European Journal of Operational Research 230 (2013) 464–474

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solving the shortest path tour problem

P. Festa [a], F. Guerriero [b,*], D. Laganà [b], R. Musmanno [b]

[a] Department of Mathematics and Applications, University of Napoli "Federico II", Compl. MSA, Via Cintia, 80126 Naples, Italy
[b] Department of Mechanical, Energy and Management Engineering, University of Calabria, Ponte Pietro Bucci, Building 41/C, 87036 Arcavacata di Rende (CS), Italy

ARTICLE INFO

Article history:
Received 27 March 2012
Accepted 16 April 2013
Available online 7 May 2013

Keywords:
Network flows
Shortest path
Structure path constraints
Labeling method

ABSTRACT

In this paper, we study the shortest path tour problem in which a shortest path from a given origin node to a given destination node must be found in a directed graph with non-negative arc lengths. Such path needs to cross a sequence of node subsets that are given in a fixed order. The subsets are disjoint and may be different-sized. A polynomial-time reduction of the problem to a classical shortest path problem over a modified digraph is described and two solution methods based on the above reduction and dynamic programming, respectively, are proposed and compared with the state-of-the-art solving procedure. The proposed methods are tested on existing datasets for this problem and on a large class of new benchmark instances. The computational experience shows that both the proposed methods exhibit a consistent improved performance in terms of computational time with respect to the existing solution method.

1. Introduction

The shortest path tour problem ($\mathcal{SPTP}$) is a variant of the shortest path problem ($\mathcal{SPP}$) and appeared for the first time in the scientific literature in Bertsekas's dynamic programming and optimal control book [2].

The $\mathcal{SPTP}$ has been recently studied by Festa in [13]. The paper of Festa is the first systematic contribution for solving the $\mathcal{SPTP}$. The author proved that the problem belongs to the complexity class **P**. The polynomial Karp-reduction of the $\mathcal{SPTP}$ to the single-source single-destination $\mathcal{SPP}$ involves the construction of an expanded graph in which different algorithms for the $\mathcal{SPP}$ were tested and compared on pseudo-randomly generated instances. The results presented in [13] showed that Dijkstra's algorithm outperforms all the competitor algorithms.

Applications of the $\mathcal{SPTP}$ arise for example in the context of the manufacture workpieces, where a robot has to perform at least one operation selected from a set of S types of operations. In such case, the problem may be modeled as a $\mathcal{SPTP}$ in which operations are associated with nodes of a directed graph and the time needed for a tool change is represented by the distance between two nodes (see [13]).

The main scientific contribution of this paper consists in analyzing some basic theoretical properties of the $\mathcal{SPTP}$, in designing a dynamic programming-based algorithm ($\mathcal{DPA}$) for solving it, and showing how an ad hoc algorithm for acyclic graphs may be used to solve the $\mathcal{SPTP}$ after efficiently reducing it to a classical $\mathcal{SPP}$ through the method referred to as modified graph algorithm ($\mathcal{MGA}$).

The remainder of the paper is organized as follows. The problem is formally described in Section 2. The state-of-the-art algorithm to address the $\mathcal{SPTP}$ is presented in Section 3. Some properties concerning the reducibility of the problem to a classical $\mathcal{SPP}$ and the relevant consequences in terms of solvability are described in Section 4. A dynamic programming algorithm is illustrated in Section 5. Computational results and the analysis of the performance of the proposed algorithms are presented in Section 6. The paper ends with some concluding remarks stated in Section 7.

2. Problem description

Consider a directed graph $G = (N,A)$ defined by a set of nodes $N := \{1, \ldots, n\}$ and a set of arcs $A := \{(i,j) \in N \times N: i \neq j\}$, where $|A| = m$. A non-negative length c_{ij} is assigned to each arc $(i,j) \in A$. Let $F(i) := \{j \in N: (i,j) \in A\}$ and $B(i) := \{j \in N: (j,i) \in A\}$ be the *forward star* and *backward star* associated with each node $i \in N$, respectively. Moreover, let S denote a certain number of node subsets $T_1, \ldots, T_S$ such that $T_h \cap T_k = \emptyset, h, k = 1, \ldots, S, h \neq k$.

Given two nodes $i_1, i_\pi \in N, i_1 \neq i_\pi$, the *path* P_{i_1,i_π} from i_1 to i_π is defined as a sequence of nodes $P_{i_1,i_\pi} = \{i_1, \ldots, i_\pi\}$ such that $(i_j, i_{j+1}) \in A, j = 1, \ldots, \pi - 1$. Observe that $i_j, j = 1, \ldots, \pi$, represents the node index occurring in position j in path P_{i_1,i_π}. A path P_{i_1,i_π} is said to be *elementary* whether $i_l \neq i_j$, $l, j = 1, \ldots, \pi$ and $l \neq j$. We refer to the length of path P_{i_1,i_π} as $l(P_{i_1,i_\pi})$ representing the sum of the lengths of the arcs connecting consecutive nodes in P_{i_1,i_π}, i.e., $l(P_{i_1,i_\pi}) = \sum_{j=1}^{\pi-1} c_{j,j+1}$.

The $\mathcal{SPTP}$ aims at finding a shortest path $\mathcal{P}_{s,d}$ from origin node $s \in V$ to destination node $d \in V$ in the directed graph G, such that it

* Corresponding author. Tel.: +39 0984 494620.
E-mail addresses: paola.festa@unina.it (P. Festa), guerrier@unical.it (F. Guerriero), demetrio.lagana@unical.it (D. Laganà), musmanno@unical.it (R. Musmanno).

0377-2217/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2013.04.029

visits successively and sequentially the following subsets T_k, $k = 0, \ldots, S+1$, such that $T_0 = \{s\}$ and $T_{S+1} = \{d\}$. Note that sets T_k, $k = 1, \ldots, S$, must be visited in exactly the same order in which they are defined.

Consequently, a path P_{i_1,i_π} is said to be a *feasible solution* for the $\mathcal{SPTP}$ if:

$$\begin{aligned} &\exists\, g_0, g_1 \ldots g_{S+1} \in [1,\pi] : g_0 < g_1 < \cdots < g_{S+1}, \\ &i_{g_0} \in P_{i_1,i_\pi} \cap T_0, i_{g_1} \in P_{i_1,i_\pi} \cap T_1, \ldots, i_{g_{S+1}} \in P_{i_1,i_\pi} \cap T_{S+1}. \end{aligned} \tag{1}$$

Conditions (1) mean that an increasing sequence of natural numbers exists such that the corresponding nodes of the path P_{i_1,i_π} belong to the ordered sequence of subsets $T_0, T_1, \ldots, T_{S+1}$. A small instance of the $\mathcal{SPTP}$ is depicted in Fig. 1, where $N = \{s = 1, 2, 3, 4, 5, 6, d = 7\}$, $S = 2, T_0 = \{s = 1\}$, $T_1 = \{3\}$, $T_2 = \{2, 4\}$, $T_3 = \{d = 7\}$. The shortest path from node 1 to node 7 is $\mathcal{P}_{1,7} = \{1, 3, 7\}$ with length 5, while the shortest path tour between the same origin and destination nodes is $P_{1,7} = \{1, 3, 2, 3, 7\}$ with length 11. Such path is not elementary, since it passes twice through node 3.

3. The state-of-the-art

The state-of-the-art consists of the expanded graph method proposed by Festa in [13], and referred to as $\mathcal{EGA}$ in the sequel. A brief description of how the $\mathcal{EGA}$ works is given in the following.

The $\mathcal{EGA}$ relies on a polynomial-time reduction algorithm that transforms any $\mathcal{SPTP}$ instance defined on a single-stage graph G into a single-source single-destination $\mathcal{SPP}$ instance defined on a multi-stage graph $G' = (V', A')$ with $S+2$ stages, each replicating G, and such that $V' = \{1, \ldots, (S+2)n\}$ and $|A'| = (S+1)m$. More precisely, the reduction algorithm performs the following operations:

i. $V' := \{1, \ldots, (S+2)n\}$; $A' := \emptyset$;
ii. at each iteration, an arc(a,b) is added to A'. In particular, for each stage $k \in \{0, \ldots, S\}$, for each node $v \in \{1, \ldots, n\}$, and for each adjacent node $w \in FS(v)$, $(a,b) = (v + kn, w + (k+1)n)$ with length c_{vw}, if $w \in T_{k+1}$; $(a,b) = (v + kn, w + kn)$ with length c_{vw}, otherwise.

Since $|A'| = (S+1)m$, the computational complexity of the reduction algorithm is $O(Sm)$.[1] Once the multi-stage graph G' is obtained, to solve the resulting $\mathcal{SPP}$ any shortest path algorithm can be applied. By applying Dijkstra's algorithm that uses a binary heap for storing temporary node labels, the overall worst case computational complexity of $\mathcal{EGA}$ is $O(|A'|\log|V'| + |V'|\log|V'|)$, which is dominated by $O(|A'|\log|V'|)$, that is $O(Sm\log n)$.

4. A modified graph method

In this section, we will focus on some basic properties related to the reducibility of any $\mathcal{SPTP}$ instance into a single-source single-destination $\mathcal{SPP}$ instance.

4.1. On the reduction of the $\mathcal{SPTP}$ to the $\mathcal{SPP}$

Given an instance of the $\mathcal{SPTP}$ on a directed graph $G = (N, A)$ the following definition is applied.

Definition 1. Let $G^{(a)} = (N^{(a)}, A^{(a)}, c^{(a)})$ be a weighted directed graph obtained from G in such a way that:

- $N^{(a)} = \bigcup_{k=0}^{S+1} T_k$;
- $A^{(a)} = \bigcup_{k=0}^{S} A_k^{(a)}$, where $A_k^{(a)} := \{(i,j) \in T_k \times T_{k+1} : i \in T_k \text{ and } j \in T_{k+1}\}$;
- $c^{(a)} : A^{(a)} \mapsto \mathbb{Z}^+$ is a function that associates an integer non-negative number $c_{ij}^{(a)}$ to each arc $(i,j) \in A^{(a)}$, where $c_{ij}^{(a)} := l(P_{i,j})$ is the length of a shortest path from node $i \in T_k$ to node $j \in T_{k+1}$ on graph G.

The following property holds for the arc set $A^{(a)}$.

Property 1. *Let $G^{(a)}$ be the weighted directed graph associated with an instance of the $\mathcal{SPTP}$, then $|A^{(a)}| \leqslant n(n-2)$.*

Solving a $\mathcal{SPTP}$ instance may be performed by finding a shortest path in $G^{(a)}$. Indeed, the following property holds.

Property 2. *Every path $P_{s,d}^{(a)}$ from s to d in $G^{(a)}$ defines a $\mathcal{SPTP}$ solution in G with the same cost, and vice versa.*

Such a property derives from the construction of graph $G^{(a)}$ given in Definition 1.

The optimal cost of any $\mathcal{SPTP}$ instance is equal to the cost of the shortest path from node s to node d computed on $G^{(a)}$, as shown in the following property.

Property 3. *The cost of a shortest path $\mathcal{P}_{s,d}^{(a)}$ in $G^{(a)}$ is equal to the cost of an optimal $\mathcal{SPTP}$ in G.*

Proof. Such property can be proved by contradiction. From Property 2, it follows that the shortest path $\mathcal{P}_{s,d}^{(a)}$ in $G^{(a)}$ corresponds to a feasible path tour $P_{s,d}^*$ in G. Hence, suppose that a shortest path tour $P_{s,d}$ different from $P_{s,d}^*$ exists in G, such that $l(P_{s,d}) < l\left(P_{s,d}^*\right)$. This means that a feasible path $P_{s,d}^{(a)'}$ exists in $G^{(a)}$, associated with $P_{s,d}$, whose cost is less than the cost of the shortest path $P_{s,d}^{(a)}$. This conclusion contradicts the initial assumption. □

4.2. Solving the $\mathcal{SPTP}$ as $\mathcal{SPP}$

The framework of the proposed $\mathcal{MGA}$ is sketched in Algorithm 1:

Algorithm 1. Modified graph algorithm

Step 1 (*Initialization*)
Compute a shortest path from each node $i \in T_k$ to each node $j \in T_{k+1}$, $k = 0, \ldots, S$.
Step 2 (*Graph Construction*)
Build the digraph $G^{(a)} = (N^{(a)}, A^{(a)})$ as detailed in Definition 1.
Step 3 (*Topological enumeration of $N^{(a)}$*)
Step 4 (*Shortest Path Computation*)
Let $l(\mathcal{P}_{1,i})$ denote the length of the shortest path from node $s = 1$ to node $d = i$, and $p(i)$ denote the predecessor node of i in $\mathcal{P}_{1,i}$.
Set $l(\mathcal{P}_{1,1}) := 0, p(1) := 1$ and $iteration_{\mathcal{MGA}} := 0$.
for all $j = 2, \ldots, |N^{(a)}|$ **do**
  $l(\mathcal{P}_{1,j}) = \min_{i \in N^{(a)} : (i,j) \in A^{(a)}} \{l(\mathcal{P}_{1,i}) + c_{ij}\}$. Update $p(j)$ with the value of i whereby the minimum of $l(\mathcal{P}_{1,j})$ occurs.
  Update the number of iterations as $iteration_{\mathcal{MGA}} = iteration_{\mathcal{MGA}} + 1$.
end for

[1] For bounding the computational complexity of an algorithm, several *asymptotic notations* are used [6]. If n is the input size, the computational complexity of an algorithm $f(n)$ is $O(g(n))$, if there exist positive constants a and n_0 such that $0 \leqslant f(n) \leqslant ag(n)$ for all $n \geqslant n_0$.

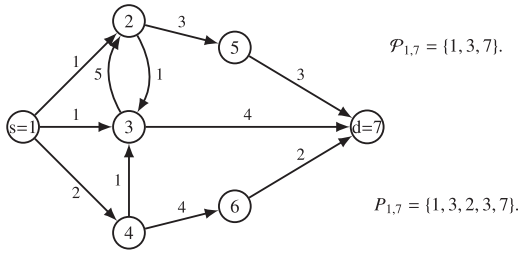


Fig. 1. A $\mathcal{SPTP}$ instance on a small directed graph G.

Theorem 1. *The worst case computational complexity of $\mathcal{MGA}$ is $O(n^3)$.*

Proof. In order to evaluate the computational complexity of the proposed algorithm, it is useful to observe that the total number C of shortest paths to be computed in Step 1 is given by:

$$C = \sum_{i=0}^{S} |T_i||T_{i+1}|.$$

Note that $C = O(n^2)$ and the worst case arises when $S = 2$ and $|T_i| = \lfloor \frac{n-2}{2} \rfloor, i = 1, 2$. In this latter case, Step 1 reduces to computing a shortest path between each pair of nodes $i, j \in N^{(a)}$ such that $i \in T_1$ and $j \in T_2$ and can be solved in the worst case in $O(mn + n^2 \log n)$ by applying Dijkstra's algorithm with Fibonacci's heap. As already underlined, the complexity of Step 2 is $O(Sm)$. Step 3 is performed in $O(|N^{(a)}|) = O(n)$, while the theoretical complexity of the shortest path problems in acyclic graphs is linear in the number of arcs, so that the complexity of Step 4 in Algorithm 1 is $O(|A^{(a)}|)$, which is $O(n^2)$. Since $m = O(n^2)$, it follows that the problem can be solved in the worst case in $O(n^3)$. □

5. A dynamic programming based algorithm

In this section, we describe a dynamic programming algorithm ($\mathcal{DPA}$) to solve the $\mathcal{SPTP}$. $\mathcal{DPA}$ has been designed by considering the $\mathcal{SPTP}$ as an extension of the weight constrained shortest path problem, where the resource r_i associated to a path $P_{s,i}$ from node s to node i represents the index of the last set T_k, $k = 0, \ldots, S + 1$, visited along the path and the resource consumption of an arc is label-dependent.

Recent scientific literature on the study of the classical weight constrained shortest path problem includes among others a paper of Dumitrescu and Boland [10], where the authors propose several alternative algorithms, including an exact method based on the weight-scaling approach, later improved by the same researchers in [11]. In 2009, Muhandiramge and Boland [19] designed a method to find simultaneous solution of Lagrangean dual problems interleaved with preprocessing. Dynamic programming approaches for the weight constrained shortest path problem and the shortest path problem with resource constraints have been proposed in [1,12,20].

In our case, viewing the $\mathcal{SPTP}$ as an extension of the weight constrained shortest path problem, to each path $P_{s,i}$ from node s to node i is associated a label y_i. It stores information about the length of the path and the resource consumption along the path, that is, $y_i = (l(P_{s,i}), r_i)$. With reference to the same network depicted in Fig. 1, label y_2 associated with the subpath $P_{1,2} = \{1, 3, 2\}$ takes the following form: $y_2 = (6, 2)$. Similarly, subpath $P_{1,3} = \{1, 2, 3\}$ has label $y_3 = (2, 1)$.

It is worth observing that more than one path $P_{s,i}$ may exist to reach node i starting from node s. This means that several labels $y_i = (l(P_{s,i}), r_i)$ can be associated with each node i and they are stored in the set $D(i)$.

Let $P'_{s,i}$ and $P''_{s,i}$ be two distinct paths from node s to node i and let r'_i and r''_i denote the resource consumption along path $P'_{s,i}$ and $P''_{s,i}$, respectively.

From the definition of resource consumption, it follows that for the path $P'_{s,i}$ a sequence of natural numbers $g'_0,\ g'_1 \ldots g'_{r'_i}$ exists such that $g'_0 < g'_1 < \cdots < g'_{r'_i}$ and $i_{g'_0} \in P'_{s,i} \cap T_0, \ldots, i_{g'_{r'_i}} \in P'_{s,i} \cap T_{r'_i}$. Similarly, for the path $P''_{s,i}$ it is possible to find a sequence of natural number $g''_0, g''_1 \ldots g''_{r''_i}$ such that $g''_0 < g''_1 < \ldots < g''_{r''_i}$ and $i_{g''_0} \in P''_{s,i} \cap T_0, \ldots, i_{g''_{r''_i}} \in P''_{s,i} \cap T_{r''_i}$.

Suppose that $r'_i \geqslant r''_i$, there exists a path $P'''_{i,d}$ such that the path resulting from the concatenation of $P''_{s,i}$ with $P'''_{i,d}$, say $\widehat{P}_{s,d} = P''_{s,i} \oplus P'''_{i,d}$, is a feasible solution for the $\mathcal{SPTP}$. The following result holds.

Lemma 2. *If $P''_{s,i} \oplus P'''_{i,d}$ is a feasible $\mathcal{SPTP}$ solution, then $P'_{s,i} \oplus P'''_{i,d}$ is also a feasible $\mathcal{SPTP}$ solution.*

Proof. Since $P''_{s,i} \oplus P'''_{i,d}$ is a feasible $\mathcal{SPTP}$ solution, there exists a sequence of natural numbers $g'''_{r''_i+1}, \ldots, g'''_{S+1}$ such that $g'''_{r''_i+1} < \cdots < g'''_{S+1}$ and $i_{g'''_{r''_i+1}} \in P'''_{i,d} \cap T_{r''_i+1}, \ldots, i_{g'''_{S+1}} \in P'''_{i,d} \cap T_{S+1}$.

Owing to the feasibility of path $P''_{s,i} \oplus P'''_{i,d}$ and $r'_i \geqslant r''_i$, it is possible to extract the subsequence $g'''_{r'_i+1}, \ldots, g'''_{S+1}$ from the sequence $g'''_{r''_i+1}, \ldots, g'''_{S+1}$, such that $g'''_{r'_i+1} < \cdots < g'''_{S+1}$ and $i_{g'''_{r'_i+1}} \in P'''_{i,d} \cap T_{r'_i+1}, \ldots, i_{g'''_{S+1}} \in P'''_{i,d} \cap T_{S+1}$. Consequently, $P'_{s,i} \oplus P'''_{i,d}$ is a feasible path for the $\mathcal{SPTP}$. □

Let us assume that $l\left(P'_{s,i}\right) < l\left(P''_{s,i}\right)$. It follows that path $P''_{s,i}$ can be discarded.

Lemma 3. *If $l\left(P'_{s,i}\right) < l\left(P''_{s,i}\right)$, then $l\left(P'_{s,i} \oplus P'''_{i,d}\right) < l\left(P''_{s,i} \oplus P'''_{i,d}\right)$, thus from path $P''_{s,i}$ it is not possible to generate the optimal $\mathcal{SPTP}$ solution.*

Proof. Lemma 3 can be proved by contradiction. Thus, for the purpose of contradiction, let us assume that $l\left(P'_{s,i} \oplus P'''_{i,d}\right) \geqslant l\left(P''_{s,i} \oplus P'''_{i,d}\right)$. Such an expression can be rewritten as follows: $l\left(P'_{s,i}\right) + l\left(P'''_{i,d}\right) \geqslant l\left(P''_{s,i}\right) + l\left(P'''_{i,d}\right)$. It follows that $l\left(P'_{s,i}\right) - l\left(P''_{s,i}\right) \geqslant 0$. This means that $l\left(P'_{s,i}\right) \geqslant l\left(P''_{s,i}\right)$, contradicting the assumption. □

Lemma 3 suggests us to define a dominance relation between labels. In particular the following results can be drawn.

Definition 2. Let $y'_i = \left(l\left(P'_{s,i}\right), r'_i\right)$ and $y''_i = \left(l\left(P''_{s,i}\right), r''_i\right)$ be two labels associated with paths $P'_{s,i}$ and $P''_{s,i}$, respectively. If $l\left(P'_{s,i}\right) < l\left(P''_{s,i}\right)$ and $r'_i \geqslant r''_i$, then y''_i is dominated by y'_i and path P''_{si} can be discarded.

Definition 3. Let $y'_i = \left(l\left(P'_{s,i}\right), r'_i\right)$ and $y''_i = \left(l\left(P''_{s,i}\right), r''_i\right)$ be two labels. Such labels are said to be equivalent if $l\left(P'_{s,i}\right) = l\left(P''_{s,i}\right)$ and $r'_i = r''_i$.

Starting from label $y^0_s = (0, 0)$ associated with the origin node s, the solution space is explored in order to obtain efficient solutions for each node. Through the algorithm iterations, a label $y''_j = \left(l\left(P''_{sj}\right), r''_j\right)$ is generated starting from a label $y'_i = \left(l\left(P'_{s,i}\right), r'_i\right)$ using the following updating rules:

$$l\left(P''_{s,j}\right) = l\left(P'_{s,i}\right) + c_{ij}; \tag{2}$$

$$r''_j = \begin{cases} r'_i + 1, & \text{if } r'_i + 1 \leqslant S+1 \text{ and } j \in T_{r'_i+1}; \\ r'_i, & \text{otherwise.} \end{cases} \tag{3}$$

At the end of the algorithm, $D(j)$, $j \in N$, is an efficient set, that is, it contains efficient and feasible solutions. In addition, $l^*_d = l\left(P^*_{s,d}\right) : y^* = \left(l\left(P^*_{s,d}\right), S+1\right) \in D(d)$ represents the length of the optimal solution of the shortest path tour from node s to node d.

Let L be the set storing the labels associated with the partial paths to be processed, the steps of the proposed labeling method are depicted in Algorithm 2.

An upper bound on $l(P_{s,d})$, named $h_{\max}$, may be obtained by considering that, in the worst case, a tour contains $S+1$ subpaths, each of them involving $n-1$ arcs having the maximum cost.

Therefore it results that

$$l(P_{s,d}) \leqslant h_{\max} \leqslant (S+1)(n-1)c_{\max}, \quad c_{\max} = \max_{(i,j)\in A} c_{ij}. \tag{4}$$

Algorithm 2. Multidimensional labeling algorithm

Step 1 (*Initialization*)
Set: $P^0_{s,s} := \{s\}, y^0_s := (0, r^0_s)$ with $r^0_s := 0$, and $iteration_{\mathcal{DPA}} := 0$.
Set: $L := \{y^0_s\}, D(s) := \{y^0_s\}, D(j) := \emptyset, \forall j \in N, j \neq s$.
Step 2 (*Label Selection*)
Select and delete from L a label y'_i.
Update the number of iterations as
  $iteration_{\mathcal{DPA}} = iteration_{\mathcal{DPA}} + 1$.
Step 3 (*Label Extension*)
for all $(i,j) \in A$ **do**
  Set:
  $\overline{P}_{s,j} := P'_{s,i} \cup \{j\}$.
  $l(\overline{P}_{s,j}) := l\left(P'_{s,i}\right) + c_{ij}$.
  if $r'_i < S+1$ **then**
    Set: $\bar{k} := r'_i + 1$.
    if $j \in T_{\bar{k}}$ **then**
      Set $\bar{r}_j := \bar{k}$.
    else
      Set $\bar{r}_j := r'_i$.
    end if
  else
    Set: $\bar{r}_j := r'_i$.
  end if
  Set $\bar{y}_j := (l(\overline{P}_{s,j}), \bar{r}_j)$.
  if $\bar{y}_j$ is not dominated by any label y'_j belonging to $D(j)$ **then**
    Remove from $D(j)$ and from L all labels y'_j that are dominated by $\bar{y}_j$.
    Add $\bar{y}_j$ to $D(j)$ and to L.
  end if
end for
Step 4 (*Termination check*)
if $L = \emptyset$ **then**
  STOP
else
  Go to Step 2.
end if

In view of the relation (4), the following property stands.

Theorem 4. *The computational complexity of $\mathcal{DPA}$ is $O(S^2n^3c_{max})$.*

Proof. The number of iterations performed by the algorithm depends on the total number B of different labels generated. There are at most $O(B) = O(Sh_{\max})$ different labels. Each label may be replicated on different nodes, but it is expanded at most $\sum_{i\in N} |F(i)| = m$ times, and each expansion requires $O(1)$ times, then the total number of performed iterations is:

$$O(Bm) = O(S^2nmc_{\max}).$$

Since $O(m) = O(n^2)$, the resulting overall complexity is $O(S^2n^3c_{\max})$ in the worst case. □

The running time of Algorithm 2 could be improved by considering a pruning strategy based on the path length. Let l_i be the least length from node i to node d and let UB be the length of a feasible solution for the $\mathcal{SPTP}$. A label y'_i can be discarded if $l\left(P'_{s,i}\right) + l_i > UB$. In addition, whenever a feasible $\mathcal{SPTP}$ solution $P'_{s,d}$ is found such that $l\left(P'_{s,d}\right) < UB$, then UB is updated.

6. Computational results

In this section, we discuss and compare the computational results obtained with the $\mathcal{DPA}$, $\mathcal{MGA}$ and $\mathcal{EGA}$. The experiments were carried out on a PC equipped with one Intel Core X5680 Processor@3.33 GHz, with 50 Gbyte RAM. All algorithms were coded in *java*.

The main goal of these experiments is to study the behavior of the proposed algorithms (i.e., $\mathcal{DPA}$, $\mathcal{MGA}$) and compare them with the state-of-the-art algorithm (i.e., $\mathcal{EGA}$). A computational analysis is outlined to show how the performance of $\mathcal{DPA}$ and $\mathcal{MGA}$ are affected by network topology and characteristics, i.e., number and size of subsets T_k, $k = 0, \ldots, S+1$. In all the test problems, the following setting is used: $s = 1$ and $d = n$. Details related to the basic shortest path algorithms used inside the proposed methods are given below.

As far as the construction of $G^{(a)}$ in the $\mathcal{MGA}$ is concerned, the Floyd–Warshall algorithm [15] is used to compute the length of the shortest path between all node-pairs of G. Observe that graph $G^{(a)}$ is built explicitly within $\mathcal{MGA}$. The shortest path solutions in $\mathcal{EGA}$ is provided by Dijkstra's algorithm [7] with binary heaps [17]. It is worth observing that for $\mathcal{EGA}$ an iteration corresponds to process the node just selected from the heap, while for $\mathcal{MGA}$ and $\mathcal{DPA}$ an iteration is defined implicitly inside the corresponding algorithmic sketches.

Given that the performance of $\mathcal{DPA}$ is strongly influenced by an Upper Bound (UB) on the minimum cost, a feasible solution is found heuristically by selecting a node sequence $\{1, i_1, i_2, \ldots, i_S, n\}$ starting from the source node 1 and ending with the destination node n, and such that $i_k \in T_k$, $k = 1, \ldots, S$. The length of the path associated with this sequence is evaluated as the shortest path cost obtained by applying Dijkstra's algorithm between two consecutive nodes in the sequence.

The collected computational results are reported in Tables A.1–A.9 reported in the Appendix. For each instance and each algorithm, the number of iterations and the execution times are detailed. The column headings of such tables are defined as follows: "Test" denotes the instance name; "Time" denotes the computational time in milliseconds (ms) required by each one of the proposed algorithms in order to find the optimal $\mathcal{SPTP}$ solution; "Iterations" denotes the overall number of iterations performed by each of the designed algorithm; "FW Time" denotes the Floyd–Warshall running time performed by $\mathcal{MGA}$ on G and required to construct $G^{(a)}$; "SpTime" indicates the running time required by $\mathcal{EGA}$ to compute the shortest path on the expanded graph; the running time required by $\mathcal{EGA}$ to build the expanded graph is reported in column "EgTime". Due to the use of *java* as programming language, the running time required by $\mathcal{DPA}$, $\mathcal{MGA}$ and $\mathcal{EGA}$ are averaged over 30 runs, executed for each $\mathcal{SPTP}$ instance.

6.1. Test problems

Three classes of networks are considered: grid random, fully random, and fully dense networks. The rationale of this choice is motivated by the fact that these networks are well recognized and used extensively to assess the behavior of the solution algorithms for the classical $\mathcal{SPP}$ and some of its variants (see for example, [4,5,16,9,8]). For all test networks, the arc lengths are chosen according to a uniform distribution in the range from [1,1000].

6.1.1. Grid random networks

Such problems are generated by using the Gridgen generator written by Bertsekas [3]. Problem sizes are outlined in Table 1. Both square (G_τ, with $\tau = 1, 2, 3$) and rectangular grid networks (G_τ, with $\tau = 4, 5, 6$) are designed. The nodes are arranged in a planar grid. Nodes s and d are placed at the corners. Each pair of adjacent nodes are connected in both directions. Sets T_k, $k = 1, \ldots, S$, are generated by randomly selecting one node at a time and placing it into T_k up to achieve the corresponding size.

6.1.2. Fully random networks

A set of nine networks of different size and density (i.e., the number of arcs over the number of nodes) is designed. These networks, named R_τ, $\tau = 1, \ldots, 9$, in the sequel, are generated by using the public domain Netgen program [18]. Table 2 reports the sizes of the random networks. For each problem, the number of nodes, the number of arcs, and the density are provided.

6.1.3. Fully dense networks

Three fully dense networks are generated by using the Compligen generator written by Bertsekas [3]. Each of these problems, referred to as C_τ, $\tau = 1, 2, 3$, in the sequel, includes all the possible $n(n-1)$ arcs. The number of nodes is set equal to 100 for C_1, 300 for C_2, and 500 for C_3.

6.1.4. $\mathcal{SPTP}$ datasets

For each network of the type described above, a set of $\mathcal{SPTP}$ s is built by varying the number of node subsets S, and the size of $|T_k|$, $k = 1, \ldots, S$, for a given S. More precisely, let $n' = n - 2$ be the number of nodes in the original network from which the source and destination nodes are discarded, four instances of the $\mathcal{SPTP}$ are obtained by setting S in $\mathcal{S} = \{5, 10, 15, 20\}$. For each value of S, three $\mathcal{SPTP}$ s are designed by varying $|T_k|$ in the set

$$\mathcal{T}_S = \left\{ |T_S|_{lower} := \left\lfloor \frac{N'}{3S} \right\rfloor, |T_S|_{middle} := \left\lfloor \frac{N'}{2S} \right\rfloor, |T_S|_{upper} := \left\lfloor \frac{N'}{S} \right\rfloor \right\}.$$

More generally, it is:

$$\mathcal{T}_S = \left\{ \left\lfloor \rho \frac{N'}{S} \right\rfloor : \rho \in \left\{ \frac{1}{3}, \frac{1}{2}, 1 \right\} \right\},$$

where ρ represents the consistency of sets T_k, $k = 1, \ldots, S$, associated with S. Observe that $|\mathcal{T}_S| = 3$ for each $S \in \mathcal{S}$, and it will be referred to as $|\mathcal{T}|$ in the sequel. In this way, a set of 12 benchmark instances remains associated with each original network.

Table 1
Grid random networks.

Problem	Dimension	Nodes	Arcs
G_1	25×25	625	2400
G_2	30×30	900	3480
G_3	50×50	2500	9800
G_4	25×50	1250	4850
G_5	30×60	1800	7020
G_6	50×100	5000	19,700

Table 2
Fully random networks.

Problem	Nodes	Arcs	Density
R_1	300	1500	5
R_2	300	3000	10
R_3	300	4500	15
R_4	500	2500	5
R_5	500	5000	10
R_6	500	7500	15
R_7	1000	5000	5
R_8	1000	10,000	10
R_9	1000	15,000	15

In what follows, we use the notation $type^{S_{|\mathcal{T}|}}$ to identify each $\mathcal{SPTP}$ instance. Indeed, $R_1^{5_8}$ refers to the fully random problem R_1, with $S = 5$ node subsets $T_1, \ldots, T_5$, such that $|T_1| = \ldots = |T_5| = 8$, and $|T_0| = |T_6| = 1$. The $\mathcal{SPTP}$ instances are available for download at the following URL: http://uweb.deis.unical.it/guerriero/benchmark-instances-for-sptp. In addition, computational experiments were carried out on the instances defined by Festa in [13] and provided by the same author.

6.2. Statistics

We look at several statistics to evaluate the performance of the proposed algorithms on a large number of different-sized test problems. The statistics we will consider are defined below:

(a) *n-Average execution time*: mean value of the overall execution times related to the $\mathcal{SPTP}$ instances associated with a network with a given topology (grid random, fully random, fully dense), and a given number n of nodes.
(b) *S-Average execution time*: mean value of the average execution times associated with a set of $\mathcal{SPTP}$ instances defined by all the networks with a given topology (grid random, fully random, fully dense), and a given value of S.
(c) *S-Average iteration number*: computed as for the S-Average execution time with respect to the number of iterations.
(d) *ρ-Average execution time*: average execution time computed with respect to a given consistency value, for a given S and for all the networks with a given topology (grid random, fully random, fully dense).

6.3. Results on grid random networks

In this section, the behavior of the proposed methods is studied for the set of $\mathcal{SPTP}$ s based on grid random networks. Computational results are detailed in Tables A.1–A.2 of the Appendix. Generally, the computational results show how the performances of the proposed algorithms seem to be affected by the structure of the networks, expressed by means of the number of nodes n, and density $\delta = \frac{m}{n}$. In general, the computational effort increases with n.

This trend is well highlighted in Fig. 2, where the n-Average execution times are plotted as a function of the number of nodes. The n-Average time of $\mathcal{MGA}$ increases more quickly than $\mathcal{DPA}$ as long as n increases beyond 1250. Since the most computational part of $\mathcal{MGA}$ is represented by Floyd–Warshall algorithm, that is involved in construction of $G^{(a)}$, the reported computational times of $\mathcal{MGA}$, that is, the times required by Floyd–Warshall, grow according to the computational complexity $O(n^3)$. Therefore, for grid networks with large value of n, that is, n greater than 1250, the time required by $\mathcal{MGA}$ is more expensive than the one implied by $\mathcal{DPA}$. On the other hand, since paths $P_{s,d}$ in grid network with a large number of nodes n contain a large number of intermediate nodes (at least $g_x + g_y - 1$ nodes in a $g_x \times g_y$ grid), then the possibility to meet several sets T_k, $k = 1, \ldots, S$, could increases with n.

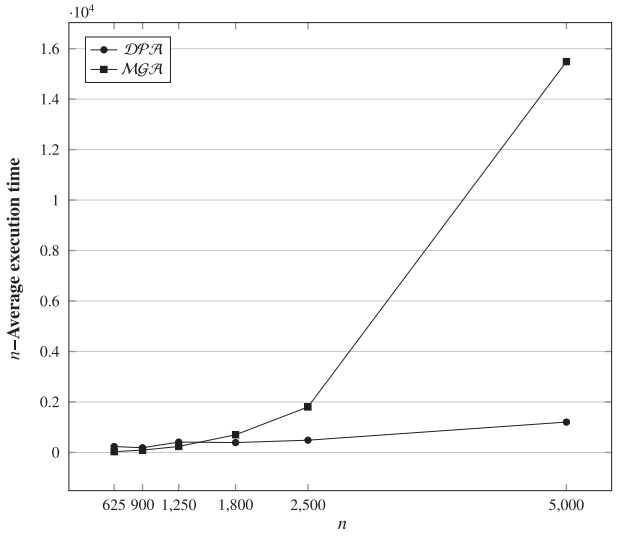


Fig. 2. n-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ on grid random network-based test problems.

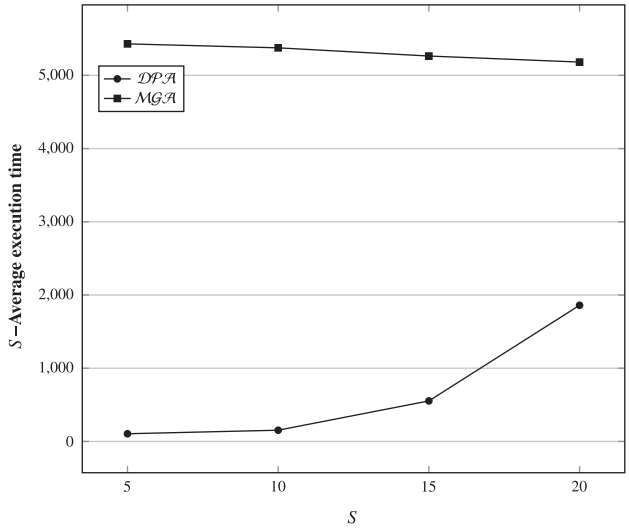


Fig. 3. S-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ on rectangular grid random network-based test problems.

Therefore, the convergence of $\mathcal{DPA}$ towards the optimal $\mathcal{SPTP}$ solution does not deteriorate with the increase of n in the same manner as it occurs for $\mathcal{MGA}$.

The same trend occurs if iteration numbers are considered instead of average times (see Tables A.1–A.2 of the Appendix).

Figs. 3 and 4 help to explain the differences underlined so far. In case of $\mathcal{DPA}$, the more high the value of S becomes, the more increase in the execution time occurs. This is motivated by the theoretical computational complexity of $\mathcal{DPA}$ that increases explicitly more than linearly with S.

On the contrary, the theoretical computational complexity of $\mathcal{MGA}$ varies implicitly with S, depending on whether it is building $G^{(a)}$ or finding the shortest path on $G^{(a)}$. More precisely, the increase of S reduces the cardinality of sets T_k, $k = 1, \ldots, S$, and thus the number of arcs of $G^{(a)}$. In fact, $|A^{(a)}| = O\left(\frac{(\rho n)^2}{S}\right)$; therefore, the time to build $G^{(a)}$ is reduced also as well as the time for finding the shortest path on $G^{(a)}$. Observe that for square grid random network-based $\mathcal{SPTP}$ instances there exists a value of S from which $\mathcal{MGA}$ outperforms $\mathcal{DPA}$.

Table 3 brings out clearly the difference in the relationship between the S-Average iteration number performed by each algorithm and the number S of sets. Such a table shows that the average number of iterations performed by $\mathcal{DPA}$ increases with S, while it decreases for $\mathcal{MGA}$. In effect, the number of labels generated by $\mathcal{DPA}$ for each node increases with S, and then the number of iterations also increases. As mentioned earlier, the running time of $\mathcal{MGA}$ decreases with the increase of S.

The impact of ρ on the performance of the solving algorithms emerges in Figs. 5 and 6. They show how the average execution times vary with respect to the values of ρ. More precisely, for a given S the time performance of $\mathcal{DPA}$ decreases significantly with the values of ρ. The rationale of such a behavior can be explained by taking into account that large value of ρ, that is large size of T_k with $k = 1, \ldots, S$, makes easier to improve the resource labels and also generate new labels with increasing resources allowing to remove a large number of dominated labels from L, so that the algorithm converges quickly to the optimal solution. This is specially true for large values of S. For $\mathcal{MGA}$ an opposite trend is observed. The occurrence of increasing value of ρ for a given S, that is increasing size of sets T_k with $k = 1, \ldots, S$, affects the construction of $G^{(a)}$ by enlarging the size of $N^{(a)}$, and increases also the number of arcs belonging to $A^{(a)}$ and connecting pairs of disjoint nodes of

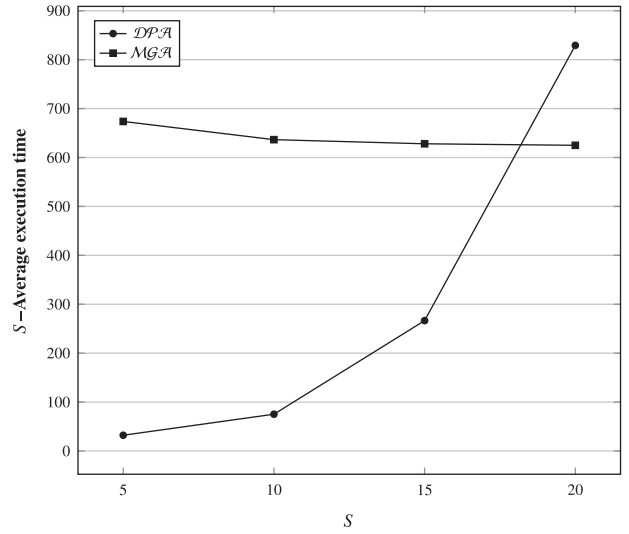


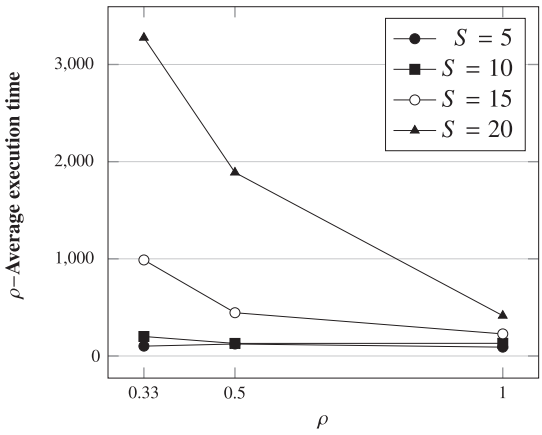
Fig. 4. S-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ on square grid random network-based test problems.

$N^{(a)}$. Moreover, the running time to find the shortest path on $G^{(a)}$ grows with the size of T_k, that depends directly on ρ for a given S. Therefore, the more the performance of $\mathcal{MGA}$ deteriorates with the increase of ρ, the lower are the values of S.

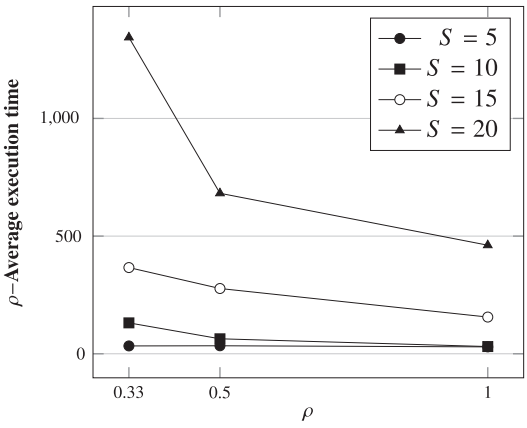
The computational results collected for the grid random network-based $\mathcal{SPTP}$ s indicate that $\mathcal{DPA}$ behaves the best. In

Table 3
S-Average iteration numbers performed by $\mathcal{MGA}$ and $\mathcal{DPA}$ on grid random network-based test problems.

S	Rectangular grid		Square grid	
	$\mathcal{DPA}$	$\mathcal{MGA}$	$\mathcal{DPA}$	$\mathcal{MGA}$
5	5538.67	708031.78	3575.56	176570.44
10	21184.78	396883.22	18522.33	98674.33
15	154647.10	274393.78	72033.78	68055.56
20	501962.20	208135.44	218226.60	51685.44

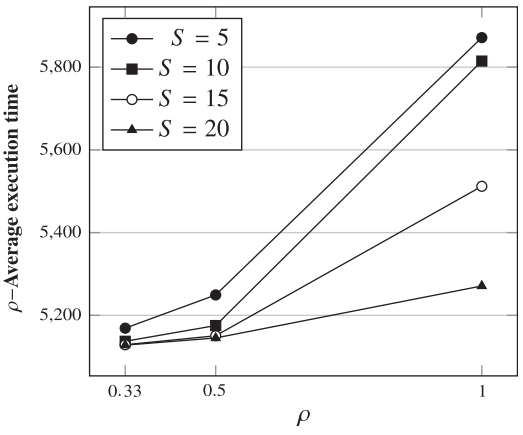


(a) Rectangular grid

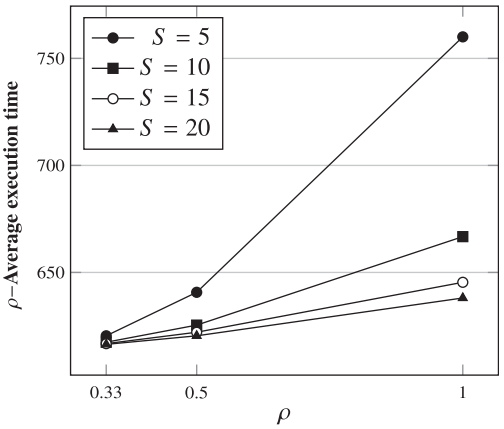


(b) Square grid

Fig. 5. ρ-Average execution time of $\mathcal{DPA}$ on grid random network-based test problems.



(a) Rectangular grid



(b) Square grid

Fig. 6. ρ-Average execution time of $\mathcal{MGA}$ on grid random network-based test problems.

particular, on the rectangular grid networks $\mathcal{DPA}$ is on average 7.95 times faster than $\mathcal{MGA}$. A similar behavior can be observed for the square grid random networks where the average execution time of $\mathcal{DPA}$ is 2.13 times faster than the average time required by $\mathcal{MGA}$.

6.4. Computational results on fully random networks

The behavior observed for grid random networks remains confirmed in case of fully random networks. Fig. 7 shows how the n-Average execution time, for both $\mathcal{DPA}$ and $\mathcal{MGA}$, increases with n, even if the slope related to $\mathcal{DPA}$ is clearly higher that the one corresponding to $\mathcal{MGA}$. The cost of an iteration of $\mathcal{DPA}$ is greater than the computational effort required to execute an iteration of $\mathcal{MGA}$. Fully random graphs are expected to contain $P_{s,d}$ paths with less and less arcs as δ increases for a fixed number of nodes n. Furthermore, the cost of the single iteration of $\mathcal{DPA}$ clearly grows with δ. On the other hand, for sufficiently small n, the cost of Floyd–Warshall algorithm that is involved in construction of $G^{(a)}$ is limited, and $\mathcal{MGA}$ outperforms $\mathcal{DPA}$; note that this happens for both fully random and grid random networks, with a number of nodes $n \leqslant 1250$. Definitively, the computational results collected on fully random networks confirm that $\mathcal{DPA}$ can outperform $\mathcal{MGA}$, but only for sufficiently large n.

Fig. 8 shows clearly how the S-Average time of $\mathcal{DPA}$ increases with S, while the average performance of $\mathcal{MGA}$ seems not be influenced by the values of S. Table 4 indicates that the average number of iterations of $\mathcal{DPA}$ increases more than linearly with S, while the average number of iterations of $\mathcal{MGA}$ decreases less than linearly with S. The rationale is the same highlighted for the grid random network-based $\mathcal{SPTP}$ s.

Finally, the impact of ρ on the performance of both algorithms is numerically underlined by looking inside Tables A.3–A.5, and globally depicted in Figs. 9 and 10.

In particular, Fig. 9 shows the impact of ρ on the performance of $\mathcal{DPA}$ for a given S. The increase of ρ enlarges the size of T_k, where $k = 1, \ldots, S$, so that the improvement of the resource labels becomes easier. In case of $\mathcal{MGA}$ the trend is opposite as follows from Fig. 10. In such a case, the execution times of $\mathcal{MGA}$ increase with ρ, so the performance of $\mathcal{MGA}$ decreases with ρ. More precisely, the size of T_k, with $k = 1, \ldots, S$, increases with ρ, and the impact of such an increase has an effect on the computational complexity of $\mathcal{MGA}$.

The computational results collected for the fully random network-based $\mathcal{SPTP}$s indicate that $\mathcal{MGA}$ outperforms $\mathcal{DPA}$. More precisely, $\mathcal{MGA}$ is on average 13.81 times faster than $\mathcal{DPA}$.

6.5. Computational results on fully dense networks

Concerning experiments on fully dense networks, since n assumes the smallest value with respect to that occurring in grid and fully random networks, then the computational time required by Floyd–Warshall is very limited compared to the one implied in

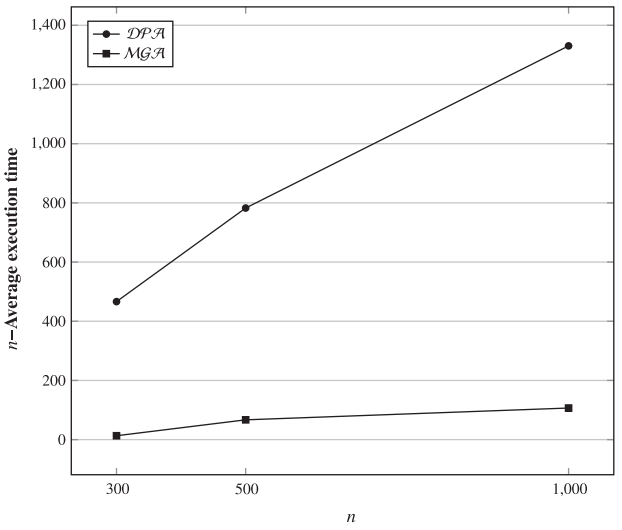


Fig. 7. n-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ for the $\mathcal{SPTP}$ s based on fully random network-based test problems.

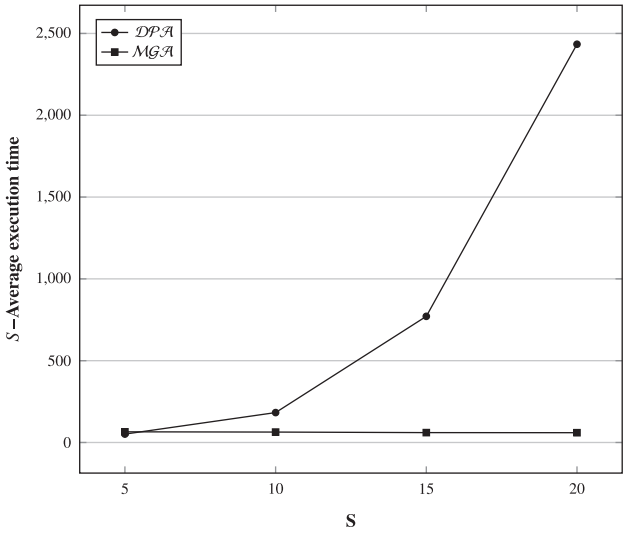


Fig. 8. S-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ for the $\mathcal{SPTP}$ s based on fully random network-based test problems.

an iteration of $\mathcal{DPA}$. Therefore, the performance of $\mathcal{MGA}$ is clearly better than $\mathcal{DPA}$. Figs. 11 and 12 show how the n-Average execution time and the S-Average execution time of $\mathcal{DPA}$ quickly increases with n, while they have a very slow increase in case of $\mathcal{MGA}$.

Table 5 brings out that the S-Average number of iterations of $\mathcal{DPA}$ increases monotonically and more than in a linear way with S, while it decreases less than linearly with S in case of $\mathcal{MGA}$. Table A.6 gives numerically evidence of the above considerations.

The trend of the ρ-Average execution time for $\mathcal{DPA}$ and $\mathcal{MGA}$ is coherent with the dependence already observed for the test problems based on grid and fully random network-based $\mathcal{SPTP}$s, as confirmed by Figs. 13 and 14. The rationale of these behaviors is implied into the nature of the computational complexity of $\mathcal{DPA}$ and $\mathcal{MGA}$ that is emphasized by this type of networks.

The computational results summarized and discussed on fully dense network-based $\mathcal{SPTP}$s show that $\mathcal{MGA}$ clearly outperforms $\mathcal{DPA}$. In fact, $\mathcal{MGA}$ is on average 814.02 times faster than $\mathcal{DPA}$.

Table 4
S-Average iteration numbers performed by $\mathcal{DPA}$ and $\mathcal{MGA}$ on fully random network-based test problems.

S	$\mathcal{DPA}$	$\mathcal{MGA}$
5	1851.741	32025.778
10	32184.519	17713.556
15	152739.593	12294.667
20	404469.926	9066.444

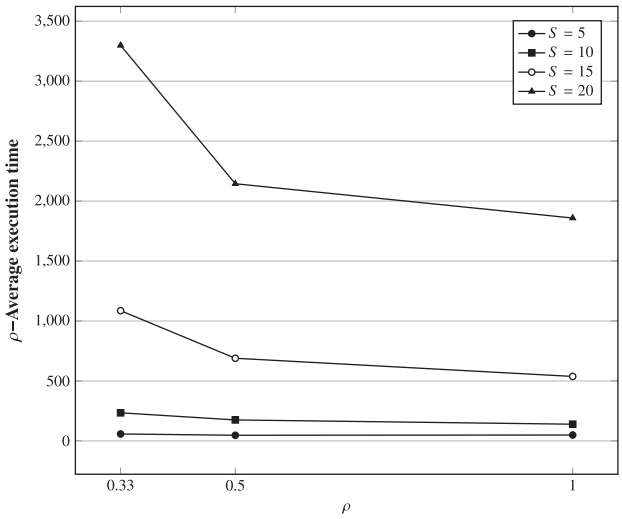


Fig. 9. ρ-Average execution time of $\mathcal{DPA}$ for fully random network-based test problems.

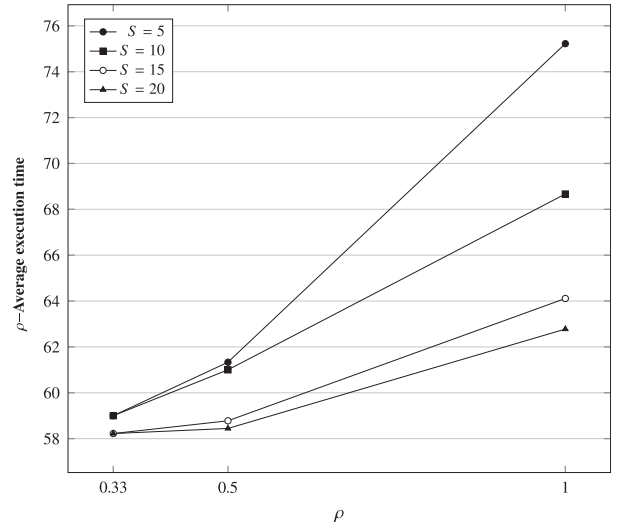


Fig. 10. ρ-Average execution time of $\mathcal{MGA}$ for fully random network-based test problems.

6.6. Comparison with the state-of-the-art algorithm

In this section, for each set of test problems, the best performing algorithm selected between $\mathcal{DPA}$ and $\mathcal{MGA}$ is compared with the algorithm devised by Festa in [13]. In particular, the comparison involves the average computational times and the number of iterations performed by the best algorithm between $\mathcal{DPA}$ and $\mathcal{MGA}$, and $\mathcal{EGA}$.

The performance of $\mathcal{EGA}$ with respect to $\mathcal{MGA}$ is affected by the topology of graph G' that is not necessarily acyclic, unlike $G^{(a)}$ that is acyclic and explicitly built within $\mathcal{MGA}$. The computational results obtained by testing $\mathcal{EGA}$ on the considered three random

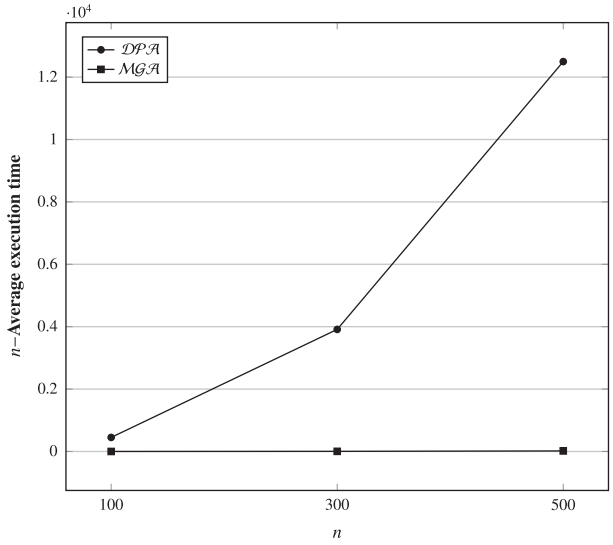


Fig. 11. n-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ on fully dense network-based test problems.

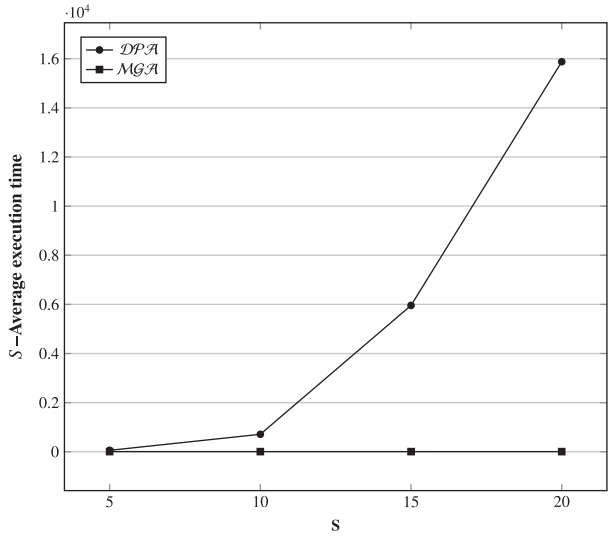


Fig. 12. S-Average execution time of $\mathcal{DPA}$ and $\mathcal{MGA}$ on fully dense network-based test problems.

Table 5
S-Average iteration numbers performed by $\mathcal{DPA}$ and $\mathcal{MGA}$ on fully dense network-based test problems.

S	$\mathcal{DPA}$	$\mathcal{MGA}$
5	2352.111	8272.444
10	34001.222	4491.889
15	206849.222	3123.556
20	494807.944	2286.778

datasets are reported in Tables A.7–A.9. In order to assess the behavior of $\mathcal{EGA}$ on these test problems, let us to consider them separately.

The computational results obtained by testing $\mathcal{EGA}$ on the first set of the $\mathcal{SPTP}$ instances based on grid random networks underline that the best algorithm between $\mathcal{DPA}$ and $\mathcal{MGA}$ outperforms $\mathcal{EGA}$. In particular, $\mathcal{DPA}$ is on average 28.27 and 18.19 times faster

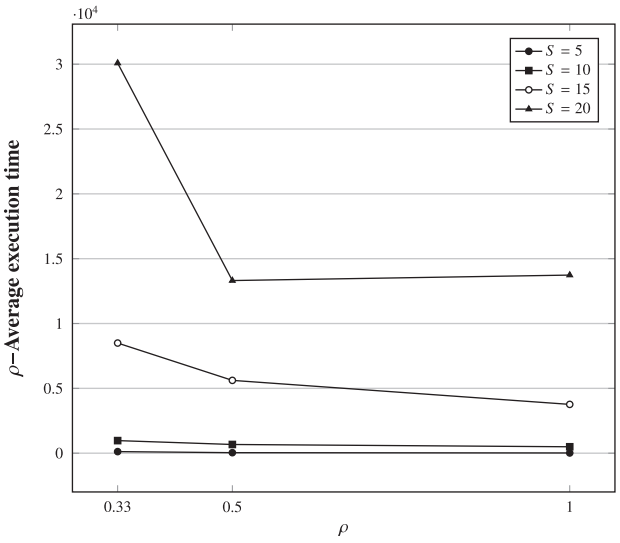


Fig. 13. ρ-Average execution time of $\mathcal{DPA}$ on fully dense network-based test problems.

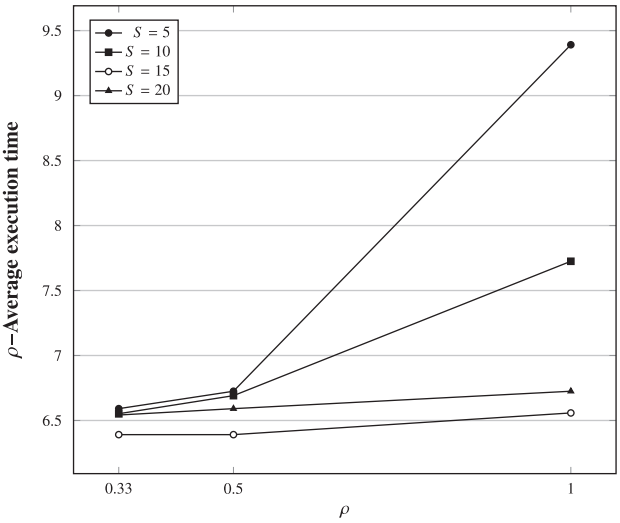


Fig. 14. ρ-Average execution time of $\mathcal{MGA}$ on fully dense network-based test problems.

than $\mathcal{EGA}$ on the rectangular and square grid random networks, respectively. $\mathcal{MGA}$ performs better than $\mathcal{EGA}$ with an average time reduction by a factor of 21.94 on fully random network-based $\mathcal{SPTP}$ instances. Finally, $\mathcal{MGA}$ clearly outperforms $\mathcal{EGA}$ on fully dense random network-based $\mathcal{SPTP}$ instances by a factor of 301.13.

As for the datasets provided by Festa in [13], named $\mathcal{TF}$ in the sequel, they refer to the same three network topologies already defined. Tables 6–8 report the characteristics of the $\mathcal{SPTP}$ instances belonging to $\mathcal{TF}$ and defined on the basis of grid random networks G^{TF}, fully random networks R^{TF} and fully dense networks C^{TF}, respectively. Observe that these networks are designed by using the generator described in [14]. The source and destination nodes are randomly selected among all the nodes, according to a uniform distribution. Each node of the networks belongs to a set T_k, where $k = 0, \ldots, S+1$. Moreover, the $\mathcal{SPTP}$ instances in the $\mathcal{TF}$ datasets

Table 6
$\mathcal{SPTP}$ s of $\mathcal{TF}$ based on grid random networks.

Problem	Dimension	Nodes	Arcs	S
G_1^{TF}	10×10	100	400	10
G_2^{TF}	10×10	100	400	30
G_3^{TF}	10×10	100	400	50
G_4^{TF}	10×10	100	400	70
G_5^{TF}	25×6	150	600	15
G_6^{TF}	25×6	150	600	45
G_7^{TF}	25×6	150	600	75
G_8^{TF}	25×6	150	600	105

Table 7
$\mathcal{SPTP}$ s of $\mathcal{TF}$ based on fully random networks.

Problem	Nodes	Arcs	Density	S
R_1^{TF}	150	600	4	15
R_2^{TF}	150	600	4	45
R_3^{TF}	150	600	4	75
R_4^{TF}	150	600	4	105
R_5^{TF}	150	1200	8	15
R_6^{TF}	150	1200	8	45
R_7^{TF}	150	1200	8	75
R_8^{TF}	150	1200	8	105

Table 8
$\mathcal{SPTP}$ s of $\mathcal{TF}$ based on fully dense networks.

Problem	Nodes	Arcs	S
C_1^{TF}	60	3540	6
C_2^{TF}	60	3540	18
C_3^{TF}	60	3540	30
C_4^{TF}	60	3540	42
C_5^{TF}	100	9900	10
C_6^{TF}	100	9900	30
C_7^{TF}	100	9900	50
C_8^{TF}	100	9900	70

are generated for each network and value of S specified in the last column of Tables 6–8. For a detailed description of how such instances are generated, the reader is referred to [13].

The computational results collected on the instances of $\mathcal{TF}$ can be found in Tables 9–11. They show that the best performing algorithm is $\mathcal{MGA}$, even on the grid network-based instances. This behavior is coherent with the trend analyzed in Section 6.3. The rationale lies in the fact that the $\mathcal{SPTP}$ instances of $\mathcal{TF}$ have a very small number of nodes, thus the running time of $\mathcal{MGA}$ is slower than that required by $\mathcal{DPA}$.

In comparison with $\mathcal{EGA}$, one may observe that $\mathcal{MGA}$ is on average 2051.50 times faster than $\mathcal{EGA}$ on grid network-based instances, 3415 times faster in case of fully random network-based instances, and 3213.46 times faster in case of fully dense network-based instances. This behavior may be explained by observing that the $\mathcal{SPTP}$ instances of $\mathcal{TF}$ are designed in such a way that the size of expanded graph built in the method proposed in [13] grows linearly with S since for each of the S node subsets it replicates the original graph, whereas working on the modified graph $G^{(a)}$ determines considerable improvements in performance.

Table 9
Computational results on the grid network-based $\mathcal{SPTP}$ s of $\mathcal{TF}$.

Test	$\mathcal{EGA}$			$\mathcal{MGA}$		
	SpTime	EgTime	Iterations	FW time	Time	Iterations
G_1^{TF}	15	5	785	0.25	0.25	813
G_2^{TF}	83	26	2803	0.25	0.25	295
G_3^{TF}	160	29	4758	0.25	0.25	199
G_4^{TF}	345	17	6712	0.25	0.25	145
G_5^{TF}	77	15	1844	0.25	0.25	1026
G_6^{TF}	311	12	6308	0.25	0.25	500
G_7^{TF}	988	6	10,956	0.25	0.25	301
G_8^{TF}	2002	12	15,380	0.25	0.25	210
Average	497.63	15.25	6193.25	0.25	0.25	436.13

Table 10
Computational results on the random network-based $\mathcal{SPTP}$ s of $\mathcal{TF}$.

Test	$\mathcal{EGA}$			$\mathcal{MGA}$		
	SpTime	EgTime	Iterations	FW time	Time	Iterations
R_1^{TF}	71	70	1752	0.25	0.25	1544
R_2^{TF}	298	86	6187	0.25	0.25	493
R_3^{TF}	874	26	10,863	0.25	0.25	293
R_4^{TF}	1596	37	15,306	0.25	0.25	211
R_5^{TF}	57	2	1810	0.25	0.25	1321
R_6^{TF}	478	6	6378	0.25	0.25	473
R_7^{TF}	1076	14	10,863	0.25	0.25	298
R_8^{TF}	2119	20	15,336	0.25	0.25	208
Average	821.13	32.63	8561.88	0.25	0.25	605.13

Table 11
Computational results on the fully dense network-based $\mathcal{SPTP}$ s of $\mathcal{TF}$.

Test	$\mathcal{EGA}$			$\mathcal{MGA}$		
	SpTime	EgTime	Iterations	FW time	Time	Iterations
C_1^{TF}	3	2	238	0.13	0.13	343
C_2^{TF}	30	9	888	0.13	0.13	158
C_3^{TF}	99	14	1581	0.13	0.13	115
C_4^{TF}	113	65	2374	0.13	0.13	87
C_5^{TF}	86	156	619	0.13	0.13	523
C_6^{TF}	209	104	2766	0.13	0.13	307
C_7^{TF}	544	266	4730	0.13	0.13	192
C_8^{TF}	1201	441	6791	0.13	0.13	139
Average	285.63	132.13	2498.38	0.13	0.13	233

7. Conclusions

This paper exhibits the results of the study concerning a variant of the $\mathcal{SPP}$, called the Shortest Path Tour Problem ($\mathcal{SPTP}$), in which the shortest path from a given origin node to a given destination node flows through a given number of node subsets ordered according to a given sequence. We provide two competitive solving algorithms. The former is based on an explicit reduction of the $\mathcal{SPTP}$ into an instance of the $\mathcal{SPP}$, the latter on a dynamic programming method where the $\mathcal{SPTP}$ is considered as an extension of the resource constrained shortest path problem. Some basic properties are highlighted and an extensive computational analysis of both the proposed methods is carried out on new large size benchmark instances, and the dataset considered in [13]. The numerical results show that the performance of the proposed algorithms depends mainly on the structure of the networks. More precisely, the dynamic programming-based algorithm ($\mathcal{DPA}$)

outperforms the modified graph-based algorithm ($\mathcal{MGA}$) on the $\mathcal{SPTP}$ instances that are generated from grid random networks, while the latter is more competitive than the former as long as fully random and fully dense networks are involved. This appears from the computational results obtained with the new datasets. The numerical experiments carried out on the $\mathcal{TF}$ dataset show that the most efficient proposed algorithm outperforms clearly the state-of-the-art solution strategy [13]. The obtained results also suggest that these methods could be extended to address other variants of the classical $\mathcal{SPP}$, which represent a new and challenging research area.

Acknowledgements

The authors would like to convey their deep appreciation to the anonymous referees for their valuable comments, that permitted to improve significantly the overall quality of the paper.

Appendix. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.ejor.2013.04.029.

References

[1] Y.P. Aneja, V. Aggarwal, K.P.K. Nair, Shortest chain subject to side constraints, Networks 13 (2) (1983) 295–302.
[2] D.P. Bertsekas, Dynamic Programming and Optimal Control, 3rd ed., vol. I, Athena Scientific, 2005.
[3] D.P. Bertsekas, Linear Network Optimization: Algorithms and Codes, M.I.T. Press, Cambridge, Massachusettes, 1991.
[4] D.P. Bertsekas, A simple and fast label correcting algorithm for shortest paths, Networks 23 (7) (1993) 703–709.
[5] B.V. Cherkassky, A.V. Goldberg, T. Radzik, Shortest path algorithms: theory and experimental evaluation, Mathematical Programming 73 (1996) 129–174.
[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[7] E. Dijkstra, A note on two problems in connexion with graphs, Numerical Mathematics 1 (1959) 269–271.
[8] L. Di Puglia Pugliese, F. Guerriero, Shortest path problem with forbidden paths: the elementary version, European Journal of Operational Research 227 (2) (2013) 254–267.
[9] L. Di Puglia Pugliese, F. Guerriero, Dynamic programming approaches to solve the shortest paths problem with forbidden paths, Optimization Methods and Software 28 (2) (2013) 221–255.
[10] I. Dumitrescu, N. Boland, Algorithms for the weight constrained shortest path problem, International Transactions in Operational Research 8 (1) (2001) 15–29.
[11] I. Dumitrescu, N. Boland, Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem, Networks 42 (3) (2003) 135–153.
[12] D. Feillet, P. Dejax, M. Gendreau, C. Gueguen, An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems, Networks 44 (3) (2004) 216–229.
[13] P. Festa, Complexity analysis and optimization of the shortest path tour problem, Optimization Letters 6 (1) (2012) 163–175.
[14] P. Festa, S. Pallottino, A Pseudo-Random Networks Generator, Technical Report, Department of Mathematics and Applications R. Caccioppoli, University of Napoli FEDERICO II, Italy, 2003.
[15] R.W. Floyd, Algorithm 97: shortest path, Communications of the ACM 5 (6) (1962) 345.
[16] F. Guerriero, V. Lacagnina, R. Musmanno, A. Pecorella, A class of label-correcting methods for the K shortest paths problem, Operations Research 49 (3) (2001) 423–429.
[17] G.C. Hunt, M.M. Michael, S. Parthasarathy, M.L. Scott, An efficient algorithm for concurrent priority queue heaps, Information Processing Letters 60 (1996) 151–157.
[18] D. Klingman, A. Napier, J. Stutz, NETGEN-A program for generating large scale (un)capacitated assignment, transportation and minimum cost flow network problems, Management Science 20 (1974) 814–822.
[19] R. Muhandiramge, N. Boland, Simultaneous solution of Lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem, Networks 53 (4) (2009) 358–381.
[20] G. Righini, M. Salani, New dynamic programming algorithms for the resource constrained elementary shortest path problem, Networks 51 (3) (2008) 155–170.