the time the tour starts. This enables to deal with real life challenges such as road congestion. A tailored labeling algorithm with time windows and pickup and delivery (TLTWPD) is developed to find the optimal tour. In the labeling algorithm, new dominance criteria are introduced to discard unpromising labels. Our computational results demonstrate that the algorithm is capable of solving instances with up to 40 locations and 20 pickup and delivery requests to optimality given a pre-set maximum memory allowance. Furthermore, we present a restricted dynamic programming heuristic algorithm to improve the computation time. This heuristic does not guarantee optimality since it restricts the considered solution space. However, the results of the relatively fast heuristic demonstrate that the solution found is close to optimal (with an average gap of 0.01%).

Key words: Traveling salesman problem, pickup and delivery, tailored labeling algorithm, time windows, profits

History:

1. Introduction.

In this paper, we introduce the time-dependent profitable pickup and delivery traveling salesman problem with time windows (TDPPDTSPTW). This problem consists of a single vehicle without capacity limit and a set of requests which have a pick-up and a delivery node. Both pickup and delivery nodes have a time window in which they should be served. Moreover, a delivery node can only be served after the pickup node of the same request is visited. For each served request a profit is collected. Moreover, a time-dependent travel time is assigned to each edge linking two nodes to

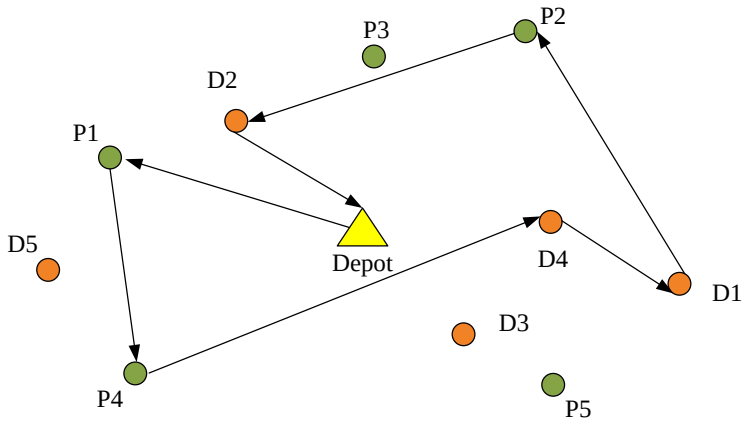

Figure 1 An Illustration of a Solution in the TDPPDTSPTW Network.

capture travel speed variation during a day. The objective is to determine a tour for the vehicle that starts and ends at the depot and that maximizes the difference between the total collected profits and total tour duration. In Figure 1 an example of such an route is given, in which $\mathrm{N_P}i$ and $\mathrm{N_D}i$ represent the pickup node and delivery node of request i respectively.

This problem is NP-hard because it is an extension of the traveling salesman problem with pickup and delivery (TSPPD), which itself is an extension of traveling salesman problem (TSP). In contrast to the TSPPD and TSP, in the TDPPDTSPTW, it is not necessary to visit all the requests (See figure 1).

The TDPPDTSPTW has practical applications in many routing and scheduling problems. For instance, the parcel and post industry is nowadays faceing a number of challenges, as they are forced to improve services without increasing prices. Global competition forces them to extend their business far from their traditional service area. On the one hand, serving a customer may be attractive because it generates profits for the providers, or it increases the capacity utilization of the vehicle. On the other hand, the additional cost for serving the customer may not be economically beneficial. The problem becomes to decide which of these potential customers to serve and how to construct the vehicle route in such a way that an appropriate objective function is optimized.

Another example is the Share-a-Ride problem (SARP) introduced by Li et al. (2014), in which passengers and parcels are simultaneously handled within the same transportation network. This problem allows rejections of both people and freight requests (e.g., if the capacity is not large enough). The time-dependent travel times are used to capture the crowded urban traffic situation. In real life, travel time depends on the departure time and the traveling speed of a route is nomally slower at peak hours. Moreover, events as extreme weather or road construction activities might also significantly increase the travel times.

The main contributions of this paper are summarized as follows. First, we introduce a new class of models by extending the classical TSPPD to the time-dependent profitable pickup and

delivery traveling salesman problem with time windows (TDPPDTSPTW). Secondly, we propose a tailored labeling algorithm with time windows and pickup and delivery to solve this problem. For this labeling algorithm, new dominance criteria are introduced. Lastly, a restricted dynamic programming heuristic is introduced with a high solution quality and lower computation times than the labeling algorithm.

The remainder of the paper is structured as follows. Section 2 provides a brief review of the existing work that is related to this paper. Section 3 defines the TDPPDTSPTW and introduces mathematical formulation of the problem. In Section 4, we present the tailored labeling algorithm with time windows and pickup and delivery (TLTWPD). Section 5 describes the restricted dynamic programming heuristic algorithm. Finally, computational results are reported in Section 6, followed by conclusions in Section 7.

2. Literature review

There are three classes of problems closely related to the problem studied in this paper: the traveling salesman problem with pickup and delivery (TSPPD); the time-dependent vehicle routing problem (TDVRP); and the traveling salesman problem with profits (TSP with profits).

2.1. The traveling salesman problem with pickup and delivery (TSPPD)

Our problem extends the TSPPD by considering time-dependent travel time. The TSPPD is firstly introduced by Ruland and Rodin (1997) which also has many applications (e.g., dial-a-ride systems and courier services). However, only limited research can be found about it. Moreover, the TSPPD appears as a subproblem in the pickup and delivery problem (PDP) which is much better studied in the literature.

Currently, the most popular methodology for solving the TSPPD is branch-and-cut. Ruland (1994) and Ruland and Rodin (1997) consider the undirected case and develop four classes of valid inequalities that are embedded in a branch-and-cut algorithm to solve at most 15 pickup and delivery requests. Recently, Dumitrescu et al. (2010) study the same problem, the authors analyze its polyhedral structure and propose new valid inequalities that are shown to be facets for the TSPPD polytope. Their algorithm is capable of solving instances involving up to 35 pickup and delivery requests to optimality.

The TSPPD appeared as pricing problem for the PDP that is usually named the elementary shortest path problem with time windows, capacity and pickup and delivery (ESPPTWCPD). Sol (1994), Sigurd and Pisinger (2004) and Ropke et al. (2009) presented a labeling algorithm with several different dominance rules to solve this problem to optimality.

2.2. The time-dependent vehicle routing problem (TDVRP)

Another related problem is TDVRP. Although the TDVRP has attracted the attention of many researchers, literature on this subject remains scarce. The pioneering work is done by Malandraki and Daskin (1992) and Malandraki and Dial (1996). They both proposed a mixed integer linear program and several heuristics to solve the problem are provided. The First-In-First-Out (FIFO) property, which implies that for every arc a later departure time results in a later (or equal) arrival time, is an intuituive and desirable property for time dependent problems. Ichoua et al. (2003) and Dabia et al. (2013) consider the TDVRP with travel time variability using "constant speed" time periods which do not allow passing. The idea of constant speed time periods is adopted in our problem as well.

Moreover, due to the complexity of the time-dependent problem, most of the existing algorithms are based on heuristics. In van Woensel et al. (2008) a tabu search heuristic is used to solve the capacitated vehicle routing problem with time dependent travel times. An approximation based on queuing theory and on the volumes of vehicles in a link is used to determine travel speed. Donati et al. (2008) developed a multi ant colony system for the TDVRP and Ibaraki et al. (2008) proposed an iterated local search heuristic for the the time-dependent vehicle routing problem with time windows (TDVRPTW). Recently, Dabia et al. (2013) developed a branch-and-price algorithm for TDVRPTW, where a tailored labeling algorithm is presented to solve the time-dependent shortest path problem with resource constraint (TDSPPRC), which is the pricing problem in the algorithm.

2.3. The traveling salesman problem with profits (TSP with profits)

The proposed problem also extends the traveling salesman problem with profits (TSP with profits), in which profits are associated with each request and the overall goal is to find a shortest path with maximal collected profits. This also means that in contrast to the original TSP, not all nodes have to be visited. In comparison with our study, it does not include time dependency and requests with pickup and delivery.

According to Feillet et al. (2005), TSPs with profits consist of three generic problems, depending on the way the two terms *profits* and *travel time* are adressed in the objective function and constraints. They can be summarized in the following categories: the profitable tour problem (PTP); the prize-collecting traveling salesman problem (PCTSP); and the orienteering problem (OP).

The profitable tour problem (PTP): Dell' Amico et al. (1995) studied the profitable tour problem (PTP) where both the profits and the travel time are combined in the objective function. Our study builds on the PTP by having the the profits and the travel time in the objective as well.

The prize-collecting traveling salesman problem (PCTSP): In the prize-collecting TSP (PCTSP) the objective is similar to PTP but a constraint is added to ensure that a minimum amount of profits must be collected on the tour. The original definition of PCTSP by Balas (1989) also adds penalty terms for unvisited vertices into the objective function.

The orienteering problem (OP): An abundant number of publications are devoted to the orienteering problem (OP), which gives the travel cost term as a constraint and aims to maximize the collected profits subject to a constraint for the maximum allowed tour length. This problem is also known as the selective traveling salesman problem (Laporte and Martello (1990)).

A variant of the OP is the time-dependent orienteering problem (TDOP) with time-dependent travel times . Fomin and Lingas (2002) provide a $(2+\epsilon)-$approximation algorithm for the TDOP which runs in polynomial time if the ratio between the minimum and maximum travel time between any two sites is constant. Li (2011) designed a novel dynamic labeling algorithm for the TDOP in which time is is measured in discrete units. Therefore, the FIFO property may not be satisfied in their model. Verbeeck et al. (2014) recently devised a fast solution method for the TDOP with time windows based on the ant colony optimization algorithm. For more details about the OP and its variants, readers are referred to Vansteenwegen et al. (2011).

3. Problem description and mathematical formulation

In this section, we introduce the notation that is used throughout the paper. We then present a mathematic formulation for the problem.

3.1. Problem definition

The Time-Dependent Profitable Pickup and Delivery Traveling Salesman Problems with Time windows (TDPPDTSPTW) is defined as follows. We consider a set of n requests $R_1, ..., R_n$, where R_i $(i = 1, ..., n)$ is associated with a pickup node i and a delivery node $n+i$. Let $G = (N, A)$ be an undirected graph, where $N = \{0, 1, ..., 2n+1\}$ is the set of all nodes. 0 and $2n+1$ are the origin and destination depots of the vehicle. We define the subsets $N_P = \{1, ..., n\}$ and $N_D = \{n+1, ..., 2n\}$ as the pickup and delivery nodes, respectively. Each pickup node $i \in N_P$ is associated with a profit r_i. Moreover, a time window $[e_j, l_j]$ is associated with each node $j \in N_P \cup N_D$, where e_j and l_j represent the earliest and latest time respectively at which service may start at node j. The vehicle needs to wait until time e_j, if arriving at j before e_j; and arriving later than l_j is not allowed. We denote $[e_0, l_0]$, $[e_{2n+1}, l_{2n+1}]$ as the time windows of the origin and the destination depot, respectively. Let s_i be the service time of node $i \in N$. Without loss of generality, we assume $s_0 = s_{2n+1} = e_0 = 0$. Let $\tau_{ij}(t_i)$ denote the travel time from node i to node j, which depends on the departure time t_i at node i. We consider only one vehicle without capacity limit and define the set of feasible arcs

as $A = \{(i,j) \in N \times N : i \neq j \text{ and } e_i + s_i + \tau_{ij}(e_i + s_i) \leq l_j\}$. This means that an arc from node i to node j is only included if it is possible to traverse from node i to j while respecting the time windows of both node i and node j.

The planning horizon is divided into several time zones. Each arc $(i,j) \in A$ has a speed profile associated to it, which consists of a constant speed within each time zone. By using those stepwise speed functions, the FIFO property holds for every arc in the graph G (i.e. a later departure always leads to a later arrival and therefore overtakings will not occur). The speed profiles can be different for each arc.

Figure 2 depicts a speed profile and the corresponding travel time function for some arc (i,j). We denote the points a, b, c, d, and e where the speed changes as *speed breakpoints*. There are also *travel time breakpoints* in the travel time function. These are defined as the departure times which ensure to arrive at node j exactly at a speed breakpoint (e.g., $a^{'}$ is the departure time at node i to arrive at node j at time a) using the method as described in Ichoua et al. (2003).

The travel time function is piecewise linear and can be represented by the breakpoint values. Note that in case of time-dependent travel times, the triangle inequality does not necessarily hold. Intuitively, when the direct link between node h and l is heavily congested, we may reach destination node l earlier by taking a diverted routed (i.e., via one or several other nodes) than by the direct route from node h.

Because of the FIFO property of the travel time functions, a later departure at the depot 0 always results in a later arrival time at node i. Therefore, if a path is infeasible with a certain departure time t_0 at the origin depot(i.e., at least one node's time window in the path is violated), it will also be infeasible for any departure time $t^{'} \geq t_0$ at the origin depot.

Given a path $p = (v_0, v_1, ..., v_k)$ with $v_0 = 0$, we define $\delta^p_{v_i}(t)$ as the ready time function at node v_i given a departure time t at node 0. This ready time function is nondecreasing in t and can be recursively calculated for each node in the path as follows:

$$\delta^p_{v_i}(t) = \begin{cases} t & \text{if } i = 0, \\ \max\{e_{v_i} + s_{v_i}, \delta^p_{v_{i-1}}(t) + \tau_{v_{i-1},v_i}(\delta^p_{v_{i-1}}(t)) + s_{v_i}\} & \text{otherwise.} \end{cases} \quad (1)$$

In case $i \neq 0$, the calculation includes two piecewise linear functions. This means that we can represent the ready time function by using the *ready time function breakpoints*. These are either breakpoints of the predecessor node's ready time function, breakpoints of the travel time function, or due to the time window of node v_i.

The duration of the path given a departure time t at node 0 can be calculated as $\delta^p_{v_k}(t) - t$, which is also a piecewise linear function. In this problem we aim to minimize the total duration of the used tour in the solution instead of minimizing the sum of the arc cost. Therefore, the minimum duration can be computed by considering the breakpoints of the ready time function.

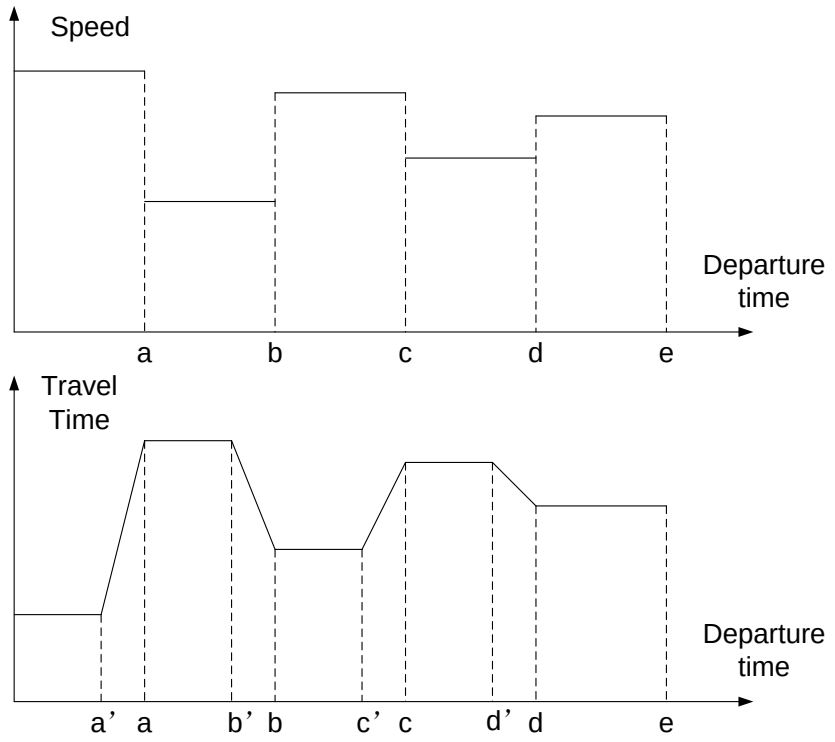

Figure 2 **Speed and Travel Time Functions.**

3.2. Mathematical formulation of the TDPPDTSPTW

For every arc $(i,j) \in A$, we denote T_{ij} as the set of time zones of the corresponding travel time function $\tau_{ij}(t_i)$. The number of time zones in set T_{ij} can be described as $\mid T_{ij} \mid$. A time zone $T_m \subset T_{ij}$, is defined by two consecutive travel time breakpoints, $T_m = [w_m, w_{m+1}]$. As $\tau_{ij}(t_i)$ is linear in each time zone. Using w_m, w_{m+1}, $\tau_{ij}(w_m)$ and $\tau_{ij}(w_{m+1})$ we can easily calculate the corresponding slope θ_m and its intersection η_m with the y-axis. Therefore

$$\tau_{ij}(t_i) = \theta_m t_i + \eta_m. \qquad \forall t_i \in T_m \tag{2}$$

Furthermore, let x_{ij}^m be a binary variable that takes value 1 if and only if the vehicle traveres the arc $(i,j) \in A$ and its departure time is in time zone m, and let y_i be a binary variable that equals 1 if and only if node $i \in N_P \cup N_D$ is visited. We also denote t_{ij}^m as the departure time in time zone T_m when traveling from i to j. t_{ij}^m is such that

$$t_{ij}^m = \begin{cases} t_i & \text{if } x_{ij}^m = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Consequently, when traveling from i to j, we have that:

$$t_i = \sum_{j \in \mathcal{N} \setminus \{0\}} \sum_{m=0}^{|T_{ij}|} t_{ij}^m. \tag{4}$$

This means that the travel time function $\tau_{ij}(t_i)$ of arc (i,j) at node i can be expressed as:

$$\tau_{ij}(t_i) = \sum_{m=0}^{|T_{ij}|} (\theta_m t_{ij}^m + \eta_m x_{ij}^m). \tag{5}$$

The Mixed Integer Programming formulation for the TDPPDTSPTW is described as follows:

$$\max \sum_{j \in \mathcal{N}_\mathcal{P}} r_j y_j - (t_{2n+1} - t_0) \tag{6}$$

subject to

$$\sum_{j \in \mathcal{N}_\mathcal{P}} \sum_{m=0}^{|T_{0j}|} x_{0j}^m = 1 \tag{7}$$

$$\sum_{i \in \mathcal{N}_\mathcal{D}} \sum_{m=0}^{|T_{i,2n+1}|} x_{i,2n+1}^m = 1 \tag{8}$$

$$\sum_{i \in \mathcal{N} \setminus \{2n+1\}} \sum_{m=0}^{|T_{ij}|} x_{ij}^m = y_j \qquad \forall j \in \mathcal{N} \setminus \{0\} \tag{9}$$

$$\sum_{i \in \mathcal{N} \setminus \{2n+1\}} \sum_{m=0}^{|T_{ik}|} x_{ik}^m - \sum_{j \in \mathcal{N} \setminus \{0\}} \sum_{m=0}^{|T_{kj}|} x_{kj}^m = 0 \qquad \forall k \in \mathcal{N} \setminus \{0, 2n+1\} \tag{10}$$

$$\sum_{j \in \mathcal{N} \setminus \{0\}} \sum_{m=0}^{|T_{ij}|} x_{ij}^m - \sum_{j \in \mathcal{N} \setminus \{0\}} \sum_{m=0}^{|T_{n+i,j}|} x_{n+i,j}^m = 0 \qquad \forall i \in \mathcal{N}_\mathcal{P} \tag{11}$$

$$t_j \geq (1 + \theta_m) t_{ij}^m + \eta_m x_{ij}^m + s_j x_{ij}^m \qquad \forall i \in \mathcal{N} \setminus \{2n+1\}, j \in \mathcal{N} \setminus \{0\} \tag{12}$$

$$t_{n+i} \geq t_i \qquad \forall i \in \mathcal{N}_\mathcal{P} \tag{13}$$

$$t_i = \sum_{j \in \mathcal{N} \setminus \{0\}} \sum_{m=0}^{|T_{ij}|} t_{ij}^m \qquad \forall i \in \mathcal{N} \setminus \{2n+1\} \tag{14}$$

$$w_m x_{ij}^m \leq t_{ij}^m \leq w_{m+1} x_{ij}^m \qquad \forall i, j \in \mathcal{N}, \forall m, m+1 \in \mid T_{ij} \mid \tag{15}$$

$$e_i y_i \leq t_i \leq l_i y_i \qquad \forall i \in \mathcal{N}_\mathcal{P} \cup \mathcal{N}_\mathcal{D} \tag{16}$$

$$x_{ij}^m, y_i \in \{0,1\} \qquad \forall i, j \in \mathcal{N}, \forall m \in \mid T_{ij} \mid \tag{17}$$

The objective function (6) aims to find a tour that maximizes the collected profits minus the total traveling cost. Constraints (7)-(8) guarantee that the path starts in the origin depot 0 and ends in the destination depot $2n+1$. Constraints (9) guarantee that every node, except the nodes representing the start and end depots, is visited at most once. Constraints (10) keeps the flow conservation. Constraints (11) ensure that it is not possible to visit only the pickup node or only the delivery node of a certain request. Constraints (12) guarantee that the departure time of a succeeding node in the route is large or equal to the sum of the departure time of the previous node together with the travel time between these two nodes and the required service time. Precedence

constraints (13) ensure that for each request i, the pickup node is visited before the delivery node. Constraints (14) is formulated as mentioned before in (4). Constraints (15)-(16) force the departure time of each request to be in the given time zone and the given time window. Due to the computational inefficiency in solving large-scale instances with a commercial ILP solver, we develop a tailored labeling algorithm with time windows and pickup and delivery to solve this problem.

4. A tailored labeling algorithm with time windows and pickup and delivery (TLTWPD)

In order to solve the TDPPDTSPTW, we introduce a new exact dynamic programming algorithm, that is named as the tailored labeling algorithm with time windows and pickup and delivery (TLTWPD). Ropke et al. (2009) developed a labeling algorithm to solve the pickup and delivery problems with time windows (PDPTW). However, this time-independent algorithm is only efficient if the triangle inequality holds. More recently, Dabia et al. (2013) proposed a tailored labeling algorithm for the time-dependent vehicle routing problem with time windows. Their algorithm has great potential for the time-dependent routing problem without precedence constraints. However, in contrast, we have to deal with the precedence constraints. Furthermore, note that our proposed algorithm for the TDPPDTSPTW can be generalized to solve other time-dependent routing problems with precedence constraints.

The algorithm starts from the depot 0 and progressively extends all feasible paths untill their reach the end depot $2n + 1$. Moreover, to speed up our tailored labeling algorithm, instead of starting the label extension only from the origin depot 0, we simulaneously generate labels both from the origin depot to its successors and from the destination depot to its predecessors. Whereas both forward labels and backward labels are extended until some fixed time t_m(e.g., the middle of the planning horizon) is reached but not further. At the end, the complete paths are generated by merging forward labels and backward labels. This approach has shown great potential for improving the running time of related resource constrained shortest path problems (see, e.g. Righini and Salani (2006) and Dabia et al. (2013)). The forward TLTWPD algorithm is introduced in Section 4.1, followed by backward TLTWPD algorithm in section 4.2. Finally, we discuss the way to merge forward and backward labels in Section 4.3.

4.1. The Forward TLTWPD Algorithm

For each forward label L_f of partial we denote $p(L_f)$ as the partial path and store the following data:

- $v(L_f)$ the last node visited on the partial path $p(L_f)$.
- $O(L_f)$ the set of incomplete requests, i.e., the pickup node served but not the delivery node.

- $U(L_f)$ the set of requests for which the pickup nodes are already visited along the partial path $p(L_f)$. It contains both the incomplete requests and the complete requests. Therefore, $O(L_f) \subseteq U(L_f)$.
- $\delta_{L_f}(t)$ the ready time at $v(L_f)$ when departed at the origin depot at t and reached $v(L_f)$ through the partial path $p(L_f)$.
- $r(L_f)$ the overall profits collected with the requests visited on the partial path $p(L_f)$.

We extend a label $L_f^{'}$ along an arc $(v(L_f^{'}), j)$, only when the extension is feasible:

$$\delta_{L_f^{'}}(0) + \tau_{v(L_f^{'}),j}(\delta_{L_f^{'}}(0)) + s_j \leq \min\{t_m, l_j + s_j\} \qquad \forall j \in \mathcal{N}/\{0\} \tag{18}$$

Inequality (18) ensures time window feasibility and guarantees the extention is stopped when t_m is reached. Furthermore, $L_f^{'}$ and j must satisfy one of the following three conditions:

$$1 \leq j \leq n \quad \wedge \quad j \notin U(L_f^{'}) \tag{19}$$

$$n+1 \leq j \leq 2n \quad \wedge \quad j - n \in O(L_f^{'}) \tag{20}$$

$$j = 2n+1 \quad \wedge \quad O(L_f^{'}) = \emptyset \tag{21}$$

Condition (19) states that j is not visited before, if it is a pickup node. Condition (20) assure that if j is a delivery node, the corresponding pickup node should have already been visited. Condition (21) states that if j is the end depot then all visited requests should have been completed. In the presence of those conditions, only elementary paths that satisfy pairing constraint (11) are generated. If the extension along the arc $(v(L_f^{'}), j)$ is feasible, then a new label L_f is created. The information in label L_f is updated as follows:

$$v(L_f) = j \tag{22}$$

$$\delta_{L_f}(t) = \max\{e_j + s_j, \delta_{L_f^{'}}(t) + \tau_{L_f^{'},j}(\delta_{L_f^{'}}(t)) + s_j\} \tag{23}$$

$$r(L_f) = \begin{cases} r(L_f^{'}) + r_j & \text{if } j \in N_P, \\ r(L_f^{'}) & \text{otherwise.} \end{cases} \tag{24}$$

$$O(L_f) = \begin{cases} O(L_f^{'}) \cup \{j\} & \text{if } j \in N_P, \\ O(L_f^{'}) \setminus \{j-n\} & \text{if } j \in N_D. \end{cases} \tag{25}$$

$$U(L_f) = \begin{cases} U(L_f^{'}) \cup \{j\} & \text{if } j \notin N_P, \\ U(L_f^{'}) & \text{otherwise.} \end{cases} \tag{26}$$

Equations (22)-(24) set the current node, the ready time function and the collected profits of the new label, respectively. Equation (25) updates the set of incomplete requests $O(L_f)$. Equation (26) updates the set of requests $U(L_f)$. Here, let $\text{dom}(L_f)$ and $\text{img}(L_f)$ be the domain and image of the

ready time function $\delta_{L_f}(t)$ respectively. If the partial path is feasible, departure at time 0 from the origin depot is always feasible. Therefore, $\text{dom}(L_f)$ always takes the form of $[0,t]$ for some $t \geq 0$. When $v(L_f) = 2n+1$, the objective of the path corresponding to L_f is:

$$obj(L_f) = r(L_f) - \min_{t \in Dom(L_f)} \{\delta_{L_f}(t) - t\} \tag{27}$$

In the labeling algorithm, all possible extentions are processed and stored for each label. However, the number of labels that can be processed is typically very large and computationly expensive. Therefore, a dominance test is established between pairs of labels ending in the same node to reduce the number of labels, so that the algorithm records only non-dominated labels. The dominance test is the following. Let $E(L_f)$ denote the set of feasible extentions of the label L_f to the end depot. More specifically, $E(L_f)$ is the set of all partial paths that can depart at node $v(L_f)$ at time $\delta_{L_f}(0)$ or later and reach the destination depot without violating precedence constraints and time windows at pickup and delivery nodes. If $L \in E(L_f)$, we define $L_f \bigoplus L$ as the label resulting from extending L_f by L. Therefore, in Proposition 1, the sufficient conditions $(1)-(6)$ are introduced.

PROPOSITION 1. *Label L_f^2 is dominated by label L_f^1 if*

1. $v(L_f^1) = v(L_f^2)$
2. $O(L_f^1) = O(L_f^2)$
3. $U(L_f^1) \subseteq U(L_f^2)$
4. $dom(L_f^2) \subseteq dom(L_f^1)$
5. $\delta_{L_f^1}(t) \leq \delta_{L_f^2}(t), \forall t \in dom(L_f^2)$
6. $r(L_f^1) \geq r(L_f^2)$

The proof of this proposition is given in the appendix. Dominance as introduced in Proposition 1 has several weaknesses and may not be able to eliminate too many unpromising labels. On the one hand, $O(L_f^1) = O(L_f^2)$ and $U(L_f^1) \subseteq U(L_f^2)$ typically implies $r(L_f^1) \leq r(L_f^2)$ because the collected profit increase when visiting more pickup nodes. Hence conditions $2, 3$ and 6 are all true only in case of equality. On the other hand, labels corresponding to paths with a very long duration will typically end up in an unattractive route if the collected profit is high enough to compensate the traveling cost. Such labels can be hard to dominate. In Figure 3, the numbers associated with the arcs represent travel times and the numbers associated with the nodes represent profits, and both partial paths P_1 and P_2 end at the same pickup node S_3. Futhermore, a path's objective value is equal to the sum of the collected profits corresponding to the request nodes visited along that path reduced by the total travel time duration cost. It shows that, compare to P_1, P_2 collects 20 units extra profits with the expense of 200 units extra traveling cost. Therefore, extending P_1 clearly results in a better solution. Because of condition 6, the label representing partial path P_2 will not

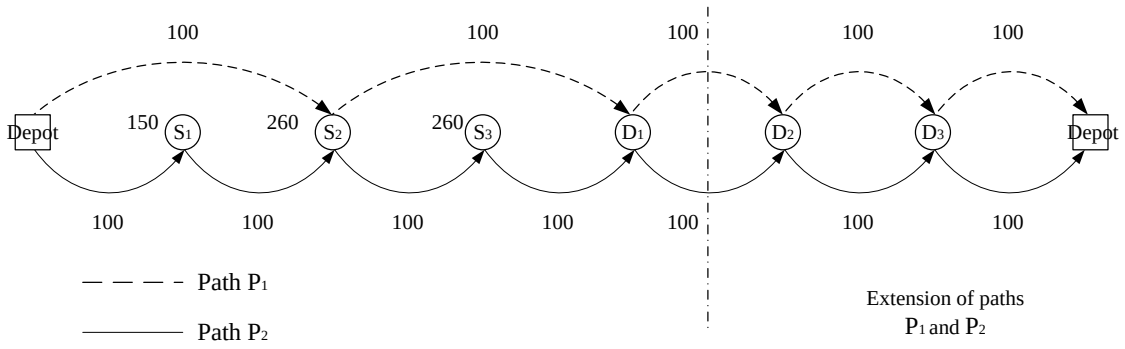

Figure 3 Example of Dominance Test.

be dominated by the one representing partial path P_1. Based on the above mentioned issue, we introduce a new dominance criterion to allow a label L_f^1 to dominate another label L_f^2, if label L_f^1 is "much better" than L_f^2 either the profit dimension or in the ready time function. We define the interval

$$I \subseteq (-\infty, \max(dom(L_f^1)) - \max(dom(L_f^2))) \tag{28}$$

Based on I, we also define a real number $\phi(L_f^1, L_f^2)$,

$$\phi(L_f^1, L_f^2) = \max\{x \in I : \delta_{L_f^1}(\max\{0, t+x\}) \leq \delta_{L_f^2}(t), \forall t \in dom(L_f^2)\} \tag{29}$$

When $\phi(L_f^1, L_f^2)$ is positive, the value describes how much the departure time of the partial path represented by label L_f^1 can be postponed, compare to path $p(L_f^2)$, and still reach node $v(L_f^1)$ at the same time or earlier than path $p(L_f^2)$. if $\phi(L_f^1, L_f^2)$ is negative, it measures how much earlier path $p(L_f^1)$ need to depart at the origin depot, compared to path $p(L_f^1)$, to ensure that node $v(L_f^1)$ is reached at the same time or earlier than path $p(L_f^2)$. In Figure 4, we depict several simple example: If there is no intersection between label L_f^1 and L_f^2, $\phi(L_f^1, L_f^2)$ is positive when $\max(\text{dom}(L_f^1)) > \max(\text{dom}(L_f^2))$ (see Figure 4(a)), or negative when $\max(\text{dom}(L_f^1)) < \max(\text{dom}(L_f^2))$ (see Figure 4(b)). Otherwise, $\phi(L_f^1, L_f^2)$ can only be negative (see Figure 4(c)). Note that, we use $\phi(L_f^1, L_f^2)$ to relax condition 4 and 5 of Proposition 1.

Another improvement is, that for every L_f, we extend $U(L_f)$ to the set $\widetilde{U}(L_f)$ by adding requests that their pickup nodes are unreachable from $v(L_f)$. Time-dependent travel times cannot guarantee the triangle inequality. Therefore, a node that cannot be directly reached from node $v(L_f)$ might be indirectly reached via a diverted route. However, a lower bound of the earliest ready time at any pickup or delivery node following node $v(L_f)$ in the partial path by $t_e = \min_{j \in N_P \cup N_D}\{\delta_{L_f}(0) + \tau_{v(L_f)j}(\delta_{L_f}(0))\}$. Any node j with $b_j < t_e$ will be unreachable from $v(L_f)$. Especially, when $j \in N_D$, the partial path corresponding to L_f can be removed, because it will lead to an unfeasible solution. This test can be done quickly, although we might fail to find all unreachable requests. Finally, we state the improved dominance test in Proposition 2 as follows:

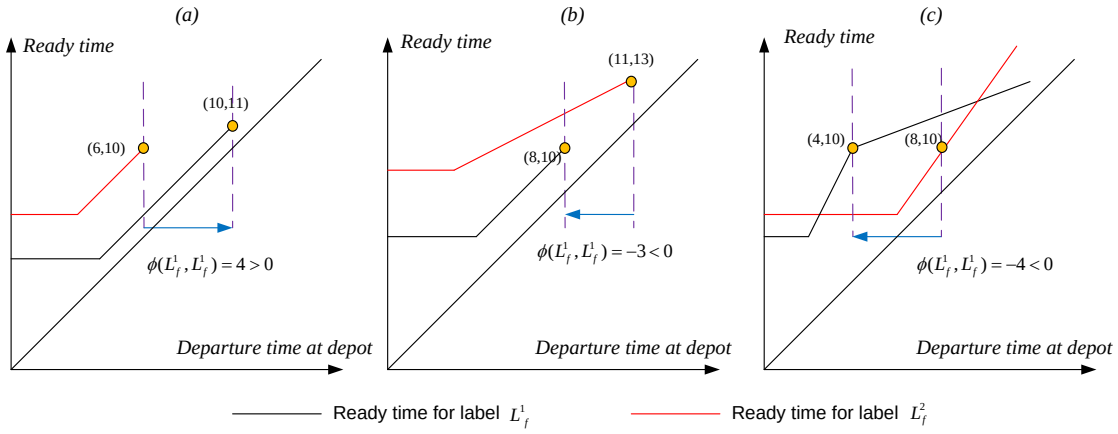

Figure 4 **Illustration of $\phi(L_f^1, L_f^2)$.**

PROPOSITION 2. *Label L_f^2 is dominated by label L_f^1 if*

1. $v(L_f^1) = v(L_f^2)$
2. $U(L_f^1) \subseteq \widetilde{U}(L_f^2)$
3. $O(L_f^1) = O(L_f^2)$
4. $\delta_{L_f^1}(0) \leq \delta_{L_f^2}(0)$
5. $r(L_f^1) \geq r(L_f^2) - \phi(L_f^1, L_f^2)$

PROOF OF PROPOSITION 4.2. Consider two labels L_f^1 and L_f^2 that satisfy the five conditions in proposition 2. For any $L \in E(L_f^2)$, we need to show that (1) $L \in E(L_f^1)$ and (2) $obj(L_f^1 \bigoplus L) \geq obj(L_f^2 \bigoplus L)$.

As to the first point, the path ($p(L_f^1 \bigoplus L)$ is elementary because of condition 2 and 3, and it is not violate time windows because of the FIFO assumption along with Condition 4 that ensure us to reach node $v(L_f^1)$ at the same time or earlier by using path $p(L_f^1)$ than by using path $p(L_f^2)$, given that we depart at the depot early enough.

To show the second point, let $L_f^{1*} = L_f^1 \bigoplus L$ and $L_f^{2*} = L_f^2 \bigoplus L$. We also denote $t_0^2 = argmin_{t \in dom(L_f^{2*})}\{\delta_{L_f^{2*}}(t) - t\}$ as the optimal departure time from the depot for path $p(L_f^{2*})$. The objective value of the path is:

$$\begin{aligned} obj(L_f^{2*}) &= r(L_f^{2*}) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) \\ &= r(L_f^2) + r(L) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) \end{aligned} \tag{30}$$

Now consider the path $p(L_f^{1*})$ resulting from extending L_f^1 by L. We denote $r(L)$ as the sum of the profits associated with the nodes visited along path $p(L)$. Moreover, consider a departure time

at the depot of this path $t_0^1 = \max\{0, t_0^2 + \Phi(L_f^1, L_f^2)\}$. This departure time ensures that we reach node $v(L_f^1)$ at time $t_0^* = \delta_{L_f^2}(t_0^2)$ or earlier, meaning $\delta_{L_f^1}(t_0^1) \leq \delta_{L_f^2}(t_0^2)$. t_0^2. Due to the definition of $\Phi(L_f^1, L_f^2)$, t_0^2 is also a feasible departure time for label L_f^{1*}. Moreover, t_0^1 provides an lower bound on $obj(L_f^{1*})$:

$$obj(L_f^{1*}) \geq r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0^1) - t_0^1) \tag{31}$$

Furthermore:

$$\begin{aligned}
&r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0^1) - t_0^1) \\
&\geq (r(L_f^1) + r(L)) - (\delta_{L_f^{2*}}(t_0^2) - \max\{0, t_0^2 + \Phi(L_f^1, L_f^2)\}) \\
&\geq (r(L_f^1) + r(L)) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2 - \Phi(L_f^1, L_f^2)) \\
&\geq (r(L_f^2) - \Phi(L_f^1, L_f^2) + r(L)) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2 - \Phi(L_f^1, L_f^2)) \\
&= (r(L_f^{2*}) + r(L)) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) = obj(L_f^{2*})
\end{aligned} \tag{32}$$

Note that the first inequality uses $\delta_{L_f^1}(t_0^1) \leq \delta_{L_f^2}(t_0^2)$ and the FIFO property together with the definition of t_0^1. The second inequality is derived by the simple fact that $\forall x \in R : -\max\{0, x\} \leq -x$, and the third inequality uses condition 5 of Proposition 2. □

4.2. The Backward TLTWPD Algorithm

In the backward TLTWPD algorithm, labels are extended from the end depot $2n+1$ to its predecessors. For a label L_b, we associate the following components:

- $v(L_b)$ the first node visited on the partial path $p(L_b)$.
- $O(L_b)$ the set of incomplete requests, i.e., the delivery has been served but not the pickup node.
- $U(L_b)$ the set of requests that their delivery nodes have already been visited along the partial path $p(L_b)$. It contains both the incomplete requests and the complete requests. Therefore, $O(L_b) \subseteq U(L_b)$.
- $\delta_{L_b}(t)$ the arrival time at the end node $2n+1$ through the partial path represented by L_b and when leaving node $v(L_b)$ at time t.
- $r(L_b)$ the overall profits collected with the requests completed on the partial path $p(L_b)$.

Let $\text{dom}(L_b)$ be the domain of the function $\delta_{L_b}(t)$ and define $t(L_b) = \max(\text{dom}(L_b))$. Thus, We extend a label $L_b^{'}$ along an arc $(j, v(L_b^{'}))$ to create a new label L_b. The extension is legal only if

$$t(L_b) \geq \max\{t_m, e_j + s_j\} \quad \text{if } j \in N_P \cup N_D \tag{33}$$

Inequality (33) ensure time window feasibility for the request node and the extension will be stopped when t_m is reached. Furthermore, $L_b^{'}$ and j must satisfy one of the following three conditions:

$$1 \leq j \leq n \quad \wedge \quad j+n \in O(L_b^{'}) \tag{34}$$

$$n+1 \leq j \leq 2n \quad \wedge \quad j \notin U(L_b^{'}) \tag{35}$$

$$j=0 \quad \wedge \quad O(L_f^{'}) = \emptyset \tag{36}$$

If the extension along the arc $(j, L_b^{'})$ is feasible, the information in label L_b is set as follows:

$$v(L_b) = j \tag{37}$$

$$\delta_{L_b}(t) = \delta_{L_b^{'}}(\max\{e_{v(L_b^{'})}, t + \tau_{jv(L_b^{'})}(t)\} + s_{v(L_b^{'})}) \tag{38}$$

$$r(L_b) = \begin{cases} r(L_b^{'}) + r_j & \text{if } j \in N_P, \\ r(L_b^{'}) & \text{otherwise.} \end{cases} \tag{39}$$

$$O(L_b) = \begin{cases} O(L_b^{'}) \setminus \{j\} & \text{if } j \in N_P, \\ O(L_b^{'}) \cup \{j-n\} & \text{if } j \in N_D. \end{cases} \tag{40}$$

$$U(L_b) = \begin{cases} U(L_b^{'}) \cup \{j-n\} & \text{if } j \in N_D, \\ U(L_b^{'}) & \text{otherwise.} \end{cases} \tag{41}$$

Dominance of the backward algorithm can be constructed in the same way as in the case of the forward algorithm, because the arrival time functions are nondecreasing and linear stepwise as before. Furthermore, similar to forward algorithm, in Proposition 3, we denote $\widetilde{U}(L_b)$ as the set of requests of which either the delivery has already been visited or the pickup or delivery are unreachable from $v(L_b)$. Since travel times do not necessarily satisfy the triangle inequality, we define $t_l = \{\max\{t : t + \tau_{jv(L_b)}(t) = t(L_b)\}\}$. Any pickup and delivery node j with $a_j + s_j > t_l$ will be unreachable from the partial path $p(L_b)$.

We define $\phi(L_b^1, L_b^2)$ (see figure 5) as

$$\phi(L_b^1, L_b^2) = \max\{x \in R : \delta_{L_b^1}(t) + x \leq \delta_{L_b^2}(t), \quad \forall t \in dom(L_b^2)\} \tag{42}$$

Proposition 3. *Label L_b^2 is dominated by label L_b^1 if*

1. $v(L_b^1) = v(L_b^2)$
2. $U(L_b^1) \subseteq \widetilde{U}(L_b^2)$
3. $O(L_b^1) = O(L_b^2)$
4. $t(L_b^1) \geq t(L_b^2)$
5. $r(L_b^1) \geq r(L_b^2) - \phi(L_b^1, L_b^2)$

The proof of proposition 3 is given in the appendix.

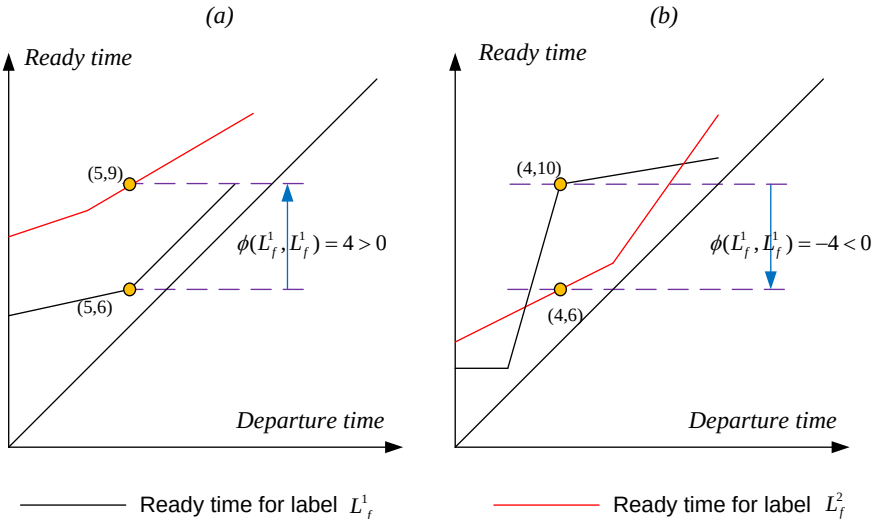

Figure 5 **Illustration of $\phi(L_f^1, L_f^2)$.**

4.3. Merging forward and backward labels

When all forward and backward labels are generated, they are merged to construct feasible profitable tours. A forward label L_f and a backward label L_b can be merged if the following conditions are satisfied:

1. $v(L_f) = v(L_b)$
2. $O(L_f) \cap O(L_b) = \{v(L_f)\}$
3. $U(L_f) = U(L_b)$
4. $Img(L_f) \cap dom(L_b) \neq \emptyset$

The resulting path $p(L) = (p(L_f) \bigoplus p(L_b))$ has the following attributes:

1. $v(L) = 2n + 1$
2. $r(L) = r(L_f) + r(L_b) - r_{v(L_f)}$
3. $O(L) = \emptyset$
4. $U(L) = U(L_f)$
5. $\delta_L(t) = \delta_{L_b}(\delta_{L_f}(t)), \forall t \in dom(L_f)$ such that $\delta_{L_f}(t) \in dom(L_b)$

However, this bidirectional labeling algorithm can generate duplicate solutions. Consider a feasible solution p^* including nodes i, j and k in this order. Each node $x \in p^*$ associated with a forward label $L_f(x)$ and a backward label $L_b(x)(v(L_f(x)) = v(L_b(x) = x)$. Therefore, the path p^* can be obtained by merging $L_f(i)$ with $L_b(i)$ as well as merging $L_f(j)$ with $L_b(j)$. To overcome this drawback, we devised an additional test: we accept a solution only when an further extension of the forward label is impossible. In our example, assume that the extension from $L_f(i)$ to node

j is feasible and extention from $L_f(j)$ to node k is infeasible, because of the predefined fixed time t_m. We generate solution p^* by merging $L_f(j)$ and $L_b(j)$ instead of $L_f(i)$ and $L_b(i)$. The test is performed constantly for each candidate pair of Labels, since this extention feasibility is detected by directly comparing fixed time with the sum of earliest departure time and travel time. Moreover, this test guarantees that each path is generated only once.

5. Restricted dynamic programming heuristic algorithm

In order to avoid the exponential computation time and to save limited memory storage, Malandraki and Dial (1996) proposed a restricted dynamic programming heuristic algorithm for the TSP. The main concept is to limit the number of partial paths for further extension in every stage by a given parameter B. Moreover, in each stage all partial paths have the same number of visited nodes. Therefore, partial paths with low costs are more likely to become part of the solution than one with higher costs.

Malandraki and Dial (1996) show that increasing the value of B results in better solutions, but also in substantial higher computation times. Note that the setting $B = 1$ results in a nearest neighborhood heuristic and setting $B = \infty$ makes it an exact dynamic programming algorithm.

Gromicho et al. (2012) propose a restricted dynamic programming algorithm for solving realistic VRPs and restrict the state space even further, by using a form of beam search (Bisiani. (1987)), which means that each partial path is only expanded to a number of its nearest feasible nodes by a given parameter E.

The same principle with some minor changes can also be applied to the TLTWPD for the TDPPDTSPTW. To restrict the state space for the TDPPDTSPTW, we also select the best B partial paths in each stage. Expanding a partial path to a pickup node may improve or deteriorate the objective function depending on the profits and traveling cost of visiting that node. However, expanding to a delivery node can only increase the traveling cost. Therefore, for every expansion of a partial path the $E/2$ best expanded partial paths ended in pickup node and the $E/2$ best expanded partial paths ended in delivery node are selected. Furthermore, in each stage we select the best $B/2$ partial paths that expand to a pickup node and the best $B/2$ partial paths that expand to a delivery node for further expansion.

Note that there is no optimality guarantee anymore with such a restricted dynamic programming heuristic algorithm.

6. Computational Experiments

The algorithms are coded in JAVA and all computations are carried out on a server with four CPU's (2.4 GHz/6 cores) and 32 GB RAM with time limit of 120 hours (5 days). For our numerical

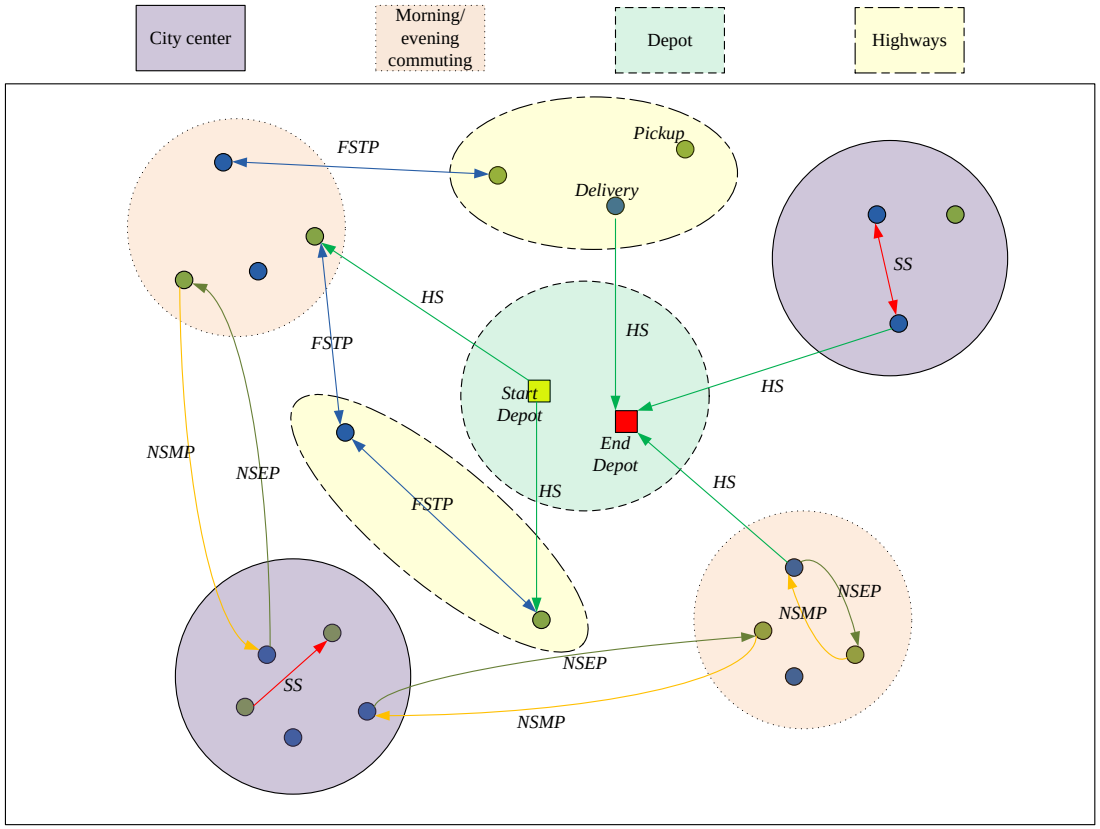

Figure 6 **An Illustration of a TDPPDTSPTW Instance with Different Speed Profiles.**

study, we created new data sets based on the ones available for related problems (Ropke et al. (2009), Dumitrescu et al. (2010) and Verbeeck et al. (2014)).

In all instances, a profit varying form 40 to 100 is randomly assigned to each pickup node of the request, and the coordinates of each pickup and delivery location are randomly selected from the instances used by Ropke et al. (2009), that follow a uniform distribution over the $[0, 50]^2$ square. Furthermore, the planning horizon covers 14 working hours (840 minutes, from 7 am to 9 pm) and a minute is set to be one unit of time. Furthermore, in each instance, the time windows of all the pickup and delivery nodes have the same width and there are 3 types of widths considered in our data sets (i.e., 30 minutes, 60 minutes and 90 minutes). The instances follow a naming convention of *prob-nx-T* in which n denotes the number of requests to be served which varies from 8 to 20. For each number of requests five instances are generated . The x denotes one of the letters a, b, c, d and e that is used to distinguish between instances with the same size. At last T represents the type of time windows that are used: 1 for instances with 30 minutes time windows, 2 for instances with 60 minutes time windows and 3 for instances with 90 minutes time windows.

Road congestion is handled by a so-called speed model which consists of different speed profiles. It is used to determine the travel time between two nodes on a specific departure time. This speed

model for the TDPPDTSPTW is based on the speed model of Verbeeck et al. (2014) for the TDOP and Dabia et al. (2013) for the TDVRPTW. Furthermore, without loss of generality, we assume that breakpoints are the same for all speed profiles as congestion tends to happen around the same time regardless of the speed profiles' type. Moreover, the links are not randomly assigned to a speed profile during the construction of the problem instance. More specifically, the pickup and delivery of each request is randomly assigned to one of the three perdefined areas: morning and evening commuting area, city center and highways. Then, the speed profile is assigned according to the type of the tail node and head node of each link (e.g. depot or request node) and areas of those two nodes located.

In this speed model each speed profile has four non-overlapping time periods with constant speed, reflecting two congested periods and two periods with normal traffic conditions. Five speed profiles are included (see Figure 6 and Table 1):

- Slow speed (SS): these links represent a busy central business district (CBD) with a lot of traffic during the whole day.
- Normal speed with morning peak (NSMP): these links represent roads leading from a residential area to the CBD. These roads are in most cases congested in the morning.
- Normal speed with evening peak (NSEP): these links represent roads leading from a CBD to a residential zone. The roads typically encounter evening congestion.
- Fast speed with two peaks (FSTP): these links represent roads near the highway with a morning and evening peak in both directions.
- High speed (HS): these links represent the links connected to the request nodes and the depot.

Table 1 Speed Profiles

Congestion description	Morning peak	Normal	Evening peak	Normal
Time periods	7 am-9 am	9 am-5 pm	5 pm-7 pm	7 pm-9 pm
1.SS	0.5	0.81	0.5	0.81
2.NSMP	0.67	1.33	0.88	1.33
3.NSEP	0.88	1.33	0.67	1.33
4.FSTP	0.85	1.5	0.85	1.5
5.HS	1.0	2.0	1.0	2.0

6.1. TLTWPD without dominance versus TLTWPD with dominance

In Table 2 and Table 3, we illustrate the strength of the TLTWPD with dominance over the labeling algorithm without dominance. In Table 2, it shows that a significant number of undesired labels are removed by using our dominance rules (columns 7-11). The comparison of the processing time of the TLTWPD with dominance and its corresponding algorithm without dominance is described

in Table 3. We report the optimal solution in row "Optimal value". Moreover, we report the time (in seconds) needed to solve an instance for both versions of the algorithm (rows 2-3). Clearly, in some cases (see instance Prob 8a-2), the performance of the TLTWPD with dominance is far better than the version without dominance. In other cases (see instances Prob 8a-3 and Prob 8e-3), only the TLTWPD with dominance can solve it to optimality within the available system memory. Note that "-" means the instances can not be solved because the memory of the system is exceeded during the procedure.

Table 2 The Effect of Dominance (on Number of Generated Labels)

	Forward TLTWPD without Dominance					Forward TLTWPD with Dominance				
Last node $v(L_f)$ visited on the Partial Path	Prob 8a-2	Prob 8b-2	Prob 8c-2	Prob 8d-2	Prob 8e-2	Prob 8a-2	Prob 8b-2	Prob 8c-2	Prob 8d-2	Prob 8e-2
0	1	1	1	1	1	1	1	1	1	1
1	8	324	21384	207	10374	8	200	2637	123	869
2	13200	9	48	6	546	1909	9	27	6	101
3	36	36	648	3933	31122	26	27	102	959	2527
4	14509	3	48	2	8	2043	3	34	2	8
5	748	3564	1944	15732	546	331	1662	291	2869	131
6	1	1	1	207	3458	1	1	1	134	292
7	2	2376	39	13	14	2	1158	27	13	14
8	12	108	7128	1311	17	12	67	879	338	14
9	24	432	21384	414	31122	18	243	2637	193	2502
10	37444	12	183	21	1092	2187	11	59	21	200
11	48	36	1296	5244	41496	35	27	188	956	3283
12	37818	9	171	2	42	2046	9	53	2	32
13	704	7128	2592	15732	1092	215	2131	290	2869	198
14	462	1	1	414	3458	216	1	1	184	292
15	22583	4752	156	24	39	2349	2074	55	24	27
16	528	216	7128	3933	61	155	126	879	948	35
17	9660	840	3320	2160	6588	281	113	349	196	348
Average	7655	1103	3320	3748	7282	658	437	473	547	604

Table 3 The Effect of Dominance (on Processing Time)

	Instance									
Bi-directional TLTWPD	Prob 8a-2	Prob 8b-2	Prob 8c-2	Prob 8d-2	Prob 8e-2	Prob 8a-3	Prob 8b-3	Prob 8c-3	Prob 8d-3	Prob 8e-3
Optimal value	145.63	125.18	197.64	67.59	222.99	165.01	138.08	227.64	92.48	245.74
CPU time(s) without dominance	6.33	0.81	1.63	0.39	3.74	-	1.75	3.17	1.28	-
CPU time(s) with dominance	0.67	0.27	1.14	0.3	1.09	2.9	0.48	1.15	0.74	3.1

6.2. Bidirectional versus Monodirectional TLTWPD algorithm

In Table 4, we report the gains of using a bidirection search (column 5) over the monodirectional search (column 3-4). Since all algorithms lead to the optimal value it is presented only once (column 2). However, the bidirectional search show greater potential power in terms of computation time to solve the instances. The monodirectional version is hardly able to solve instances with more than 9 requests and wide time windows within three minutes. This is mainly because of the fact

that the number of labels that need to be processed in the bidirectional version is considerably less compared to the monodirectional TLTWPD algorithm.

Table 4 Monodirectional Algorithm vs. Bidirectional Algorithm

Instance	Optimal Value	Processing Time (s)		
		Monodirectional Forward	Monodirectional Backward	Bidirectional
Prob8a-1	120.72	3.83	5.34	0.47
Prob8b-1	76.41	1.12	0.8	0.21
Prob8c-1	134.18	2.51	1.96	0.46
Prob8d-1	30.78	0.9	1.43	0.18
Prob8e-1	192.84	5.05	3.02	0.7
Prob8a-2	145.63	8.36	8.89	0.67
Prob8b-2	125.18	5.26	4.86	0.27
Prob8c-2	197.64	4.44	3.47	1.14
Prob8d-2	67.59	2.22	1.1	0.3
Prob8e-2	222.99	6.57	3.44	1.09
Prob8a-3	165	65.45	46.97	2.9
Prob8b-3	138.08	13.2	6.48	0.48
Prob8c-3	227.64	5.3	4.64	1.15
Prob8d-3	92.48	6.85	3.23	0.74
Prob8e-3	245.74	21.81	121.2	3.1
Prob9a-1	47.53	0.73	0.67	0.28
Prob9b-1	129.82	11.21	7.01	0.68
Prob9c-1	212.41	59.3	14.69	1.09
Prob9d-1	125.34	14.78	16.14	0.33
Prob9e-1	124.22	59.89	34.56	0.56
Prob9a-2	92.44	7.54	13	2.25
Prob9b-2	159.82	115.92	117.56	3.33
Prob9c-2	276.76	319.32	159.42	4.9
Prob9d-2	137.98	92.13	73.13	0.68
Prob9e-2	124.22	254.17	109.36	0.79
Prob9a-3	152.26	316.32	467.78	79.64
Prob9b-3	189.82	191.46	252	5.89
Prob9c-3	299.16	587.99	824.32	24.74
Prob9d-3	151.51	164.56	196.07	1.14
Prob9e-3	139.59	571.54	538.02	2.08

6.3. Performance of the bidirectional TLTWPD algorithm

We now evaluate the performance of the bidirectional TLTWPD algorithm. Each instance with 10 requests is solved once with the proposed TLTWPD algorithm and once with MIP model described in section 3 by using Gurobi 5.6.1 with its default settings. Table 5 indicates that Gurobi is able to find the optimal solution, but in a substantially larger amount of time than the TLTWPD algorithm. The average CPU time required by Gurobi is approximately 165.04 seconds where the same statistic for the Bidirectional TLTWPD to produce the reported solutions is approximately 12.06 seconds.

Table 5 Computational Results for the Instances with 10 Requests

Instance	Optimal Value	Gurobi CPU seconds	TLTWPD CPU seconds
Prob10a-1	55.28	8.77	0.98
Prob10b-1	134.78	7.75	3.37
Prob10c-1	222.69	26.82	4.07
Prob10d-1	186.75	9.98	0.56
Prob10e-1	143.49	10.02	1.34
Prob10a-2	117	9.05	1.39
Prob10b-2	214.28	8.78	7.21
Prob10c-2	310.25	55.8	36.21
Prob10d-2	191.81	48.42	1.73
Prob10e-2	173.49	35.88	2.02
Prob10a-3	163.92	8.52	8.19
Prob10b-3	244.28	1583.47	39.70
Prob10c-3	329.98	568.62	67.14
Prob10d-3	202.31	68.61	2.74
Prob10e-3	203.49	25.17	4.32

The results reported in Table 6 show that the algorithm is able to solve most of the instances with up to 12 requests within 15 minutes. The wider the time windows the more time it takes to obtain the optimal solutions, since wide time windows add more complexity to the proposed problem by having a larger flexibility.

Table 6 Computational Results for the Instances with 11 and 12 Requests

Instance	Optimal Value	Time(s)	Instance	optimal value	Time(s)	Instance	optimal value	Time(s)
Prob11a-1	76.73	1.16	Prob11a-2	133.07	3.58	Prob11a-3	177.3	13.6
Prob11b-1	192.53	7.21	Prob11b-2	278.66	20.22	Prob11b-3	364.99	85.57
Prob11c-1	195.87	4.35	Prob11c-2	299.23	105.54	Prob11c-3	361.3	848.64
Prob11d-1	192	2.28	Prob11d-2	250.26	12.2	Prob11d-3	305.88	36.33
Prob11e-1	134.19	2.65	Prob11e-2	164.19	3.4	Prob11e-3	224.19	10.2
Prob12a-1	72.29	1.65	Prob12a-2	123.39	3.11	Prob12a-3	155.94	28.46
Prob12b-1	236.46	9.1	Prob12b-2	341.69	55.09	Prob12b-3	394.03	174.74
Prob12c-1	221.33	5.84	Prob12c-2	349.11	265.29	Prob12c-3	378.95	3778.36
Prob12d-1	194.45	5.82	Prob12d-2	269.93	25.74	Prob12d-3	288.5	99.27
Prob12e-1	196.85	4.67	Prob12e-2	226.85	5.34	Prob12e-3	256.85	16.96

In Table 7, most instances with up to 20 requests could be solved as well. Due to the memory limit, some instances in this table cannot be solved with the current algorithm, especially the instances with medium time windows. As can be seen the computation times can increase to several days.

Table 7 Instances with 13, 14, 15, 16, 17, 18, 19 and 20 Requests

Instance	Optimal Value	Time(s)	Instance	optimal value	Time(s)
Prob13a-1	117.72	2.12	Prob13a-2	185.81	5.77
Prob13b-1	284.06	77.33	Prob13b-2	386.06	345.32
Prob13c-1	239.48	56.6	Prob13c-2	400.94	6829.21
Prob13d-1	244.65	3.71	Prob13d-2	309.76	39.67
Prob13e-1	194.33	4.63	Prob13e-2	224.33	7.22
Prob14a-1	118.59	2.17	Prob14a-2	187.33	10.92
Prob14b-1	236.46	407.86	Prob14b-2	388.09	579.7
Prob14c-1	245.74	258.16	Prob14c-2	449.36	362031.71
Prob14d-1	205.13	3.6	Prob14d-2	310.92	81
Prob14e-1	253.99	3.73	Prob14e-2	303.99	30.73
Prob15a-1	160.88	3.06	Prob15a-2	190.88	47.24
Prob15b-1	230.86	550.44	Prob15b-2	386.71	314.46
Prob15c-1	327.09	1112.3	Prob15c-2	430.39	2d5h
Prob15d-1	238.02	14.49	Prob15d-2	383.5	616.27
Prob15e-1	262.14	21.86	Prob15e-2	360.49	1763.32
Prob16a-1	181.12	16.24	Prob16a-2	263.53	1094.15
Prob16b-1	347.78	8926.75	Prob16b-2	526.44	1d6h
Prob16c-1	388.76	2206.04	Prob16c-2	569.35	3d5h
Prob16d-1	268.83	1128.9	Prob16d-2	429.68	14644.43
Prob16e-1	373.70	23.36	Prob16e-2	459.70	170.62
Prob17a-1	147.06	21.86	Prob17a-2	262.41	507.1
Prob17b-1	414.25	11213.97	Prob17b-2	-	-
Prob17c-1	342.71	2483.01	Prob17c-2	-	-
Prob17d-1	366.06	467.38	Prob17d-2	500.66	27402.66
Prob17e-1	430.95	8989.86	Prob17e-2	504.28	815.2
Prob18a-1	224.86	395.09	Prob18a-2	300.61	1700.65
Prob18b-1	356.25	1d15h	Prob18b-2	-	-
Prob18c-1	416.54	3477.76	Prob18c-2	-	-
Prob18d-1	319.49	3358.58	Prob18d-2	538.98	31824.23
Prob18e-1	483.7	214.54	Prob18e-2	513.70	2168.91
Prob19a-1	273.52	450.31	Prob19a-2	338.61	49570.31
Prob19b-1	411.56	4903.78	Prob19b-2	-	-
Prob19c-1	410.4	18616.45	Prob19c-2	-	-
Prob19d-1	405.91	577.91	Prob19d-2	610.26	33924.18
Prob19e-1	501.88	438.87	Prob19e-2	541.09	10725.64
Prob20a-1	285.16	2316.52	Prob20a-2	438.67	41946.24
Prob20b-1	-	-	Prob20b-2	-	-
Prob20c-1	413.22	11038.24	Prob20c-2	-	-
Prob20d-1	467.49	3776.55	Prob20d-2	-	-
Prob20e-1	501.88	869.77	Prob20e-2	649.88	81791.4

6.4. Performance of the restricted dynamic programming heuristic algorithm

We also have conducted computational experiments to analyze the solution quality produced by restricted dynamic programming heuristic algorithm. In Table 7 we have seen that the exact procedure has enormous computation times and problems with the memory restrictions. In Table 8 we solve the same instances but now with the restricted dynamic programming heuristic. The restricted dynamic programming heuristic algorithm is run with different values for B.

Considering only the 60 instances for which we know the optimal solution, it turns out that with a value of $B = 1000$ the algorithm leads in on average 24 seconds to the optimal solution for 45 (75%) of the 60 instances. With a value of $B = 5000$ this increases considerably to 93% of the instances, but the average computation time is now tenfold (245s). Increasing the value of B to 10000 ensures that 97% of the instances are solved to optimality in on average 504 seconds.

Table 8 Impact of Different B Values on Instances in Table 7

Instance	Optimal Value	B=1000			B=5000			B= 10000		
		Value	Time(s)	% gap	Value	Time(s)	% gap	Value	Time(s)	% gap
Prob14a-1	118.59	118.59	0.39	0	118.59	0.58	0	118.59	2.22	0
Prob14b-1	236.46	236.46	6.29	0	236.46	1.45	0	236.46	12.67	0
Prob14c-1	245.74	245.74	9	0	245.74	16.46	0	245.74	20.62	0
Prob14d-1	205.13	205.13	2.96	0	205.13	5.6	0	205.13	4.51	0
Prob14e-1	253.99	253.99	4.6	0	253.99	2.29	0	253.99	3.09	0
Prob14a-2	187.33	187.33	2.31	0	187.33	7.69	0	187.33	4.65	0
Prob14b-2	388.09	388.09	9.61	0	388.09	27.01	0	388.09	106.22	0
Prob14c-2	449.36	428.5	7.4	4.64	433.11	104.33	3.62	449.36	247.46	0
Prob14d-2	310.92	310.92	16.6	0	310.92	26.52	0	310.92	20.72	0
Prob14e-2	303.99	303.99	6.8	0	303.99	17.46	0	303.99	5.12	0
Prob15a-1	160.88	160.88	4.03	0	160.88	2.39	0	160.88	2.32	0
Prob15b-1	230.86	230.86	3.3	0	230.86	3.86	0	230.86	11.5	0
Prob15c-1	327.09	327.09	3.91	0	327.09	10.26	0	430.38	567.7	0
Prob15d-1	238.02	238.02	4.73	0	238.02	7.62	0	238.02	6.21	0
Prob15e-1	262.14	262.14	7.73	0	262.14	6.35	0	262.14	5.43	0
Prob15a-2	190.88	190.88	8.24	0	190.88	2.5	0	190.88	8.72	0
Prob15b-2	386.71	386.71	10.06	0	386.71	43.31	0	386.71	68.04	0
Prob15c-2	430.39	398.47	20.99	7.42	430.39	183.58	0	430.39	583.58	0
Prob15d-2	383.5	383.5	11.76	0	383.5	24.04	0	383.5	55.9	0
Prob15e-2	360.49	360.49	16.49	0	360.49	16.07	0	360.49	17.22	0
Prob16a-1	181.13	181.13	7.16	0	181.13	5.26	0	181.13	4.36	0
Prob16b-1	347.78	347.78	22.79	0	347.78	103.73	0	347.78	355.52	0
Prob16c-1	388.76	388.76	13.65	0	338.76	39.67	0	388.76	256.46	0
Prob16d-1	268.83	268.83	12.62	0	268.83	8.39	0	268.83	54.16	0
Prob16e-1	373.3	373.7	6.07	0	373.3	10.81	0	373.3	9.25	0
Prob16a-2	263.53	263.53	17.52	0	263.53	16.49	0	263.53	77.1	0
Prob16b-2	526.44	475.79	38.25	9.62	526.44	231.04	0	526.44	1034.59	0
Prob16c-2	569.35	520.41	13.5	8.6	565.01	336.06	0.76	567.93	819.13	0.25
Prob16d-2	429.88	429.68	16.08	0	429.68	161.07	0	429.68	262.52	0
Prob16e-2	459.7	459.7	9.88	0	459.7	83.96	0	459.7	45.08	0
Prob17a-1	147.06	147.06	5.77	0	147.06	6.76	0	147.06	5.66	0
Prob17b-1	414.25	414.25	33.87	0	414.25	538.03	0	414.25	1471.2	0
Prob17c-1	342.71	339.73	9.42	0.87	342.71	341.49	0	342.71	257.73	0
Prob17d-1	366.06	366.06	19.88	0	366.06	91.56	0	366.06	46.93	0
Prob17e-1	430.95	430.95	35.46	0	430.95	12.41	0	430.95	22.28	0
Prob17a-2	262.41	262.41	27.13	0	262.41	119.08	0	262.41	85.58	0
Prob17b-2	-	553.17	55.73	-	563.15	320.39	-	596.99	802.6	-
Prob17c-2	-	478.06	15.05	-	569.31	571.69	-	571.14	1146.89	-
Prob17d-2	500.66	493.24	23.03	1.48	500.66	299.23	0	500.66	700.54	0
Prob17e-2	504.28	504.28	14.27	0	504.28	168.7	0	504.28	238.02	0
Prob18a-1	224.86	224.86	7.52	0	224.86	15.20	0	224.86	8.39	0
Prob18b-1	356.25	259.13	27.41	27.26	356.25	570.05	0	356.25	1595.13	0
Prob18c-1	416.54	391.9	15.01	5.92	416.54	330.65	0	416.54	282.04	0
Prob18d-1	319.49	319.49	26.74	0	319.49	119.55	0	319.49	57.24	0
Prob18e-1	483.7	483.7	2.87	0	483.7	137.81	0	483.7	84.68	0
Prob18a-2	300.61	300.61	50.22	0	300.61	187.77	0	300.61	240.6	0
Prob18b-2	-	553.77	48.1	-	553.77	624.94	-	606.56	231.65	-
Prob18c-2	-	555.3	55.41	-	555.3	323.55	-	555.3	1035.26	-
Prob18d-2	538.98	517.61	32.2	0	538.98	312.07	0	538.98	475.17	0
Prob18e-2	513.7	513.7	38.36	0	483.7	210.43	0	513.7	438.61	0
Prob19a-1	273.52	273.52	10.09	0	273.52	27.25	0	273.52	31.11	0
Prob19b-1	411.56	267.56	23.59	34.99	411.56	499.79	0	411.56	1055.17	0
Prob19c-1	410.4	386.03	25.98	5.94	410.4	193.25	0	410.4	489.21	0
Prob19d-1	405.91	349.56	18.27	13.88	405.91	191.64	0	405.91	82.87	0
Prob19e-1	501.88	501.88	33.79	0	501.88	300.1	0	501.88	302.36	0
Prob19a-2	338.61	338.61	38.4	0	338.61	281.31	0	338.61	1074.03	0
Prob19b-2	-	561.04	57.88	-	563.56	584.45	-	592.09	1456.65	-
Prob19c-2	-	561.83	31.36	-	594.55	459.97	-	594.55	1101.69	-
Prob19d-2	610.26	564.01	44.24	7.58	575.79	547.33	5.65	610.26	1177.36	0
Prob19e-2	541.09	541.09	45.19	0	541.09	497.62	0	541.09	610.57	0
Prob20a-1	285.16	285.16	21.06	0	285.16	39	0	285.16	57.4	0
Prob20b-1	-	269.38	33.07	-	468.83	184.61	-	468.83	1709.1	-
Prob20c-1	413.22	381.96	17.18	7.56	413.22	468.99	0	413.22	334.41	0
Prob20d-1	467.49	459.49	23.81	1.71	467.49	321.94	0	467.49	740.09	0
Prob20e-1	501.88	501.88	43.37	0	501.88	1154.24	0	501.88	1902.5	0
Prob20a-2	438.67	438.67	99.84	0	438.67	601.08	0	438.67	1090.78	0
Prob20b-2	-	561.38	37.61	-	594.2	1020.78	-	594.2	1939.41	-
Prob20c-2	-	594.98	90.34	-	629.09	983.19	-	682.83	1020.53	-
Prob20d-2	-	642.82	91.93	-	645.45	898.51	-	645.61	3184.15	-
Prob20e-2	649.88	599.09	31.83	7.82	646.09	1403.8	0.58	646.2	2034.67	0.57
Average			23.96	2.08		245.05	0.15		504.18	0.01

The two instances which were not solved to optimality had respectively a gap of 0.25 and 0.57. This means that, compare to the TLTWPD, the average gap of restricted dynamic programming heuristic algorithm over the 60 instances is just 0.01%.

Note that all the instances which the exact TLTWPD algorithm was not able to solve (e.g. $Prob19b-2$ in Table 7) can be solved within one hour with this restricted dynamic programming algorithm.

Secondly, in Table 9, we presents the impact of different E values on the performances of the algorithm with a fixed value for $B = 5000$. We set the values of E to n, $0.25n$, $0.125n$ and fractional values are rounded to its closed integer. The results indicate that the computation times decrease if the value for E decreases. As can be seen in Table 9 the solution quality is almost maintained if the value of E is set equal to $0.25n$, except for instance $Prb19a-1$. Moreover, the solution of $Prb19d-2$ is even improved. Remember from Table 8 that this instance was not solved to optimality with $B = 5000$ and $E = n$.

When we further reduce E to $0.125n$, the computation times are about 33% of the original computation times with $E = n$. However, the solution quality is deteriorated in most cases. It is also worth to mention that the solution quality of two instances are improved. That is because smaller values for E allow only label expansions to near neighbors, which sometimes can escape the local optimum by including "unpromising" labels for further iterations.

Table 9 Impact of Different E Values on Instances with 19 and 20 Requests when B=5000

	E=n		E=0.25n			E=0.125n		
Instance	Value	Time(s)	Value	Time(s)	% gap	Value	Time(s)	% gap
Prob19a-1	273.52	27.25	242.5	116.52	11.34	242.5	27.67	11.34
Prob19b-1	411.56	499.79	411.56	102.75	0	411.56	50.18	0
Prob19c-1	410.4	193.25	410.4	132.25	0	360.4	36.52	12.18
Prob19d-1	405.91	191.64	405.91	114.92	0	405.91	50.2	0
Prob19e-1	501.88	300.1	501.88	216.36	0	501.88	61.19	0
Prob19a-2	338.61	281.31	338.61	175.06	0	334.61	40.11	1.18
Prob19b-2	592.09	584.45	592.09	424.15	0	548.59	118.58	7.35
Prob19c-2	594.55	459.97	594.55	333.79	0	574.93	318.25	3.3
Prob19d-2	605.31	547.33	610.26	470.95	-0.82	610.26	332.95	-0.82
Prob19e-2	541.09	497.62	541.09	413.32	0	531.88	176.05	1.7
Prob20a-1	285.16	39	285.16	29.96	0	275.22	27.63	3.49
Prob20b-1	468.83	184.61	468.83	122.98	0	468.83	36.99	0
Prob20c-1	413.22	468.99	413.22	313.76	0	413.22	36.26	0
Prob20d-1	467.49	321.94	467.49	181.31	0	467.49	64.54	0
Prob20e-1	501.88	262.46	501.88	122.93	0	457.88	79.36	8.77
Prob20a-2	438.67	601.08	438.67	338.77	0	438.67	64.11	0
Prob20b-2	594.2	1020.78	594.2	468.5	0	582.63	167.84	1.95
Prob20c-2	629.09	983.19	629.09	572.74	0	623.52	262.62	0.89
Prob20d-2	645.61	898.51	645.61	676.6	0	662.95	289.64	-2.69
Prob20e-2	649.88	1403.8	649.88	623.82	0	593.88	237.47	8.62
Average		488.35		297.57	0.53		123.91	2.86

7. Conclusion

This paper presents the time-dependent profitable pickup and delivery traveling salesman problem with time windows, which combines the characteristics of the traveling salesman problem with pickup and delivery with the traveling salesman problem with profits. Moreover, the time-varying traveling speed is considered to capture the real world problem of road congestion.

We proposed a tailored labeling algorithm with time windows and pickup and delivery for solving the TDPPDTSPTW. Considering time-dependent travel times increases the complexity of the problem. New and strong dominance rules are presented to reduce the number of labels. Computational results show that most instances with up to 20 requests can be solved to optimality within the given memory limit, but also that several instances with only 14 requests remain unsolved.

To reduce the computation time and memory usage a restricted dynamic programming heuristic algorithm is implemented. This heuristic is able to find solutions for all instances with good qualities (on average 0.01% gap for instances of which we know the optimal solution) and reasonable computational time (on average 504 seconds).

Obviously, the performance of our exact algorithm critically depends on the tailored dominance criterion that we generated. It shows great potential when there are significant profits varieties among all the requests and relatively tight time windows attached to all the vertices. Finding more efficient dominance criteria is important in future research.

In addition, solving realistic extensions of the TDPPDTSPTW also seems an attractive research direction. More specifically, the time-dependent variant of the pickup and delivery problem with time windows (TDPDPTW) and the time-dependent team orienteering problem (TDTOP) are very interesting as it aims to optimize the routes of a fleet of vehicles, instead of one vehicle only. When column generation is utilized to solve those two problems in an exact way, our proposed problem can be seen as the pricing problem.

Acknowledgments

The authors gratefully acknowledge funds provided by China Scholarship Council(CSC).

Appendix

PROOF OF PROPOSITION 4.1. Consider two labels L_f^1 and L_f^2 that satisfy the six conditions in proposition 4.1. We need to show that (1) any feasible extention of L_f^2 to destination depot $2n+1$ is also a feasible extention of L_f^1 and (2) for any feasible extention L of L_f^2, the value of $obj(L_f^1 \bigoplus L)$ is larger than or equal to the value of $obj(L_f^2 \bigoplus L)$.

Regarding the first point, let L be a feasible extention of L_f^2. If such a path does not exist then obviously one can remove label L_f^2. We have that (a) the path $p(L_f^1 \bigoplus L)$ is elementary because of condition 2 and

3, (b) $p(L_f^1 \bigoplus L)$ is not violating any time windows if the origin depot 0 is left at any time $t \in dom(L_f^2)$ because of condition 4, 5 and FIFO assumption.

To show the second point, consider a feasible path p extending $p(L_f^2)$ to $2n+1$ and let $L_f^{2*} = L_f^2 \bigoplus L$ and $t_0 = arg\min_{t \in Dom(L_f^{2*})}\{\delta_{L_f^{2*}}(t) - t\}$. It is the optimal departure time form the depot given the path $p(L_f^{2*})$. The reduced cost of the path is

$$obj(L_f^{2*}) = r(L_f^{2*}) - (\delta_{L_f^{2*}}(t_0) - t_0) = (r(L_f^2) + r(v(L))) - (\delta_{L_f^{2*}}(t_0) - t_0) \tag{43}$$

Here $r(L)$ is the sum of the profits associated with the requests nodes visited along path p. Now consider the same path p to extend label L_f^1. Departing origin depot 0 at time t_0 is feasible for path $p(L_f^{1*})$ (using condition 4 as well as condition 5 and FIFO assumption) and induces an upper bound on the reduced cost of L_f^{1*}:

$$obj(L_f^{1*}) \geq r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0) - t_0) \tag{44}$$

Condition 5 implies that $\delta_{L_f^1}(t_0) \leq \delta_{L_f^2}(t_0)$, and using the FIFO property this means that $\delta_{L_f^{1*}}(t_0) \leq \delta_{L_f^{2*}}(t_0)$. We also have that $r(L_f^{1*}) = r(L_f^1) + r(L) = r(L_f^2) + r(L) = r(L_f^{2*})$ because of condition 2 and 3. In combination this means that

$$obj(L_f^{1*}) \geq r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0) - t_0) \geq r(L_f^{2*}) - (\delta_{L_f^{2*}}(t_0) - t_0) = obj(L_f^{2*}) \tag{45}$$

□

PROOF OF PROPOSITION 4.3. Similar to PROPOSITION 4.2, and by using conditions 2, 3 and 4, we can prove that any feasible extention to L_b^2 is also feasible for L_b^1

To show that the objective value of any path $p(L_b^1 \bigoplus L)$ is larger than or equal to the objective value of path $p(L_b^2 \bigoplus L)$ for any feasible extention L of L_b^2, we let $L_b^{2*} = L_b^2 \bigoplus L$, and $t^* = arg\min_{t \in dom(L_b^{2*})}\{\delta_{(L_b^{2*})}(t) - t\}$, that is, the optimal departure time from the depot given the path corresponding to L_b^{2*}. The objective value of the path is

$$obj(L_b^{2*}) = r(L_b^{2*}) - (\delta_{L_b^{2*}}(t^*) - t^*) = r(L_f^2) + r(L) - (\delta_{L_f^{2*}}(t^*) - t^*) \tag{46}$$

Here $r(L)$ is the sum of the profits associated with the nodes visited along the extention L. Now consider the path $p(L_b^{1*}) = p(L_b^1 \bigoplus L)$, it depart from the origin depot at time t^* along path $p(L_b^*)$ and reach the node $v(L_b^2)$ at the same time as path $p(L_b^{2*})$, but may arrive to node $2n+1$ earlier or later. Therefore, departure time t^* at the depot will provide an upper bound on the objective value of path $p(L_b^{1*})$. That is

$$obj(L_f^{1*}) = r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0^1) - t_0^1) \tag{47}$$

Consequently:

$$\begin{aligned}
&r(L_b^{1*}) - (\delta_{L_b^{1*}}(t^*) - t^*) \\
&\geq (r(L_b^1) + r(L)) - (\delta_{L_b^{2*}}(t^*) - t^* - \phi(L_b^1, L_b^2)\}) \\
&\geq (r(L_b^2) - \phi(L_b^1, L_b^2) + r(L)) - (\delta_{L_b^{2*}}(t^*) - t^* - \phi(L_b^1, L_b^2)) \\
&= (r(L_b^{2*}) + r(L)) - (\delta_{L_f^{2*}}(t^*) - t^*) = obj(L_b^{2*})
\end{aligned} \tag{48}$$

Therefore, we have that $obj(L_f^{1*}) \geq obj(L_f^{2*})$. Note that the first inequality uses the definition of $\phi(L_b^1, L_b^2)$ and the second inequality uses condition 5 of Proposition 4.3. □

References

Balas, E. 1989. The prize-collecting traveling salesman problem. *Networks* **19**(6) 621–636.

Bisiani., R. 1987. Beam search. S. Shapiro, ed., *Encyclopedia of Artificial Intelligence*. Wiley and Sons, 56–58.

Dabia, S., S. Ropke, T. van Woensel, T de Kok. 2013. Branch and cut and price for the time dependent vehicle routing problem with time windows. *Transportation Science* **47**(3) 380–396.

Dell' Amico, M., F. Maffioli, P. Varbrand. 1995. On prize-collecting tours and the asymmetric traveling salesman problem. *International Transactions in Operational Research* **2**(3) 297–308.

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

Dumitrescu, I., S. Ropke, J.F. Cordeau. 2010. The traveling salesman problem with pickup and delivery: polyhedral results and branch-and-cut algorithm. *Mathematic Programming Series A* **121**(2) 269–305.

Feillet, D., P. Dejax, M. Gendreau. 2005. Traveling salesman problems with profits. *Transportation Science* **39**(2) 188–205.

Fomin, F., A. Lingas. 2002. Approximation algorithms for time-dependent orienteering. *Information Processing Letters* **83**(2) 57 – 62.

Gromicho, J.A.S., J.J. van Hoorn, A.L. Kok, J.M.J. Schutten. 2012. Restricted dynamic programming: A flexible framework for solving realistic VRPs. *Computers and Operations Research* **39**(5) 902–909.

Ibaraki, T., S. Imahori, K. Nonobe, K. Sobue, T. Uno, M. Yagiura. 2008. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics* **156**(11) 2050–2069.

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

Laporte, G., S. Martello. 1990. The selective traveling salesman problem. *Discrete Applied Mathematics* **26**(2-3) 193–207.

Li, B., D. Krushinsky, H.A. Reijers, T. van Woensel. 2014. Offline share-a-ride problem: Taxi sharing between passenger and package. *European Journal of Operational Research* **238**(1) 31–40.

Li, J. 2011. Model and algorithm for time-dependent team orienteering problem. in S.Lin and X.Huang(Eds). *Advanced research on computer education. Simulation and modeling, communications in computer and information science* **175** 1–7.

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

Malandraki, C., R.B. Dial. 1996. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research* **90**(1) 45–55.

Righini, G., M. Salani. 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization* **3**(3) 255–273.

Ropke, S., J.-F. Cordeau, G. Laporte. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* **43**(3) 267–286.

Ruland, K. S. 1994. Polyhedral solution to the pickup and delivery problem. *Ph.D. Thesis* Sever Institute, Washington University in St.Louis.

Ruland, K.S., E.Y. Rodin. 1997. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers Mathematics with Applications* **33**(12) 1–13.

Sigurd, M., D. Pisinger. 2004. Scheduling transportation of live animals to avoid the spread of diseases. *Transportation Science* **38**(2) 197–209.

Sol, M. 1994. Column generation techniques for pickup and delivery problems. *PhD thesis, Technische Universiteit Eindhoven* .

Vansteenwegen, P., W. Souffriau, D. Van Oudheusden. 2011. The orienteering problem: a survey. *European Journal of Operational Reseach* **209**(1) 1–10.

Verbeeck, C., E. H. Aghezzaf, P. Vansteenwegen. 2014. A fast solution method for the time-dependent orienteering problem with time windows. *European Journal of Operational Research* **236**(2) 419–432.

van Woensel, T., L. Kerbache, H. Peremans, N. Vandaele. 2008. Vehicle routing with dynamic travel times: a queueing approach. *European Journal of Operational Research* **186**(3) 990–1007.