Optimization Letters (2020) 14:579–609
https://doi.org/10.1007/s11590-018-1342-y

ORIGINAL PAPER

Exact and heuristic dynamic programming algorithms for the traveling salesman problem with flexible time windows

Ramon Faganello Fachini[1] · Vinícius Amaral Armentano[1]

Received: 6 October 2017 / Accepted: 15 October 2018 / Published online: 22 October 2018

Abstract
This article proposes extensions of exact and heuristic dynamic programming algorithms for the traveling salesman problem with *flexible* time windows, which are a limited enlargement of the generally referred to as *hard* time windows. The service of a customer can be started before or after the hard time window at a penalty cost. The addressed problem thus requires the determination of a sequence of customers and their respective service start times in order to minimize the sum of traveling cost with earliness and lateness cost. Computational tests are conducted on a variety of symmetric and asymmetric instances proposed in the literature, and the advantages of flexible windows are stressed.

Keywords Traveling salesman problem · Flexible time windows · Dynamic programming · Labeling algorithm

1 Introduction

The aim of the traveling salesman problem with time windows (TSPTW) is to find a minimum cost route visiting a set of customers, where each customer must be visited only once under the *hard* constraint that the service starts within a given time window. This implies that the salesman has to wait if he arrives before the start of the time window. The problem is NP-hard because it encompasses the traveling salesman

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11590-018-1342-y) contains supplementary material, which is available to authorized users.

✉ Ramon Faganello Fachini
ramonfachini@gmail.com

Vinícius Amaral Armentano
vinicius@densis.fee.unicamp.br

1 Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, SP 13083-852, Brazil

problem, and furthermore, Savelsbergh [30] has shown that the problem of finding a feasible solution for TSPTW is NP-complete.

The TSPTW has applications in the job shop scheduling where the setup time of each job depends on the previous job [5], in the single machine discrete lot-sizing and scheduling problem with sequence dependent setup costs [19], and in the approach cluster-first, route-second for the vehicle routing problem with time windows (VRPTW) [24]. In addition, the feasibility problem of the TSPTW becomes a separation subproblem when a branch-and-cut approach is used to solve the VRPTW [23]. The literature on exact algorithms for the TSPTW is extensive and for a thorough review on this topic, see [6].

When the hard time window constraints may be violated then the time windows become *soft* constraints, as suggested by Sexton and Choi [32] for the single vehicle pickup and delivery problem, and the deviations of starting the service before or after the time windows imply a penalty cost function that is added to the total vehicle traveling time. In real world situations, a soft time window corresponds to a limited increase of the hard time window which, with the magnitude of the penalty cost parameter, reflects the importance of a customer being served closer to the time window.

The relaxation of the hard time window constraints leads to a greater feasible space and, as a consequence, we can have an optimal cost lower than the optimal cost associated with the hard constrained time windows. Let $p_i(s_i)$ denote the penalty cost function, where s_i is the service start time outside the time window $[e_i, l_i]$ of customer i. Several variants of a linear penalty cost $p_i(s_i) = \alpha_i(e_i - s_i),\ \ s_i < e_i$ and $p_i(s_i) = \beta_i(s_i - l_i),\ \ s_i > l_i$, where α_i and β_i are positive real parameters that have been proposed in the literature of the VRPTW, are mentioned in the sequel.

Koskosidis et al. [24] and Liberatore et al. [26] suggest an unlimited starting time service, whereas Balakrishnan [2] proposes a maximum waiting time $W_{\max}$ and a maximum penalty cost $P_{\max}$. Chiang and Russel [9] allow a maximum time window enlargement M, i.e., $[e_i - M, l_i + M]$ and a maximum waiting time $W_{\max}$. In [33], earlier starting $s_i < e_i$ is not allowed, but unlimited lateness $s_i > l_i$ is permitted. Qureshi et al. [27] suggest a *semi soft* penalty cost such that the upper limit of the time window is extended to $l_i' > l_i$, the penalty cost is linear in the interval $\left[l_i, l_i'\right]$, and service start must occur in the interval $\left[e_i, l_i'\right]$. The limit l_i' is defined as the maximum late arrival penalty equivalent to the time and cost of an additional single vehicle route only serving customer i. Bhusiri et al. [8] include the maximum limit of late arrival time in the definition of l_i' and propose a lower limit of the soft window e_i', expressed as the maximum early arrival penalty equivalent to the cost of an additional single vehicle route only serving customer i. The penalty cost is linear in the intervals $\left[e_i', e_i\right]$ and $\left[l_i, l_i'\right]$, and service start must occur in the interval $\left[e_i, l_i'\right]$. The authors claim that waiting time to start service incurs extra costs such as labour operating cost, maintenance cost and parking fee, in addition to the loss of opportunity to generate more profits. For this reason, the objective is to minimize the total waiting time as well as the routing cost that includes the penalty cost. This is accomplished by setting the service start time as the arrival time at each customer.

Ibaraki et al. [22] propose a comprehensive approach to the vehicle routing problem with hard and soft time windows for the case in which the problem is decoupled into two subproblems, as suggested by Sexton and Bodin [31], Sexton and Choi [32] and

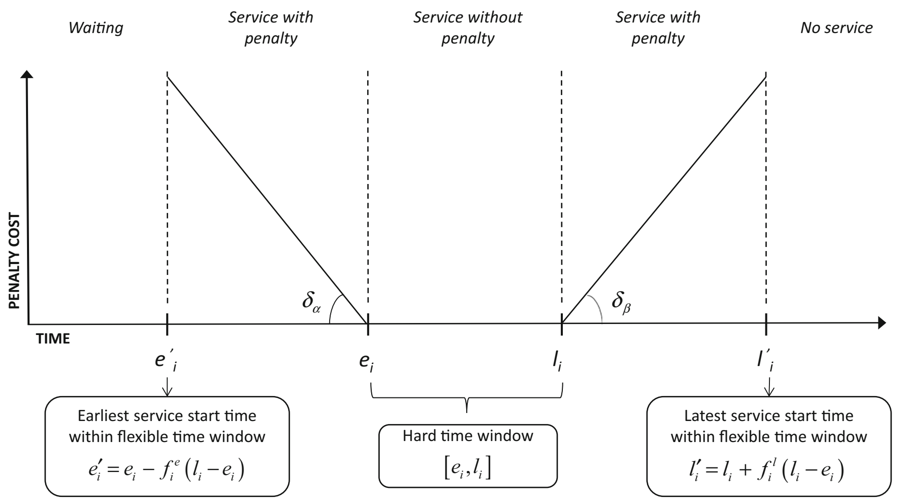


Fig. 1 Flexible time windows

Dumas et al. [15]. The former consists of determining the routes and the latter determines the optimal service start time of each customer in order to minimize the penalized total cost for violating the hard time windows. The penalty function for each customer can be non-convex provided that it is piecewise linear and lower semicontinuous, i.e., $p_i(t) \leq \lim_{\varepsilon \to 0} \min\{p_i(t+\varepsilon),\ p_i(t-\varepsilon)\}$.

The term *flexible* time window has been suggested by Taş et al. [34] and consists of an enlargement of the hard time window of customer i by associated fractions f_i^e and f_i^l such that the flexible time window $\left[e'_i,\ l'_i\right]$ has limits expressed by $e'_i = e_i - f_i^e(l_i - e_i)$ and $l'_i = l_i + f_i^l(l_i - e_i)$. The service at customer i cannot start outside the flexible window, but can start in the intervals $\left[e'_i,\ e_i\right]$ and $\left[l_i,\ l'_i\right]$ with linear penalty cost $p_i(s_i) = \delta_\alpha(e_i - s_i),\ \ e'_i \leq s_i < e_i$ and $p_i(s_i) = \delta_\beta(s_i - l_i),\ \ l_i < s_i \leq l'_i$, where δ_α and δ_β are positive real parameters (see Fig. 1). Feasible vehicle routes are constructed by a tabu search algorithm and the optimal service start of each customer is determined by a linear programming (LP) formulation in order to minimize the total penalty cost of each route for a given sequence of customers in that route. Vidal et al. [35] stress that most timing problems, for example, vehicle routing and machine scheduling can be formulated by means of a LP formulation, as suggested by the pioneer articles proposed by Sexton and Bodin [31], Sexton and Choi [32], Dumas et al. [15] for vehicle routing, and Fry and Leong [20] for machine scheduling.

In this article, we propose extensions of exact and heuristic dynamic programming algorithms for the traveling salesman problem with flexible time windows (TSPFlexTW). The objective is to determine a sequence of customers and their respective service start times so as to minimize the sum of traveling costs with earliness and lateness costs. The sets of states or labels generated by the proposed dynamic programming algorithms are explored according to label-correcting strategies. Regarding exact algorithms, we extend the bi-directional resource-bounded dynamic programming proposed by Li [25], and with reference to heuristic dynamic programming, we extend the algorithm with precedence constraints devised by Balas and Simonetti [4].

The main contributions of this article are threefold:

1. The proposition of label resources, resource extension functions and dominance criteria to be used in label-correcting algorithms for routing and scheduling problems with flexible time window constraints.
2. The design of exact and heuristic dynamic programming algorithms for the TSPFlexTW, which extend solution methods originally developed for the TSPTW.
3. The extensive computational experimentation on a variety of symmetric and asymmetric instances from the literature and the assessment of the benefits gained by flexible time windows compared to the hard time windows.

The remainder of the article is organized as follows. Section 2 introduces the problem description. Sections 3 and 4 describe the extension of exact and heuristic dynamic programming algorithms for the TSPFlexTW. Computational results are reported in Sect. 5, and conclusions are outlined in Sect. 6.

2 Problem description

The TSPFlexTW is defined on a complete graph $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{0, 1, \ldots, n, n+1\}$ is the set of nodes and $\mathcal{A} = \{(i, j) : i \in \mathcal{N} \backslash \{n+1\},\ j \in \mathcal{N} \backslash \{0\},\ i \neq j\}$ represents their connecting arcs. Let $\mathcal{C} = \{1, \ldots, n\}$ be the set of customers, while nodes 0 and $n+1$ denote the depot as route starting and destination points, respectively. For each customer $i \in \mathcal{C}$, there is a service time W_i, a hard time window $[e_i, l_i]$ and its fractions f_i^e and f_i^l for the time window enlargement. The time windows $[e_0, l_0]$ and $[e_{n+1}, l_{n+1}]$ at the depot represent the planning horizon. Following Taş et al. [34], each flexible time window $\left[e_i', l_i'\right]$ has limits expressed by $e_i' = \max\left\{e_i - f_i^e(l_i - e_i), 0\right\}$ and $l_i' = l_i + f_i^l(l_i - e_i)$. Servicing a customer within $\left[e_i', e_i\right]$ is penalized by δ_α for one unit of earliness, while servicing a customer within $\left[l_i, l_i'\right]$ is penalized by δ_β for one unit of lateness. Waiting up to e_i is allowed at no penalty cost, but customers cannot be served after l_i'. Moreover, each arc $(i, j) \in \mathcal{A}$ has an associated traveling cost c_{ij} and a traveling time t_{ij}. If $c_{ij} = c_{ji}$ and $t_{ij} = t_{ji}$ the TSPFlexTW is symmetric, otherwise is asymmetric.

Consider the following variables:

- w_i: service start time at node $i \in \mathcal{N}$;
- α_i: service earliness at node $i \in \mathcal{N}$;
- β_i: service lateness at node $i \in \mathcal{N}$;

$$x_{ij} = \begin{cases} 1 \text{ if node } j \in \mathcal{N} \text{ is visited immediately after visiting node } i \in \mathcal{N}; \\ 0 \text{ otherwise.} \end{cases}$$

In the following, the TSPFlexTW is formulated as a mixed integer programming (MIP) model, a special case of the vehicle routing problem with flexible time windows (VRPFlexTW) proposed by Taş et al. [34].

$$\min \sum_{i\in \mathcal{N}\setminus\{n+1\}} \sum_{j\in \mathcal{N}\setminus\{0\},\,(i,j)\in\mathcal{A}} c_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \left(\delta_\alpha \alpha_i + \delta_\beta \beta_i\right) \tag{1}$$

$$\sum_{i\in \mathcal{N}\setminus\{n+1\}} x_{ij} = 1 \;\; j \in \mathcal{N}\setminus\{0\},\; j \neq i \tag{2}$$

$$\sum_{j\in \mathcal{N}\setminus\{0\}} x_{ij} = 1 \;\; i \in \mathcal{N}\setminus\{n+1\},\; i \neq j \tag{3}$$

$$w_i + W_i + t_{ij} - [M_{ij}(1 - x_{ij})] \leq w_j \;\; i \in \mathcal{N}\setminus\{n+1\},\; j \in \mathcal{N}\setminus\{0\},\; i \neq j \tag{4}$$

$$e'_i \leq w_i \leq l'_i \quad i \in \mathcal{N} \tag{5}$$

$$\alpha_i \geq e_i - w_i \quad i \in \mathcal{N} \tag{6}$$

$$\beta_i \geq w_i - l_i \quad i \in \mathcal{N} \tag{7}$$

$$(w_i, \alpha_i, \beta_i) \geq 0,\; x_{ij} \in \{0,1\}\; i, j \in \mathcal{N},\; i \neq j \tag{8}$$

where $M_{ij} = \max\left\{l'_i - e'_j + t_{ij} + W_i,\ 0\right\}$ (see [13]). The objective function (1) minimizes the overall cost, which includes traveling cost and service penalty. Constraints (2) and (3) state that every node must be visited exactly once. Constraints (4) guarantee that the service start time at a customer must be greater or equal to the sum of the service start time with the service time and the traveling time of its immediate predecessor. Constraints (5) ensure that the service start times are within the given flexible time windows, while constraints (6) and (7) connect the node service start times with the service earliness and service lateness, respectively. Constraints (8) indicate the domain of the variables.

3 Exact dynamic programming algorithms for the TSPFlexTW

The proposed dynamic programming algorithms are based on the algorithm conceived by Desrochers [12] for the resource constrained shortest path problem and extended to its elementary version by Feillet et al. [16]. Such algorithms include resources to the Ford-Bellman algorithm (see [1] for an introduction to labeling algorithms).

An essential feature of dynamic programming is to structure an optimization problem in *stages* corresponding to optimization subproblems that are solved sequentially. Associated with each stage are the *states* that convey the relevant information of the prior history in order to make future decisions [7, 10].

In constrained shortest path problems, stages are associated with the nodes of the graph and each state corresponds to a feasible path from the depot 0 to a node i with characteristics of a particular problem. The states of a node i are represented by labels (R, C, i), where each component of the vector R denotes the consumption of a different resource and C is the cost along the path. Another fundamental component of dynamic programming is a recursive equation or a resource extension function associated with each arc of the graph and each resource. In addition, dominance rules are applied in order to fathom dominated states.

In this section, we extend the bi-directional resource-bounded dynamic programming devised by Li [25] to two variants of dynamic programming algorithms for the TSPFlexTW.

3.1 Bi-directional resource-bounded dynamic programming for the TSPTW

Li [25] suggests the dynamic programming technique with a label-correcting algorithm for the TSPTW that takes advantage of the bi-directional bounded search framework. Let node $o = 0$ denote the depot as the route starting point and node $d = n+1$ refer to the depot as the route destination point. In a bi-directional search, labels are extended both forward from node o to its successors and backward from node d to its predecessors. Such a label extension may result in a smaller number of non-dominated labels than in mono-directional dynamic programming [29]. The entire route is obtained by joining paths from the forward and backward extensions. Hereafter, we define the components of the aforementioned algorithm.

3.1.1 Label extension

Let $L_i^{FP} = (V, s, i)$ denote a forward label, where i is the last reached node of the forward path $FP = (0, \ldots, i)$, s represents *the earliest time* when service can start at node i, and V is a vector that indicates the node sequence of $FP = (0, \ldots, i)$. Let V_i be the position that node i is visited, then if node i is the kth visited node, $V_i = k$. The accumulated traveling cost of $L_i^{FP} = (V, s, i)$ is represented by $c\big(L_i^{FP}\big)$. Let V^{IF} denote the vector associated with the initial forward (IF) label, which is defined as $\big(V^{IF}, 0, 0\big)$ with $V_0^{IF} = 1,\ V_h^{IF} = 0,\ \ h \in \mathcal{C} \cup \{n+1\}$ and $c\big(V^{IF}, 0, 0\big) = 0$. The forward extension of a label (V, s, i) to node j along arc (i, j) produces the new label $\big(V', s', j\big)$ with cost $c\big(V', s', j\big)$ by means of the following resource extension functions: $V_h' = V_h,\ \ \text{if}\ h \neq j,\ V_h' = V_i + 1,\ \ \text{if}\ h = j$, as node j is the successor of node i; the earliest time when service can start, i.e., $s' = \max\big\{s + W_i + t_{ij},\ \ e_j\big\}$, since there is a waiting time if node j is reached before the start of corresponding time window e_j; the accumulated travel cost at node j is increased by c_{ij}, i.e., $c\big(V', s', j\big) = c(V, s, i) + c_{ij}$.

The backward label extension is a mirror of the forward label extension. A backward path is represented by $BP = (n+1, \ldots, i)$ with respective label $L_i^{BP} = (V, s, i)$ and cost $c\big(L_i^{BP}\big)$. Let T be the maximum feasible arrival time at the destination node $n+1$, i.e., $T = \max_{h \in \mathcal{C}}\big\{l_h + W_h + t_{h,\ n+1}\big\}$ [29]. The initial label of the backward extension is $\big(V^{IB}, T, n+1\big)$ with $V_{n+1}^{IB} = 1,\ V_h^{IB} = 0,\ \ h \in \mathcal{C} \cup \{0\}$ and $c\big(V^{IB}, T, n+1\big) = 0$. In this case, V^{IB} denotes the vector associated with the initial backward (IB) label. Consider that label (V, s, i) is backward extended to $\big(V', s', j\big)$, where the computation of V' is identical to the forward extension as shown above, while $s' = \min\big\{s - W_j - t_{ji},\ \ l_j\big\}$ since s' is *the latest time* when service can start at node j, and $c\big(V', s', j\big) = c(V, s, i) + c_{ji}$.

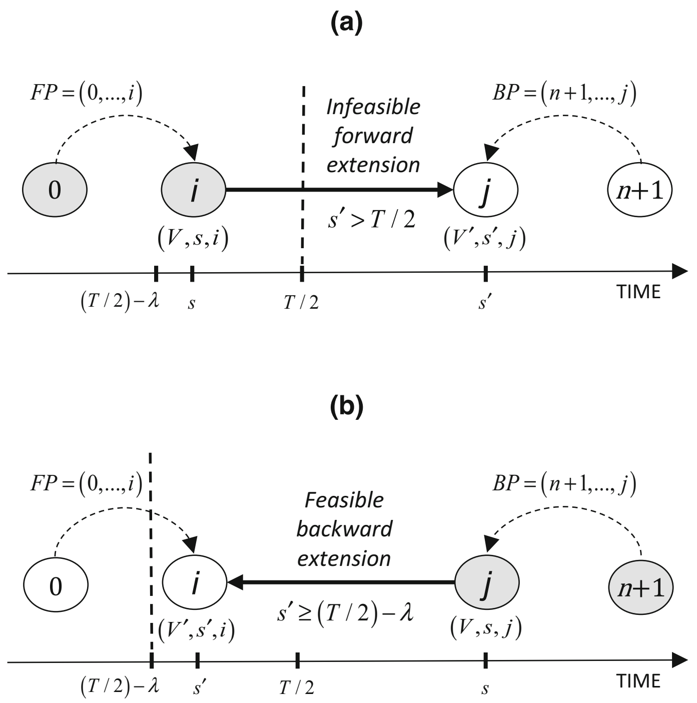


Fig. 2 Time resource bounding scheme. **a** Infeasible forward extension, **b** feasible backward extension

Righini and Salani [29] suggest that the forward and the backward extensions are allowed only if $s' \leq T/2$ and $s' \geq T/2$, respectively. A *half-way point* that minimizes the *half-way function* $\left|s'_{FP} - s'_{BP}\right|$ among *neighbours* of a pair of nodes i and j is then chosen to join their associated labels. In this way, each path from node $o = 0$ to node $d = n + 1$ is generated only once.

Li [25] addresses the case where a forward label at node i can be directly extended to reach a node j or otherwise that a backward label at node j can be extended to reach a node i. The author argues that the conditions $s' \leq T/2$ and $s' \geq T/2$ do not guarantee the previous extensions, and thus a feasible and possibly optimal path from node $o = 0$ to node $d = n + 1$ is lost.

Let $\lambda = \max_{h \in \mathcal{N}\setminus\{n+1\}, \ell \in \mathcal{N}\setminus\{0\}, h \neq \ell}\{W_h + t_{h\ell}\}$ denote the maximum arc traversal time in the graph G. Figure 2 illustrates that a feasible solution is not lost if the forward extension from (V, s, i) to $\left(V', s', j\right)$ is allowed only if the earliest time $s' \leq T/2$, and the backward extension from (V, s, j) to $\left(V', s', i\right)$, only if the latest time $s' \geq (T/2) - \lambda$ [25]. Note that if condition $s' \geq T/2$ were applied to the latest time instead of $s' \geq (T/2) - \lambda$, this would disallow the backward extension depicted in Fig. 2b, therefore, the feasible $o - d$ path would be lost.

The forward label extension is feasible if (1) node j has not previously been visited, i.e., $V_j = 0$, (2) s' is no later than the upper limit of time window $\left[e_j, l_j\right]$, i.e., $s' \leq l_j$, (3) a look-ahead step, in which the new label $\left(V', s', j\right)$ can be further propagated to visit all non-visited nodes while satisfying their time windows, and (4) $s' \leq T/2$. The same feasibility conditions (1) and (3) on the forward extension apply to the backward procedure, whereas conditions (2) and (4) are replaced by $s' \geq e_j$ and $s' \geq (T/2) - \lambda$, respectively.

3.1.2 Label dominance

Dominance tests are performed during forward and backward label extensions. Define the set $U = \{h | V_h \neq 0, h \in \mathcal{N}\}$ as the nodes visited by a label (V, s, i). Regarding *forward dominance*, let $\left(V^1, s^1, i\right)$ and $\left(V^2, s^2, i\right)$ be two labels with the same reached node i and identical sets $U^1 = U^2$. Label $\left(V^1, s^1, i\right)$ dominates $\left(V^2, s^2, i\right)$ if

$$c\left(V^1, s^1, i\right) \leq c\left(V^2, s^2, i\right) \tag{9}$$

$$s^1 \leq s^2 \tag{10}$$

and at least one of these inequalities is strict. With respect to *backward dominance*, the dominance rule (10) is replaced by

$$s^1 \geq s^2 \tag{11}$$

If the triangular inequality holds for the arc traveling costs c_{ij} and times t_{ij}, the condition $U^1 = U^2$ can be replaced by $U^2 \subseteq U^1$ for both forward and backward extensions [25].

3.1.3 Label joining

Forward and backward labels are joined to yield TSPTW feasible routes. Let $L_i^{FP} = \left(V^F, s^F, i\right)$ be a forward label and $L_i^{BP} = \left(V^B, s^B, i\right)$ be a backward label with the same reached node i. Their concatenation $L_i^{FP} \oplus L_i^{BP}$ consists of a feasible route if all nodes are visited only once, and if the starting time at the forward label is less than or equal to the starting time of the backward label, i.e., $s^F \leq s^B$. The cost of the resulting route is $c\left(L_i^{FP}\right) + c\left(L_i^{BP}\right)$.

Figure 3 depicts a pseudo-code of the algorithm. Let $ForL$, $BackL$ and $ActL$ be linked lists that store the forward labels, the backward labels and the active label pool, respectively. The algorithm is started by placing the first forward label and the first backward label into the active label pool $ActL$ (lines 1–3). Nodes as well as forward and backward labels are then repeatedly treated until $ActL$ becomes empty (lines 4–19). Herein, $Extend^F((V, s, i), j)$ (line 8) and $Dominance^F\left(ForL, \left(V', s', j\right)\right)$ (line 9) represent the label extension and label dominance procedures in the forward extension. Analogously, $Extend^B((V, s, i), j)$ (line 13) and $Dominance^B\left(BackL, \left(V', s', j\right)\right)$ (line 14) are applied to the backward extension. Finally, for each node $i \in \mathcal{N}$ the label joining is performed by the function $Join(i)$ (lines 20–22) and the minimum cost route is obtained (line 23).

Algorithm Bi-directional resource-bounded dynamic programming

```
   // Initialization //
1  $ForL \leftarrow \{(V^{IF}, 0, 0)\}$, $BackL \leftarrow \{(V^{IB}, T, n+1)\}$
2  $ActL \leftarrow ForL \cup BackL$
3  $ForL \leftarrow \emptyset$, $BackL \leftarrow \emptyset$
4  While $ActL \neq \emptyset$ do
5      Remove label $(V, s, i)$ from the head of $ActL$
6      If $(V, s, i)$ is a forward label then
           // Forward extension //
7          For all outgoing nodes $j$ of $i$ do
8              $(V', s', j) \leftarrow Extend^{F}((V, s, i), j)$
               // Forward dominance checking //
9              $ForL \leftarrow Dominance^{F}(ForL, (V', s', j))$
10         End For
11     Else
           // Backward extension //
12         For all incoming nodes $j$ of $i$ do
13             $(V', s', j) \leftarrow Extend^{B}((V, s, i), j)$
               // Backward dominance checking //
14             $BackL \leftarrow Dominance^{B}(BackL, (V', s', j))$
15         End For
16     End If
17     $ActL \leftarrow ForL \cup BackL$
18     $ForL \leftarrow \emptyset$, $BackL \leftarrow \emptyset$
19 End While
   // Label joining //
20 For all nodes $i$ from 0 to $n+1$ do
21     $Join(i)$
22 End For
23 Output the TSPTW minimum cost route
```

Fig. 3 Bi-directional resource-bounded dynamic programming algorithm

3.2 Bi-directional resource-bounded dynamic programming algorithms for the TSPFlexTW

We suggest two alternative forms of label correcting that includes label representation and the respective procedures for label extension and dominance.

3.2.1 Label-correcting LP

Let the label $L_i^P = (V, s, i)$ denote a path both for a forward label and a backward label for the TSPFlexTW. It differs from the TSPTW with respect to the flexible time windows $\left[e_i', l_i'\right]$, $i \in \mathcal{N}$ that replace the hard time windows $[e_i, l_i]$, $i \in \mathcal{N}$ in all forward and backward calculations.

In addition to the accumulated traveling cost of the forward or backward label $c\left(L_i^P\right)$, there is a penalty cost corresponding to the minimum earliness and lateness cost $pen\left(L_i^P\right)$ associated with starting service within the flexible time windows but earlier or later than the limits of the hard time windows. Such a minimum cost is obtained by means of the LP model suggested by Taş et al. [34]:

$$pen\left(L_i^P\right) = \min \sum_{j \in \bar{\mathcal{N}}} \left(\delta_\alpha \alpha_j + \delta_\beta \beta_j\right) \tag{12}$$

$$w_j + W_j + t_{jh} \le w_h \; j,h \in \bar{\mathcal{N}},\; (j,h) \in \bar{\mathcal{A}} \tag{13}$$

$$e'_j \le w_j \le l'_j \; j \in \bar{\mathcal{N}} \tag{14}$$

$$\alpha_j \ge e_j - w_j \; j \in \bar{\mathcal{N}} \tag{15}$$

$$\beta_j \ge w_j - l_j \; j \in \bar{\mathcal{N}} \tag{16}$$

$$\left(w_j, \alpha_j, \beta_j\right) \ge 0 \; j \in \bar{\mathcal{N}} \tag{17}$$

where $\bar{\mathcal{N}} \subset \mathcal{N}$ is the set of nodes visited in the path P and $\bar{\mathcal{A}} \subset \mathcal{A}$ is the set of traversed arcs of this path. The objective function (12) expresses the minimization of penalty costs along the path P. Constraints (13) describe the compatibility between the service start time of two subsequent nodes in path P. Constraints (14) impose that the service start time at each node in path P is limited by a flexible time window. Constraints (15) connect the earliness with the service start time, while constraints (16) link the lateness and the service start time. Non-negativity constraints on variables are represented by (17). Before solving this linear program for a forward or a backward label $L_i^P = (V, s, i)$, the service start variable w_i regarding the last reached node i is fixed at the current value of its time resource, i.e., $w_i = s$.

Regarding forward dominance, let $\left(V^1, s^1, i\right)$ and $\left(V^2, s^2, i\right)$ be two labels with the same reached node i. Then the dominance criterion (9) is replaced by

$$c\left(V^1, s^1, i\right) + pen\left(V^1, s^1, i\right) \le c\left(V^2, s^2, i\right) + pen\left(V^2, s^2, i\right) \tag{18}$$

with the dominance criterion (10). The backward dominance criteria are given by (11) and (18).

3.2.2 Label-correcting EL

Additional resources of accumulated earliness (E) $early\left(L_i^P\right)$ and accumulated lateness (L) $late\left(L_i^P\right)$ are added to both forward and backward label L_i^P. They were inspired by the accumulated waiting time $wa\left(L_i^P\right)$ and the accumulated delay time $de\left(L_i^P\right)$ proposed in [8] for the vehicle routing problem with soft time windows constraints. The resulting label is denoted $L_i^P = (V, s, i, early, late)$ with accumulated traveling cost $c\left(L_i^P\right)$ and penalty cost $pen\left(L_i^P\right)$. The flexible time windows $\left[e'_i, l'_i\right]$, $i \in \mathcal{N}$ also replace the hard time windows $[e_i, l_i]$, $i \in \mathcal{N}$ in all label extensions.

The label extension for three first components of $(V, s, i, early, late)$ follows that of TSPTW presented in Sect. 3.1.1. Let $s\left(L_j^P\right)$ denote the service start time at node $j \in \mathcal{N}$ and assume the extension of L_i^P along arc $(i, j) \in \mathcal{A}$ towards node $j \in \mathcal{N}$. The accumulated earliness $early\left(L_i^P\right)$ and lateness $late\left(L_i^P\right)$ are extended along the forward and backward paths in the following way:

$$early\left(L_j^P\right) = \begin{cases} early\left(L_i^P\right) + \left(e_j - s\left(L_j^P\right)\right), & \text{if } s\left(L_j^P\right) < e_j \\ early\left(L_i^P\right), & \text{if } s\left(L_j^P\right) \geq e_j \end{cases} \tag{19}$$

$$late\left(L_j^P\right) = \begin{cases} late\left(L_i^P\right) + \left(s\left(L_j^P\right) - l_j\right), & \text{if } s\left(L_j^P\right) > l_j \\ late\left(L_i^P\right), & \text{if } s\left(L_j^P\right) \leq l_j \end{cases} \tag{20}$$

The forward label dominance is determined by inequalities (9), (10) and the additional resource inequalities

$$early\left(L_i^1\right) \leq early\left(L_i^2\right) \tag{21}$$

$$late\left(L_i^1\right) \leq late\left(L_i^2\right) \tag{22}$$

The backward dominance criteria are given by (9), (11), (21) and (22). In contrast to the label-correcting *LP*, the label-correcting *EL* temporarily disregards the value of penalty costs $pen\left(L_i^P\right)$ until the label joining procedure.

The two variants of the exact dynamic programming algorithm (EDPA) are denoted according to the label-correcting procedures *LP* and *EL*, i.e., EDPALP and EDPAEL.

3.2.3 Common label joining for label-correcting procedures

The last step of the label-correcting procedures *LP* and *EL* shares a common label joining procedure such that the overall costs of the concatenated paths $L_i^{FP} \oplus L_i^{BP}$ are given by $c\left(L_i^{FP}\right) + c\left(L_i^{BP}\right) + pen\left(L_i^{FP} \oplus L_i^{BP}\right)$, where $pen\left(L_i^{FP} \oplus L_i^{BP}\right)$ is determined by the solution of LP model (12)–(17). In order to avoid the time-consuming computation of such a linear program for every pair of forward and backward paths to be concatenated we proceed in the following way. Store the current minimum overall cost found so far and call it *best*. Then the computation of $pen\left(L_i^{FP} \oplus L_i^{BP}\right)$ for the next pair of paths is performed only if $c\left(L_i^{FP}\right) + c\left(L_i^{BP}\right) < best$ holds for such a concatenated route $L_i^{FP} \oplus L_i^{BP}$, otherwise it cannot improve $best$ since $pen\left(L_i^{FP} \oplus L_i^{BP}\right) \geq 0$.

4 Heuristic dynamic programming algorithms for the TSPFlexTW

Desaulniers et al. [11] propose a heuristic dynamic programming for the subproblem encountered when solving the VRPTW by column generation, which corresponds to an elementary shortest path problem with resource constraints. Two different heuristic strategies are suggested, the one that eliminates non-promising arcs in the set $\mathcal{A}$, and the other a modified dominance rule which is applied to eliminate a large number of labels.

In a first attempt to obtain effective heuristics for the TSPFlexTW, we implemented such heuristic strategies in the dynamic programming based on the label-correcting *LP* and the label-correcting *EL* without success. The arc elimination failed to significantly speed up the algorithms. On the other hand, the modified dominance rule often was too aggressive and fathomed all labels that could yield TSPFlexTW feasible solutions.

We then adopted an alternative approach by considering the dynamic programming for the traveling salesman problem with precedence constraints (TSPPC) presented by Balas and Simonetti [4]. If such constraints are absent, then the dynamic programming can be used as a heuristic that finds in linear time a local optimum over an exponential size neighbourhood. In the following sections, we first describe the approach for the TSPTW of Balas and Simonetti [4], and then extend it by including the label-correcting *LP* and the label-correcting *EL* for the TSPFlexTW.

4.1 General heuristic dynamic programming for the TSPTW

The TSPTW can be addressed with a dynamic programming algorithm originally proposed for the TSPPC. Let σ denote an initial ordering of the nodes $i \in \{0, \dots, n\}$ with depot 0 in the first position, and assume that k is a positive integer. The TSPPC aims to find the minimum cost permutation π of σ satisfying $\pi(0) = 1$ and $\pi(i) < \pi(j)$ for all pairs of nodes (i, j) such that $\sigma(i) + k \leq \sigma(j)$, where notations $\sigma(\,)$ and $\pi(\,)$ are used to describe the positions of nodes in σ and π, respectively. This means that aside from depot 0, which is fixed in the first position, any customer i that precedes a customer j by k or more positions in the initial ordering σ must also precedes customer j in any feasible permutation π. Figure 4 illustrates examples of feasible and infeasible permutations for $\sigma = (0, 5, 6, 7, 8, 9, 2, 3, 1, 4)$ and $k = 4$. A dynamic programming formulation has been proposed in [3] for the TSPPC, which finds the optimal solution in linear time with $n + 1$ (i.e., depot 0 plus n customers), but exponential time with the parameter k.

The above-mentioned property still holds when k is replaced by a parameter that depends on the node i, $k(i)$, $i \in \{0, \dots, n\}$. This fact enabled Balas and Simonetti [4] to translate the customer time windows into precedence requirements and suggest an efficient implementation of the Balas [3] dynamic programming for the TSPTW. Since the complexity of this algorithm is exponential with $k(i)$, $i \in \{0, \dots, n\}$, then computer space and/or time may restrict the values of these parameters in order to obtain a viable heuristic dynamic programming in linear time with $n + 1$.

The heuristic algorithm comprises three main stages. In the first stage, the TSPTW is reformulated as a TSPPC. In the second stage, an auxiliary graph G^* that bounds the complexity of the algorithm is constructed. In the third stage, the problem is solved by determining a shortest path in G^*. These stages are summarized as follows.

4.1.1 TSPTW–TSPPC reformulation

The heuristic dynamic programming starts by generating an initial ordering σ, in which $\sigma(0) = 1$ and the customers are sorted in non-decreasing order of their time window midpoints $(e_i + l_i)\big/2,\ i \in \mathcal{C}$.

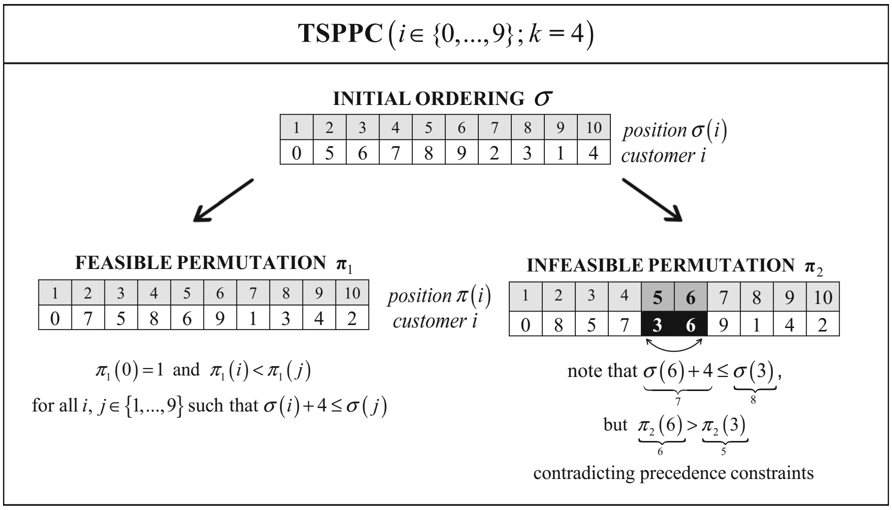

TSPPC $(i \in \{0,...,9\}; k = 4)$
INITIAL ORDERING σ
1 2 3 4 5 6 7 8 9 10 position $\sigma(i)$
0 5 6 7 8 9 2 3 1 4 customer i
FEASIBLE PERMUTATION π_1
1 2 3 4 5 6 7 8 9 10 position $\pi(i)$
0 7 5 8 6 9 1 3 4 2 customer i
$\pi_1(0)=1$ and $\pi_1(i)<\pi_1(j)$
for all $i, j \in \{1,...,9\}$ such that $\sigma(i)+4 \le \sigma(j)$
INFEASIBLE PERMUTATION π_2
1 2 3 4 5 6 7 8 9 10
0 8 5 7 3 6 9 1 4 2
note that $\underbrace{\sigma(6)}_{7}+4 \le \underbrace{\sigma(3)}_{8}$,
but $\underbrace{\pi_2(6)}_{6} > \underbrace{\pi_2(3)}_{5}$
contradicting precedence constraints

Fig. 4 Examples of TSPPC feasible and infeasible permutations

The ordering σ now can be used to derive precedence constraints. Node i placed in position $\sigma(i)$ has to precede node j whose position is $\sigma(j)$ in any feasible route such that $e_j + W_j + t_{ji} > l_i$. Hence, if j_0 is the smallest index such that $e_j + W_j + t_{ji} > l_i$ for all j with $\sigma(j) \geq j_0$, then $k(i)$ is defined as $k(i) = \min\{j_0 - \sigma(i), K\}$, where K is an upper bound on $k(i)$ used to avoid the assignment of impractical values to this parameter. Such a procedure is applied to every node in order to generate a TSPPC from the underlying TSPTW.

4.1.2 The auxiliary graph G*

The core of the heuristic algorithm here described consists of the construction of the auxiliary graph $G^* = (\mathcal{N}^*, \mathcal{A}^*)$ associated with the TSPPC earlier obtained. This graph contains a set Γ^* of $n + 2$ node layers, one layer for each position in the route, with the depot appearing at the beginning and at the end of the route, both as starting node $o = 0$, the only node in layer Γ^*_1, and the destination node $d = n + 1$, the only node in layer Γ^*_{n+2}. There is a one-to-one correspondence between TSPPC feasible routes and $o - d$ paths in G^*.

Let r be a counter assigning unique numbers to the nodes of a layer Γ^*_i, $i \neq 1$, $i \neq n+2$. Each node $(i, j, r) \in \Gamma^*_i$ represents a family of permutations π with depot 0 placed in position 1 and customer j placed in position i (i.e., $\pi(0) = 1$ and $\pi(j) = i$). Customer j must be one of those ranked between positions $\max\{2, i - K + 1\}$ and $\min\{n + 1, i + K - 1\}$ in the initial ordering σ. Moreover, there is a pair of sets

$$S^-_{(i,j,r)} = \{h : h \in \mathcal{C}, \sigma(h) \geq i, \pi(h) < i\}$$

and

$$S^{+}_{(i,j,r)} = \{h : h \in \mathcal{C}, \sigma(h) < i, \ \pi(h) \geq i\}$$

associated with each node $(i,\ j,\ r) \in \Gamma^*_i$. The former consists of customers ranked i or higher in σ but visited in a position lower than i in the permutations π considered by this node, and the latter comprises customers ranked below i in σ but visited in a position greater than or equal to i in such permutations. The pair $\left(S^{-}_{(i,j,r)},\ S^{+}_{(i,j,r)}\right)$ enables the identification of all customers visited before j in a given node $(i,\ j,\ r)$ and provides an efficient routine to ascertain the compatibility between nodes of consecutive layers. The arcs of G^* only join nodes of consecutive layers with verified compatibility, i.e., nodes that can be part of the same feasible solution. Figure 5 depicts the auxiliary graph G^* corresponding to a TSPPC in which $\sigma(i) = (0, 1, 2, 3, 4, 5, 6, 7)$ and $k(i) = 3$, $i \in \{0, \ldots, 7\}$. To elucidate the structure of the layers, nodes of two feasible $o - d$ paths are highlighted as well as their respective components.

Because each layer Γ^*_i, $i \neq 1$, $i \neq n+2$ shares a common structure of its node set and of the arc set $\mathcal{A}^* \cap \left(\Gamma^*_i,\ \Gamma^*_{i+1}\right)$, a copy is generated in advance and then retrieved for all $1 < \ i < n+2$ during the construction of G^*. The time complexity of finding the shortest path $o - d$ in this auxiliary graph is strongly related to the number of arcs $|\mathcal{A}^*|$. Since $\left|\Gamma^*_i\right| \leq (K+1)2^{K-2},\ \ 1 < \ i < n+2$, and no node of G^* has an indegree greater than K, the bound on the number of arcs of the auxiliary graph is given by $|\mathcal{A}^*| \leq K(K+1)2^{K-2}(n+1)$. Despite the inherent intricacy of building G^*, detailed guidelines are presented in [3] and [4].

4.1.3 Shortest path computation

Once G^* has been built, the TSPTW solution is obtained by solving a shortest path problem with time windows (SPPTW) defined on this auxiliary graph. The referred SPPTW can be computed by a simplified label-correcting method whose components are detailed next.

The algorithm for the SPPTW consists of a mono-directional dynamic programming in which a label is denoted by $L^P_{i,r} = (s,\ i,\ r)$ with accumulated traveling cost $c(s,\ i,\ r)$. Indexes i and r assign such a label to the rth node of layer Γ^*_i, i.e., $(i,\ j,\ r) \in \Gamma^*_i$. On the other hand, resource s stands for *the earliest time* when service can start at customer j associated with node $(i,\ j,\ r) \in \Gamma^*_i$.

The initial label $(0, 1, 1)$ has cost $c(0, 1, 1) = 0$ and is associated with the node $(1, 0, 1) \in \Gamma^*_1$. The *forward extension* of a label $(s,\ i,\ r)$ along arc $\mathcal{A}^* \cap \left((i,\ j,\ r) \in \Gamma^*_i,\ (i+1,\ h,\ \ell) \in \Gamma^*_{i+1}\right)$ towards node $(i+1,\ h,\ \ell)$ produces the new label $\left(s',\ i+1,\ \ell\right)$ with cost $c\left(s',\ i+1,\ \ell\right)$ by means of the extensions

$$s' = \max\left\{s + W_j + t_{jh},\ e_h\right\} \tag{23}$$

$$c\left(s', i+1, \ell\right) = c(s, i, r) + c_{jh} \tag{24}$$

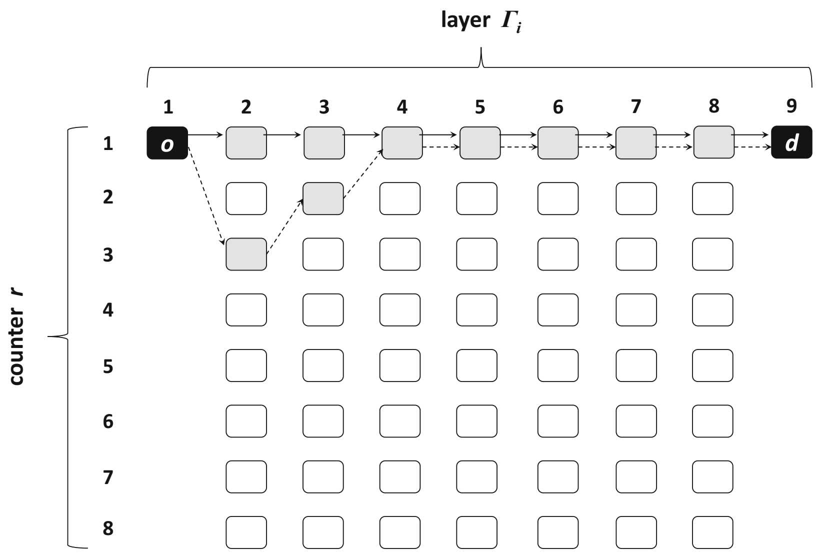
which are feasible only if

$$s' \leq l_h \tag{25}$$

TSPPC $\left(i\in\{0,\ldots,7\};\,k(i)=3, i\in\{0,\ldots,7\}\right)$

INITIAL ORDERING σ

1	2	3	4	5	6	7	8	*position* $\sigma(i)$
0	1	2	3	4	5	6	7	*customer i*



feasible path (i): $0-1-2-3-4-5-6-7-8^*$

(i,j,r)	(1,0,1)	(2,1,1)	(3,2,1)	(4,3,1)	(5,4,1)	(6,5,1)	(7,6,1)	(8,7,1)	(9,8*,1)
(S^-,S^+)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)

feasible path (ii): $0-2-1-3-4-5-6-7-8^*$

(i,j,r)	(1,0,1)	(2,2,3)	(3,1,2)	(4,3,1)	(5,4,1)	(6,5,1)	(7,6,1)	(8,7,1)	(9,8*,1)
(S^-,S^+)	(∅, ∅)	(∅, ∅)	({2},{1})	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)	(∅, ∅)

****customer*** $j=\mathbf{8}\rightarrow$ destination depot

Fig. 5 Example of graph G^*

Regarding the *dominance procedure*, a label $\left(s^1, i, r\right)$ dominates another label $\left(s^2, i, r\right)$ if

$$s^1 \leq s^2 \tag{26}$$

$$c\left(s^1, i, r\right) \leq c\left(s^2, i, r\right) \tag{27}$$

Algorithm Heuristic dynamic programming

```
    // Initialization //
1   Set the values of $K$ and $q$
    // TSPTW–TSPPC reformulation //
2   Generate $\sigma = (0,...,n)$ with $\sigma(0) = 1$ and customers sorted in non-decreasing order of $(e_i + l_i)/2$
3   Compute $k(i)$ values by taking their upper bound $K$ into account
    // $G^*$ construction//
4   Construct the auxiliary graph $G^*$
    // SPPTW computation //
5   $NodL(1,0,1) \leftarrow \{(0,1,1)\}$
6   $i \leftarrow 1$
7   While $i < n+2$ do
        // Label extension //
8       For all nodes $(i,j,r)$ in layer $\Gamma_i^*$ do
9           For all labels $(s,i,r)$ stored in $NodL(i,j,r)$ do
10              For all outgoing nodes $(i+1,h,\ell)$ of $(i,j,r)$ do
11                  $(s',i+1,\ell) \leftarrow Extend((s,i,r),(i+1,h,\ell))$
                    // Dominance checking //
12                  $NodL(i+1,h,\ell) \leftarrow Dominance(NodL(i+1,h,\ell),(s',i+1,\ell),q)$
13              End For
14          End For
15      End For
16      $i \leftarrow i+1$
17  End While
18  Output the TSPTW route corresponding to the minimum cost label stored in $NodL(n+2, n+1, 1)$
```

Fig. 6 Heuristic dynamic programming algorithm

and at least one of these inequalities is strict. Additionally, a maximum number of q non-dominated labels is imposed on any node of G^* in order to avoid an excessive computation time. For each node $(i, j, r) \in G^*$, only the best q labels $\left(L_{i,r}^1, \ldots, L_{i,r}^q\right)$ with respect to the accumulated traveling cost are stored.

Figure 6 shows a pseudocode of the heuristic dynamic programming. To initialize the algorithm, we set the values of the parameters K and q (line 1). We then reformulate the TSPTW as a TSPPC (lines 2 and 3) and construct the auxiliary graph G^* (line 4). Afterwards, a simplified labeling procedure solves the TSPTW by computing a SPPTW on G^* (lines 5–18). This procedure is composed of a label extension procedure (lines 8–15) and a label dominance checking (line 12). Herein, $NodeL$ denotes the set of labels associated with a node $(i, j, r) \in G^*$, $Extend((s, i, r), (i+1, h, \ell))$ and $Dominance\left(NodL(i+1, h, \ell), \left(s', i+1, \ell\right), q\right)$ represent the label extension and dominance procedures, respectively. At the end of the algorithm, the TSPTW route corresponding to the minimum cost label stored in $NodL(n+2, n+1, 1)$ is outputted.

4.2 Adaptations to the TSPFlexTW context

This section presents two variants of a dynamic programming algorithm for the TSPFlexTW, which are adapted from the Balas and Simonetti [4] method. The first one, called *heuristic LP*, employs the label-correcting *LP* technique and the second one, called *heuristic EL*, makes use of the label-correcting *EL* technique.

Both approaches start by reformulating the TSPFlexTW as a TSPPC. For this purpose, the hard time windows are replaced by the flexible time windows in the required calculations. The heuristic variants then proceed to construct the auxiliary graph G^*. Once these steps have been carried out, the label-correcting *LP* and *EL* are applied.

As far as the label-correcting *LP* is concerned, the additional cost-related resource $pen(s, i, r)$ is assigned to the label (s, i, r). Its forward propagation employs the same resource extension functions (23) and (24) as well as the same feasibility condition (25) except that the flexible time windows replace the hard ones in the calculations. Regarding label dominance, if the inequality (26) is satisfied for a pair of labels $\left(s^1, i, r\right)$ and $\left(s^2, i, r\right)$, their penalty costs are obtained by means of the linear program (12)–(17). Thus, label $\left(s^2, i, r\right)$ is fathomed if the inequality

$$c\left(s^1,i,r\right) + pen\left(s^1,i,r\right) \le c\left(s^2,i,r\right) + pen\left(s^2,i,r\right) \tag{28}$$

also holds and at least one of these inequalities is strict.

On the other hand, the label-correcting *EL* is based on the extended label definition $(s, i, r, early, late)$ with accumulated traveling cost $c(s, i, r, early, late)$ and penalty cost $pen(s, i, r, early, late)$. In the course of label extension, the penalty costs are temporarily disregarded and the resources of accumulated earliness and lateness are recursively updated along the paths by means of the extension functions (19) and (20). The label dominance checking is determined by rules (21), (22), (26) and (27).

In both heuristic variants, only the best q labels $\left(L^1_{i,r}, \dots, L^q_{i,r}\right)$ with respect to the accumulated traveling cost are stored for a given node $(i, j, r) \in G^*$. Therefore, at the end of the extension procedure a set of at most q non-dominated labels $L^P_{n+2,\ 1}$, $P = 1, \dots, q$ associated with the node $(n+2,\ n+1,\ 1) \in G^*$ is obtained. Each non-dominated label stands for a TSPFlexTW feasible route and its overall cost is given by $c\left(L^P_{n+2,\ 1}\right) + pen\left(L^P_{n+2,\ 1}\right)$, where $pen\left(L^P_{n+2,\ 1}\right)$ is determined by the solution of LP model (12)–(17). In both heuristic variants, the feasible route with the minimum overall cost $\min_{P=1,\dots,q}\left\{c\left(L^P_{n+2,\ 1}\right) + pen\left(L^P_{n+2,\ 1}\right)\right\}$ corresponds to the best solution found.

The two variants of the heuristic dynamic programming algorithm (HDPA) are denoted according to the label-correcting procedures *LP* and *EL*, i.e., HDPALP and HDPAEL.

5 Computational experiments

This section describes the computational results of the proposed algorithms, which were coded in C# and run on an Intel® Core™ i7-4790 CPU at 3,6 GHz with 16 GB RAM. For all labeling methods, the Gurobi v.6.05 software was used to solve the LP model (12)–(17).

The algorithms were tested on two sets of instances. The first one consists of 135 symmetric Euclidean instances described in [14], with number of customers between 20 and 200 and time window widths ranging from 20 to 100 time units. They are grouped into 27 classes of five instances with an equal number of customers and time window widths. The name of each instance indicates the number of customers $|\mathcal{C}|$, the width of the time windows and the instance number in the class, e.g., n20w20.001 is the first instance of the class of five instances with 20 customers and time windows of 20 units. Since no travel time and travel cost matrices are available, these values are calculated by truncated integer Euclidean distances and then adjusted to maintain the validity of the triangular inequality by setting $t_{ij} = t_{ih} + t_{hj}$, if $t_{ih} + t_{hj} < t_{ij}$ and $c_{ij} = c_{ih} + c_{hj}$, if $c_{ih} + c_{hj} < c_{ij}$ for each triple $(i, j, h) \in \mathcal{N}$ [6].

The second set is adapted from the real-world asymmetric traveling salesman (ATSP) instances proposed in [17, 18], which are part of the TSPLIB [28]. These instances consist of 17 ATSP problems coming from pharmaceutical product delivery in Bologna. In the same manner as Dumas et al. [14], we generated time windows for the ATSP customers. For each problem, we first constructed a second nearest neighbour ATSP tour to determine the service start time at each customer. Then, four instances were obtained by generating time windows around such service start times. Each side of a time window consists of a random number drawn from a uniform distribution in the interval $[0, width/2]$, where $width = 50, \ 100, \ 200$ or 400. The service time is zero for all nodes. This procedure generated 68 instances grouped in 17 classes, with 33–170 customers per instance, and time windows between 50 and 400 units. The instance names specify the number of customers $|\mathcal{C}|$ and the width of its time windows, e.g., ftv33w50 contains 33 customers and presents time windows of 50 units.

The rest of this section is organized as follows. Section 5.1 assesses the performance of the variants EDPALP and EDPAEL of the exact dynamic programming algorithm, while the two variants HDPALP and HDPAEL of the heuristic dynamic programming algorithm are analyzed in Sect. 5.2. Finally, Sect. 5.3 compares the best solutions found for the TSPFlexTW symmetric instances with the corresponding TSPTW exact results to highlight the benefits gained by flexible time windows.

5.1 Results of the exact variants EDPALP and EDPAEL

The symmetric and asymmetric instances were adapted to the TSPFlexTW context by setting costs $(\delta_\alpha, \delta_\beta)$ to (0.50, 1.00) and fractions (f_i^e, f_i^l) to (0.10, 0.10) for all $i \in \mathcal{N}$. Such parameter settings correspond to the medium flexibility considered by Taş et al. [34] in their main experiments. Then, EDPALP and EDPAEL were applied with a computing time limit of 3600 s.

For each class of symmetric instances, Table 1 presents the results of exact methods by using the following notation: Solved (number of instances for which the optimal solution was found within the time limit), CPU (mean time in seconds), $|ForL|$ (mean number of non-dominated forward labels), $|BackL|$ (mean number of non-dominated backward labels) and '–' (no optimal solution was found within the time limit). EDPALP solved 92 (68%) instances with up to 200 customers, while EDPAEL solved 65 (48%) instances with up to 100 customers. Regarding the 65 instances for which both methods found the optimal solution, the mean computing time of EDPALP is significantly lower than that of EDPAEL. Furthermore, EDPALP consistently generated less non-dominated labels than EDPAEL. These results suggest that the label-correcting *LP* procedure coupled with EDPALP yields a better dominance procedure, which fathoms more dominated labels and speed up the exact bi-directional labeling algorithm.

We remark that the exact dynamic programming algorithms are sensitive to large numbers of customers as well as to wide time windows. For example, both EDPALP and EDPAEL solved most of the instances with time windows of 80 units and involving up to 40 customers. However, none of the algorithms coped with time windows with more than 40 units for instances with 100 customers.

Analogous information is shown in Table 2 for each class of asymmetric instances, which provides further evidence of the stronger dominance rules of EDPALP when compared to EDPAEL. EDPALP solved 59 (86%) instances whereas EDPAEL solved 54 (79%). In terms of the 54 instances solved by both variants, EDPALP presented lower CPU times and total of forward and backward labels. Taken together, the results obtained for symmetric and asymmetric instances underline the advantage of EDPALP with reference to exact dynamic programming. Detailed results, for each instance, are provided in Section A of the Supplementary Electronic Material.

Mono-directional counterparts of the bi-directional algorithms EDPALP and EDPAEL were considered in a second experiment in order to investigate the benefits stemming from the bi-directional bounded search framework. In such counterparts, labels are mono-directionally extended by just considering the forward propagation direction. They were applied to solve the same 203 symmetric and asymmetric instances of the TSPFlexTW within an identical processing time limit. The obtained results were compared to those of the bi-directional algorithms as shown in Table 3, where columns '#Labels' report the total number of non-dominated labels in memory at the end of execution of each algorithm. The bi-directional version of EDPALP was able to solve 8 more instances, reduced the mean computing time by around 33% and the mean number of labels by around 20% when compared to the mono-directional one. With reference to EDPAEL, the bi-directional version solved 8 more instances, reduced the mean computing time by around 64% and the mean number of labels by around 42% when compared to the mono-directional one. These results are consistent with those of Righini and Salani [29] and subsequent studies, which have shown that bi-directional labeling algorithms often outperform their mono-directional counterparts.

Since the bi-directional version of EDPALP achieved the best overall performance among the exact variants, it was tested against an alternate optimal approach, i.e., a state-of-art MIP solver. For this purpose, we used Gurobi v.6.05 branch-and-cut to directly solve model (1)–(8) with default parameters except limiting the total processing time to 3600 s. Table 4 displays such a comparison for symmetric and asymmetric instance sets. The interested reader is referred to Section B of the Supplementary Electronic Material for the detailed results obtained by the MIP solver.

Gurobi v.6.05 obtained optimal solutions for 92 (68%) symmetric instances with up to 150 customers. Note that EDPALP solved the same number of instances, and

Table 1 EDPALP and EDPAEL results for symmetric instances

Class	EDPALP				EDPAEL			
	Solved	$\lvert ForL \rvert$[a]	$\lvert BackL \rvert$[a]	CPU	Solved	$\lvert ForL \rvert$[a]	$\lvert BackL \rvert$[a]	CPU
n20w20	5/5	36.40	22.2	0.05	5/5	43.20	26.80	0.02
n20w40	5/5	107.00	55.4	0.18	5/5	170.40	110.20	0.03
n20w60	5/5	446.00	115.00	0.67	5/5	1891.80	133.40	0.25
n20w80	5/5	1008.80	220.60	1.40	5/5	2988.60	603.60	0.54
n20w100	5/5	3822.80	808.60	6.57	5/5	8224.40	2266.00	2.38
n40w20	5/5	146.40	70.80	0.20	5/5	790.60	155.60	0.08
n40w40	5/5	1192.80	187.20	1.96	5/5	16718.00	666.40	12.52
n40w60	5/5	13643.80	400.00	39.36	4/5	34814.75	1053.75	764.49
n40w80	5/5	44047.40	3331.60	207.65	4/5	118729.50	23277.50	1559.38
n40w100	3/5	100978.67	29835.33	2188.25	0/5	–	–	3600.00
n60w20	5/5	712.80	107.60	1.04	5/5	10207.60	891.60	2.87
n60w40	5/5	3824.20	2392.60	14.06	4/5	15417.25	2061.00	726.68
n60w60	5/5	59722.60	1478.80	508.65	3/5	87264.67	3839.67	1532.83
n60w80	2/5	148663.00	25026.00	2731.12	0/5	–	–	3600.00
n60w100	0/5	–	–	3600.00	0/5	–	–	3600.00
n80w20	5/5	1687.60	402.60	3.69	5/5	74321.00	5315.00	279.02
n80w40	5/5	25389.20	1716.80	74.86	2/5	56508.50	50085.00	2251.30
n80w60	2/5	118627.50	69409.50	2933.70	0/5	–	–	3600.00
n80w80	0/5	–	–	3600.00	0/5	–	–	3600.00
n100w20	5/5	32086.60	857.20	268.53	3/5	168674.00	4229.33	1959.99
n100w40	5/5	61362.00	7605.40	388.00	0/5	–	–	3600.00
n100w60	0/5	–	–	3600.00	0/5	–	–	3600.00

Table 1 continued

Class	EDPALP				EDPAEL			
	Solved	$\|ForL\|$[a]	$\|BackL\|$[a]	CPU	Solved	$\|ForL\|$[a]	$\|BackL\|$[a]	CPU
n150w20	3/5	50286.33	924.67	1615.47	0/5	–	–	3600.00
n150w40	1/5	197350.00	4384.00	3477.59	0/5	–	–	3600.00
n150w60	0/5	–	–	3600.00	0/5	–	–	3600.00
n200w20	1/5	221844.00	1444.00	3564.44	0/5	–	–	3600.00
n200w40	0/5	–	–	3600.00	0/5	–	–	3600.00
All	92/135				65/135			
Mean[b]		5062.35	548.11	18.46		32822.42	4319.83	145.57
Overall mean		28845.13	4193.92	1334.35		32822.42	4319.83	1936.76

[a]Computed only for the instances that could be solved within the time limit
[b]Means for the 65 instances that could be solved by both EDPALP and EDPAEL

Table 2 EDPALP and EDPAEL results for asymmetric instances

Class	EDPALP				EDPAEL			
	Solved	$\|ForL\|$[a]	$\|BackL\|$[a]	CPU	Solved	$\|ForL\|$[a]	$\|BackL\|$[a]	CPU
ftv33	4/4	126.75	31.50	0.15	4/4	310.00	48.75	0.04
ftv35	4/4	256.25	41.25	0.28	4/4	307.75	71.50	0.03
ftv38	4/4	238.25	33.50	0.20	4/4	321.25	98.00	0.03
ftv44	4/4	206.75	315.25	0.60	4/4	564.50	969.00	0.14
ftv47	4/4	955.50	140.75	1.37	4/4	5651.25	739.50	1.56
ftv55	4/4	12359.25	325.25	36.86	3/4	881.67	380.00	900.10
ftv64	4/4	2756.00	553.50	6.24	4/4	65431.50	3735.25	494.00
ftv70	3/4	383.67	109.00	900.47	3/4	1554.33	185.67	900.20
ftv90	3/4	508.33	1563.00	902.93	3/4	2860.00	33300.00	958.10
ftv100	3/4	989.00	512.33	901.73	3/4	4593.00	3004.33	901.78
ftv110	4/4	15172.50	11664.75	117.87	3/4	4018.33	6761.00	907.59
ftv120	3/4	1913.67	888.00	905.91	3/4	12853.33	1802.67	908.22
ftv130	3/4	1875.67	847.00	904.95	3/4	6487.67	20988.00	921.21
ftv140	3/4	6339.33	767.67	916.80	3/4	79217.00	2326.00	1230.37
ftv150	3/4	6754.67	324.67	922.75	2/4	2214.00	375.50	1800.48
ftv160	3/4	40248.67	359.67	1094.54	2/4	15770.00	225.50	1804.50
ftv170	3/4	22583.67	557.33	1009.32	2/4	14399.50	220.50	1804.16
All	59/68				54/68			
Mean[b]		1163.93	400.67	3.58		12824.22	4269.15	69.07
Overall mean		6323.31	1189.98	507.23		12824.22	4269.15	796.03

[a]Computed only for the instances that could be solved within the time limit
[b]Means for the 54 instances that could be solved by both EDPALP and EDPAEL

Table 3 Comparison between mono-directional and bi-directional versions of EDPALP and EDPAEL

Algorithm	Mono-directional version			Bi-directional version		
	Solved	#Labels	CPU	Solved	#Labels	CPU
EDPALP	143/203	22320.41[a]	200.63[a]	151/203	17840.36[a]	132.61[a]
EDPAEL	111/203	32553.79[b]	141.05[b]	119/203	18668.95[b]	49.55[b]

[a]Means for the 143 instances that could be solved by both versions of EDPALP
[b]Means for the 111 instances that could be solved by both versions of EDPAEL

Table 4 Comparison between EDPALP and Gurobi v.6.05

Instance set	EDPALP		Gurobi v.6.05	
	Solved	CPU	Solved	CPU
Symmetric	92/135	139.08[a]	92/135	181.19[a]
Asymmetric	59/68	35.45[b]	67/68	25.65[b]

[a]Means for the 81 symmetric instances that could be solved by both methods
[b]Means for the 59 asymmetric instances that could be solved by both methods

successfully coped with up to 200 customers. Regarding the 81 symmetric instances that could be solved by both methods, Gurobi v.6.05 required a mean CPU time around 30% longer than EDPALP. Turning now to asymmetric instances, Gurobi v.6.05 solved 67 (98%) out of the 68 instances to optimality and consumed lower computing times for those 59 that also could be solved by EDPALP. Overall, these results suggest an advantage of EDPALP for the symmetric set and a disadvantage for the asymmetric one. Such a disagreement may be explained by the absence of the triangular inequality property for the asymmetric instances. As stated by Li [25], the exact dynamic programming algorithm is significantly faster for situations in which the triangular inequality holds since the strengthened dominance condition mentioned in Sect. 3.1.2 can be applied.

5.2 Results of the heuristic variants HDPALP and HDPAEL

Based on the prior experimentation carried out in [4], we tested the sets of values {8, 10, 12, 14} and {10, 15, 20} for the parameters K and q, respectively. Within a processing time limit of 3600 s, the best values (K, q) found were (8, 20) for HDPALP and (12, 20) for HDPAEL. Note that, if the largest required $k(i)$, i.e., $\max_{i\in\{0,\ldots,n\}}\{j_0 - \sigma(i)\}$, does not exceed its imposed upper bound K and if no more than q non-dominated labels are generated for each node $(i, j, r) \in G^*$, the heuristic dynamic programming algorithms become exact. However, the investigated sets entail instances with values of $\max_{i\in\{0,\ldots,n\}}\{j_0 - \sigma(i)\}$ beyond 20, which are too large for practical use [5].

After the determination of these parameters, both heuristic variants were applied to the TSPFlexTW instances by keeping the same flexible time window and cost settings employed in the previous section. Table 5 summarizes the results obtained for symmetric and asymmetric instances. For each set, we report the number of instances for which a feasible solution was found (Solved), the mean of best objective costs found by the heuristic method in question (Obj.), its gap relative to the available optimal solutions (Opt.), i.e., 100(Obj − Opt)/Opt, and CPU (mean time in seconds). Detailed results, for each instance, can be found in Section C of the Supplementary Electronic Material.

This table is revealing in several ways. As far as symmetric problems are concerned, HDPALP solved 120 (88%) instances, presented a mean gap of 0.73% relative to the available optimal solutions and required a mean computational time of approximately 223 s. On the other hand, HDPAEL solved 130 (96%) instances, achieved a mean gap of 0.23% and expended a mean computational time of approximately 93 s. Moreover, for the 119 instances solved by both methods, the reported means show that HDPAEL provided better quality solutions and consumed lower CPU times. With reference to asymmetric problems, both algorithms solved all instances with gaps smaller than 1.00% in regard to the available optimal solutions. HDPAEL was a little faster, while HDPALP obtained a smaller mean gap.

From these results, the effectiveness of the heuristic dynamic programming variants can be analyzed. Within the given time limit, HDPALP solved 188 (92%) instances and HDPAEL solved 198 (97%) out of the 203 instances, whereas their exact bi-directional counterparts EDPALP and EDPAEL solved 151 (74%) and 119 (58%) instances, respectively. Such findings are not surprising since the number of labels may grow exponentially with the number of arcs in the forward and backward paths regarding exact dynamic programming. Conversely, the complexity of the heuristic variants is bounded by $|\mathcal{A}^*| \leq K(K+1)2^{K-2}(n+1)$ [3], with parameters K and q that control the combinatorial explosion of labels propagated along arcs in $\mathcal{A}^*$. Therefore, with an appropriated parameter setting (K, q), the heuristic algorithms can cope with instances which are intractable for their exact counterparts.

To get additional support for the evaluation of the heuristics, we applied non-parametric Wilcoxon signed rank test [21]. This test compares the expected values $E_{HDPALP}[OC]$ and $E_{HDPAEL}[OC]$, where OC is the random variable that represents the objective cost obtained by the variants HDPALP and HDPAEL for a set of instances. Ranks are assigned to each difference and we test the null hypothesis $E_{HDPALP}[OC] = E_{HDPAEL}[OC]$ with the alternative hypothesis $E_{HDPALP}[OC] < E_{HDPAEL}[OC]$ or $E_{HDPALP}[OC] > E_{HDPAEL}[OC]$ at a significance level $\alpha = 0.05$. For the asymmetric problems, the null hypothesis $E_{HDPALP}[OC] = E_{HDPAEL}[OC]$ was accepted. However, the Wilcoxon test showed that HDPAEL performance was superior to that of HDPALP for symmetric instances. Since the numbers of non-dominated labels stored by both heuristic variants become similar under the restrictions imposed by parameter q, this advantage might be related to the HDPAEL faster dominance procedure, which does not require time-consuming LP calculations.

Table 5 HDPALP and HDPAEL results

Instance set	HDPALP				HDPAEL			
	Solved	Obj.	Gap%	CPU	Solved	Obj.	Gap%	CPU
Symmetric	120/135	563.81[a]	0.73[a]	223.88[a]	130/135	565.56[a]	0.23[a]	93.46[a]
		562.36[b]	0.73[b]	230.60[b]		557.58[b]	0.18[b]	78.92[b]
Asymmetric	68/68	4094.61[a]	0.03[a]	138.57[a]	68/68	4096.87[a]	0.05[a]	9.46[a]

[a]Overall means

[b]Means for the 119 symmetric instances that could be solved by both HDPALP and HDPAEL

5.3 TSPFlexTW versus TSPTW

We also analyzed the traveling cost savings resulting from flexible time windows. Optimal solution costs reported in [14] for the TSPTW symmetric instances were compared to the corresponding solution costs obtained for the TSPFlexTW. Three tests were carried out by considering different flexible time window scenarios: (1) we kept the medium flexibility settings $\left(f_i^e, f_i^l\right) = (0.10, 0.10)$ for all $i \in \mathcal{N}$ and $\left(\delta_\alpha, \delta_\beta\right) = (0.50, 1.00)$; (2) we increased the values of flexibility fractions to $\left(f_i^e, f_i^l\right) = (0.25, 0.25)$ for all $i \in \mathcal{N}$ but kept the values of cost coefficients $\left(\delta_\alpha, \delta_\beta\right) = (0.50, 1.00)$; and (3) we kept the values of flexibility fractions $\left(f_i^e, f_i^l\right) = (0.10, 0.10)$ for all $i \in \mathcal{N}$ but applied higher deviation costs $\left(\delta_\alpha, \delta_\beta\right) = (2.00, 4.00)$. The higher values of parameters considered in scenarios (2) and (3) were also extracted from Taş et al. [34]. For all scenarios, we applied EDPALP to solve the TSPFlexTW instances with a computing time limit of 3600 s and, whenever it fails, HDPAEL is employed with $K = 12$ and $q = 20$.

Table 6 summarizes the above-mentioned tests which are fully described in Section D of the Supplementary Electronic Material. Column 1 shows the names of instance classes. The next five columns refer to results obtained when solving TSPFlexTW scenario (1). Column 2 reports the number of instances for which a feasible solution was found (Solved). Columns 3–6 present the mean of TSPFlexTW best objective costs obtained by EDPALP or HDPAEL which includes traveling and penalty costs (Obj.), its associated mean traveling cost (Trav.), the mean percentage difference $\Delta_1\%$ between TSPFlexTW objective costs and TSPTW optimal costs (Opt.), i.e.,$100(\text{Opt} - \text{Obj})/\text{Obj}$, and the mean percentage difference $\Delta_2\%$ between TSPFlexTW traveling costs and TSPTW optimal costs, i.e., $100(\text{Opt} - \text{Trav})/\text{Trav}$. Analogous information is shown in Columns 7–11 and in Columns 12–16 for scenarios (2) and (3), respectively. The bottom of this table provides the summary of the percentage differences obtained for the three scenarios investigated. We remark that the TSPFlexTW is a relaxation of the TSPTW, thus its optimal solution at most would lead to the same objective cost as the TSPTW optimal cost. As a consequence, the percentage differences defined above are non-negative whenever the TSPFlexTW is solved to optimality, i.e., $\Delta_1\%, \Delta_2\% \geq 0$. Nevertheless, since EDPALP fails in some instances, which are heuristically solved by HDPAEL, negative percentage differences are possible, i.e., $\Delta_1\%, \Delta_2\% < 0$.

From the examination of Table 6, it is possible to attest the advantages of flexible time windows. With reference to scenario (1), percentage differences ranging from -8.89 to 12.00% and from -6.59 to 13.42% were obtained for $\Delta_1\%$ and $\Delta_2\%$, respectively. The flexible time windows leaded to lower overall costs for 73 (54%) instances and lower traveling costs for 82 (61%) instances than the hard time windows. Only few heuristic solutions found by HDPAEL presented higher overall costs (15%) or higher travelling costs (6%) than the corresponding TSPTW. The TSPFlexTW algorithm variants failed in 4 (3%) instances. As far as scenario (2) is concerned, the higher flexibility fractions leaded to even better comparative figures resulting in overall cost savings for 77 (57%) instances and traveling cost savings for 104 (77%) instances when compared to TSPTW. The values of $\Delta_1\%$ range between -9.12 and 17.22%,

Table 6 Comparison of the optimal TSPTW solutions with the best solutions found for the TSPFlexTW

Class	TSPFlexTW scenario (1)					TSPFlexTW scenario (2)					TSPFlexTW scenario (3)				
	Solved	Obj.	Trav.	Δ_1%	Δ_2%	Solved	Obj.	Trav.	Δ_1%	Δ_2%	Solved	Obj.	Trav.	Δ_1%	Δ_2%
n20w20	5/5	360.48	360.20	0.21	0.29	5/5	358.80	356.40	0.69	1.42	5/5	361.20	361.20	0.00	0.00
n20w40	5/5	312.65	311.20	1.26	1.74	5/5	307.83	303.80	2.90	4.25	5/5	314.40	313.60	0.65	0.99
n20w60	5/5	299.80	299.00	3.67	3.99	5/5	290.18	281.00	7.33	11.03	5/5	302.20	299.00	2.75	3.99
n20w80	5/5	303.34	297.80	2.66	4.65	5/5	288.10	274.80	8.37	13.97	5/5	309.80	305.40	0.40	1.94
n20w100	5/5	268.90	268.20	2.14	2.40	5/5	268.90	268.20	2.14	2.40	5/5	271.00	268.20	1.41	2.40
n40w20	5/5	484.36	483.40	0.45	0.64	5/5	478.58	473.40	1.72	2.88	5/5	485.92	485.00	0.13	0.31
n40w40	5/5	455.59	454.40	1.24	1.53	5/5	453.73	447.20	1.68	3.27	5/5	459.16	454.40	0.40	1.53
n40w60	5/5	409.16	406.60	1.66	2.32	5/5	399.28	386.00	3.96	7.46	5/5	413.00	411.40	0.72	1.07
n40w80	5/5	392.12	387.40	1.94	3.23	5/5	397.43	388.60	0.57	2.88	5/5	398.80	393.00	0.23	1.71
n40w100	5/5	373.60	370.60	0.94	1.72	5/5	382.20	353.00	− 1.17	6.84	5/5	383.72	371.40	− 1.47	1.52
n60w20	5/5	576.90	575.80	0.89	1.10	5/5	575.65	569.40	1.10	2.25	5/5	579.00	577.40	0.48	0.79
n60w40	5/5	578.65	573.40	2.03	3.02	5/5	574.30	565.80	2.84	4.36	5/5	585.00	578.40	0.89	2.08
n60w60	5/5	550.04	545.60	1.83	2.67	5/5	550.58	536.20	1.72	4.61	5/5	553.00	551.80	1.27	1.50
n60w80	5/5	499.53	491.20	1.59	3.31	4/5	502.13	478.50	3.57	8.75	5/5	515.44	497.00	− 1.56	2.16
n60w100	5/5	516.23	499.60	− 0.24	3.16	5/5	530.63	491.40	− 2.89	5.01	5/5	566.12	499.60	− 8.44	3.16
n80w20	5/5	673.74	672.80	0.45	0.60	5/5	669.25	663.20	1.13	2.04	5/5	675.80	674.60	0.13	0.32
n80w40	5/5	624.61	620.80	0.93	1.60	5/5	622.93	603.00	1.41	4.67	5/5	629.20	628.80	0.12	0.18
n80w60	4/5	591.14	587.50	0.66	1.27	5/5	616.70	576.60	− 1.42	5.43	4/5	601.30	588.25	− 0.98	1.14
n80w80	5/5	608.17	597.00	− 2.07	− 0.30	5/5	619.23	578.00	− 3.94	2.98	5/5	641.68	597.00	− 6.87	− 0.30
n100w20	5/5	754.50	753.20	0.41	0.58	5/5	750.65	744.60	0.93	1.77	5/5	756.92	755.60	0.09	0.27
n100w40	5/5	689.68	687.00	1.76	2.17	5/5	696.28	680.20	0.78	3.17	5/5	697.32	688.60	0.68	1.93
n100w60	5/5	685.34	676.20	1.75	3.17	2/5	686.50	642.00	− 3.70	2.82	5/5	712.76	676.20	− 2.22	3.17

Table 6 continued

Class	TSPFlexTW scenario (1)					TSPFlexTW scenario (2)					TSPFlexTW scenario (3)				
	Solved	Obj.	Trav.	$\Delta_1\%$	$\Delta_2\%$	Solved	Obj.	Trav.	$\Delta_1\%$	$\Delta_2\%$	Solved	Obj.	Trav.	$\Delta_1\%$	$\Delta_2\%$
n150w20	5/5	866.24	862.00	0.25	0.76	5/5	864.43	853.40	0.48	1.75	5/5	874.44	864.80	− 0.69	0.43
n150w40	5/5	832.66	828.40	0.21	0.71	5/5	856.43	823.20	− 2.71	1.35	5/5	845.16	828.80	− 1.23	0.67
n150w60	3/5	787.77	779.67	1.31	2.37	3/5	846.63	794.33	− 5.65	0.47	3/5	812.07	779.67	− 1.68	2.37
n200w20	5/5	1001.17	994.60	0.81	1.49	5/5	1008.25	982.00	0.12	2.82	5/5	1020.64	997.20	− 1.13	1.23
n200w40	4/5	975.89	965.50	− 0.51	0.58	3/5	982.07	963.33	− 3.50	1.00	4/5	1007.05	965.50	− 3.60	0.58
All	131/135					127/135					131/135				
$\Delta_1\% > 0$	73					77					34				
$\Delta_1\% = 0$	38					17					61				
$\Delta_1\% < 0$	20					33					36				
Min $\Delta_1\%$	− 8.89					− 9.12					− 17.53				
Max $\Delta_1\%$	12.00					17.22					9.38				
$\Delta_2\% > 0$	82					104					60				
$\Delta_2\% = 0$	41					17					62				
$\Delta_2\% < 0$	8					6					9				
Min $\Delta_2\%$	− 6.59					− 2.00					− 6.59				
Max $\Delta_2\%$	13.42					23.89					13.08				

whereas the values of $\Delta_2\%$ range between -2.00 and 23.89%. As a counterpart of such an increased flexibility, the TSPFlexTW algorithms failed in 8 (6%) instances because taking wider flexible time windows into account implies a higher computational effort. Finally, as expected, scenario (3) shows that higher penalty coefficients can possibly eliminate the benefits of flexible time windows in terms of overall cost reduction since higher penalties incur. However, traveling cost savings were preserved for a large number of instances (44%). The values of $\Delta_1\%$ vary from -17.53 to 9.38%, while the values of $\Delta_2\%$ are between -6.59 and 13.08%.

We note that the greatest cost savings were obtained for the instances with $20 \leq |\mathcal{C}| \leq 60$, since most of them could be solved to optimality by EDPALP. Furthermore, wide time windows increase the TSPFlexTW instance difficult for a fixed $|\mathcal{C}|$ and may lead to reduced and/or negative cost savings. For example, the solutions obtained on the class n80w40 presented mean percentage differences $\Delta_1\% = 0.93/\Delta_2\% = 1.60$ for scenario (1), $\Delta_1\% = 1.41/\Delta_2\% = 4.67$ for scenario (2), and $\Delta_1\% = 0.12/\Delta_2\% = 0.18$ for scenario (3). On the other hand, the mean outcomes found on the class n80w80, whose time windows are 40 units larger, were $\Delta_1\% = -2.07/\Delta_2\% = -0.30$, $\Delta_1\% = -3.94/\Delta_2\% = 2.98$ and $\Delta_1\% = -6.87/\Delta_2\% = -0.30$ for scenarios (1), (2) and (3), respectively.

6 Conclusions

We have considered flexible time windows in the traveling salesman problem (TSPFlexTW) which are an expansion of the hard time windows (TSPTW). The service of a customer can be started before or after the hard time window at a penalty cost added to the objective function. This leads to a problem that requires the determination of a sequence of customers and their respective service start times in order to minimize the sum of traveling costs with earliness and lateness costs. Flexible time windows result in a greater feasible space, which may lead to cost savings.

We have proposed two alternative extensions of exact and heuristic dynamic programming algorithms originally proposed for the TSPTW to solve the TSPFlexTW. Such algorithms make use of new label-correcting techniques, called label-correcting *LP* and label-correcting *EL*, and they were tested on a variety of symmetric and asymmetric instances from the literature.

The obtained results indicate that label-correcting *LP* leads to more effective exact methods when compared to label-correcting *EL*. Moreover, bi-directional versions of these approaches were superior to their mono-directional counterparts. The most efficient exact method outperformed a state-of-art MIP solver for instances where the triangular inequality holds and stronger dominance rules can be applied. We also observed that exact algorithms are sensitive to large numbers of customers as well as to wide time windows.

Regarding heuristic methods, label-correcting *EL* outperformed label-correcting *LP* for symmetric instances, while both presented a similar performance for asymmetric instances. The most efficient heuristic quickly solved 198 out of the 203 instances with up to 200 customers and time windows of up to 400 units by yielding a mean gap smaller than 0.20% in regard to the available optimal solutions.

The benefits gained by TSPFlexTW compared to the TSPTW have been also assessed. The flexible time windows leaded to overall cost savings of up to 17.22% and traveling cost savings of up to 23.89% depending on the flexibility settings employed. Wider flexible time windows provided greater cost savings, whereas higher penalty coefficients reduced such gains. Only few heuristic solutions found for harder instances presented higher overall costs or higher travelling costs than the corresponding TSPTW.

The algorithmic techniques suggested here may be applied to other types of routing and scheduling problems with flexible time window constraints. Furthermore, the proposed dynamic programming algorithms can be embedded in optimization methods under the cluster-first, route-second approach for the VRPFlexTW.

Acknowledgements This research was funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq-Brazil, Grants 141064/2015-3 and 305577/2014-0) and the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP-Brazil, Grants 2016/01860-1 and 2016/06566-4). We are grateful for the valuable suggestions made by the editor and referees.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Upper Saddle River, NJ (1993)
2. Balakrishnan, N.: Simple heuristics for the vehicle routeing problem with soft time windows. J. Oper. Res. Soc. **44**(3), 279–287 (1993)
3. Balas, E.: New classes of efficiently solvable generalized traveling salesman problems. Ann. Oper. Res. **86**, 529–558 (1999)
4. Balas, E., Simonetti, N.: Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study. INFORMS J. Comput. **13**(1), 56–75 (2001)
5. Balas, E., Simonetti, N., Vazacopoulos, A.: Job shop scheduling with setup times, deadlines and precedence constraints. J. Sched. **11**(4), 253–262 (2008)
6. Baldacci, R., Mingozzi, A., Roberti, R.: New state-space relaxations for solving the traveling salesman problem with time windows. INFORMS J. Comput. **24**(3), 356–371 (2012)
7. Bellman, R.: Dynamic Programming. Princeton University Press, Upper Saddle River, NJ (1957)
8. Bhusiri, N., Qureshi, A.G., Taniguchi, E.: The trade-off between fixed vehicle costs and time-dependent arrival penalties in a routing problem. Transp. Res. Part E Log. **62**, 1–22 (2014)
9. Chiang, W.C., Russell, R.A.: A metaheuristic for the vehicle routeing-problem with soft time windows. J. Oper. Res. Soc. **55**(12), 1298–1310 (2004)
10. Denardo, E.V.: Dynamic Programming: Models and Applications. Prentice Hall, Upper Saddle River, NJ (1982)
11. Desaulniers, G., Lessard, F., Hadjar, A.: Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transp. Sci. **42**(3), 387–404 (2008)
12. Desrochers, M.: An algorithm for the shortest path problem with resource constraints. Technical report G-88-27, GERAD, Montreal (1988)
13. Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F.: Time constrained routing and scheduling. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, F.L. (eds.) Handbooks in Operations Research and Management Science. Network Routing, vol. 8, pp. 35–139. Elsevier, Amsterdam (1995)
14. Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M.M.: An optimal algorithm for the traveling salesman problem with time windows. Oper. Res. **43**(2), 367–371 (1995)
15. Dumas, Y., Soumis, F., Desrosiers, J.: Optimizing the schedule for a fixed vehicle path with convex inconvenience costs. Transp. Sci. **24**, 145–152 (1990)
16. Feillet, D., Dejax, P., Gendreau, M., Gueguen, C.: An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks **44**(3), 216–229 (2004)

17. Fischetti, M., Toth, P.: A polyhedral approach to the asymmetric traveling salesman problem. Manag. Sci. **43**(11), 1520–1536 (1997)
18. Fischetti, M., Toth, P., Vigo, D.: A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper. Res. **42**(5), 846–859 (1994)
19. Fleischmann, B.: The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. Eur. J. Oper. Res. **75**(2), 395–404 (1994)
20. Fry, T.D., Leong, G.K.: Single machine scheduling: a comparison of two solution procedures. OMEGA Int. J. Manag. Sci. **15**(4), 277–282 (1987)
21. Golden, B.L., Stewart, W.R.: Empirical analysis of heuristics. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The traveling salesman problem, pp. 207–250. Wiley, Chichester (1985)
22. Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., Yagiura, M.: Effective local search algorithms for routing and scheduling problems with general time window constraints. Transp. Sci. **39**(2), 206–232 (2005)
23. Kohl, N., Desrosiers, J., Madsen, O.B.G., Solomon, M.M., Soumis, F.: 2-path cuts for the vehicle routing problem with time windows. Transp. Sci. **33**(1), 101–116 (1999)
24. Koskosidis, Y.A., Powell, W.B., Solomon, M.M.: An optimization based heuristic for vehicle routeing and scheduling with soft time window constraints. Transp. Sci. **26**, 69–85 (1992)
25. Li, J.Q.: A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Working paper, University of California, Berkeley (2009)
26. Liberatore, F., Righini, G., Salani, M.: A column generation algorithm for the vehicle routing problem with soft time windows. 4OR. Q. J. Oper. Res. **9**(1), 49–82 (2011)
27. Qureshi, A., Taniguchi, E., Yamada, T.: An exact solution approach for vehicle routing and scheduling problems with soft time windows. Transp. Res. Part E Log. **45**(6), 960–977 (2009)
28. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
29. Righini, G., Salani, M.: Symmetry helps: bounded bidirectional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optim. **3**(3), 255–273 (2006)
30. Savelsbergh, M.W.P.: Local search in routing problems with time windows. Ann. Oper. Res. **4**(1), 285–305 (1985)
31. Sexton, T.R., Bodin, L.D.: Optimizing single vehicle many-to-many operations with desired delivery times: I. Sched. Transp. Sci. **19**(4), 378–410 (1985)
32. Sexton, T.R., Choi, Y.: Pickup and delivery of partial loads with soft time windows. Am. J. Math. Soc. **6**, 369–398 (1986)
33. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transp. Sci. **31**(2), 170–186 (1997)
34. Taş, D., Jabali, O., Woensel, T.V.: A vehicle routing problem with flexible time windows. Comput. Oper. Res. **52**, 39–54 (2014)
35. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Timing problems and algorithms: time decisions for sequences of activities. Networks **65**(2), 102–128 (2015)