Comput Optim Appl (2016) 63:559–583
DOI 10.1007/s10589-015-9773-1

# On the shortest path problem with negative cost cycles

**Luigi Di Puglia Pugliese**[1] · **Francesca Guerriero**[1]

Received: 14 February 2014 / Published online: 18 July 2015


**Abstract** In this paper, the elementary single-source all-destinations shortest path problem is considered. Given a directed graph, containing negative cost cycles, the aim is to find paths with minimum cost from a source node to each other node, that do not contain repeated nodes. Two solution strategies are proposed to solve the problem under investigation and their theoretical properties are investigated. The first is a dynamic programming approach, the second method is based on the solution of the $k$ shortest paths problem, where $k$ is considered as a variable. Theoretical aspects related to the innovative proposed strategies to solve the problem at hand are investigated. The practical behaviour of the defined algorithms is evaluated by considering random generated networks and instances derived from vehicle routing benchmark test problems.

**Keywords** Shortest paths · Negative cost cycles · Dynamic programming · $k$ shortest paths

## 1 Introduction

The Shortest Path Problem (*SPP*) is one of the most studied problems in network optimization [15,23,28]. The problem comes up in practice and arises as sub-problem in many network optimization algorithms. Solution approaches for the *SPP* have been studied for a long time (e.g., [8,23,29,44]). More recently, some improvements and

✉ Luigi Di Puglia Pugliese
luigi.dipugliapugliese@unical.it

Francesca Guerriero
francesca.guerriero@unical.it

[1] Department of Mechanical, Energy and Management Engineering, University of Calabria, 87036 Rende, CS, Italy

computational studies have been presented by Ahuja et al. [1], Gabow and Tarjan [31] and Goldberg [34]. The reader is referred to the works of Cherkassky et al. [12] and Gallo and Pallottino [32] for detailed surveys.

In its basic formulation, the objective is to determine a minimum cost path through a network from a given source node to a destination node. Several polynomial time solution approaches have been developed in the scientific literature to address the *SPP* in the case there are no negative cost cycles (e.g., see [16,23,42]). On a network with negative arc costs, but with all cycles having non-negative costs, the best currently known time bound $\mathcal{O}(nm)$ is achieved by the Bellman-Ford algorithm [8,29,44], where $n$ and $m$ denote the number of nodes and arcs in the network, respectively. If the arc costs are non-negative, implementations of Dijkstra's algorithm [23] achieve better bounds. In particular, an implementation presented by Fredman and Tarjan [30] runs in $\mathcal{O}(m + n \log n)$ time.

An interesting extension of the *SPP* is represented by the $k$ shortest paths problem (*kSPP*), whose aim is to find a set of $k$ shortest paths for each node by considering the first, the second and so on up to the $k$-th shortest path. Many solution approaches have been defined to solve the *kSPP* (e.g., see [36]) and its loopless counterpart (e.g., see [43,55]). Dreyfus [25] and Yen [55] cite several additional papers on this subject going back as far as 1957.

When negative cost cycles are present in the network, the *SPP* is not well defined, that is, a finite optimal solution does not exist. In this case, the problem is to check whether a simple cycle, whose arc costs sum up to a negative number, is present in the network. This problem is known as Negative Cost Cycle Detection Problem (*NCCDP*). It is worth noting that the *NCCDP* is polynomial. Several procedures have been defined for the *NCCDP*. The Bellman-Ford algorithm is one of the earliest and to date asymptotically fastest algorithm for the *NCCDP* [12]. Recently, Subramani [51] has introduced a new approach for the *NCCDP*; the proposed solution strategy is based on exploiting the connections between the *NCCDP* and the problem of checking whether a system of difference constraints is feasible.

The scientific literature provides several works dealing with algorithms that list up all cycles in directed as well as undirected graphs in which arc costs are not considered [11,47]. Yamada and Kinoshita [54] addressed the problem of finding all cycles in a directed graph with negative costs and they proposed a recursive algorithm. All the cited references do not solve the *SPP* in presence of negative cost cycles, rather they enumerate all the cycles in the network or check whether a negative cost cycle exists.

Solving the *SPP* in general graph means to find a shortest elementary path, i.e., a shortest path with distinct nodes. The elementarity requirement in a network with negative cost cycles has been studied in the context of branch and price algorithm for the Vehicle Routing Problem (*VRP*), see for instance [17]. Indeed, the pricing problem is formulated on a cyclic graph with no restriction in the cost sign. Besides the elementarity requirement, constraints on the consumption of resource along paths are introduced. The scientific literature refers to this problem as the Resource Constrained Elementary Shortest Path Problem (*RCESPP*). Starting with the seminal work of Beasley and Christofides [6], a variety of solution approaches have been developed to solve to optimality the *RCESPP* (see, e.g., [20,22,39]). The single-source single-destination version of the Elementary Shortest Path Problem (*ESPP*) can be viewed

as a particular instance of the *RCESPP*, where the resource consumption constraints are removed.

In this paper, we deal with the more general case in which an elementary shortest path must be found for all nodes and resource consumptions are not taken into account. In other words, we address the single-source all-destinations version of the *ESPP*. In order to solve to optimality the single-source all-destinations *ESPP*, we define a dynamic programming approach, applying the concepts introduced by Boland et al. [10] and Righini and Salani [48] for the *RCESPP*. In addition, an innovative method, that is, a dynamic $k$ shortest paths algorithm, is designed. The value of $k$ is a variable and may assume an arbitrary value for each node of the network.

An in depth analysis is carried out to evaluate the theoretical complexity of the proposed solution approaches and an extensive computational phase is conducted in order to asses the efficiency of the defined optimal solution strategies. The proposed methods are able to solve also the single-source single-destination *ESPP*. However, it is worth observing that well-tailored solution approaches for the single-source single-destination version can be devised but this topic is out of the scope of this work. The reader is referred to [24,38] for the latter version of the problem.

Studying the *ESPP* is motivated by different main issues. First, the solution approaches for the *ESPP* proposed in this paper, can be used to determine lower bounds on the optimal cost and to solve the dual Lagrangean problem associated with the *RCESPP*. This information can be used to improve the performance of the state-of-the-art methods.

Secondly, the *ESPP* models the currency conversion problem [49]. Indeed, when the nodes represent currencies and the arcs the transactions with costs equal to exchange rates, the problem is to find the maximum product path, that is, the best exchange sequence. The problem can be easily transformed in the *SPP* and, since the graph can contain negative cost cycles, it can be viewed as an instance of the *ESPP*.

In addition, the *ESPP* is the pricing problem when the multiple traveling salesman problem (*mTSP*) is solved with column-generation approaches (see, e.g. [7]). Even though the *mTSP* can be used to represent a great variety of real-life applications, the scientific literature has not devoted too much attention to this problem. The proposed solution approaches for the *mTSP* are based on either the branch and bound scheme [2,33] or focus on the transformation of the *mTSP* to the *TSP* [45,46]. For this reason, the *ESPP* as pricing problem of the *mTSP* has not been addressed and exact algorithms are not available.

The paper is organized as follows. The proposed solution approaches, along with their theoretical analysis are presented in Sect. 2. Section 3 is devoted to the discussion of the computational results. Section 4 presents our conclusions. The paper ends with the Appendix, where a toy example is used to show the operations executed by the proposed algorithms.

## 2 Proposed solution approaches

The solution approaches proposed in this paper are based on the idea to compute the minimum set of paths for each node such that an elementary shortest path is found for all nodes.

### 2.1 Notations and definitions

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph where $\mathcal{N}$ is the set of $n$ nodes, whereas $\mathcal{A}$ denotes the set of $m \leq |\mathcal{N} \times \mathcal{N}|$ arcs. $\mathcal{N}$ contains also the source node $s$. A cost $c_{ij}$ is associated with each arc $(i, j) \in \mathcal{A}$. A cycle on node $i$ is defined as an ordered sequence of nodes $C_i = (i, j_1, \ldots, j_g, i)$, where $j_h \neq j_k$ for all pairs $h, k \in \{1, \ldots, g\}$ such that $h \neq k$. It can also be viewed as a sequence of consecutive $g + 1$ arcs $\big((i, j_1), \ldots, (j_{g-1}, j_g), (j_g, i)\big) \in \mathcal{A}$. It is worth observing, as shown by Allender [3], that the number of cycles in a complete graph is equal to $\tilde{C} = \sum_{\rho=1}^{n} \frac{n!(\rho-1)!}{\rho!(n-\rho)!}$. A path $\pi_{si}$ from node $s$ to node $i$ is a sequence of nodes $\pi_{si} = (s = i_1, \ldots, i_l = i)$, $l \geq 2$ and a corresponding sequence of $l - 1$ arcs such that the $h$-th arc in the sequence is $(i_h, i_{h+1}) \in \mathcal{A}$ for $h = 1, \ldots, l - 1$. Thus, each path contains at least one arc. Let $M_\pi(j)$ be the multiplicity of node $j$ in path $\pi$, that is, $M_\pi(j) = |\{v : 1 \leq v \leq l, i_v = j\}|$. The path $\pi$ is said to be an *elementary* path if $M_\pi(j) = 1$, for all $j \in \pi$. A cycle $C_i$ on node $i \in \mathcal{N}$ is said to be a negative cost cycle (*NCC*) if $c(C_i) = \sum_{(h,k)\in C_i} c_{hk} < 0$.

Let $\pi_{sj}$ be an elementary path from node $s$ to node $j$ such that $i \in \pi_{sj}$ for some $i \in \mathcal{N}\backslash\{s\}$. Let $\pi_{si} = \pi_{sj} \cup \{(j, i)\}$ be the path obtained by extending path $\pi_{sj}$ to node $i$ through arc $(j, i) \in \mathcal{A}$. Since $i \in \pi_{sj}$, a cycle $C_i = \pi_{ij} \cup \{j, i\}$ is created.

**Proposition 1** *Given a path $\pi_{sj}$ and its sub-path $\pi_{si}$, $C_i$ is a NCC iff $c(\pi_{si}) > c(\pi_{sj}) + c_{ji}$, $(j, i) \in \mathcal{A}$.*

*Proof* The cost of the path $\pi_{sj}$ can be viewed as the sum of the cost of the sub-path $\pi_{si}$ and the sub-path from $i$ to $j$, that is $c(\pi_{sj}) = c(\pi_{si}) + c(\pi_{ij})$. It follows that

$$c(\pi_{si}) > c(\pi_{sj}) + c_{ji} \Leftrightarrow c(\pi_{si}) > c(\pi_{si}) + c(\pi_{ij}) + c_{ji} \Leftrightarrow 0 > c(\pi_{ij}) + c_{ji} = c(C_i). \tag{1}$$

This concludes the proof. □

Since the directed graph $\mathcal{G}$ is not assumed to be acyclic and the arc costs are not constrained in sign, there may be *NCCs* in $\mathcal{G}$. The *ESPP*, from the source $s$ to all other nodes, consists in finding the paths $\pi_{si}$, $\forall i \in \mathcal{N}\backslash\{s\}$ such that $c(\pi_{si})$ is minimized and $M_{\pi_{si}}(j) = 1$, $\forall j \in \pi_{si}$.

It is well known that the Bellman's optimality principle does not apply for this problem, indeed sub-paths of the optimal path for some node may not be optimal. For more details, the reader is referred to [18].

### 2.2 Dynamic multi-dimensional labelling approach

The approach, presented in this section, uses the concept of *critical node* introduced by Kohl [41]. In particular, a node $i$ is said to be critical if there exists a cycle $C_i$ that is a *NCC*. A binary variable is introduced for each critical node in order to keep track whether the node is visited. This binary variable can be viewed as a resource associated with node $i$, bounded to be less than or equal to one (see [6]).

Let $S = \{i_1, i_2, \ldots, i_{|S|}\}$, be the set of critical nodes and $r[p]$, $p = 1, \ldots, |S|$ be the binary variable associated with node $i_p \in S$.

We denote as $y_i = (c(\pi_{si}), r_i)$ the label that contains the cost and the resource consumptions vector of the path $\pi_{si}$ from node $s$ to node $i$. The resource consumptions vector $r_i$ has the following form: $r_i[p] \geq 1,\ \forall p : j_p \in \pi_{si}$ and $r_i[p] = 0,\ \forall p : j_p \notin \pi_{si}$. In other words, the vector $r_i$ keeps track of the nodes $j \in S$ that are visited along the path $\pi_{si}$.

**Definition 1** Let $y_i^1 = (c(\pi_{si}^1), r_i^1)$ and $y_i^2 = (c(\pi_{si}^2), r_i^2)$ be two labels associated with node $i$. The label $y_i^2$ is dominated by the label $y_i^1$ if $c(\pi_{si}^1) \leq c(\pi_{si}^2), r_i^1[p] \leq r_i^2[p]$ for $p = 1, \ldots, |S|$ and at least one inequality is strict.

**Definition 2** A label $y_i = (c(\pi_{si}), r_i)$ associated with node $i$ is said to be efficient or Pareto-optimal if there does not exist a label $\bar{y}_i$ that dominates it.

**Definition 3** A label $y_i = (c(\pi_{si}), r_i)$ associated with node $i$ is said to be feasible if $r_i[p] \leq 1,\ \forall p = 1, \ldots, |S|$.

A set $D(i)$, containing the efficient labels to node $i$, and the set $FS(i) = \{j : (i, j) \in \mathcal{A}\}$ of successor nodes, are associated with each node $i \in \mathcal{N}$.

The main difficulty is that the set $S$ of critical nodes must be known in advance. the enumerative procedure of Yamada and Kinoshita [54] could be executed to determine the entire set of negative cost cycles. The nodes belonging to each negative cost cycle can be used to initialize set $S$. However, this strategy could be inefficient, since only a sub-set of all nodes belonging to all negative cost cycles need to be considered. In addition, the procedure defined by Yamada and Kinoshita [54] is not polynomial.

If the set $S$ is not known in advance, we can modify the solution approaches presented in [10,48], in order to define well-tailored procedure for the *ESPP*. In particular, starting from $S = \emptyset$, a label-setting algorithm is executed. The search is stopped when a *NCC*, named $C_i$, is detected. The set $S$ is incremented by adding node $i$ and the label-setting algorithm is run again. The procedure terminates when no *NCC* is detected. This approach is similar to the general state-space augmenting algorithm proposed in [10] with the difference that the label-setting algorithm is stopped when a negative cost cycle is detected. In Algorithm 1, we describe the steps of the aforementioned procedure, whereas the truncated label-setting algorithm is depicted in Algorithm 2.

### *2.2.1 Correctness of the dynamic multi-dimensional labelling approach*

In this section, we prove the correctness of the proposed solution approach. In particular, at the end of Algorithm 1, the label $y_i$ with the smallest cost among those belonging to the set $D(i)$ is associated with the least cost elementary path from $s$ to $i$.

**Proposition 2** *At the end of Algorithm 2, one of the following two situations can occur:*

1. *a NCC is detected, that is, the stop conditions of line* 13 *are verified;*
2. *if the procedure is stopped by condition of line* 27*, then all the efficient and feasible paths from s to each other node $i \in \mathcal{N}\backslash\{s\}$ are determined.*
   (a) *At the end of each iteration, the following conditions hold:*
      *i.* $D(s) = \{y_s^1\} = \{(0, 0)\}$;

**Algorithm 1** : Dynamic Multi-Dimensional Labelling Approach

1: **Step 1** *(Initialization phase)*
2: Set: $\zeta = 0$; $S^{\zeta} = \emptyset$
3:
4: **Step 2** *(Run the truncated label-setting algorithm)*
5: Run Algorithm 2 with critical nodes restricted to $S^{\zeta}$ and get $C$.
6:
7: **Step 3** *(Cycle detection)*
8: **if** $C \neq \emptyset$ **then**
9: $S^{\zeta+1} = S^{\zeta} \cup \{j\}$, $M_C(j) = 2$;
10: $\zeta = \zeta + 1$;
11: Go to **Step 2**.
12: **else**
13: STOP.
14: **end if**

*ii.* $\forall j \in \mathcal{N}$, *if* $D(j) \neq \emptyset$ *[i.e.:* $D(j) = \{y_j^1, y_j^2, \ldots, y_j^{\Xi}\}$*] and* $j \neq s$, *then* $y_j^{\xi}, \xi = 1, \ldots, \Xi$ *is a label related to a feasible path from node* $s$ *to node* $j$ *and* $D(j)$ *is an efficient set.*

*(b) Upon termination of the algorithm, if* $D(j) \neq \emptyset$, $j \in \mathcal{N}$ *and* $j \neq s$, *then* $D(j)$ *contains the labels of all efficient and feasible paths from node* $s$ *to node* $j$.

*Proof* Let us consider case 1. The conditions of line 13 derive from Proposition 1. Thus, if they are verified, then a *NCC* is detected. If case 1 does not occur, then all *NCCs* have been forbidden and the algorithm terminates when the list $L$ is found empty, that is, case 2.

In what follows, we prove case 2.

Let us consider case 2a. Condition 2(a)i holds because, initially, $D(s) = \{y_s^1\} = \{(0, 0)\}$ and, by the rules of the algorithm, $D(s)$ cannot change.

We prove condition 2(a)ii by induction on the iteration count. Indeed, initially, condition 2(a)ii holds, since node $s$ is the only node for which the set $D(s)$ is nonempty. Suppose that 2(a)ii holds for some node $j$ at the beginning of some iteration. Let $y_i^{\delta}$ be the label removed from $L$.

If $i = s$ (which only occurs at the first iteration) and $\delta = 1$, then at the end of this iteration, we have $D(j) = y_j^1$ for all successor nodes $j$ of $s$ such that the corresponding path $\pi_{sj}$ is feasible, $D(j) = \emptyset$ for all other nodes $j \neq s$, $j \notin FS(i)$. Thus, the set of labels has the required property.

If $i \neq s$, then $y_i^{\delta}$ is the label of some feasible path $\pi_{si}^{\delta}$ starting from $s$ and ending to $i$ that is not dominated by the other paths in $D(i)$, by the induction hypothesis. If the set $D(j)$ changes, for some node $j$, such that $j \in FS(i)$, as a result of the iteration, a new feasible label $\bar{y}_j = (c(\bar{\pi}_{sj}), \bar{r}_j)$ is obtained for node $j$. The created label is related to the feasible path $\pi_{sj}$ consisting of path $\pi_{si}^{\delta}$ followed by the arc $(i, j)$. Finally, note that, by the rules of the algorithm, the newly created label is added to $D(j)$ only if it is an efficient label. This completes the induction proof of 2(a)ii.

**Algorithm 2** : Truncated label-setting algorithm with $S^{\zeta}$

**Step 1** *(Initialization phase)*
Set: $y_s^1 = (0, 0)$; $D(s) = \{y_s^1\}$; $D(j) = \emptyset,\ \forall j \in \mathcal{N}\backslash\{s\}$; $L = \{y_s^1\}$; $C = \emptyset$.

**Step 2** *(Label selection)*
Select the lexicografically minimal label $y_i^{\xi}$ from the list $L$ and remove it from $L$.

**Step 3** *(Label extension)*
**for all** $j \in FS(i)$, $j \neq s$ **do**
  Set: $\bar{\pi}_{sj} = \pi_{si}^{\xi} \cup \{(i, j)\}$; $c(\bar{\pi}_{sj}) = c(\pi_{si}^{\xi}) + c_{ij}$; $\bar{r}_j[p] = r_i^{\xi}[p]$, $p = 1, \ldots, |S^{\zeta}|$;
  **if** $j \in S^{\zeta}$ **then**
    $\bar{r}_j[\hat{p}] = \bar{r}_j[\hat{p}] + 1$ with $i_{\hat{p}} = j$.
  **end if**
  **if** $j \notin S^{\zeta}$ and $j \in \pi_{si}^{\xi}$ and $c(\bar{\pi}_{sj}) < c(\pi_{sj}^{\xi})$ **then**
    STOP. A *NCC* $C$ is detected.
    Return $C$.
  **else**
    **if** $\bar{r}_i[p] \leq 1,\ p = 1, \ldots, |S^{\zeta}|$ **then**
      **if** $\bar{y}_j = (c(\bar{\pi}_{sj}), \bar{r}_j)$ is not dominated by any label in $D(j)$ **then**
        $D(j) = D(j) \cup \{\bar{y}_j\}$; $L = L \cup \{\bar{y}_j\}$;
        remove from $D(j)$ and $L$ all the labels that are dominated by $\bar{y}_j$.
      **end if**
    **end if**
  **end if**
**end for**

**Step 4** *(Termination check)*
**if** $L = \emptyset$ **then**
  STOP. The label $y_i = argmin_{\bar{y} \in D(i)}\{c(\bar{\pi}_{si})\}$ is associated with the optimal elementary path from $s$ to $i$, $\forall i \in N$.
  Return $C$.
**else**
  Go to **Step 2**.
**end if**

Let now consider condition 2b. Using part 2(a)ii, we have that, at each iteration, $\forall j \in \mathcal{N}$ such that $D(j) \neq \emptyset$, $D(j)$ is an efficient set. Thus, the property mentioned is also satisfied when the algorithm terminates. In addition, the way in which the candidate list $L$ is updated and the termination condition (i.e., the algorithm terminates when there are no more labels left to be scanned) guarantee that all the labels with the potential to determine a new label for at least one node are scanned during the execution of the algorithm. □

**Proposition 3** *(*Correctness of Algorithm 1*) At the end of Algorithm 1, the label $y_i$ with the smallest cost among those belonging to the set $D(i)$ is associated with the least cost elementary path from $s$ to $i$, $\forall i \in \mathcal{N}$.*

*Proof* From Proposition 2, it follows that all the efficient and feasible labels $y_i, \forall i \in \mathcal{N}$, are determined. Since the set $D(i)$ is an efficient set, the path $\bar{\pi}_{si} = argmin_{y \in D(i)}\{c(\pi_{si})\}$ has minimum cost among all the feasible paths from $s$ to $i$, that is, $\bar{\pi}_{si}$ is the optimal solution $\forall i \in \mathcal{N}$. □

#### 2.2.2 Complexity analysis

The complexity of Algorithm 1 is $\mathcal{O}(n2^{2n})$, in the worst case. This result is formally stated in the following lemma.

**Lemma 1** *In the worst case, the complexity of Algorithm 1 is* $\mathcal{O}(n2^{2n})$.

*Proof* The total number of efficient labels is at most $2^{|S|}$. Thus, at most $2^{|S|}$ labels are selected from set $L$. Since each label contains $|S| + 1$ elements, the total number of operations required by the dominance check is bounded by $|S|2^{|S|}$. Thus, the complexity of Algorithm 2 is $\mathcal{O}(|S|2^{2|S|})$. A node $i$ is added to the set $S$ at each iteration of Algorithm 2. Since the cardinality of set $S$ is $n$ in the worst case, the complexity of Algorithm 1 is $\mathcal{O}(n2^{2n})$. □

The scientific literature proposes a relaxation of the elementary constraint by considering a surrogate resource $\gamma$ which counts the number of nodes in the path.

This idea was proposed by Feillet et al. [26] and applied by Righini and Salani [48] for solving the *RCESPP*. It is worth observing that [26,48] consider the single-source single-destination case. In their approach, a labelling algorithm is executed and labels with $\gamma > |\mathcal{N}|$ are declared unfeasible. At the end of the algorithm, if the path is acyclic, then it is the optimal elementary path, otherwise, the repeated nodes are added to the set $S$. In our algorithm, the labelling procedure (Algorithm 2) is stopped when a cycle is detected, thus the resource $\gamma$ does not make sense. The approach to augment the set $S$ is the same of that proposed by Boland et al. [10] and by Righini and Salani [48]. The difference is that in the latters the optimal path has to be found for a single destination. For more details, the reader is referred to [19,22]. In [37] the authors proposed a new state-space reduction technique combined with the strategies defined in [48]. This strategy cannot be extended to the single-source all-destinations version of the *ESPP*.

For a numerical example illustrating the operations executed by Algorithm 1 the reader is referred to the Appendix.

### 2.3 Labelling approach based on the $k$ shortest paths problem

In this section, we describe a labelling approach in which the first $k$ paths are determined for each node $i \in \mathcal{N}$. The aim of the procedure is to determine the smallest set of paths for each node $i$. Under this respect, $k$ is a variable and can take different values for each node. The main idea is to start with $k = 1$ for each node. Whenever a *NCC* is detected, $k$ is incremented for a specific node and the labelling algorithm is run again. When no *NCC* is detected, from the set of paths associated with each node, the optimal solution is chosen.

It is worth observing that when the *kSPP* is solved, the paths are ranked by non-decreasing cost. In our context, it is necessary to store equivalent paths. This is necessary because paths with the same cost can have a different structure, that is, they can contain different nodes.

Let $K_i$ be the number of best paths from node $s$ to node $i$. The set $\Pi_i$ contains the $K_i$ paths ordered by non-decreasing cost, that is, $c(\pi_{si}^k) \leq c(\pi_{si}^{k+1}),\ \forall k = 1, \ldots, K_i - 1$. For the case of $q$ equivalent paths, that is, $c(\pi_{si}^k) = c(\pi_{si}^{k+1}),\ k = h, \ldots, h+q$, with $h \geq 1$ and $h + q \leq K_i$ we have that $\pi_{si}^k \neq \pi_{si}^p,\ k = h, \ldots, h+q-1$ and $p = k+1, \ldots, h+q-1$.

**Definition 4** Set $\Pi_i, \forall\, i \in \mathcal{N}$ is said to be *path-elementary*, if each path $\pi_{si}^k \in \Pi_i, k = 1, \ldots, K_i$ does not contain repeated nodes.

A vector $y_i \in \mathbb{R}^{K_i}$ is associated with each node $i$ and stores the costs of the paths belonging to the set $\Pi_i$, that is, $y_i[k]$ represents the cost of path $\pi_{si}^k \in \Pi_i$.

Every time a *NCC* $C$ is detected, the number of paths that have to be found for each node $i \in C$ is increased, that is, $K_i = K_i + 1,\ \forall i \in C$, and the labelling algorithm is executed again. When no *NCC* is detected, $y_i[1]$ is the cost of the optimal elementary shortest path from node $s$ to node $i \in \mathcal{N}$.

The proposed solution approach iteratively solves the *kSPP* for a fixed value of $K_i,\ \forall i \in \mathcal{N}$. The steps of the labelling procedure are depicted in Algorithm 3. It is iteratively run after that $K_i,\ \forall i \in \mathcal{N}$, is coherently updated. The dynamic labelling solution strategy is reported in Algorithm 4.

### *2.3.1 Correctness of the dynamic labelling approach*

In this section, we prove the correctness of the proposed solution approach. In particular, at the end of Algorithm 4, $\pi_{si}^1$ is the least cost elementary path from $s$ to $i$.

**Proposition 4** *At the end of Algorithm 3, we have one of the following two cases:*

1. *a NCC is detected, that is, conditions of line 9 are verified;*
2. *if condition of line* 30 *is verified, then all the first $K_i$ elementary paths are determined for each node $i \in \mathcal{N}$, that is, sets $\Pi_i,\ \forall i \in \mathcal{N}$, are path-elementary.*
   (a) *At the end of each iteration, the following conditions hold:*
      i. $y_s[1] = 0$;
      ii. $\forall i \in \mathcal{N} \backslash \{s\}$, *if* $y_i[k] \neq +\infty$, *then* $\pi_{si}^k$ *is an elementary path.*
   (b) *Upon termination of the algorithm, $\Pi_i,\ \forall i \in \mathcal{N}$, is path-elementary.*

*Proof* Let us consider case 1: The first part of conditions in line 9, that is, $y_i[k] \neq +\infty,\ \forall k \in \{1, \ldots, K_i\}$, checks whether all the $K_i$ paths to node $i$ are determined. The second one checks whether it is not possible to generate at least one elementary path. The third part derives from Proposition 1, thus if it is verified for some $\bar{k}$, a *NCC* is found and the algorithm terminates. It follows that the presence of a *NCC* is verified only if the entire set of paths from node $s$ to the considered node $i$ are determined. If case 1 does not occur, then the determined value $K_i$ is sufficient to skip *NCCs* involving node $i,\ \forall i \in \mathcal{N}$, and the algorithm terminates when the list $L$ is found empty, that is, case 2.

**Algorithm 3** : Truncated labelling algorithm for dynamic *kSPP*

**Step 1** *(Initialization phase)*
Set: $y_s[1] = 0; y_i[k] = +\infty, k = 1, \ldots, K_i, \forall i \in \mathcal{N}\backslash\{s\}; \pi_{ss}^1 = \{s\}, \Pi_s = \{\pi_{ss}^1\}; \Pi_i = \emptyset, \forall i \in \mathcal{N}\backslash\{s\}; L = \{s\}; C = \emptyset.$

**Step 2** *(Node selection)*
Select a node $i$ from $L$ and delete it from $L$.

**Step 3** *(Label extension)*
**for all** $j \in FS(i)$, $j \neq s$ **do**
  **if** $y_i[k] \neq +\infty,\ \forall k \in \{1, \ldots, K_i\}$ and $\nexists \hat{k} : j \notin \pi_{si}^{\hat{k}}$ and $\exists \bar{k} : j \in \pi_{si}^{\bar{k}}, y_i[\bar{k}] + c_{ij} < y_j[1]$ **then**
    STOP. A cycle $C = \pi_{si}^{\bar{k}} \cup \{(i, j)\}$ is detected.
    Return $C$.
  **else**
    **for all** $k = 1, \ldots, K_i : y_i[k] \neq +\infty, j \notin \pi_{si}^k$ **do**
      **for all** $\xi = 1, \ldots, K_j$ **do**
        **if** $y_i[k] + c_{ij} < y_j[\xi]$ or $(y_i[k] + c_{ij} = y_j[\xi]$ and $\pi_{si}^k \cup \{(i, j)\} \neq \pi_{sj}^{\xi})$ **then**
          $y_j[\delta + 1] = y_j[\delta], \delta = \xi, \ldots, K_j - 1;$
          $\pi_{sj}^{\delta+1} = \pi_{sj}^{\delta}, \delta = \xi, \ldots, K_j - 1;$
          $y_j[\xi] = y_i[k] + c_{ij};$
          $\pi_{sj}^{\xi} = \pi_{si}^k \cup \{(i, j)\};$
          $\Pi_j = \Pi_j \cup \left\{\pi_{sj}^{\xi}\right\}$
          add node $j$ to $L$ if it does not already belong to it.
          BREAK
        **end if**
      **end for**
    **end for**
  **end if**
**end for**

**Step 4** *(termination check)*
**if** $L = \emptyset$ **then**
  STOP. $y_i[1]$ is the cost of the optimal elementary path $\pi_{si}^1$.
  Return $C$.
**else**
  Go to **Step 2**.
**end if**

Case 2: Let us consider case 2a. Condition 2(a)i holds because, initially, $y_s[1] = 0$ and, by the rules of the algorithm, $y_s[1]$ cannot change.

We prove condition 2(a)ii by induction on the iteration count. Indeed, initially, condition 2(a)ii holds, since node $s$ is the only node for which the cost is not equal

**Algorithm 4** : Dynamic Labelling Approach

**Step 1** *(Initialization phase)*
Set: $\zeta = 0$; $K_s = 1$; $K_i^{\zeta} = 1\ \forall i \in \mathcal{N}\backslash\{s\}$.

**Step 2** *(Run the truncated labelling algorithm)*
Run Algorithm 3 with $K_i^{\zeta}, i \in \mathcal{N}$ and get $C$.

**Step 3** *(Cycle detection)*
**if** $C \neq \emptyset$ **then**
  $K_i^{\zeta+1} = K_i^{\zeta} + 1,\ \forall i \in C,\ K_i^{\zeta+1} = K_i^{\zeta},\ \forall i \in \mathcal{N}\backslash C$;
  $\zeta = \zeta + 1$;
  Go to **Step 2**.
**else**
  STOP.
**end if**

to $+\infty$. Suppose that 2(a)ii holds for some node $j$ at the beginning of some iteration. Let $i$ be the node removed from $L$.

If $i = s$ (which only occurs at the first iteration), then at the end of this iteration, we have $y_j[1] \neq +\infty$ for all successor nodes $j$ of $s$ and the corresponding path $\pi_{sj} = \{(s, j)\}$ is elementary, and $y_j[1] = +\infty$ for all other nodes $j \neq s,\ j \notin FS(i)$. Thus, the set of paths has the required property.

If $i \neq s$, then $y_i[k]$ is the cost of the $k$-th path $\pi_{si}^k$ starting from $s$ and ending to $i$ that does not contain repeated nodes, by the induction hypothesis. If $y_j[k]$ changes, for some node $j \in FS(i)$ and $k$, then a new path is obtained for node $j$. The created path $\pi_{sj}^k$ consists of path $\pi_{si}^k$ followed by the arc $(i, j)$. Finally, note that, by the rules of the algorithm, the newly created path is elementary because $j \notin \pi_{si}^k$. This completes the induction proof of 2(a)ii.

Let now consider condition 2b. Using part 2(a)ii, we have that, at each iteration, $\pi_{sj}^k,\ \forall j \in \mathcal{N}$ such that $y_j[k] \neq +\infty$ is an elementary path. Thus, the property mentioned is also satisfied when the algorithm terminates. In addition, the way in which the candidate list $L$ is updated and the termination condition (i.e., the algorithm terminates when there are no more nodes left to be scanned) guarantee that all the nodes with the potential to determine a new path for at least one node are scanned during the execution of the algorithm. □

**Proposition 5** *(*Correctness of Algorithm 4*) At the end of Algorithm 4, $\pi_{si}^1$ is the optimal path from node $s$ to node $i$, $\forall i \in \mathcal{N}$.*

*Proof* The algorithm terminates when no cycle is detected. From Proposition 4, we know that if no cycles are detected, then Algorithm 3 provides the path-elementary set $\Pi_i,\ \forall i \in \mathcal{N}$, containing the first $K_i$ elementary paths (see part 2 of Proposition 4). Since the paths are ordered in non-decreasing order of the cost, the path $\pi_{si}^1$ belonging to each set $\Pi_i$ represents the least cost path. In addition, being $\Pi_i$ path-elementary, $\pi_{si}^1$ is the optimal path. □

*2.3.2 Complexity analysis*

Let $K$ be the highest number of paths to some node $i$ at the end of the Algorithm 4, that is, $K = \max_{i \in \mathcal{N}}\{|K_i|\}$. In the worst case, $|K_i|$ is equal to the number of cycles involving node $i$. We know that the number of cycles $C_i$ in a complete network is equal to $\sum_{\rho=1}^{n-1} \prod_{\beta=1}^{\rho}(n-\beta)$, where $\rho$ indicates the number of nodes, different from $i$, included in the cycles. Since $K_i$ is incremented for each node $i$ included in the detected cycles $C$ (see line 9 of Algorithm 4), $|K_i|$ assumes, in the worst case, the following value: $\sum_{\rho=1}^{n-1}(\rho+1)\prod_{\beta=1}^{\rho}(n-\beta)$.

**Lemma 2** *In the worst case, the complexity of the Algorithm 4 is* $\mathcal{O}(n^2K^4)$.

*Proof* The running time of one iteration of the inner forloop in line 14 of Algorithm 3 is $\mathcal{O}(K)$. Since the forloops of line 13 and 14 of Algorithm 3 take $K^2$, the number of iterations in lines 13 – 25 of Algorithm 3 is $\mathcal{O}(K^3)$. The $FS$ contains at most $n-1$ elements, thus lines 8 – 27 of Algorithm 3 perform $\mathcal{O}(nK^3)$ operations. The forloop of line 8 of Algorithm 3 is invoked $nK$ times. Consequently, Algorithm 3 takes $\mathcal{O}(n^2K^4)$. □

A numerical example illustrating the operations executed by Algorithm 4 is reported in the Appendix.

## 3 Computational experiments

The aim of this section is to evaluate the behaviour of the proposed solution approaches and to compare them in terms of computational cost. The proposed algorithms have been coded in Java and tested by using an Intel(R) core(TM) i7 CPU M620, 2.67 GHz, ram 4.00 GB, under a Microsoft 7 operating system. In the next section, we present the considered instances and how they were generated.

### 3.1 Test problems

The computational results are carried out on two groups of test problems. The first one is composed of random networks, the instances belonging to the second group are derived from *VRP* benchmark test problems.

The test problems of the first group (i.e., fully random networks) have been generated randomly by using the Netgen generator of Klinglman et al. [40]. In particular, we consider networks with number of nodes $n$ belonging to {300, 350, 400, 450, 500} and for each value of $n$ we consider three arc densities, that is, 10, 30 and 50. With these parameters, the minimum and maximum number of arcs are 3000 and 25000, respectively. The cost $c_{ij}, \forall(i,j) \in \mathcal{A}$ is randomly generated from the interval [0, 100].

For each network, a given number of instances have been generated, in such a way that at least a fixed number of negative cost cycles is present. The procedure used to build the test problems is detailed in what follows. Let $\#c$ be the number of negative cost cycles and let $leng$ be the number of arcs belonging to a given cycle. For each fully random network, we have generated 30 instances by letting $\#c = 1, \ldots, 30$. In

**Table 1** Characteristics of the *CVRP* benchmark instances

| Set | Paper | $\|Set\|$ | Nodes | |
|---|---|---|---|---|
| | | | Min | Max |
| $F$ | [27] | 3 | 45 | 135 |
| $A-A$ | [4] | 27 | 32 | 80 |
| $A-B$ | [4] | 23 | 31 | 78 |
| $A-P$ | [4] | 24 | 16 | 101 |
| $CE$ | [14] | 13 | 13 | 101 |
| $S$ | [50] | 6 | 50 | 100 |
| $B$ | [53] | 60 | 100 | 100 |
| $CMT$ | [13] | 14 | 50 | 199 |
| $T$ | [52] | 13 | 75 | 385 |
| $GWKC$ | [35] | 20 | 200 | 483 |

other words, each instance has a number of cycles at least equal to $\#c$. The value $leng$ has been randomly chosen in the interval $[2, 5]$.

The procedure used to build the test instances relies on the solution of the shortest path problem for each fully random network, obtaining a tree $T$. Let $\pi_{ij}$ be the path from node $i$ to node $j$ in $T$. Exactly $\#c$ cycles are chosen by selecting $\#c$ arcs $(j, i) \in \mathcal{A}\backslash T$. Each cycle $C_i^{\#c}$, $\#c = 1, \ldots, 30$ is composed by the arcs belonging to the path $\pi_{ij}$ and by the arc $(j, i)$. Starting from each leaf node and exploring the related branch by following only inverse arcs, the first path $\pi_{ij}$, from which it is possible to generate a cycle with $leng$ nodes is selected. The cost $c_{ji}$ of arc $(j, i)$ is modified in order to obtain a new cost $\bar{c}_{ji} = c_{ji} - c(C_i^{\#c}) - 1$. It follows that evaluating the cost of the cycle $C_i^{\#c}$ considering the new cost $\bar{c}_{ji}$ we have $\bar{c}(C_i^{\#c}) = -1$.

It is worth observing that the procedure described above does not ensure that exactly $\#c$ negative cost cycles are introduced, but the value $\#c$ indicates the minimum number of negative cost cycles present in the network.

The second group of instances are derived from Capacited *VRP* (*CVRP*) test problems taken from the scientific literature. We have considered ten sets of test problems. Each set is associated with the scientific contribution in which the corresponding test problems have been introduced for the first time.

Table 1 shows the name of each set, the paper in which the related test problems have been introduced, the number of problems and the characteristics of the instances in terms of number of nodes (i.e., the maximum and the minimum number of nodes over the networks belonging to the considered set are reported).

In order to test the algorithm for the *ESPP* on the *CVRP* test problems, we have considered the related *RCESPP*, that is, the pricing problem obtained when a branch and price approach is used to solve the *CVRP*. First, we have modified the *CVRP* test problems by adding a prize at each node. As described in [48], the value of each prize can be chosen in the range $[1, 20]$. The cost of a path is the sum of the cost associated with the arcs minus the prizes collected at the nodes. Second, we get the *ESPP* instances by considering the Lagrangean relaxation of the *RCESPP*. Indeed, given a Lagrangean multiplier, the problem is an instance of the *ESPP*. For each

*RCESPP* test problem we have considered 20 *ESPP* instances by choosing several values for the Lagrangean multiplier. These values are selected in a such way that instances with different degree of complexity are obtained: the lower the value of the Lagrangean multiplier, the more difficult the *ESPP* instances. As suggested in [6], the Lagrangean multiplier is set equal to $0.1, 0.2, \ldots, 2$ for the $1^{st}$, the $2^{nd}$, $\ldots$, the $20^{th}$ *ESPP* instances. Thus we have tested 4060 *ESPP* instances.

### 3.2 Test codes

The codes considered in this work are named $DMLA$, implementing Algorithm 1 described in Sect. 2.2; and the code $DLA$ is related to Algorithm 4, defined in Sect. 2.3.

Considering $DLA$, in order to select a node from $L$, see line 5 of Algorithm 3, we implement the FIFO strategy. In addition, at each iteration, the costs for each node are inizialized by considering the partial solutions obtained in the previous iteration. In particular, the set $L$ contains all nodes such that at least one path has been determined in the previous iteration and the labels are those available at the end of the previous iteration. Of course, for the nodes belonging to the detected cycle, the dimension of the associated label is incremented by one. This type of initialization speeds up the search process at each iteration.

The same type of initialization is not possible for $DMLA$. Indeed, at each iteration, an additional resource is introduced, augmenting the dimension of each label. Since the efficiency of a solution is affected by the dimension of the associated label, a solution that in the previous iteration was dominated, with the new definition of the label could represent an efficient sub-path. For this reason, in $DMLA$ the initialization is that reported in Algorithm 2.

### 3.3 Computational results

In this section, we analyze the behaviour of the proposed solution approaches. In the next section we focus on the results collected when solving the first group of instances, that is, the fully random networks. In Sect. 3.3.2, the resulting best performing algorithm on the first group of instances is tested on the second one.

#### *3.3.1 Computational results on the fully random networks*

The collected results are organized in three classes. The first one is related to the instances with $\#c \in [1, 10]$, the second class to those with $\#c \in [11, 20]$ and the results obtained on the instances with $\#c \in [21, 30]$ belong to the third class. In what follows, we present a summary of the collected computational results. A more detailed accounting of the experiments is given in [18].

**Results for** $DMLA$. The results are collected in Table 2, where the average execution time (column time) and the average number of iterations (column iter) are reported for each value of the indicator $\#c$. It is worth observing that the column iter reports the average number of executions of Algorithm 2 in Algorithm 1. As expected, the

**Table 2** Average execution time in ms and average number of iterations obtained by $DMLA$ when solving the first, the second and the third class of test problems

| $DMLA$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| #$c$ | Time | Iter | #$c$ | Time | Iter | #$c$ | Time | Iter |
| 1 | 68.64 | 1.07 | 11 | 3411.22 | 18.00 | 21 | 49217.28 | 39.33 |
| 2 | 127.92 | 2.27 | 12 | 4501.15 | 20.07 | 22 | 57689.17 | 41.53 |
| 3 | 217.36 | 3.67 | 13 | 6140.20 | 22.40 | 23 | 73947.59 | 43.60 |
| 4 | 434.72 | 5.80 | 14 | 7985.17 | 24.67 | 24 | 93819.00 | 45.13 |
| 5 | 582.40 | 7.40 | 15 | 11858.16 | 27.53 | 25 | 117422.99 | 46.93 |
| 6 | 885.05 | 9.33 | 16 | 14540.33 | 29.93 | 26 | 148047.03 | 48.47 |
| 7 | 1215.77 | 11.00 | 17 | 20057.57 | 32.53 | 27 | 208258.21 | 50.73 |
| 8 | 1886.57 | 13.20 | 18 | 26119.77 | 34.93 | 28 | 351173.85 | 53.93 |
| 9 | 2338.98 | 14.60 | 19 | 33535.01 | 36.93 | 29 | 438542.73 | 55.47 |
| 10 | 3063.86 | 17.00 | 20 | 42470.75 | 39.47 | 30 | 599554.48 | 57.87 |
| AVG | 1082.13 | 8.53 | | 17061.93 | 28.65 | | 213767.23 | 48.30 |

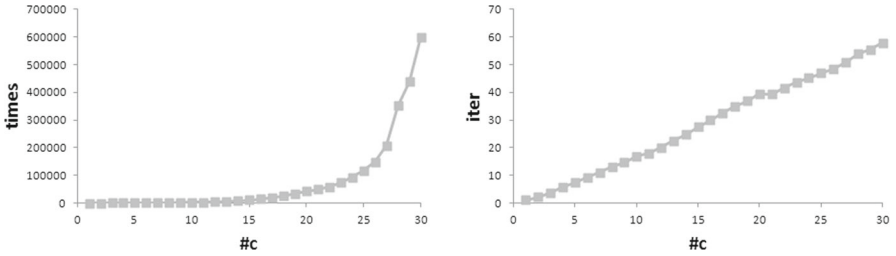



**Fig. 1** Average execution time and average number of iterations of $DMLA$ as a function of the indicator #$c$ of the number of negative cost cycles

computational results underline that the higher the number of negative cost cycles, the higher the execution time. Table 2 shows that the computational cost for solving the second and the third class of instances is 15.77 and 197.54 times higher than the execution time required to solve the instances of the first class.

This behaviour can be justified by considering the number of iterations. Indeed, $DMLA$ performs, on average, 8.53, 28.65 and 48.30 iterations, for the instances belonging to the first, the second and the third class, respectively. Figure 1 shows the trend of the average computational cost and the average number of iterations with respect to the indicator #$c$ of the number of negative cost cycles.

From Figure 1, it is clear that the computational cost grows exponentially with respect to the increase of the number of negative cost cycles, whereas, a linear trend is observed for the number of iterations. This behaviour is due to the fact that the higher the number of negative cost cycles, the higher the execution time per iteration. In particular, the time per iteration grows exponentially with the indicator #$c$ of the number of negative cost cycles. This trend is shown in Figure 2.

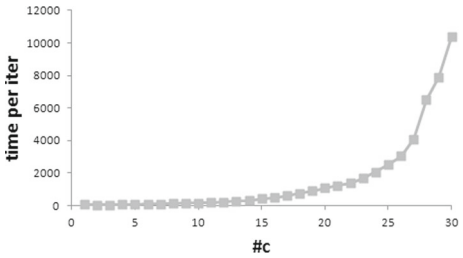



**Fig. 2** Average execution time per iteration of $DMLA$ as a function of the indicator #$c$ of the number of negative cost cycles

**Table 3** Average execution time in ms and average number of iterations obtained by $DLA$ when solving the first, the second and the third class of test problems

| $DLA$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| #$c$ | Time | Iter | #$c$ | Time | Iter | #$c$ | Time | Iter |
| 1 | 98.80 | 3.27 | 11 | 573.04 | 33.20 | 21 | 1045.21 | 71.13 |
| 2 | 133.12 | 4.87 | 12 | 635.44 | 36.33 | 22 | 1128.41 | 76.87 |
| 3 | 151.84 | 6.73 | 13 | 659.36 | 38.60 | 23 | 1210.57 | 85.73 |
| 4 | 166.40 | 8.93 | 14 | 693.68 | 40.73 | 24 | 1290.65 | 92.07 |
| 5 | 197.60 | 11.13 | 15 | 747.77 | 46.07 | 25 | 1278.17 | 95.00 |
| 6 | 210.08 | 12.47 | 16 | 887.13 | 54.00 | 26 | 1361.37 | 102.87 |
| 7 | 234.00 | 15.87 | 17 | 957.85 | 57.73 | 27 | 1551.69 | 113.07 |
| 8 | 276.64 | 18.67 | 18 | 975.53 | 60.67 | 28 | 1982.25 | 135.87 |
| 9 | 372.32 | 22.47 | 19 | 925.61 | 59.60 | 29 | 2415.94 | 162.07 |
| 10 | 392.08 | 25.00 | 20 | 1054.57 | 66.27 | 30 | 2482.50 | 176.40 |
| AVG | 223.29 | 12.94 | | 811.00 | 49.32 | | 1574.67 | 111.11 |

A possible explanation of this trend can be found by considering the dimension of the state-space induced by the definition of the labels. Indeed, the state-space is composed by states associated with efficient partial solutions. The number of Pareto-optimal solutions is strongly related to the dimension of the labels. Indeed, the higher the number of negative cost cycles, the higher the dimension of the labels due to the fact that a higher number of additional node resources must be introduced. In addition, as shown in [5,9,21], the number of non-dominated solutions increases exponentially with the size of the labels. As a consequence, the state-space grows exponentially with the increase of the number of the negative cost cycles.

**Results for** $DLA$. In Table 3, we report the average computational cost under column time and the average number of times Algorithm 3 is called by Algorithm 4 under column iter, for each value of the indicator #$c$. As expected, the higher the number of negative cost cycles the higher the execution time. Indeed, the computational cost for solving the instances belonging to the second and the third class is 3.63 and 7.05 times higher than that required for the instances of the first class. This behaviour is justified by the number of iterations executed by $DLA$. In particular, $DLA$ performs 3.81 and 8.59 times higher number of iterations for the second and the third class of

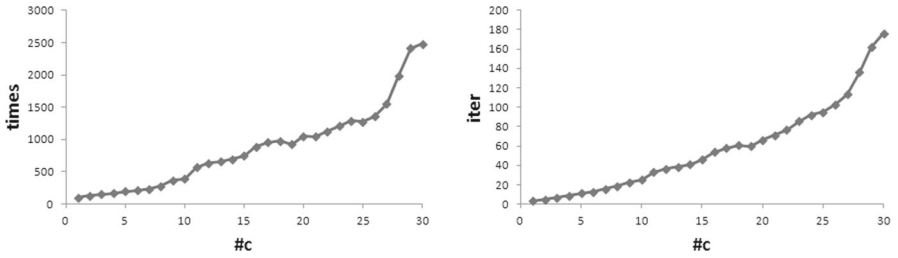



**Fig. 3** Average execution time and average number of iterations of $DLA$ as a function of the indicator $\#c$ of the number of negative cost cycles

**Table 4** Average execution time in ms and average number of iterations varying the density

| Density | $DMLA$ | | $DLA$ | |
|---|---|---|---|---|
| | Time | Iter | Time | Iter |
| 10 | 56.14 | 28.23 | 0.30 | 52.51 |
| 30 | 83.49 | 28.62 | 0.64 | 50.56 |
| 50 | 92.29 | 28.63 | 1.67 | 70.29 |

instances, respectively, than that executed by the algorithm for solving the instances belonging to the first class.

Figure 3 shows the trend of the computational cost and the number of iterations executed by $DLA$.

From Figure 3, it is evident that both the execution time and the number of iterations present the same trend. In particular, both parameters show a linear trend. However, the average execution time and the average number of iterations grow faster for $\#c > 26$ than the increasing observed for $\#c \leq 26$.

**Comparison** The results underline that $DLA$ behaves the best. Indeed, on average, $DMLA$ is 88.89 times slower than $DLA$. However, it is worth observing that the number of iterations performed by $DLA$ is 2.03 times higher than that of $DMLA$. The worst performance in terms of computational cost of $DMLA$ is due to the average time per iteration. Indeed, the time per iteration obtained with $DLA$ is 93.87 times lower than that of $DMLA$.

Table 4 shows the performance of $DMLA$ and $DLA$ varying the density, that is, the ratio between the number of arcs and the number of nodes. Density does not seem to strongly affect the number of iterations executed by $DMLA$, that are almost the same for all three different values of density tested. In addition, the computational time of $DMLA$ increases, on average, by the 45 % and 11 % when the density value varies from 10 to 30 and from 30 to 50, respectively. In addition, the execution time is less than doubled when the density increases from 10 to 50. The results summarized in Table 4 underline a different situation for $DLA$, whose performance is more affected by the density than $DMLA$. Indeed, when the density increases from 10 to 30 and from 30 to 50 the execution time of $DLA$ increases more than 2 and 2.5 times, respectively. In addition, the computational time is multiplied by a factor of five when the density increases from 10 to 50.

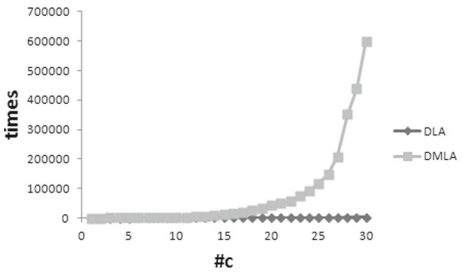



**Fig. 4** Average execution time in ms of $DMLA$ and $DLA$ as a function of the indicator #$c$ of the number of negative cost cycles

From Figure 4, it is possible to observe that the higher the number of negative cost cycles, the higher the difference, in terms of execution time, between $DLA$ and $DMLA$. More specifically, $DLA$ is 4.85, 21.04 and 135.75 times faster than $DMLA$ for the instances belonging to the first, the second and the third class, respectively.

The $DLA$ behaves better than $DMLA$ for two main reasons: 1) $DMLA$ is strongly affected by the dimension of the label; 2) $DLA$ takes advantage of the fact that it maintains the partial solutions at each iteration. This aspect justifies the lower computational cost per iterations.

### *3.3.2 Computational results on CVRP benchmark instances*

In this section, we evaluate the behaviour of the best performing algorithm, that is, $DLA$ by considering the $CVRP$ benchmark instances. An execution time limit of one hour for each instance has been imposed.

The results collected on the corresponding *ESPP* instances are summarized in Table 5. We report the name of the set under the first column, the column *min* $\lambda$ reports the minimum value of the Lagrangean multiplier such that at least one instance is solved, the last three columns show the numerical results obtained. In particular, for each set, we have considered average results for each value of the Lagrangean multiplier. Under column *min*, we report the minimum values related to the time (in seconds), number of iterations (iter), number of distinct cycles (cycles) detected during the execution of the $DLA$ and the percentage (%) of the instances solved, evaluated over all the average results, obtained for each value of the multiplier. Under column *max* the maximum values and under column *mdm* the medium values on all the instances associated with each set. It is worth observing that the first six sets, that is, $F$, $A-A$, $A-B$, $A-P$, $CE$, and $S$ contain instances associated with $CVRP$ test problems for which the optimal solution is known. The remaining four sets are composed of instances for which the related $CVRP$ test problems are not solved to optimality.

The average results reported in Table 5 (see row AVG) underline that on all the considered test problems, in the medium case, the 60 % of instances are solved within the time limit of one hour. In the worst case, only the 21 % of instances is solved, whereas 95 % of instances are solved in the best case (see row AVG, column max of Table 5). It is worth observing that for the first six sets, at least one instances with the Lagrangean multiplier equal to 0.1 is solved (see column *min* $\lambda$ of Table 5). For the remaining

**Table 5** Results obtained on the instances derived from *CVRP* benchmark test problems

| Set | Min λ | | Min | mdm | Max |
|---|---|---|---|---|---|
| $F$ | 0.1 | Time | 0.00 | 0.61 | 3.57 |
| | | Iter | 1.00 | 50.35 | 269.00 |
| | | Cycles | 0.00 | 2.73 | 13.00 |
| | | % Solved | 33 % | 83 % | 100 % |
| $A-A$ | 0.1 | Time | 0.04 | 74.88 | 828.83 |
| | | Iter | 11.81 | 458.45 | 2262.09 |
| | | Cycles | 2.04 | 13.29 | 33.50 |
| | | % Solved | 7 % | 79 % | 100 % |
| $A-B$ | 0.1 | Time | 0.00 | 146.92 | 1015.13 |
| | | Iter | 1.00 | 948.93 | 3227.60 |
| | | Cycles | 0.00 | 14.56 | 36.60 |
| | | % Solved | 4 % | 35 % | 96 % |
| $A-P$ | 0.1 | Time | 0.00 | 29.38 | 360.37 |
| | | Iter | 1.00 | 171.47 | 1273.20 |
| | | cycles | 0.00 | 4.28 | 15.83 |
| | | % Solved | 4 % | 73 % | 100 % |
| $CE$ | 0.1 | Time | 0.00 | 19.85 | 380.31 |
| | | Iter | 1.00 | 103.89 | 1028.00 |
| | | cycles | 0.00 | 2.64 | 8.80 |
| | | % solved | 38 % | 76 % | 100 % |
| $S$ | 0.1 | Time | 0.10 | 11.28 | 111.27 |
| | | Iter | 11.33 | 173.58 | 816.00 |
| | | Cycles | 0.33 | 3.15 | 17.00 |
| | | % Solved | 50 % | 78 % | 100 % |
| $B$ | 0.7 | Time | 0.03 | 168.22 | 941.88 |
| | | Iter | 4.12 | 997.60 | 3134.00 |
| | | Cycles | 0.95 | 18.77 | 33.00 |
| | | % Solved | 2 % | 35 % | 100 % |
| $CMT$ | 0.6 | Time | 0.19 | 305.52 | 906.53 |
| | | Iter | 61.25 | 1092.18 | 4975.75 |
| | | Cycles | 3.75 | 11.79 | 27.50 |
| | | % Solved | 14 % | 29 % | 57 % |
| $T$ | 0.6 | Time | 11.31 | 208.41 | 892.76 |
| | | Iter | 195.83 | 1248.68 | 2775.00 |
| | | Cycles | 11.67 | 25.18 | 51.60 |
| | | % Solved | 38 % | 49 % | 92 % |
| $GWKC$ | 0.3 | Time | 0.27 | 18.34 | 221.11 |
| | | Iter | 1.00 | 96.94 | 926.50 |
| | | Cycles | 0.00 | 4.73 | 17.67 |
| | | % Solved | 15 % | 64 % | 100 % |
| AVG | | Time | 1.19 | 98.34 | 566.18 |
| | | Iter | 28.93 | 534.20 | 2068.71 |
| | | cycles | 1.87 | 10.11 | 25.45 |
| | | % Solved | 21 % | 60 % | 95 % |

The time is given in seconds

sets, $DLA$ is able to solve at least one instance with the Lagrangean multiplier equal to 0.7, 0.6, 0.6, and 0.3 for set $B$, $CMT$, $T$, and $GWKC$, respectively.

In addition, even though the best performance is observed for the first six sets of instances, the results collected on the other sets can be considered satisfactory. Indeed, the average percentage of solved problem is of 17, 44 and 87 % in the worst, medium and best case, respectively.

## 4 Conclusions

In this paper we have investigated the shortest path problem in presence of negative cost cycles. This study is the first attempt to provide solution methods for the single-source all-destinations shortest path problem with negative cost cycles. The scientific literature provides strategies that are able to determine the presence of negative cost cycles and algorithms for solving the elementary shortest path problem on graph with negative cost cycles in conjunction with resource consumption constraints.

Two different strategies have been devised to solve to optimality the problem under investigation. The main idea behind the proposed solution approaches is to compute, for each node, the minimum number of paths such that the shortest paths from the source node to all others nodes are determined. In addition, the two methods dynamically increase the number of paths that have to be found for some node. The main difference among the proposed approaches is related to the way in which the number of paths is incremented. In the first method, the number of paths that have to be found is determined by considering a dummy node resource that keeps trace about the visiting at the node. This results in a constrained multi-objective shortest path in which the node associated with the dummy resource can be visited only once along the paths. The second proposed approach is based on the idea behind the $k$ shortest paths methods. In particular, the value of $k$ is different per node and it is incremented each time a further path that passes through such a node is needed in order to avoid a negative cost cycle. The theoretical complexity of the proposed solution approaches in the worst case is derived.

The proposed optimal strategies have been evaluated empirically on a large set of test problems. In particular, we have considered two groups of instances. The first one refers to fully random networks, the second group contains the instances derived from vehicle routing problem benchmarks. Referring to the first group of instances, we have considered instances with up to 500 nodes and 25000 arcs with 30 negative cost cycles at least present in the networks. The experiments underline the superiority of the innovative strategy based on the $k$ shortest paths problem. Indeed, it outperforms the dynamic programming based approach. In particular, the best performing algorithm is able to solve the instances based on fully random networks with 500 nodes, 25000 arcs and at least 30 negative cost cycles in about 5 seconds.

The proposed method based on the $k$ shortest paths problem has been tested on instances derived from the vehicle routing problem benchmarks. In particular, we have considered two classes of test problems: the first one refers to those that the literature solves to optimality, the second class contains the test problems for which only near optimal solutions are known. Starting from the vehicle routing problem

benchmarks we have derived several instances of the elementary shortest path problem with a different degree of difficulty. The computational results suggest that the best performing algorithm solves to optimality near to all the more difficult instances belonging to the first class within a reasonable amount of time. For the instances of the second class, the solution approach based on the $k$ shortest paths problem does not solve the more difficult instances. However, despite to the $NP$-hard complexity of the problem, the numerical results are satisfactory.

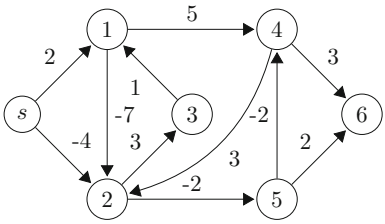
From the computational results, we can conclude that the solution approach based on the $k$ shortest paths problem is very efficient for solving the elementary shortest path problem on medium-large size fully random networks. In addition, the proposed method is able to solve in a reasonable amount of time instances derived from the vehicle routing problem benchmarks.

**Acknowledgments** The authors would like to thank Professor Francesco Scarcello for his valuable comments and helpful suggestions related to the complexity analysis of the proposed solution approaches. They also wish to thank the editor and the anonymous referees having contributed to improve the quality and readability of the paper, with their constructive suggestions and comments.

## 5 Appendix

In this appendix, we show how the proposed Algorithm 1 and 4 work by considering the instance of Figure 5.

**Fig. 5** Graph example




### 5.1 Algorithm 1

In Table 6, we report set $S$, the labels at the end of Algorithm 2, the last selected label and the detected cycle $C$. The labels reported in Table 6 have the following form: $< \pi(y) >$. The superscript reported next to the labels indicates that the related label is dominated.

At the first iteration of Algorithm 1 (see column $1^{st}$ iteration of Table 6), the label $< s, 2, 5, 4(-8) >$ is selected. When we try to extend the label to node 2 a $NCC$ is detected. Indeed, path $\pi_{s2} = \{s, 2, 5, 4\} \cup \{2\}$ has a cost equal to $-5$ that is less than the cost of path $\pi_{s2} = \{s, 2\}$ associated with label $< s, 2(-4) >$. The dummy resource is introduced to node 2. At the $2^{nd}$ iteration, the $NCC$ $\{1, 2, 3, 1\}$ is detected

**Table 6** Labels associated with each node at the end of Algorithm 2 at each iteration of Algorithm 1

| | $1^{st}$ iteration | $2^{nd}$ iteration | $3^{rd}$ iteration |
|---|---|---|---|
| $S$ | $\emptyset$ | $\{2\}$ | $\{2, 1\}$ |
| $s$ | $< s(0) >$ | $< s(0, 0) >$ | $< s(0, 0, 0) >$ |
| 1 | $< s, 1(2) >$ | $< s, 1(2, 0) >$ | $< s, 1(2, 0, 1) >$ |
| | | $< s, 2, 1(-1, 1) >$ | $< s, 2, 1(-1, 1, 1) >$ |
| 2 | $< s, 2(-4) >$ | $< s, 2(-4, 1) >^D$ | $< s, 2(-4, 1, 0) >$ |
| | | $< s, 1, 2(-5, 1) >$ | $< s, 1, 2(-5, 1, 1) >$ |
| 3 | $< s, 2, 3(-2) >$ | $< s, 2, 3(-2, 1) >^D$ | $< s, 2, 3(-2, 1, 0) >$ |
| | | $< s, 1, 2, 3(-3, 1) >$ | $< s, 1, 2, 3(-3, 1, 1) >$ |
| 4 | $< s, 2, 5, 4(-8) >$ | $< s, 2, 3, 1, 4(4, 1) >^D$ | $< s, 2, 3, 1, 4(4, 1, 1) >^D$ |
| | | $< s, 2, 5, 4(-8, 1) >^D$ | $< s, 2, 5, 4(-8, 1, 0) >$ |
| | | $< s, 1, 4(7, 0) >$ | $< s, 1, 4(7, 0, 1) >$ |
| | | $< s, 1, 2, 5, 4(-9, 1) >$ | $< s, 1, 2, 5, 4(-9, 1, 1) >$ |
| 5 | $< s, 2, 5(-6) >$ | $< s, 2, 5(-6, 1) >$ | $< s, 2, 5(-6, 1, 0) >$ |
| | | $< s, 1, 2, 5(-7, 1) >$ | $< s, 1, 2, 5(-7, 1, 1) >$ |
| 6 | $< s, 2, 5, 6(-4) >$ | $< s, 2, 5, 6(-4, 1) >^D$ | $< s, 2, 5, 6(-4, 1, 0) >^D$ |
| | | $< s, 2, 5, 4, 6(-5, 1) >^D$ | $< s, 2, 5, 4, 6(-5, 1, 0) >$ |
| | | $< s, 1, 2, 5, 4, 6(-6, 1) >$ | $< s, 1, 2, 5, 4, 6(-6, 1, 1) >$ |
| | | | $< s, 1, 2, 5, 6(-5, 1, 1) >^D$ |
| | | | $< s, 1, 4, 6(10, 0, 1) >$ |
| Last extracted label | $< s, 2, 5, 4(-8) >$ | $< s, 1, 2, 3(-3, 1) >$ | $< s, 1, 4(7, 0, 1) >$ |
| $C$ | $\{2, 5, 4, 2\}$ | $\{1, 2, 3, 1\}$ | $\emptyset$ |

and node 1 is inserted in the set $S$. The introduction of the resource at node 2 and 1 avoids the generation of *NCCs* and the optimal solution is found at the $3^{rd}$ iteration.

### 5.2 Algorithm 4

In Table 7, we report the paths and the related cost ($< \pi_{si}^k(y_i[k]) >$) at the end of each iteration of Algorithm 4 when solving the $ESPP$ on the network of Figure 5. At the $1^{st}$ iteration, node 3 is selected. Since all the conditions of line 9 are verified, Algorithm 3 terminates and the cycle $\{1, 2, 3, 1\}$ is returned. The value of $K_i, i = 1, 2, 3$ is incremented of one and Algorithm 3 is executed again. Node 4 is selected (see $2^{nd}$ iteration). When path $\pi_{s4}^1 = \{s, 1, 2, 5, 4\}$ is extended to node 2, conditions of line 9 are satisfied, thus the $NCC$ $\{2, 5, 4, 2\}$ is detected and Algorithm 3 is stopped. The number of paths that need to be found for nodes 2, 5 and 4 is increased. After other three iterations, see $3^{rd}$, $4^{th}$ and $5^{th}$ iteration of Table 7, condition of line 29 is verified. Since $C = \emptyset$, Algorithm 4 terminates and $y_i[1], \forall i \in \mathcal{N}$, is the optimal cost.

**Table 7** Paths and costs that are found for each node at the end of Algorithm 3 in each iteration of Algorithm 4

| | $1^{st}$ iteration | | $2^{nd}$ iteration | | $3^{rd}$ iteration | | $4^{th}$ iteration | | $5^{th}$ iteration | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $K_i$ | $y_i$ | $K_i$ | $y_i$ | $K_i$ | $y_i$ | $K_i$ | $y_i$ | $K_i$ | $y_i$ |
| $s$ | 1 | $< s(0) >$ | 1 | $< s(0) >$ | 1 | $< s(0) >$ | 1 | $< s(0) >$ | 1 | $< s(0) >$ |
| 1 | 1 | $< s, 1(2) >$ | 2 | $< s, 2, 3, 1(-1) >$<br>$< s, 1(2) >$ | 2 | $< s, 2, 3, 1(-1) >$<br>$< s, 1(2) >$ | 2 | $< s, 2, 3, 1(-1) >$<br>$< s, 1(2) >$ | 2 | $< s, 2, 3, 1(-1) >$<br>$< s, 1(2) >$ |
| 2 | 1 | $< s, 1, 2(-5) >$ | 2 | $< s, 1, 2(-5) >$<br>$< s, 2(-4) >$ | 3 | $< s, 1, 2(-5) >$<br>$< s, 2(-4) >$<br>$< s, 1, 4, 2(10) >$ | 4 | $< s, 1, 2(-5) >$<br>$< s, 2(-4) >$<br>$< s, 1, 4, 2(10) >$<br>$+\infty$ | 5 | $< s, 1, 2(-5) >$<br>$< s, 2(-4) >$<br>$< s, 1, 4, 2(10) >$<br>$+\infty$<br>$+\infty$ |
| 3 | 1 | $< s, 1, 2, 3(-3) >$ | 2 | $< s, 1, 2, 3(-3) >$<br>$< s, 2, 3(-2) >$ | 2 | $< s, 1, 2, 3(-3) >$<br>$< s, 2, 3(-2) >$ | 2 | $< s, 1, 2, 3(-3) >$<br>$< s, 2, 3(-2) >$ | 2 | $< s, 1, 2, 3(-3) >$<br>$< s, 2, 3(-2) >$ |
| 4 | 1 | $< s, 1, 4(7) >$ | 1 | $< s, 1, 2, 5, 4(-9) >$ | 2 | $< s, 1, 2, 5, 4(-9) >$<br>$< s, 2, 5, 4(-8) >$ | 3 | $< s, 1, 2, 5, 4(-9) >$<br>$< s, 2, 5, 4(-8) >$<br>$< s, 2, 3, 1, 4(4) >$ | 4 | $< s, 1, 2, 5, 4(-9) >$<br>$< s, 2, 5, 4(-8) >$<br>$< s, 2, 3, 1, 4(4) >$<br>$< s, 1, 4(7) >$ |
| 5 | 1 | $< s, 1, 2, 5(-7) >$ | 1 | $< s, 1, 2, 5(-7) >$ | 2 | $< s, 1, 2, 5(-7) >$<br>$< s, 2, 5(-6) >$ | 3 | $< s, 1, 2, 5(-7) >$<br>$< s, 2, 5(-6) >$<br>$+\infty$ | 4 | $< s, 1, 2, 5(-7) >$<br>$< s, 2, 5(-6) >$<br>$< s, 1, 4, 2, 5(8) >$<br>$+\infty$ |
| 6 | 1 | $+\infty$ | 1 | $< s, 1, 2, 5, 4, 6(-6) >$ | 1 | $< s, 1, 2, 5, 4, 6(-6) >$ | 1 | $< s, 1, 2, 5, 4, 6(-6) >$ | 1 | $< s, 1, 2, 5, 4, 6(-6) >$ |
| Last extracted node | | 3 | | 4 | | 4 | | 4 | | 5 |
| $C$ | | $\{1, 2, 3, 1\}$ | | $\{2, 5, 4, 2\}$ | | $\{2, 5, 4, 2\}$ | | $\{2, 5, 4, 2\}$ | | $\emptyset$ |

## References


1. Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.E.: Faster algorithms for the shortest path problem. Technical Report CS-TR-154-88. Departement of Computer Science, Princeton University (1988)
2. Ali, Al, Kennington, J.L.: The asymmetric m-traveling salesmen problem: a duality based branch-and-bound algorithm. Discret. Appl. Math. **13**, 259–276 (1986)
3. Allender, E.W.: On the number of cycles possible in digraphs with large girth. Discret. Appl. Math. **10**, 211–225 (1985)
4. Augerat, P., Belenguer, J.M., Benavent, E., Corberan, A., Naddef, D., Rinaldi, G.: Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M. Universite Joseph Fourier, Grenoble, France (1995)
5. Barndorff-Nielsen, O., Sobel, M.: On the distribution of the number of admissible points in a vector random sample. Theory Propability Appl. **11**(2), 283–305 (1966)
6. Beasley, J.E., Christofides, N.: An algorithm for the resource constrained shortest path problem. Networks **19**(4), 379–394 (1989)
7. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega **34**(3), 209–219 (2006)
8. Bellman, R.E.: On a routing problem. Q. Appl. Math. **16**(1), 87–90 (1958)
9. Bentley, J.L., Kung, H.T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. J. ACM **25**(4), 536–543 (1978)
10. Boland, N., Dethridge, J., Dumitrescu, I.: Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Oper. Res. Lett. **34**(1), 58–68 (2006)
11. Chen, S., Ryan, D.R.: A comparison of three algoritms for finding fundamental cycles in a directed graph. Networks **11**, 1–12 (1981)
12. Cherkassky, B.V., Goldberg, A.V., Radzik, T.: Shortest paths algorithms: Theory and experimental evaluation. Math. Program. **73**(2), 129–174 (1996)
13. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) Combinatorial Optimization, chapter 11. John Wiley, Chichester (1979)
14. Christofides, N., Eilon, S.: An algorithm for the vehicle dispatching problems. Oper. Res. Q. **20**(3), 309–318 (1969)
15. Dantzig, G.B.: On the shortest route through a network. Management Science, pp. 187–190, (1960)
16. Daskin, M.S.: Network and Discrete Location. Wiley, New York (1995)
17. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. **40**(2), 342–354 (1992)
18. Di Puglia Pugliese, L., Guerriero, F.: Solution approaches for the elementary shortest path problem. Technical Report 5/10, University of Calabria, LOGICA, (2010). Downloadable from http://uweb.deis.unical.it/dipugliapugliese/
19. Di Puglia Pugliese, L., Guerriero, F.: A computational study of solution approaches for the resource constrained elementary shortest path problem. Ann. Oper. Res. **201**(1), 131–157 (2012)
20. Di Puglia Pugliese, L., Guerriero, F.: A reference point approach for the resource constrained shortest path problems. Transp. Sci. **47**(2), 247–265 (2013)
21. Di Puglia Pugliese, L., Guerriero, F.: Shortest path problem with forbidden paths: the elementary version. Eur. J. Oper. Res. **227**(2), 254–267 (2013)
22. Di Puglia Pugliese, L., Guerriero, F.: A survey of resource constrained shortest path problem: exact solution appoaches. Networks **62**(3), 183–200 (2013)
23. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
24. Drexl, M., Irnich, S.: Solving elementary shortest-path problems as mixed-integer programs. OR Spectrum **36**(2), 281–296 (2014)
25. Dreyfus, S.E.: An appraisal of some shortest path algorithms. Oper. Res. **17**, 395–412 (1969)
26. Feillet, D., Dejax, P., Gendreau, M., Gueguen, C.: An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks **44**(3), 216–229 (2004)
27. Fisher, M.L.: Optimal solution of vehicle routing problems using minimum k-trees. Oper. Res. **42**, 626–642 (1994)
28. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
29. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)

30. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
31. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithm for network problems. SIAM J. Comput. **18**(5), 1013–1036 (1989)
32. Gallo, G., Pallottino, S.: Shortest path algorithms. Ann. Oper. Res. **13**(1), 1–79 (1988)
33. Gavish, B., Srikanth, K.: An optimal solution method for large-scale multiple traveling salesman problems. Oper. Res. **34**(5), 698–717 (1986)
34. Goldberg, A.V.: Scaling algorithms for the shortest path problem. In: $4^{th}$ ACM-SIAM Symposium on Discrete Algorithms, pp. 222–231 (1993)
35. Golden, B.L., Wasil, E.A., Kelly, J.P., Chao, I.-M.: Metaheuristics in vehicle routing. In: Crainic, T.G., Laporte, G. (eds.) Fleet Management and Logistics, pp. 33–56. Kluwer, Boston (1998)
36. Guerriero, F., Musmanno, R., Lacagnina, V., Pecorella, A.: A class of label-correcting methods for the $k$ shortest paths problem. Oper. Res. **49**(3), 423–429 (2001)
37. Guerriero, F., Di Puglia Pugliese, L.: Multi-dimensional labelling approaches to solve the linear fractional elementary shortest path problem with time windows. Optim. Methods Softw. **26**(2), 295–340 (2011)
38. Ibrahim, M.S., Maculan, N., Minoux, M.: A strong flow-based formulation for the shortest path problem in digraph with negative cycles. Int. Trans. Oper. Res. **16**, 361–369 (2009)
39. Irnich, S.: Resource extension functions: properties, inversion and generalization to segments. OR Spectrum **30**(1), 113–148 (2008)
40. Klingman, D., Napier, A., Stutz, J.: Netgen: A program for generating large-scale (un)capacitated assignment, transportation, and minimum cost flow network problems. Manag. Sci. **20** (1974)
41. Kohl, N.: Exact methods for time constrained routing and related scheduling problems. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby (1995)
42. Magnanti, T.L., Wong, R.T.: Network design and transportation planning: models and algorithms. Transp. Sci. **18**, 1–55 (1984)
43. Martins, E.Q., Pascoal, M.M., Santos, J.L.: Deviation algorithms for ranking shortest paths. Int. J. Found. Comput. Sci. **10**(3), 247–261 (1999)
44. Moore, E.F.: The shortest path through a maze. In: Interantional Symposium on the Theory of Switching, pp. 285–292. Harvard University Press (1959)
45. Orloff, C.S.: Routing a fleet of m vehicles to/from a central facility. Networks **4**, 147–162 (1974)
46. Rao, M.R.: A note on the multiple traveling salesman problem. Oper. Res. **28**(3), 628–632 (1980)
47. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks **5**, 237–252 (1975)
48. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks **51**(3), 155–170 (2008)
49. Sedgewick, R., Wayne, K.: Shortest paths. http://www.cs.princeton.edu/rs/AlgsDS07/, (2007)
50. Solomon, M.M.: Vehicle routing and scheduling with time window constraints: models and algorithms. PhD thesis, Department of Decision Science, University of Pennsylvania (1983)
51. Subramani, K.: A zero-space algorithm for negative cost cycle detection in networks. J. Discret. Algorithms **5**, 408–421 (2007)
52. Taillard, D.: Parallel iterative search methods for vehicle routing problems. Networks **23**, 661–673 (1993)
53. Van Bredam, A.: An Analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related. Customer-related, and time-related constraints. PhD thesis, University of Antwerp (1994)
54. Yamada, T., Kinoshita, H.: Finding all the nagative cycles in a directed graph. Discret. Appl. Math. **118**, 279–291 (2002)
55. Yen, J.Y.: Finding the k-shortest loopless paths in a network. Manage. Sci. **17**, 711–715 (1971)