Computational Optimization and Applications (2022) 83:593–614
https://doi.org/10.1007/s10589-022-00405-8

A generalized shortest path tour problem with time windows

L. Di Puglia Pugliese[1] · D. Ferone[2] · P. Festa[3] · F. Guerriero[2]

Received: 15 December 2021 / Accepted: 25 July 2022 / Published online: 10 August 2022

Abstract
This paper studies a generalization of the shortest path tour problem with time windows (*GSPTPTW*). The aim is to find a single-origin single-destination shortest path, which has to pass through an ordered sequence of not necessarily disjoint node-subsets. Each node has a time window for each node-subset to which it belongs. We investigate the theoretical properties of *GSPTPTW* and propose a dynamic programming approach to solve it. Numerical results collected on a large set of new benchmark instances highlight the effectiveness of the proposed solution approach.

Keywords Generalized shortest path tour problem · Disjoint subsets · Time windows · Dynamic programming

1 Introduction

The *Shortest Path Tour Problem* (*SPTP*) is a constrained version of the *Shortest Path Problem* (*SPP*), and it was firstly introduced in [1]. Disjoint subsets of nodes $T_1, \ldots, T_N$ characterize *SPTP*, which aims at finding a shortest path from a source node s to a destination node d, where a solution path must visit the disjoint subsets

✉ F. Guerriero
francesca.guerriero@unical.it

L. Di Puglia Pugliese
luigi.dipugliapugliese@icar.cnr.it

D. Ferone
danieleferone@gmail.com

P. Festa
paola.festa@unina.it

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Rende, Italy

[2] Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy

[3] Department of Mathematics and Applications, University of Napoli Federico II, Naples, Italy

$T_k, k = 1, \ldots, N$ according to their order. A subset T_k is said to be visited if at least one node belonging to T_k appears in a solution path. In [2], the author proved that *SPTP* is polynomially solvable by reducing it to *SPP*. Later, a dynamic programming algorithm and a "depth-first tour search algorithm" were proposed in [3, 4], respectively.

The scientific literature addressed several variants of *SPTP*. *Forward SPTP* was studied in [5, 6], in which it is possible to visit a node in T_k if and only if at least a node of each previous subsets $T_1, \ldots, T_{k-1}$ has been already visited. This variant of the problem remains solvable in polynomial time. The *Constrained SPTP* was studied for the first time in [7]. The authors proved the **NP**-hardness of *Constrained SPTP*. In this variant, each arc must appear at most once in any feasible solution path. Mathematical formulations and solution approaches for this version of *SPTP* have been proposed in [8–10]. Recently, [11] proposed a branch-and-price approach, too. *SPTP with Time Windows* (*SPTPTW*) was addressed for the first time in [12]. The authors proved that the problem is **NP**-hard and proposed a dynamic programming labeling-based algorithm for its solution.

SPTPTW shares some similarities, depicted in the following, with the shortest path problem with time windows (*SPPTW*) and the generalized vehicle routing problem with time windows (*GVRPTW*) [13]. *SPPTW*, introduced in [14], aims at finding a path from a source to a destination node with the smallest cost, such that each node visited along the path is served within its time window. Given its practical and theoretical importance, *SPPTW* has attracted significant attention from many researchers over the years [15] and both exact and heuristic strategies have been proposed for its solution (see, e.g., [16–19]). As for *SPTPTW*, only the nodes belonging to the subsets $T_k, k = 1, \ldots, N$ have to be served within their time window. *GVRPTW* is a particular instance of the generalized vehicle routing [20], where the set of nodes is partitioned into sets of customers, and each set must be visited (served) exactly once. Differently from *SPTPTW*, the sets are not ordered and the decisions involved by the solution process of the problem are both node selection and node sequencing.

In this paper, we extend the work [12] related to *SPTPTW*. In particular, we study a generalization of *SPTPTW* (*GSPTPTW*), where the assumption on the disjoint subsets T_k, $k = 1, \ldots, N$ is relaxed. Thus, we admit that a node v can belong to different subsets T_k with possibly different time windows. It follows that node v can be used to serve any of the subsets it belongs to within the associated time window. Note that *GSPTPTW* is not a forward variant.

GSPTPTW models several real situations in which a node can belong to several subsets. Different events, like exhibitions or live performances, can be held in the same physical place. On the one hand, such events can take place at different hours, and hence the same place (node) is included in different subsets, that represent different time slots of the time horizon. On the other hand, the interesting events can take place in parallel in the same place, hence this place (node) is included in different subsets with the same time window. In this case, the subsets represent different interesting events. Another application is photographic tour planning. In this context, the sun position can either positively or negatively influence the quality of the photos, e.g., it could be better to shoot during either sunrise or sunset. Each type of landscape can be modeled as a subset T_k, and each node belonging to T_k represents

the different best shooting hours. Moreover, a courier has to deliver parcels to several customers with different time availability in the same place (for example in a condominium). In this case, the subsets represent the possible availability of the customers to pick up the parcels, modeled as time windows, and the nodes represent the customers. Thus, a node can be included into several subsets, modeling the different time availability of the customer to pick up the parcels. One can also consider a courier that has to deliver parcels to several intermediate depots of the same company. Each depot can pick up the parcels in different time slots. Thus, we can model the different time slots as subsets T_k and put into each subset the depots open in the associated time slot. The courier chooses the depot to deliver the parcels among all the open ones for each time slot.

The contribution of the paper is threefold. (1) We analyze the theoretical properties of *GSPTPTW*, proving that *GSPTPTW* has the same complexity as *SPTPTW*. It is a non-trivial result since relaxing the assumption on disjoint subsets the generalized problem maintains the same complexity of the more constrained variant. (2) We retrieve a polynomial procedure to transform any instance of *GSPTPTW* to an instance of *SPTPTW*. This result allows us to use solution approaches developed for *SPTPTW* to address *GSPTPTW*. (3) We define a solution strategy to directly solve *GSPTPTW* by exploiting a dynamic programming reformulation of *GSPTPTW*. The collected numerical results underline that solving *GSPTPTW* directly, by using the proposed dynamic programming approach, is more efficient than solving the corresponding *SPTPTW*, with the state-of-the-art algorithm proposed in [12].

Hence, the present work completes the theoretical study carried out for the *SPTPTW* in [12]. Indeed, this paper and [12] represent together an overall overview on both theoretical and application aspects related to *SPTP* with time restrictions.

The paper is organized as follows. In Sect. 2, the problem is formally described. Section 3 presents the proposed solution approach based on dynamic programming. Section 4 shows the computational results carried out considering several network topologies. Finally, Sect. 5 concludes the paper providing some directions for future research.

2 The generalized shortest path tour problem with time windows

GSPTPTW is defined on a directed graph $G(V, A)$, where V is the set of n nodes and $A = \{(i,j) \in V \times V | i,j \in V \wedge i \neq j\}$ is the set of m arcs. Let $s, d \in V$ be the source and the destination nodes, respectively. Subsets $T_k \subseteq V, k = 1, \ldots, N$ are given. Let $\mathcal{T} = \bigcup_{k=1}^{N} T_k \subseteq V$ be the set of nodes included in at least one subset $T_k, k = 1, \ldots, N$. In *SPTP* and its variant with time windows, the subsets T_k are disjoint, i.e., $T_h \cap T_l = \emptyset, \forall h, l = 1, \ldots, N, h \neq l$. In *GSPTPTW* the disjunction constraint is relaxed, and this means that a node can belong to several subsets T_k. Without loss of generality, we assume that $T_1 = \{s\}$ and $T_N = \{d\}$.

A non-negative cost c_{ij} and a non-negative transit time t_{ij} are associated with each arc $(i,j) \in A$. A service time s_i^k and a time window $\left[e_i^k, l_i^k\right]$ is associated with each

node $i \in T_k$, where e_i^k and l_i^k are respectively the earliest and the latest feasible arrival time to node i when it is used to serve subset T_k.

Given two distinct nodes i_1 and i_v, a path $\pi_{i_1 i_v} = \langle i_1, \ldots, i_v \rangle$ is an ordered sequence of nodes from i_1 to i_v, such that $(i_l, i_{l+1}) \in A, l = 1, \ldots, v-1$. The cost $c\left(\pi_{i_1 i_v}\right)$ of the path $\pi_{i_1 i_v}$ is defined as the sum of the cost associated with its arcs, i.e., $c\left(\pi_{i_1 i_v}\right) = \sum_{l=1}^{v-1} c_{i_l i_{l+1}}$.

GSPTPTW aims at finding a path π_{sd}^* from the source node $s \in V$ to the destination node $d \in V$ in the directed graph G with the smallest cost. An optimal path π_{sd}^* must visit sequentially the subsets $T_k, k = 1, \ldots, N$. We note that the subsets $T_k, k = 1, \ldots, N$, must be visited in exactly the same order in which they are defined.

Let τ_{i_l} be the arrival time at node $i_l, l = 1, \ldots, v$, a path π_{sd} is said to be a feasible solution for *GSPTPTW* if it satisfies the following requirements:

$$\begin{aligned} &\exists\, g_1, \ldots, g_N : g_1 \leq g_2 \leq \ldots \leq g_N, \\ &i_{g_1} \in \pi_{sd} \cap T_1, i_{g_2} \in \pi_{sd} \cap T_2, \ldots, i_{g_N} \in \pi_{sd} \cap T_N, \end{aligned} \tag{1}$$

$$e_{i_{g_k}}^k \leq \tau_{i_{g_k}} \leq l_{i_{g_k}}^k, \forall k = 1, \ldots, N. \tag{2}$$

Nodes $i_{g_k}, k = 1, \ldots, N$, are called service nodes, since they are used to serve the subsets $T_k, k = 1, \ldots, N$, respectively. A subpath $\pi_{i_{g_k} i_{g_{k+1}}}, k = 1, \ldots, N-1$, is part of a feasible path π_{sd} and is used to connect service nodes i_{g_k} and $i_{g_{k+1}}$. On the one hand, it can be composed of not-service nodes, i.e., $\pi_{i_{g_k} i_{g_{k+1}}} = \left\langle i_{g_k}, u, \ldots, i_{g_{k+1}} \right\rangle$. On the other hand, not-service nodes are not needed to connect i_{g_k} and $i_{g_{k+1}}$, i.e., $\pi_{i_{g_k} i_{g_{k+1}}} = \left\langle i_{g_k}, i_{g_{k+1}} \right\rangle$.

Equation (2) force each service node belonging to a path π_{sd} to be visited within the time window (waiting at a service node is allowed). In a feasible path, the nodes can be visited more than once. However, N nodes, including the source and the destination, must be service nodes, one for every subset $T_k, k = 1, \ldots, N$ and the order imposed by the definition of the subsets must be guaranteed. Thus, a node i, belonging to some subset T_k, can be present in a feasible path π_{sd} as either a service node or a not-service node. In addition, since a node can be part of several subsets T_k, a feasible path can be composed of repeated service nodes. Indeed, a node can be used to serve any of the subsets to which it belongs.

Given a path π_{si}, let

- k be the index of the first subset T_k not served in π_{si};
- i^- be the node that precedes node i in π_{si};
- $\Psi(j, q)$ be a binary predicate indicating whether or not node j serves the subset T_q in π_{si}.

Starting with $\tau_s = 0$, the arrival time to a service node $i_{g_k} \in \pi_{si}$ is defined as follows:

$$\tau_{i_{g_k}} = \max\left\{ e^k_{i_{g_k}}, \tau^k_{i^-_{g_k}} + t_{i^-_{g_k} i_{g_k}} + \Psi(i^-_{g_k}, k-1) \cdot s^{k-1}_{i^-_{g_k}} \right\}; \tag{3}$$

whereas, the arrival time to a not-service node i is defined by the following equation:

$$\tau_i = \tau_{i^-} + t_{i^- i} + \Psi(i^-, k-1) \cdot s^{k-1}_{i^-}. \tag{4}$$

When a node $i \in \pi_{sd}$ can be used to serve a subset T_k, we have to mark i as either a service node or a not-service node. In the former case, we have to consider time window constraint and service time calculating τ_i as Eq. (3). If node i is marked as not-service node, then Eq. (4) is used to determine the arrival time τ_i. Note that when a node $i \in \mathcal{T}$ is used as not-service node, the time window is irrelevant. Moreover each node $i \in V \setminus \mathcal{T}$ is always a not-service node since it does not belong to any subset $T_k, k = 1, \ldots, N$.

For the sake of clarity, Fig. 1 depicts a toy *GSPTPTW* instance with $s = 1$ and $d = 8$. In this simple example, all transit times and all service times are set equal to 1. The costs are reported on the arcs. The path $\pi^1_{18} = \langle 1, 2, 3, 4, 5, 8 \rangle$ is a feasible path tour; the node 2 is the service node for T_2 and T_3, and the node 3 is the service node for T_4. The cost of π^1_{18} is $c(\pi^1_{18}) = 10$. The instance presents a second feasible path tour $\pi^2_{18} = \langle 1, 2, 3, 4, 6, 7, 5, 8 \rangle$. This path is a feasible path tour, because the nodes 2, 6 and 5 are selected as service nodes for sets T_2, T_3 and T_4, respectively. The cost of π^2_{18} is $c(\pi_2) = 8$, and the solution is optimal.

It is worth observing that node 2 belongs to two consecutive sets, i.e., T_2 and T_3. Hence node 2 can be used as service node for both sets. It exists a feasible solution to *GSPTPTW* for the instance reported in Fig. 1 that is $\pi^3_{18} = \langle 1, 2, 2, 3, 4, 6, 7, 5, 8 \rangle$ where node 2 is a service node for both T_2 and T_3 and node 5 is the service node for set T_4, with cost $c(\pi_3) = 8$.

Since *GSPTPTW* is a generalization of *SPTPTW*, each *SPTPTW* instance is also a particular *GSPTPTW* instance, where each node belongs to at most one subset. By this consideration, it follows that the proof in [12] to assert the **NP**-hardness of *SPTPTW* still remains valid for *GSPTPTW*. Therefore, the following result holds:

Theorem 1 *GSPTPTW is an* **NP***-hard problem.*

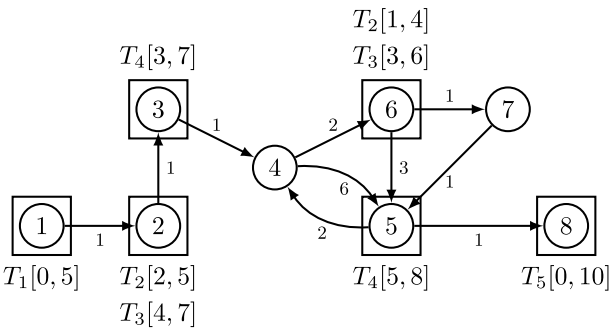

Fig. 1 A toy example

In addition, we can prove that *GSPTPTW* is not harder than *SPTPTW*.

Theorem 2 *GSPTPTW can be polynomially reduced to SPTPTW. Starting from an instance of GSPTPTW, i.e.,* $\mathcal{I}_G\left\langle V, A, \left\{T_k\right\}_{k=1}^{N}\right\rangle$ *it is possible to define an instance* $\mathcal{I}\left\langle V', A', \left\{T'_k\right\}_{k=1}^{N}\right\rangle$ *of SPTPTW. The following two properties are verified.*

1. *Subsets* $T'_k, k = 1, \ldots, N$ *are disjoint sets.*
2. *There exists a feasible path* π_{sd} *for GSPTPTW if and only if there exists a path* π'_{sd} *for SPTPTW such that* $c(\pi'_{sd}) = c(\pi_{sd})$.

Proof First, we describe the steps to construct an instance $\mathcal{I}\left\langle V', A', \left\{T'_k\right\}_{k=1}^{N}\right\rangle$ starting from $\mathcal{I}_G\left\langle V, A, \left\{T_k\right\}_{k=1}^{N}\right\rangle$. Let $\mathcal{K}(v)$ be the set of indices k such that $v \in T_k$ and let $\overline{\mathcal{T}} = \{v \in \mathcal{T} : |\mathcal{K}(v)| > 1\}$. The following operations can be performed:

- for each node $v \in \overline{\mathcal{T}}$, generate $|\mathcal{K}(v)|$ nodes, i.e., $V' = V \cup \{v_k\}_{v \in \overline{\mathcal{T}}, k \in \mathcal{K}(v)}$;
- set $e'^k_{v_k} = e^k_v$; $l'^k_{v_k} = l^k_v$; $s'^k_{v_k} = s^k_v$, $\forall v \in \overline{\mathcal{T}}, k \in \mathcal{K}(v)$;
- $A' = A \cup \{(v, v_k), (v_k, v)\}_{v \in \overline{\mathcal{T}}, k \in \mathcal{K}(v)}$ with $c_{vv_k} = c_{v_k v} = t_{vv_k} = t_{v_k v} = 0$;
- $T'_k = T_k \cup \{v_k\} \setminus \{v\}$, $\forall v \in \overline{\mathcal{T}} \cap T_k, k \in \mathcal{K}(v)$.

It follows that the dimension of the instance $\mathcal{I}$ is greater than that of $\mathcal{I}_G$. In particular, we have

- $|V'| = |V| + \sum_{v \in \overline{\mathcal{T}}} |\mathcal{K}(v)|$;
- $|A'| = |A| + 2 \sum_{v \in \overline{\mathcal{T}}} |\mathcal{K}(v)|$;
- $|T'_k| = |T_k|$.

It is worth observing that the new nodes v_k are linked to the graph through arc (v, v_k) and (v_k, v). This means that to reach node v_k, node v must be traversed.

Property 1 follows. Indeed, each node v such that $|\mathcal{K}(v)| > 1$ is not present in any $T'_k, k \in \mathcal{K}(v)$ and the duplicated nodes v_k of v are added to the associated T'_k, $\forall k \in \mathcal{K}(v)$.

Property 2 can be proved as follows.

$\Rightarrow$ Given a feasible path π_{sd} for *GSPTPTW*, it is possible to construct a path π'_{sd} in $G'(V', A')$, such that $c(\pi'_{sd}) = c(\pi_{sd})$. Let $\mathcal{K}_s(v)$ be the set of indices k for which node $v \in T_k$ is a service node for the path π_{sd}. A feasible path π'_{sd} for the constructed instance of *SPTPTW* can be obtained by adding subpaths $\langle v, v_k, v\rangle$, for all $k \in \mathcal{K}_s(v)$. Node v is marked as not-service node, since $v \notin \bigcup_{k=1}^{N} T'_k$, and the duplicated nodes $v_k, \forall k \in \mathcal{K}_s(v)$ are marked as service nodes for the subsets $T'_k, \forall k \in \mathcal{K}_s(v)$. Since $t_{vv_k} = t_{v_k v} = 0$, path π'_{sd} is feasible with respect to time

window constraints. Thus, paths π_{sd} and π'_{sd} represent the same feasible solution. In addition, being $c\big(\langle v, v_k, v\rangle\big) = 0$, it follows that $c(\pi'_{sd}) = c(\pi_{sd})$.

$\Leftarrow$ Let π'_{sd} be a feasible path in $G'(V', A')$ with cost $c(\pi'_{sd})$, and let v_k be the service node for T'_k in the path π'_{sd}.

For each $k = 1, \ldots, N$, v_k may or may not belong to T_k in G. In the first case, v_k is a service node also for T_k. In the second case, $v_k \notin T_k$ because there exists in G a node v with $|\mathcal{K}(v) > 1|$ and v_k has been created during the construction of the instance I. By construction, the path π'_{sd} must include the sub-path $\pi_{v_k} = \langle v, v_k, v\rangle$, that has both cost and travel time equal to 0. Therefore, the path π_{sd} can be obtained replacing π_{v_k} with the node v that is selected as service node for T_k. The path π_{sd} remains feasible with respect to the edge traversal constrains, since we are traversing a subset of edges of π'_{sd}. Moreover, it remains feasible respect to the time windows, since the service time of v is equal to the service time of v_k and the travel time of π_{v_k} is 0. Finally, $c(\pi'_{sd}) = c(\pi_{sd})$ because $c(\pi_{v_k}) = 0$.

□

Figure 2 reports the graph G' obtained by applying the construction procedure described in Theorem 2 to the toy instance of Fig. 1. Nodes 2_2 (6_2) and 2_3 (6_3) are the copies of node 2 (6) associated with subsets T'_2 and T'_3, respectively. The dotted arcs are those included to connect node 2 (6) with its copies characterized by both cost and time equal to zero. The optimal solution is $\pi^*_{18} = \langle 1, 2, 2_2, 2, 3, 4, 6, 6_3, 6, 7, 5, 8\rangle$, with service nodes $\{1, 2_2, 6_3, 5, 8\}$ and cost $c(\pi^*_{18}) = 8$.

From Theorems 1 and 2, the following result holds.

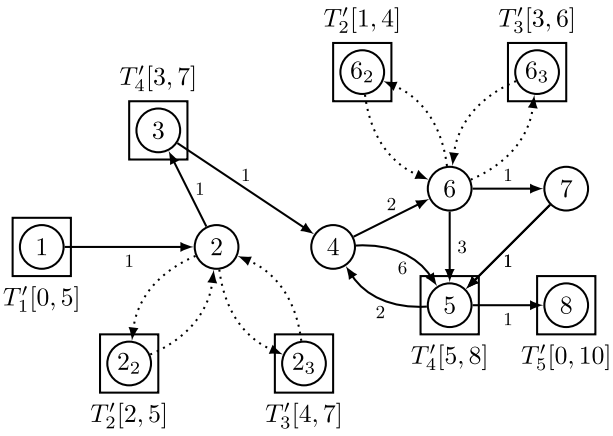

Fig. 2 The graph obtained by applying Theorem 2 to the toy example depicted in Fig. 1

Theorem 3 *GSPTPTW belongs to the same class of complexity of SPTPTW.*

Theorem 3 is a strong result. Indeed, it claims that relaxing the disjoint assumption on the subset $T_k, k = 1, \dots, N$ allows to maintain the same complexity. Thus, assuming that $T_k, k = 1, \dots, N$, are not disjoint subsets does not compromise the computational effort for solving *SPTP* and its variants.

3 Generalized dynamic programming

We represent the solution space of *GSPTPTW* as a state-space $S(Y, \Gamma)$ composed of a set of states Y and a set of transitions Γ. Each state $y_i^h \in Y$ corresponds to a feasible subpath π_{si}^h from the source node s to node $i \in V$. The superscript h means that we consider the $h-$th subpath to reach node $i \in V$. The state $y_i^h(\pi_{si}^h, c_i^h, \tau_i^h, r_i^h)$ is associated with the subpath π_{si}^h with cost $c_i^h = c(\pi_{si}^h)$, arrival time τ_i^h, and $r_i^h = k$ which represents the index of the last served subset T_k along the subpath π_{si}^h. A transition $\gamma_{ij}^{hq} = \left\langle y_i^h, y_j^q \right\rangle \in \Gamma$ exists if $(i, j) \in A$ and both conditions (1) and (2) are satisfied. In particular, the transitions are enabled by control $\phi_s\left(\gamma_{ij}^{hq}\right)$ that allows transition γ_{ij}^{hq} with j marked as service node; and control $\phi_{ns}\left(\gamma_{ij}^{hq}\right)$ that allows transition γ_{ij}^{hq} with j marked as not-service node.

Definition 1 Control $\phi_s\left(\gamma_{ij}^{hq}\right)$ is a boolean function that allows the existence of transition γ_{ij}^{hq}. In particular, if the following conditions hold

$$(i, j) \in A, \tag{5}$$

$$j \in T_{r_i^h+1}, \tag{6}$$

$$\tau_j^q = \max\left\{e_j^{r_i^h+1}, \tau_i^h + t_{ij}\right\} \leq l_j^{r_i^h+1}, \tag{7}$$

then $\phi_s\left(\gamma_{ij}^{hq}\right) = true$ and transition $\gamma_{ij}^{hq} \in \Gamma$, otherwise $\phi_s\left(\gamma_{ij}^{hq}\right) = false$ and $\gamma_{ij}^{hq} \notin \Gamma$.

Definition 2 Control $\phi_{ns}\left(\gamma_{ij}^{hq}\right)$ is a boolean function that allows the existence of transition γ_{ij}^{hq}. In particular, if $(i, j) \in A$, then $\phi_{ns}\left(\gamma_{ij}^{hq}\right) = true$ and transition $\gamma_{ij}^{hq} \in \Gamma$, otherwise $\phi_{ns}\left(\gamma_{ij}^{hq}\right) = false$ and $\gamma_{ij}^{hq} \notin \Gamma$.

Let $\bar{Y}$ be the set of generated states and let M_j be the number of states associated with node j. Algorithm 1 depicts the steps to generate the state-space $S(Y, \Gamma)$. To handle the case where a node belongs to two consecutive sets, Lines 2–9 insert null cost loops. The introduction of these loops allows to obtain tour of the form π_{18}^3 for the instance reported in Fig. 1.

Algorithm 1: Construction of $S(Y, \Gamma)$

STEP 0 (Initialization)
for $k = 2, \ldots, N$ **do**
  foreach $j \in T_k$ **do**
    if $j \in T_{k-1}$ **then**
      Add edge (j, j) to A with cost 0
      break
    end
  end
end
$\pi_{ss}^0 = \langle s \rangle$, $c_s^0 = 0$, $\tau_s^0 = 0$,$r_s^0 = 1$, $y_s^0(\pi_{ss}^0, c_s^0, \tau_s^0, r_s^0)$, $Y = \{y_s^0\}$, $\bar{Y} = \emptyset$, $M_s = 1, M_j = 0, \forall j \in V \setminus \{s\}$
STEP 1 (State selection)
Select a state $y_i^h = \arg\min_{y_i^h \in Y \setminus \bar{Y}} \{c_i^h\}$
$\bar{Y} = \bar{Y} \cup \{y_i^h\}$
STEP 2 (States and transitions generation)
for $(i, j) \in A$ **do**
  if $\phi_{ns}\left(\gamma_{ij}^{hq}\right)$ *with* $q = M_j + 1$ **then**
    Set $M_j = M_j + 1, q = M_j$
    $\pi_{sj}^q = \pi_{si}^h \cup \{j\}$; $c_j^q = c(\pi_{sj}^q)$; $\tau_j^q = \tau_i^h + t_{ij} + \Psi(i, r_i^h) \cdot s_i^{r_i^h}$; $r_j^q = r_i^h$
    $Y = Y \cup \{y_j^q(\pi_{sj}^q, c_j^q, \tau_j^q, r_j^q)\}$
    $\Gamma = \Gamma \cup \{\gamma_{ij}^{hq}\}$
  end
  if $\phi_s\left(\gamma_{ij}^{hq}\right)$ *with* $q = M_j + 1$ **then**
    Set $M_j = M_j + 1, q = M_j$
    $\pi_{sj}^q = \pi_{si}^h \cup \{j\}$; $c_j^q = c(\pi_{sj}^q)$;
    $\tau_j^q = \max\left\{e_j^{r_i^h+1}, \tau_i^h + t_{ij} + \Psi(i, r_i^h) \cdot s_i^{r_i^h}\right\}$; $r_j^q = r_i^h + 1$
    $Y = Y \cup \{y_j^q(\pi_{sj}^q, c_j^q, \tau_j^q, r_j^q)\}$
    $\Gamma = \Gamma \cup \{\gamma_{ij}^{hq}\}$
  end
end
STEP 3 (Termination check)
if $Y \setminus \bar{Y} = \emptyset$ **then**
  STOP
end
else
  Go to Step 1
end

It is worth observing that the construction of the state-space $S(Y,\Gamma)$ implicitly determines all feasible solutions. Thus, among all final states, i.e., those associated with the destination node d, an optimal path π^*_{sd} is associated with that at minimum cost, i.e., $\pi^*_{sd} = \arg\min_{y^h_d \in Y} \{c^h_d\}$. It follows that Algorithm 1 implicitly solves *GSPTPTW*. Thus, it can be used to determine an optimal solution π^*_{sd} to *GSPTPTW*.

Algorithm 1 has exponential time and exponential space complexity. However, not all states associated with feasible solutions have to be generated, rather only those with the potential of generating an optimal solution. In the following, we present some state-space reduction techniques that have a positive impact, as shown by the computational results, on the practical behavior of the proposed solution approach.

3.1 State-space reduction

The state-space $S(Y,\Gamma)$ can be reduced by eliminating states whose transitions do not allow to conduct to potentially optimal final states. This reduction can be done by applying the dominance rule proposed by [12], given below for the sake of completion.

Definition 3 Given two states y^h_i and y^q_i associated with subpath from node s to node i. State y^h_i dominates state y^q_i if the following conditions hold

$$\begin{aligned} c^h_i &\leq c^q_i, \\ \tau^h_i &\leq \tau^q_i, \\ r^h_i &\geq r^q_i, \end{aligned}$$

and at least one inequality is strictly satisfied.

The dominated states are not generated.

Definition 4 Two states y^h_i and y^q_i are said to be equivalent if $c^h_i = c^q_i$, $\tau^h_i = \tau^q_i$, and $r^h_i = r^q_i$.

If a state y^q_i is generated and there exists an equivalent state y^h_i, then the state y^q_i is not stored in Y. This rule does not compromise the optimality of the final solution determined. Indeed, the same feasible sequence of controls and states is generated starting from equivalent states. In addition, having the same cost, also the final states, associated with complete paths, present the same cost. It is also worth to note that discarding equivalent paths allows to prevent cycling on zero cost and zero time cycles, if any.

In addition, we can extend the cost and time boundings proposed in [12].

Cost bounding Given an upper bound Λ on the optimal solution cost $c(\pi^*_{sd})$, all states y^h_i such that $c^h_i \geq \Lambda$ can not be included in Y. The upper bound Λ can be computed

by solving the generalized *SPTP* (*GSPTP*) where the transit and service times are minimized and the time window constraints are removed. In order to improve the cost bounding, given a state y_i^h, we can compute a valid lower bound on the cost of a partial path from node i to node d, named lb_i^h. In particular, it represents the minimum shortest path tour cost to reach the destination node d starting from i. In this case, a state y_i^h can be omitted if $c_i^h + lb_i^h \geq \Lambda$. The lower bound lb_i^h is computed in what follows

$$lb_i^h = \begin{cases} c_{SPT}^{i,r_i^h}, & \text{if } i \text{ is marked as service node for } T_{r_i^h}; \\ \min_{j \in T_{r_i^h+1}} \left\{ c_{SP}^{ij} + c_{SPT}^{j,r_i^h+1} \right\}, & \text{otherwise,} \end{cases} \tag{8}$$

where $c_{SPT}^{i,k}$ is the minimum cost associated with the generalized shortest path tour without time window constraints from node i serving T_k to node d, considering the subsets $T_k, T_{k+1}, \ldots, T_N$, and c_{SP}^{ij} is the cost of the shortest path from node i to node j.

It is worth observing that both $c_{SPT}^{i,k}$ and c_{SP}^{ij} can be computed in polynomial time before applying Algorithm 1. Indeed, *GSPTP* can be polynomially reduced to *SPTP* by applying Theorem 2, whereas c_{SP}^{ij} is computed by applying any algorithm for *SPP*. Even though $c_{SPT}^{i,k}$ and c_{SP}^{ij} can be computed in polynomial time, it is necessary to determine $c_{SPT}^{i,k}$ for each node $i \in \mathcal{T}$ and $k \in \mathcal{K}(i)$, and c_{SP}^{ij} for each pair of nodes $i, j \in V$.

Time bounding Let t_{SP}^{ij} be the shortest transit time from node i to node j. A state y_i^h can not be generated if for all $j \in T_{r_i^h+1}$, the associated time windows are violated, i.e., the state y_i^h is omitted if $\tau_i^h + t_{SP}^{ij} > l_j^{r_i^h+1}, \forall j \in T_{r_i^h+1}$.

Algorithm 2 depicts the steps of the proposed solution approach for solving *GSPTPTW*.

Algorithm 2: GDPB

Step 0 (Preprocessing)
Compute the upper bound Λ by solving *GSPTP* minimizing transit and service times.
Compute $c_{SPT}^{i,k}, \forall i \in \mathcal{T}, k \in \mathcal{K}(i)$ and $c_{SP}^{ij}, \forall i, j \in V$ to obtain the lower bounds on cost as in (8) for performing cost bounding.
Compute $t_{SP}^{ij}, \forall i, j \in V$ to obtain the lower bounds on transit time for performing time bounding.
Step 1 (Resolution)
Apply Algorithm 1 with both cost and time bounding, obtaining an optimal solution $\pi_{sd}^* = \arg\min_{y_d^h \in Y} \left\{ c_d^h \right\}$.

Table 1 Characteristics of the random networks

Problem	Nodes	Arcs	Density
R1	300	1500	5
R2	300	3000	10
R3	300	4500	15
R4	500	2500	5
R5	500	5000	10
R6	500	7500	15
R7	1000	5000	5
R8	1000	10,000	10
R9	1000	15,000	15

Table 2 Characteristics of the grid networks

Problem	Dimension	Nodes	Arcs
G1	25×25	625	2400
G2	30×30	900	3480
G3	50×50	2500	9800
G4	25×50	1250	4850
G5	30×60	1800	7020
G6	50×100	5000	19700

3.2 A* implementation

A* technique [21] is widely used to address constrained and multiobjective shortest path problems [22–24]. In this technique, the cost of each state is evaluated by considering an approximate extension that allows to evaluate the cost of a complete solution starting from the partial path associated with the state. The cost of each state y_i^h is defined as $f_i^h = c_i^h + g_i^h$, where g_i^h is the approximation cost from node i to node d. Hence, A* technique allows to better identify states that have the potential to generate the optimal sequence of controls and states. This behavior is observed also for the problem at hand (see Sect. 4.3.1). In our context, a valid approximation cost g_i^h is represented by the cost of a shortest path tour solution, i.e., g_i^h can be set equal to lb_i^h computed in (8).

4 Computational results

We evaluate the performance of the proposed generalized dynamic programming approach with the aim of showing how the characteristics of the addressed problem influence the behavior of the proposed solution strategy. We compare the solution approach GDPB described in Algorithm 2 for *GSPTPTW* with DPB, the labelling procedure with cost and time bounding proposed in [12] for solving *SPTPTW*. In particular, starting from an instance of *GSPTPTW*, we apply Theorem 2 to construct

an equivalent instance of *SPTPTW*, then the DPB is used to solve the latter. We also analyze the benefit of using A* technique, proved to be efficient for several instances of *SPP*, and *SPTPTW*. The algorithms that implement A* technique are referred in the sequel as GDPBA* and DPBA*.

4.1 Implementation details

All the algorithms were implemented in C++, compiled with `g++ 9.4.0` under Ubuntu 20.04 using the flag `-O3`, and the experiments were run on a INTEL i5-6400@2.70 GHz processor with 8 GB of RAM. The Y set was implemented with the `std::priority_queue` of the standard C++ library, using a `std::vector` as container.

For DPB and GDPB, the comparison value is the label cost c_i^h, meanwhile for DPBA* and GDPBA* the comparison value is the label cost plus the lower bound defined in Eq. (8), i.e., $c_i^h + lb_i^h$. In both cases, the top of the heap stores the label with the minimum comparison value.

In addition to Y, to quickly access the labels of each node i, an n-dimensional array D of `std::list` is used to store pointers to the labels associated with each node. Therefore, when a new state y_i^h is generated for the node i, the list $D[i]$ is scanned to check if it exists a label that dominates y_i^h. If it is not the case, $D[i]$ is scanned again to mark as dominated all the labels that are dominated by y_i^h. These dominated labels are removed by $D[i]$ (in constant time since $D[i]$ is a `std::list`), but they are not removed from Y. Indeed, when the algorithm extracts a label from Y, it checks if the label is marked as dominated and, in this case, it discards the label without analyzing the forward star.

4.2 Instances generation

The instances are constructed from the benchmarks for *SPTP* proposed by [3], who considered three topologies of networks: complete, random, and grid networks. More specifically, the set of networks proposed in [3] is composed of three complete networks, characterized by 100, 300, and 500 nodes, respectively; nine random networks; and six grid networks, whose characteristics (i.e., number of nodes, number of arcs, and density defined as the ratio between the number of arcs and the number of nodes) are reported in Tables 1 and 2, respectively.

Table 3 Comparison between the average performance considering A* at varying the networks topologies

	DPB vs DPBA*		GDPB vs GDPBA*	
	Speed up	1	Speed up	1
Complete	1.08	1.06	1.10	1.09
Grid	2.04	52.34	1.55	58.23
Random	1.07	1.47	1.06	1.77
AVG	1.56	9.09	1.25	9.48

A number of 12 instances are built for each network considering different number of subsets N, i.e., $N \in \{5, 10, 15, 20\}$, and different number of nodes belonging to each subset, i.e., $|T_k| = \left\lfloor \rho \frac{n}{N} \right\rfloor$ with $\rho \in \left\{ \frac{1}{3}, \frac{1}{2}, 1 \right\}$. In order to generate the instances for *GSPTPTW*, we modify the instances for *SPTP* considering different parameters $\%T$ and $\%n$, where

- $\%\mathrm{T} \in \{20, 30, 40\}$ defines the percentage of subsets T whose nodes belong to any other subset.
- $\%\mathrm{n} \in \{20, 45, 60\}$ defines the percentage of nodes shared among the subsets T.

In particular, the conversion procedure takes as input an instance of *SPTP* and gives as output an instance of *GSPTPTW* applying the following modifications:

- for all $h = 2, \ldots, N-1$, randomly pick $\frac{\%n \cdot |T_h|}{100}$ nodes of T_h with uniform probability, and insert them in $\frac{\%T \cdot N}{100}$ different subsets $T_i, i \neq h$;
- generate time windows applying the procedure used in [12] for *SPTPTW*.

The parameters %T and %n characterize *GSPTPTW* instances. Indeed, for %T = %n = 0, *GSPTPTW* instances are actually instances of *SPTPTW*. Thus, the higher the value of %T and %n, the higher the difference between *GSPTPTW* and *SPTPTW* instances.

4.3 Numerical results

In this section, we present the numerical results collected on the considered instances. We first analyze the effectiveness of using A* technique (DPBA* and GDPBA*) with respect to the Dijkstra-like rule (DPB and GDPB). Then, we show the behavior of the proposed algorithms with respect to %T and %n. The behavior of the algorithms is evaluated by considering the following parameters:

Table 4 Numerical results comparing the algorithms with and without A* technique

	tp	ta	t	#l	tp	ta	t	#l
	GDPB				GDPBA*			
Complete	5.34	27.48	32.82	19339.37	5.45	24.46	29.91	17646.60
Grid	6.02	6.24	12.26	107088.02	7.55	0.37	7.92	1838.97
Random	0.44	0.15	0.59	2660.77	0.44	0.12	0.55	1502.22
AVG	3.93	11.29	15.22	43029.39	4.48	8.32	12.80	6995.93
	DPB				DPBA*			
Complete	6.88	27.84	34.72	38593.49	6.84	25.20	32.04	36305.68
Grid	16.55	19.55	36.10	233130.99	16.59	1.13	17.72	4453.53
Random	1.06	0.47	1.53	6945.48	1.06	0.37	1.43	4711.58
AVG	8.16	15.95	24.12	92889.99	8.16	8.90	17.06	15156.93

- tp, the execution time, in seconds, required by the preprocessing. It represents the time needed to compute the bounds for GDPB and GDPBA*, whereas, for DPB and DPBA*, tp is the time to construct the graph by following Theorem 2 and the time to compute the bounds;
- ta, the execution time, in seconds, of the dynamic programming algorithm;
- t, the overall execution time, i.e., t = tp + ta;
- #l, the number of examined states/labels.

4.3.1 Evaluation of A* technique

Preliminaries Before to examine in details the numerical results considering tp, ta, t, and #l, we give an overview of the effectiveness in using A* techniques at varying the network topologies. In particular, in Table 3 we shows the speed up and the number of labels generated without A* over those generated when A* is considered (column l).

Table 3 highlights that the highest benefit obtained by using A* technique is observed for grid networks, followed by complete and random ones. In particular, for grid networks, a huge reduction in the number of labels examined is observed. This behavior influences the overall execution time. Indeed, the speed up is 2.04 and 1.55 for DPBA* and GDPBA*, respectively.

Table 5 Average numerical results at varying %T

		DPBA*				GDPBA*			
	%T	tp	ta	t	#l	tp	ta	t	#l
Complete	20	4.88	37.82	42.69	38404.45	4.10	36.42	40.52	21695.16
	30	7.15	22.27	29.42	34876.04	5.61	21.78	27.39	16671.06
	40	8.49	15.52	24.02	35636.56	6.62	15.19	21.81	14573.59
	AVG	6.84	25.20	32.04	36305.68	5.45	24.46	29.91	17646.60
Random	20	0.70	0.22	0.93	3807.44	0.37	0.11	0.48	1636.64
	30	1.07	0.36	1.43	4747.70	0.44	0.11	0.56	1463.13
	40	1.39	0.52	1.92	5579.62	0.50	0.13	0.63	1406.91
	AVG	1.06	0.37	1.43	4711.58	0.44	0.12	0.55	1502.23
Grid	20	10.72	1.20	11.92	4726.24	5.22	0.48	5.70	2342.29
	30	18.44	1.09	19.52	4487.44	7.73	0.34	8.07	1768.11
	40	20.68	1.10	21.78	4142.77	9.70	0.30	10.00	1406.52
	AVG	16.61	1.13	17.74	4452.15	7.55	0.37	7.92	1838.97
AVG	20	5.43	13.07	18.51	15646.04	3.23	12.33	15.56	8558.02
	30	8.88	7.90	16.79	14703.72	4.59	7.41	12.00	6634.10
	40	10.19	5.72	15.90	15119.65	5.61	5.21	10.81	5795.67
	AVG	8.17	8.90	17.07	15156.47	4.48	8.32	12.80	6995.93

Detailed results Table 4 shows the average values of tp, ta, t, and #l at varying the network topologies. The results collected clearly show the benefit in terms of both execution time and number of generated labels obtained using A* technique. This consideration is valid for both DPB and GDPB.

In particular, DPBA* is 1.56 times faster than DPB, on average. This behavior is justified by the smaller number of generated labels. Indeed, DPBA* shows for #l a value that is 9.09 times lower than the value of #l observed for DPB, on average. The same trend is observed for GDPBA*. In particular, GDPBA* is 1.25 times faster than GDPB, and the former examines 9.48 times less number of labels than the latter, on average. We observe that the A* technique is more performing for DPB than for GDPB.

Table 4 highlights the better behavior of GDPB and GDPBA* with respect to DPB and DPBA*, respectively. In particular, GDPB is 1.58 times faster than DPB. This behavior is justified by the number of generated labels. Indeed, GDPB examines 2.15 times less number of labels than DPB, on average. The same trend is observed when A* is applied. In particular, GDPBA* is 1.33 times faster than DPBA* and the former examines 2.16 times less number of labels than those generated by DPBA*. One can readily see that the better performance of GDPB is mitigated when the A* technique is included. However, GDPBA* remains the best performing algorithm.

Due to the high effectiveness of using A* technique, in the following we consider DPBA* and GDPBA* to analyze the behavior of both the algorithms at varying %T and %n.

4.3.2 Evaluation at varying %T

Table 5 shows the average results, varying %T for both DPBA* and GDPBA*. For the sake of clarity, first, we analyze the behavior of DPBA* and GDPBA* separately, then we compare the two algorithms at varying the parameter %T.

Results for DPBA * *varying %T* Table 5 shows that the higher %T, the lower t, on average. Indeed, the overall execution time t for %T = 40 is 1.05 and 1.16 times lower than that observed for %T equal to 30 and 20, respectively. We observe that the preprocessing phase follows an inverted trend. Indeed, it requires more computational overhead for higher value of %T, on average. In particular, tp for %T = 40 is 1.14 and 1.87 times higher than that observed for %T equal to 30 and 20, respectively. This is an expected trend since the higher %T, the higher the number of nodes belonging to different subsets T_k. On the one hand, this increases the number of bounds to be computed, on the other hand, more nodes have to be included in the modified graph (see Theorem 2).

The execution time required by the algorithm, i.e., ta, decreases for high value of %T. Indeed, ta for %T equal to 40 is 1.38 and 2.28 times lower than that observed for %T equal to 30 and 20, respectively.

The decreasing of ta for higher value of %T suffices the increasing of tp, thus an overall reduction of computational overhead t is observed for high values of %T.

The higher %T, the higher tp. This trend is observed for each topology of network. Looking at Table 5, the trend of ta at varying %T is strongly influenced by the results obtained on complete network. Indeed, this topology of network is the hardest to solve. In particular, DPBA* requires ta = 25.20 s, whereas, it is equal to 0.37 and 1.13 for random and grid networks, respectively. For random networks, ta follows the same trend of tp, whereas, for grid networks, ta does not follow a regular trend varying %T. The values of ta decrease for increasing values of %T for complete networks. This trend influences the average one.

***Results for GDPBA** * *at varying %T* Table 5 shows that the computational overhead of GDPBA* decreases when %T increases. In particular, the value of t when %T is equal to 40 is 1.13 and 1.62 times lower than that observed for %T equal to 30 and 20, respectively, on average.

This behavior is influenced by ta. In particular, ta for %T = 40 is 1.42 and 2.36 times lower than that observed for %T equal to 30 and 20, respectively. The observed trend for ta is related to the number of examined labels. Indeed, GDPBA* examines for %T = 40, 1.47 and 1.14 times lower number of labels than those generated with %T equal to 30 and 20, respectively.

The preprocessing execution time follows the same trend observed for DPBA*. Indeed, tp for %T = 40 is 1.22 and 1.73 times higher than that observed for %T

Table 6 Average numerical results varying %n

		DPBA*				GDPBA*			
	%n	tp	ta	t	#l	tp	ta	t	#l
Complete	30	5.05	30.27	35.33	34473.55	4.23	29.51	33.74	19281.23
	45	6.90	23.01	29.91	35281.75	5.45	22.27	27.72	17038.40
	60	8.57	22.33	30.90	39161.75	6.66	21.61	28.27	16620.19
	AVG	6.84	25.20	32.04	36305.68	5.45	24.46	29.91	17646.60
Random	30	0.72	0.20	0.92	3840.78	0.37	0.10	0.47	1426.10
	45	1.03	0.36	1.39	4769.10	0.44	0.12	0.55	1506.55
	60	1.42	0.55	1.97	5524.87	0.50	0.14	0.64	1574.03
	AVG	1.06	0.37	1.43	4711.58	0.44	0.12	0.55	1502.23
Grid	30	11.04	1.03	12.08	4467.88	5.42	0.41	5.83	2096.85 [t]
	45	17.07	1.00	18.07	4190.53	7.43	0.32	7.75	1697.53
	60	21.74	1.36	23.09	4704.46	9.80	0.40	10.20	1722.54 [b]
	AVG	16.62	1.13	17.75	4454.29	7.55	0.37	7.92	1838.97
AVG	30	5.60	10.50	16.11	14260.74	3.34	10.01	13.35	7601.39 [t]
	45	8.33	8.12	16.46	14747.13	4.44	7.57	12.01	6747.49
	60	10.58	8.08	18.65	16463.69	5.66	7.38	13.04	6638.92 [b]
	AVG	8.17	8.90	17.07	15157.19	4.48	8.32	12.80	6995.93

equal to 30 and 20. We highlight that for GDPBA* the preprocessing refers only to the computation of the bounds.

The decreasing of ta for increasing values of %T is higher than the increasing of tp, thus an overall decreasing of t is observed for high values of %T.

The trend of tp varying %T is the same for each topology of network. Whereas, ta follows the average trend for both complete and grid networks. For the random ones, a clear trend is not observed. Since the most difficult instances to be solved are those based on complete networks, the average trend is influenced by the results obtained for the latters.

Comparing DPBA* and GDPBA* at varying %T Table 5 highlights that GDPBA* outperforms DPBA* for each value of %T. Indeed, GDPBA* is 1.18, 1.39, and 1.47 times faster than DPBA* for %T equals to 20, 30, and 40, respectively, on average. We highlight that the speed up of GDPBA* with respect to DPBA* increases for increasing value of %T. This is an expected trend, since the higher %T, the higher the difference between *GSPTPTW* and *SPTPTW* instances. Thus, the tailored algorithm GDPBA* for the *GSPTPTW* behaves better than DPBA* applied on the *SPTPTW* instances constructed from the *GSPTPTW* ones.

The worst behavior of DPBA* is mainly due to the preprocessing phase. Indeed, tp for DPBA* is 1.68, 1.97, and 1.81 times higher that that observed by GDPBA* for %T equal to 20, 30, and 40, respectively. In addition, considering only ta, GDPBA* behaves the best. In particular, ta for GDPBA* is 1.06, 1.06, and 1.09 times lower than that observed for DPBA*, considering %T equal to 20, 30, and 40, respectively. This behavior is justified by the number of labels examined by GDPBA*. In particular, the latter generates, on average, 1.82, 2.21, and 2.60 times lower labels than those generated by DPBA* for %T equal to 20, 30, and 40, respectively.

The difference between the two approaches are quite impressive for grid networks. In particular, GDPBA* is 2.09, 2.41, and 2.17 times faster than DPBA* and the former examines 2.01, 2.53, and 2.94 times less labels than those generated by the latter for %T equal to 20, 30, and 40, respectively.

4.3.3 Evaluation at varying %n

Table 6 reports the average results at varying the parameter %n for each topology of network.

We first analyse the behavior of DPBA* and GDPBA* separately, then a comparison between the two algorithms is provided considering the values of %n.

Results for DPBA* at varying %n Table 6 shows that the higher %n, the higher the computational overhead. Indeed, t for %n = 60 is 1.15 and 1.13 times higher than the value of t observed for %n equal to 45 and 30, respectively. This behavior is influenced by the preprocessing time. Indeed, the higher %n, the higher tp. An inverted trend is observed for ta. In particular, ta shows the same value for %n equal to both 60 and 45. Whereas, ta for %n = 30 is 1.29 times higher than ta for %n equal to 60.

This behavior is justified by the number of examined labels. In particular, DPBA* for %n equal to 60, examines a number of labels 1.11 and 1.15 times higher than those generated with %n equal to 45 and 30, respectively.

The trend of tp at varying %n is the same for each topology of network. In particular, it increases when %n increases. This is an expected trend. Indeed, for higher value of %n, on the one hand, a higher number of bounds have to be computed, on the other hand, a higher number of copies of the nodes have to be included in the modified graph.

The profile of ta at varying %n is different for the three topologies of networks. In particular, ta decreases for increases values of %n for complete networks. An clear trend is not observed for both random and grid networks. This behavior is related to the number of generated labels (see column #l of Table 6).

Results for GDPBA* at varying %n The overall computational overhead t seems to not be influenced by the value of %n. Indeed, it is not possible to drawn a trend on average. However, looking at tp and ta, Table 6 highlights a strong correlation between both tp and ta and %n. In particular, tp increases when %n increases. An inverted trend is observed for ta. The latter behavior is justified by the number of examined labels. Indeed, #l for %n equal to 60 is 1.01 and 1.14 times lower than the values of #l observed for %n equal to 45 and 30, respectively.

The average trend is influenced by the behavior of GDPBA* for both complete and grid networks. Indeed, ta increases for lower value of %n due to the increasing of generated labels. The trend is inverted for random networks but the average computational overhead is lower than that observed for both complete and grid networks.

Comparing DPBA* and GDPBA* at varying %n Table 6 shows the better behavior of GDPBA* than DPBA*, see column t. Indeed, the former is 1.07, 2.23, and 2.57 times faster than the latter for %n equal to 30, 45, and 60, respectively, on average.

As observed in the previous sections, the preprocessing for DPBA* requires more computational overhead than that observed for GDPBA*, and the difference of tp between the two algorithms increases for increasing values of %n. This justifies the good behavior of GDPBA* for higher values of %n. However, considering only ta, GDPBA* remains the best performing algorithm. Indeed, we observe a speed up equal to 1.02, 1.03, and 1.03 for complete networks, 2.09, 3.04, and 3.96 for random networks, and 2.53, 3.12, and 3.42 for grid networks, considering values of %n equal to 30, 45, and 60, respectively.

4.4 Final remarks

The numerical results highlight the effectiveness of the proposed solution approach tailored for *GSPTPTW*, i.e., GDPBA*. For high values of both %T and %n, *GSPTPTW* instances are more demanding in terms of computational effort to be solved to

optimality. Whilst GDPBA* behaves quite similar to DPBA* for low values of both %T and %n, the former remarkably outperforms DPBA* for high values of both %T and %n.

5 Conclusions and future work

In this paper, we presented the Generalized Shortest Path Tour Problem with Time Windows (*GSPTPTW*) where the assumption on disjoint subsets $T_k, k = 1, \dots, N$ is relaxed. Thus, each node can belong to different subsets T_k and have a possibly different time window associated with each subset it belongs. We prove that *GSPTPTW* belongs to the same class of complexity of *SPTPTW*. This is a strong result since it is possible to consider any definition of subsets T_k without compromising the complexity. We also presented a polynomial procedure to reduce a *GSPTPTW* instance to a *SPTPTW* instance.

We proposed a dynamic programming algorithm for solving *GSPTPTW*, named GDPB, and extended the cost and time bounding procedures proposed in [12]. GDPB was compared with DPB, the labelling procedure of [12], which was used to solve *SPTPTW* instances derived from *GSPTPTW* instances, by applying the proposed polynomial reduction procedure. In addition, we implemented an A* version of both GDPB and DPB.

We conducted an extensive experimental phase, in order to both study the impact of A* technique and compare the two algorithms. We considered instances generated from benchmarks for *SPTP*. The results showed that A* technique strongly increases the efficiency of the approaches. Indeed, A* speeds up both GDPB and DPB due to a reduction of the number of generated labels.

The numerical results reveal that GDPB outperforms DPB. This behaviour is justified by the lower computational overhead of both the preprocessing phase and the generalized dynamic programming approach. In addition, the higher the number of nodes belonging to different subsets, the better the performance of GDPB with respect to DPB.

As future work, it could be interesting to study *GSPTPTW* with uncertain data, using the concept of simheuristics [25] or robust opitmization [26, 27].

Funding Open access funding provided by Universitá della Calabria within the CRUI-CARE Agreement.

Data availability The data and the codes that support the findings of this study are available at the following link: https://doi.org/10.6084/m9.figshare.12444791.

Declarations

Conflict of interest The authors have no conflict of interests to declare.

References

1. Bajaj, C.P.: Some constrained shortest-route problems. Math. Methods Oper. Res. **15**(1), 287–301 (1971). https://doi.org/10.1007/BF01939836
2. Festa, P.: Complexity analysis and optimization of the shortest path tour problem. Optim. Lett. **6**, 163–175 (2012). https://doi.org/10.1007/s11590-010-0258-y
3. Festa, P., Guerriero, F., Laganà, D., Musmanno, R.: Solving the shortest path tour problem. Eur. J. Oper. Res. **230**, 464–474 (2013). https://doi.org/10.1016/j.ejor.2013.04.029
4. Bhat, S., Rouskas, G.N.: Service-concatenation routing with applications to network functions virtualization. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN), pp. 1–9. (2017). https://doi.org/10.1109/icccn.2017.8038463
5. Carrabs, F., Cerulli, R., Festa, P., Laureana, F.: On the forward shortest path tour problem. In: Sforza, A., Sterle, C. (eds.) Optimization and Decision Science: Methodologies and Applications, pp. 529–537. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-67308-0_53
6. Carrabs, F., D'Ambrosio, C., Ferone, D., Festa, P., Laureana, F.: The constrained forward shortest path tour problem: mathematical modeling and grasp approximate solutions. Networks **78**(1), 17–31 (2021). https://doi.org/10.1002/net.22010
7. Ferone, D., Festa, P., Guerriero, F., Laganà, D.: The constrained shortest path tour problem. Comput. Oper. Res. **74**, 64–77 (2016). https://doi.org/10.1016/j.cor.2016.04.002
8. de Andrade, R.C., Saraiva, R.D.: An integer linear programming model for the constrained shortest path tour problem. Electr. Notes Discrete Math. **69**, 141–148 (2018). https://doi.org/10.1016/j.endm.2018.07.019
9. Ferone, D., Festa, P., Guerriero, F.: An efficient exact approach for the constrained shortest path tour problem. Optim. Methods Softw. **35**, 1–20 (2020). https://doi.org/10.1080/10556788.2018.1548015
10. Saraiva, R.D., de Andrade, R.C.: Constrained shortest path tour problem: models, valid inequalities, and lagrangian heuristics. Int. Trans. Oper. Res. **28**(1), 222–261 (2021)
11. Martin, S., Magnouche, Y., Juvigny, C., Leguay, J.: Constrained shortest path tour problem: branch-and-price algorithm. Comput. Oper. Res. **144**, 105819 (2022). https://doi.org/10.1016/j.cor.2022.105819
12. Di Puglia Pugliese, L., Ferone, D., Festa, P., Guerriero, F.: Shortest path tour problem with time windows. Eur. J. Oper. Res. **282**(1), 334–344 (2020). https://doi.org/10.1016/j.ejor.2019.08.052
13. Moccia, L., Cordeau, J.-F., Laporte, G.: An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. J. Oper. Res. Soc. **63**(2), 232–244 (2012). https://doi.org/10.1057/jors.2011.25
14. Desrosiers, J., Pelletier, P., Soumis, F.: Plus court chemin avec contraintes d'horaires. RAIRO-Oper. Res. **17**(4), 357–377 (1983). https://doi.org/10.1051/ro/1983170403571
15. Di Puglia Pugliese, L., Guerriero, F.: A survey of resource constrained shortest path problems: exact solution approaches. Networks **62**(3), 183–200 (2013). https://doi.org/10.1002/net.21511
16. Desrochers, M., Soumis, F.: A generalized permanent labelling algorithm for the shortest path problem with time windows. INFOR Inf. Syst. Oper. Res. **26**(3), 191–212 (1988). https://doi.org/10.1080/03155986.1988.11732063
17. Gallo, G., Pallottino, S.: Shortest path algorithms. Ann. Oper. Res. **13**(1), 1–79 (1988). https://doi.org/10.1007/bf02288320
18. Guerriero, F., Di Puglia Pugliese, L., Macrina, G.: A rollout algorithm for the resource constrained elementary shortest path problem. Optim. Methods Softw. **34**(5), 1056–1074 (2019). https://doi.org/10.1080/10556788.2018.1551391
19. Powell, W.B., Chen, Z.: A generalized threshold algorithm for the shortest path problem with time windows. In: Pardalos, P.M., Du, D. (Eds.), Network Design: Connectivity and Facilities Location, Proceedings of a DIMACS Workshop, Princetin, New Jersey, USA, April 28–30, 1997, Vol. 40 of

DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS/AMS, pp. 303–318. (1998). https://doi.org/10.1090/dimacs/040/18
20. Ghiani, G., Improta, G.: An efficient transformation of the generalized vehicle routing problem. Eur. J. Oper. Res. **122**(1), 11–17 (2000). https://doi.org/10.1016/S0377-2217(99)00073-9
21. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968). https://doi.org/10.1109/TSSC.1968.300136
22. Pulido, F.-J., Mandow, L., Pérez-de-la Cruz, J.-L.: Dimensionality reduction in multiobjective shortest path search. Comput. Oper. Res. (2015). https://doi.org/10.1016/j.cor.2015.05.007
23. de las Casas, P.M., Kraus, L., Sedeño-Noda, A., Borndörfer, R.: Targeted multiobjective DIJKSTRA algorithm (2021). arXiv:2110.10978
24. Ferone, D., Festa, P., Fugaro, S., Pastore, T.: A dynamic programming algorithm for solving the k-color shortest path problem. Optim. Lett. **15**(6), 1973–1992 (2020). https://doi.org/10.1007/s11590-020-01659-z
25. Ferone, D., Festa, P., Gruler, A., Juan, A.A.: Combining simulation with a grasp metaheuristic for solving the permutation flow-shop problem with stochastic processing times. In: Winter Simulation Conference (WSC), pp. 2205–2215 (2016). https://doi.org/10.1109/WSC.2016.7822262
26. Di Puglia Pugliese, L., Guerriero, F., Poss, M.: The resource constrained shortest path problem with uncertain data: a robust formulation and optimal solution approach. Comput. Oper. Res. **107**, 140–155 (2019). https://doi.org/10.1016/j.cor.2019.03.010
27. Pessoa, A.A., Di Puglia Pugliese, L., Guerriero, F., Poss, M.: Robust constrained shortest path problems under budgeted uncertainty. Networks **66**(2), 98–111 (2015). https://doi.org/10.1002/net.21615