Journal of Discrete Algorithms 24 (2014) 12–25

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Fast shortest-paths algorithms in the presence of few destinations of negative-weight arcs

Domenico Cantone, Simone Faro *

University of Catania, Dept. of Mathematics and Computer Science, Viale Andrea Doria 6, I-95125 Catania, Italy

ARTICLE INFO

Article history:
Available online 10 April 2013

Keywords:
Shortest path
Negative edges
Computational complexity

ABSTRACT

In this paper we present hybrid algorithms for the *single-source shortest-paths* (SSSP) and for the *all-pairs shortest-paths* (APSP) problems, which are asymptotically fast when run on graphs with *few* destinations of negative-weight arcs. Plainly, the case of graphs with few sources of negative-weight arcs can be handled as well, using reverse graphs. With a directed graph with n nodes and m arcs, our algorithm for the SSSP problem has an $\mathcal{O}(\ell(m + n\log n + \ell^2))$-time complexity, where ℓ is the number of destinations of negative-weight arcs in the graph. In the case of the APSP problem, we propose an $\mathcal{O}(nm^* + n^2\log n + \ell n^2)$ algorithm, where m^* is the number of arcs participating in shortest paths. Notice that m^* is likely to be small in practice, since $m^* = \mathcal{O}(n\log n)$ with high probability for several probability distributions on arc weights.

1. Introduction

The *shortest-paths problem* on weighted directed graphs is one of the basic network optimization problems. Its importance is mainly due to its applications in various areas, such as communication and transportation. Given a source node s in a weighted directed graph G, with n nodes and m arcs, the *single-source shortest-paths problem* from s (SSSP, for short) is the problem of finding the minimum weight paths from s to all other nodes of G. The *all-pairs shortest-paths problem* (APSP, for short) consists in finding the minimum weight paths for each pair of nodes in G.

In this paper we propose two algorithms, one for the SSSP problem (based on the algorithms by Dijkstra and by Bellmand–Ford–Moore) and one for the APSP problem (based on the algorithms by Dijkstra and by Floyd, and on the Hidden-Paths algorithm), which are asymptotically fast when run on graphs with *few* destinations of negative-weight arcs. Plainly, by reversing the input graph, our algorithms achieve the same time complexities also in the case of graphs with few sources of negative-weight arcs.

Negative arc weights arise in several applications, such as in arc- and vertex-diverse path computation, in network survivability, in job scheduling with deadlines, in arbitrage problems in foreign exchange markets, and so on.

If we denote by ℓ the number of destinations of negative-weight arcs in a directed graph with n nodes and m arcs, then our proposed algorithm for the SSSP problem has an $\mathcal{O}(\ell(m + n\log n + \ell^2))$-time complexity. It easily turns out that when $\ell = o(\sqrt[3]{mn})$, our algorithm is asymptotically faster than the Bellman–Ford–Moore algorithm [1,12,30]. Additionally, for $\ell = o(h)$, where h is the minimum between n and the number of the negative-weight arcs contained in the graph, our algorithm is asymptotically faster than Yap's algorithm [40].

Concerning the APSP problem, we present an $\mathcal{O}(nm^* + n^2\log n + \ell n^2)$-time algorithm, where m^* is the number of arcs participating in shortest paths. It turns out immediately that when $\ell = o(n)$ and $m^* = o(n^2)$, our algorithm is asymptotically

* Corresponding author.
E-mail addresses: cantone@dmi.unict.it (D. Cantone), faro@dmi.unict.it (S. Faro).

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2013.03.005

faster than Floyd's algorithm [11]. Moreover, when $\ell = \mathcal{O}(m^*/n + \log n)$, our algorithm achieves the same $\mathcal{O}(nm^* + n^2 \log n)$-time complexity of the Hidden-Paths algorithm [24], which however works only with nonnegative weighted graphs. Finally, when $m^* = o(\frac{n^2}{\log^2 n})$ and $\ell = o(\frac{n}{\log^2 n})$, our algorithm achieves an $o(\frac{n^3}{\log^2 n})$-time complexity, outperforming the best known subcubic algorithms for the APSP problem. In [24] it is argued that m^* is likely to be small in practice, since $m^* = \mathcal{O}(n \log n)$ with high probability for several probability distributions on arc weights.

The paper is organized as follows. After introducing in Section 2 the relevant notations and terminology used in the paper, in Section 3 we briefly review the relevant literature on shortest-paths algorithms with particular emphasis on the algorithms which we use as building blocks in our solutions. Then, in Sections 4 and 5 we present our algorithms for the SSSP and the APSP problems, respectively, evaluating their time complexity and proving also their correctness. Finally, we draw our conclusions in Section 6.

2. Preliminaries

We begin by reviewing the relevant notations and terminology. A *directed graph* is represented as a pair $G = (V, E)$, where V is a finite set of nodes (or vertices) and $E \subseteq V \times V$ is a set of arcs (or edges) such that E does not contain any self-loop of the form (v, v). In this context, we usually put $n = |V|$ and $m = |E|$. A *path* in $G = (V, E)$ from u to v is any finite sequence $(v_0, v_1, \ldots, v_k)$ of nodes such that $v_0 = u$, $v_k = v$, and (v_i, v_{i+1}) is an arc of G, for $i = 0, 1, \ldots, k-1$. By $(u \rightsquigarrow v)$ we denote an unspecified path from u to v (possibly the edge (u, v)) and by $(v_0, v_1, \ldots, v_k \rightsquigarrow w)$ we denote the concatenation of the paths $(v_0, v_1, \ldots, v_k)$ and $(v_k \rightsquigarrow w)$. Likewise, by $(v \rightsquigarrow w \rightsquigarrow u)$ we denote the concatenation of the paths $(v \rightsquigarrow w)$ and $(w \rightsquigarrow u)$. The *length* of a path is the number of arcs participating in it (with possible repetitions); thus, the length of the path $(v_0, v_1, \ldots, v_k)$ is k. A node v_j in a path $(v_0, v_1, \ldots, v_k)$ is *internal* if $0 < j < k$. A *weight function* ω on $G = (V, E)$ is any real function $\omega : E \to \mathbb{R}$. A weight function ω can be extended over paths by putting $\omega(v_0, v_1, \ldots, v_k) = \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$. A *shortest path* from u to v (also called an *optimal path*) is a path in G whose weight is minimum among all paths from u to v. Provided that v is reachable from u and that no path from u to v goes through a negative-weight cycle, a shortest path from u to v always exists; in such a case we denote by $\delta(u, v)$ the weight of a shortest path from u to v. If v is not reachable from u, we set $\delta(u, v) = +\infty$. If there is a path from u to v through a negative-weight cycle, we put $\delta(u, v) = -\infty$. The function $\delta : V \times V \to \mathbb{R} \cup \{+\infty, -\infty\}$ is called the *distance function* on (G, ω). A path is *purely nonnegative* if all its arcs have nonnegative weight. An arc is said to be optimal if it participates in some shortest path. Observe that an arc (u, v) is optimal if and only if $\omega(u, v) = \delta(u, v)$. We denote by $m^*_{G,\omega}$ (or simply m^*, when the weighted graph (G, ω) is understood from the context) the number of optimal arcs in (G, ω). Given a weighted graph (G, ω), with $G = (V, E)$, the *reverse* of (G, ω) is the weighted graph (G^R, ω^R), where $G^R = (V, E^R)$, with $E^R = \{(v, u) \mid (u, v) \in E\}$, and $\omega^R : E^R \to \mathbb{R}$, with $\omega^R(v, u) = \omega(u, v)$, for every $(v, u) \in E^R$.

In Sections 4 and 5, we will make implicit use of the following elementary facts.

Lemma 1. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$ and let δ be the distance function on (G, ω). Let $G' = (V', E')$, with $V' \subseteq V$ and let $\omega' : E' \to \mathbb{R}$ be a weight function on G' such that*

- *$\omega'(u', v') \geqslant \delta(u', v')$, for $(u', v') \in E'$.*

Then, for each path π' in G' from u' to v', with $u', v' \in V'$, we have $\omega'(\pi') \geqslant \delta(u', v')$. □

Lemma 2. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$ and let δ be the distance function on (G, ω). Let $G' = (V, E')$, with $E' \supseteq E$, be an extension of G such that $\delta(u', v') < +\infty$, for $(u', v') \in E'$. Also, let $\omega' : E' \to \mathbb{R}$ be a weight function on G' such that*

- *$\omega'(u', v') \geqslant \delta(u', v')$, for $(u', v') \in E'$;*
- *$\omega'(u, v) \leqslant \omega(u, v)$, for $(u, v) \in E$.*

Then the distance function δ' on (G', ω') coincides with δ. □

The SSSP and APSP problems are defined precisely as follows.

Definition 3 *(Single-source shortest-paths problem).* Given a source node s in a weighted graph (G, ω), the *single-source shortest-paths problem* from s is the problem of computing the shortest-path distance $\delta(s, v)$ on (G, ω), for all $v \in V$.

Definition 4 *(All-pairs shortest-paths problem).* Given a weighted graph (G, ω), the *all-pairs shortest-paths problem* is the problem of computing the shortest-path distance $\delta(u, v)$ on (G, ω), for all $u, v \in V$.

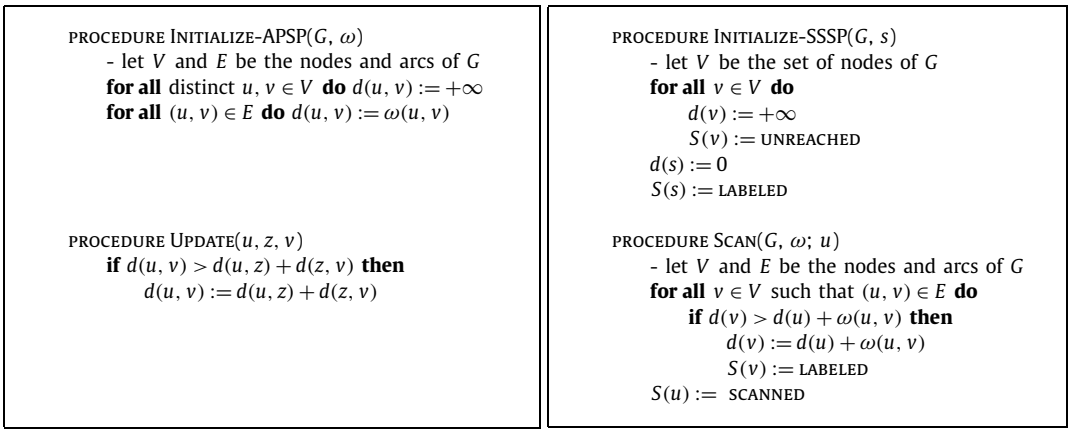

Fig. 1. (On the left) procedures INITIALIZE-APSP and UPDATE for APSP algorithms, and (on the right) procedures INITIALIZE-SSSP and SCAN for SSSP algorithms.

We will also be interested in the *single-destination shortest-paths problem* to a destination node t (SDSP, for short), in which one has to find the shortest paths from all nodes in a weighted directed graph to t. Plainly, the SDSP problem can be reduced in linear time to the SSSP problem, reversing the graph and weight.

Throughout the paper we will assume that graphs satisfy the relation $n = \mathcal{O}(m)$. This is certainly the case, for instance, when all nodes of a graph are reachable from a given source node, which is very common. For ease of presentation, we will also assume that the graphs do not contain negative cycles. However, we observe that our algorithms detect correctly the presence of negative-weight cycles (reachable from the source node, in the case of the SSSP problem). Finally, we are interested only in algorithms working on *real*-weighted graphs, where reals are manipulated only by *comparison* and *addition* operations.

Most of the algorithms working on the *comparison–addition* model are based on the general *labeling* method. Such method maintains a *shortest-path estimate* function $d : V \times V \to \mathbb{R} \cup \{+\infty\}$, where $d(u, v)$ represents the shortest-path estimate from the source u to the target v. When the source node s is understood, for simplicity we write $d(v)$ to denote the estimate function $d(s, v)$.

In the case of the APSP problem, initially one sets $d(u, v) := \omega(u, v)$ if $(u, v) \in E$ and $d(u, v) := +\infty$ otherwise (see procedure INITIALIZE-APSP(G) in Fig. 1). Subsequently, the shortest-path estimate d is updated, within calls to the procedure UPDATE(u, z, v), by assignments of the form $d(u, v) := d(u, z) + d(z, v)$, provided that $d(u, v) > d(u, z) + d(z, v)$ holds (see UPDATE(u, z, v) in Fig. 1). It turns out that the property $d(u, v) \geqslant \delta(u, v)$ is maintained as an invariant, for each ordered pair $u, v \in V$.

Concerning the SSSP problem from a source node s, the shortest-path estimate function d is initialized by putting $d(s) := 0$ and $d(v) := +\infty$ for all nodes $v \neq s$. Then the estimate function d is updated only by assignments of the form $d(v) := d(u) + \omega(u, v)$, provided that $d(v) > d(u) + \omega(u, v)$ holds. It turns out that $d(v) \geqslant \delta(s, v)$ is maintained as an invariant, for $v \in V$. Procedure SCAN(u) performs such updates for all arcs $(u, v) \in E$ (see Fig. 1).

3. Previous results

In the case of the SSSP problem, the most general solution is given by the Bellman–Ford–Moore algorithm [1,12,30], which is characterized by an $\mathcal{O}(mn)$-time complexity. However, several algorithms have been proposed over the years to efficiently handle restricted families of directed weighted graphs. For instance, the case of nonnegative-weight functions is solved efficiently by the celebrated Dijkstra's algorithm [8], which admits an $\mathcal{O}(m + n \log n)$-time implementation based on Fibonacci heaps [14].

Another particular case occurs when the graph is nearly acyclic and negative-weight arcs are allowed only outside of any cycle. In this case, the Two-Levels-Greedy algorithm proposed in [2], which is a natural combination of Dijkstra's algorithm with the linear algorithm for acyclic graphs, solves the SSSP problem in $\mathcal{O}(m + n \log k)$-time complexity, where k is the maximum cardinality of any strongly connected component in the graph.

For planar graphs, there has been a series of results yielding progressively better bounds. The first algorithm that exploited planarity was due to Lipton, Rose, and Tarjan [27], who gave an $\mathcal{O}(n^{3/2})$ algorithm. Klein, Mozes and Weimann [26] proposed an $\mathcal{O}(n^{4/3} \log^{2/3} D)$ algorithm, where D is the sum of the absolute values of the weights. Subsequently, Fakcharoenphol and Rao [10] gave an algorithm requiring $\mathcal{O}(n \log^3 n)$ time and $\mathcal{O}(n \log n)$ space. Finally, Klein et al. presented in [25] an algorithm for planar graphs with negative weights requiring linear space and $\mathcal{O}(n \log^2 n)$ time.

A restriction, particularly relevant to the present paper, concerns the number of negative-weight arcs contained in the graph. In [40], C.K. Yap describes a hybrid algorithm for the shortest-path problem between *two* nodes in a directed graph in the presence of few negative-weight arcs, with an $\mathcal{O}(h(m + n \log n + h^2))$ running time, where h is the minimum between n and the number of the negative-weight arcs contained in the graph.

```
DIJKSTRA(G, ω; s)
  - let V be the set of nodes of G
  INITIALIZE-SSSP(G, s)
  Q := Make_Queue(V, d)
  while Q ≠ ∅ do
    v := Extract_Min(Q)
    SCAN(G, ω; v)

REVERSE-DIJKSTRA(G, ω; t)
  DIJKSTRA(G^R, ω^R; t)
```

```
GENERALIZED-DIJKSTRA(G, ω; s)
  - let V and E be the nodes and arcs of G
  INITIALIZE-SSSP(G, s)
  Q := Make_Queue({s}, d)
  while Q ≠ ∅ do
    v := Extract_Min(Q)
    SCAN(G, ω; v)
    for all u ∈ V : (v, u) ∈ E do
      if S(u) = LABELED and u ∉ Q then
        Insert(u, Q)
```

Fig. 2. Dijkstra's algorithm and its reverse and generalized versions.

In the case of the APSP problem, the general case is classically solved by the dynamic programming algorithm proposed by Floyd [11] in 1962, which has a $\Theta(n^3)$ running time. The first subcubic algorithm, characterized by an $\mathcal{O}(n^3 \log^{1/3} \log n / \log^{1/3} n)$-time complexity, was proposed by Fredman [13]. Several other subcubic solutions have been presented over the years [4,5,9,17,18,36–38,41]. Among the more recent results, the $\mathcal{O}(n^3/\log n)$-time algorithm of Chan [4] deserves a special mention since it is based on a simple geometric approach and does not require explicit table lookups or word tricks. Moreover it must be noticed that the algorithm of Han [18] was the first to break the $\mathcal{O}(n^3/\log n)$ barrier by exploiting sophisticated word-packing tricks. The best result known to date is due to Han and Takaoka [19], which achieves an $\mathcal{O}(n^3 \log\log n/\log^2 n)$-time complexity. The latter result is very close to the $\mathcal{O}(n^3/\log^2 n)$ running time which is currently believed to be the natural limit of combinatorial approaches to the APSP problem.

In the restricted case in which the graph is sparse, the APSP can be solved more efficiently by Johnson's algorithm [23] which, after a preliminary reweight of the graph based on the shortest-path weights computed by a run of the Bellman–Ford–Moore algorithm from an external source, applies Dijkstra's algorithm from each node of the graph. If priority queues are implemented with Fibonacci heaps, Johnson's algorithm has an $\mathcal{O}(mn + n^2 \log n)$-time complexity and turns out to be asymptotically faster than Floyd's algorithm when $m = o(n^2)$. Recently the complexity of Johnson's algorithm has been lowered to $\mathcal{O}(mn + n^2 \log\log n)$ and $\mathcal{O}(mn + n^2\alpha(m, n))$ for directed and undirected graphs, respectively, by Pettie [34] and Pettie and Ramanchandran [35], by applying Thorup's hierarchy-based approach [39] to real-weighted graphs.

For the case of nonnegative weights, we mention the Hidden-Paths algorithm presented in [24], and developed independently in [21,29], which solves the APSP problem in $\mathcal{O}(nm^* + n^2 \log n)$ time, provided that one uses Fibonacci heaps to implement a priority queue, where m^* is the number of optimal arcs in the graph (i.e., arcs participating in shortest paths). The Hidden-Paths algorithm operates by running Dijkstra's SSSP algorithm in parallel from all nodes of the graph, using information gained at each node to reduce the work done at other nodes. When the input graph is the complete graph with arc weights chosen independently from any of a large class of probability distributions, it turns out that $m^* = \mathcal{O}(n \log n)$ with high probability (cf. [15,31]). Thus, under such assumptions, the Hidden-Paths algorithm achieves an $\mathcal{O}(n^2 \log n)$-time complexity on average (with high probability).

For the sake of completeness, we will briefly review below the shortest-paths algorithms which are used as building blocks in the design of our algorithms, to be presented in Sections 4 and 5. These are the Bellman–Ford algorithm, Dijkstra's algorithm, Floyd's algorithm, and the Hidden-Paths algorithm.

3.1. The Bellman–Ford algorithm

In the general case, namely when the underlying graph is not required to satisfy any particular topological restriction and the weight function is not restricted in any way, the best known complexity bound for the SSSP problem is reached by the $\mathcal{O}(mn)$-time algorithm due to Bellman [1], Ford [12], and Moore [30]. The algorithm maintains the set of nodes Q into a FIFO queue, initialized with the source node with null estimate distance, and stops when the queue Q is empty. Thus, the next node to be scanned is removed from the head of Q, whereas a node whose shortest-path estimate decreases is added to the tail of Q. It is to be noticed, though, that other variants [6,16,32,33] of the Bellman–Ford–Moore algorithm show a better practical behavior than the original algorithm.

3.2. Dijkstra's algorithm

The well-celebrated Dijkstra's algorithm [8] solves the SSSP problem for directed graphs with nonnegative-weight arcs only (see Fig. 2). It maintains all nodes in a min-priority queue, according to their shortest-path estimates. Then, at each iteration, it extracts the node u from the queue with the minimum shortest-path estimate $d(u)$, following a greedy strategy, and it executes procedure SCAN(u) in Fig. 1. At termination, the shortest-path estimate function $d(s, v)$ computed by Dijkstra's algorithm coincides with the shortest-path distance function $\delta(s, v)$, with v ranging in V.

The complexity of Dijkstra's algorithm depends on the implementation used for its service priority queue. A first naive implementation given in [8], in which the priority queue is maintained as a simple list, has an $\mathcal{O}(n^2)$-time complexity.

```
HIDDEN-PATHS(G, ω)
  - let V and E be the nodes and arcs of G
  INITIALIZE-APSP(G, ω)
  Q := Make_Queue({(u, v) | u, v ∈ V, u ≠ v}, d)
  while Q ≠ ∅ do
    (u ⇝ v) := Extract_Min(Q)
    Opt := Opt ∪ {(u ⇝ v)}
    for (z, u) ∈ Opt ∩ E do
      UPDATE(z, u, v)
    if (u ⇝ v) is an arc then
      for (v ⇝ z) ∈ Opt do
        UPDATE(u, v, z)
```

Fig. 3. The Hidden-Paths algorithm.

By implementing the priority queue by k-ary heaps [7] (for fixed k), one obtains an $\mathcal{O}(m \log n)$-time algorithm. Finally, we mention that using Fibonacci heaps [14] one gets an $\mathcal{O}(m + n \log n)$-time implementation.

It can be shown that in the case of nonnegative weighted graphs, every node u, when selected and removed from the queue by Dijkstra's algorithm, satisfies $d(u) = \delta(s, u)$, so that it does not need to be scanned anymore.[1] However, if negative weights are allowed, this is no longer true. Such a case can be handled by (re)inserting in the queue, if needed, any node which becomes LABELED, even if it has already been scanned. As a node may re-enter the queue several times, it turns out that the resulting GENERALIZED-DIJKSTRA algorithm, reported in Fig. 2, has an exponential worst-case running time, as proved in [22].

Given a weighted graph (G, ω) and a destination node t of G, the REVERSE-DIJKSTRA algorithm performs just a call to the standard Dijkstra's algorithm on the reverse weighted graph (G^R, ω^R) from the node t (see Fig. 2). Plainly, if the weight function ω is nonnegative, then the reverse Dijkstra's algorithm computes correctly the shortest paths to the given destination t.

3.3. Floyd's algorithm

Let $G = (V, E)$ be a directed graph, with $V = \{v_1, v_2, \ldots, v_n\}$, and let $w : E \to \mathbb{R}$ be a real-valued weight function. For $k = 1, 2, \ldots, n$, let $V_k = \{v_1, v_2, \ldots, v_k\}$. Also, put $V_0 = \emptyset$. Using a dynamic programming approach, Floyd's algorithm [11] computes a sequence of real functions $d^0, d^1, \ldots, d^n$ over $V \times V$, where, at least in the case in which G contains no negative-weight cycle, $d^k(u, v)$ is the shortest-path distance from vertex u to vertex v restricted to paths whose internal nodes are in V_k, if any such path exist, otherwise $d^k(u, v) = +\infty$, for $0 \leqslant k \leqslant n$ and $u, v \in V$. Observe that, for each pair of nodes $u, v \in V$, we have

$$d^0(u, v) = \begin{cases} w(u, v) & \text{if } (u, v) \in E \\ +\infty & \text{otherwise} \end{cases}$$

and $d^k(u, v) = \delta(u, v)$, for any $k \geqslant n$ (in the case in which G contains no negative-weight cycle). In particular at iteration i of Floyd's algorithm, the estimate function d^i is computed by executing procedure UPDATE(u, v_{i+1}, v) in Fig. 1, for all $u, v \in V$, following the dynamic programming approach. The complexity of Floyd's algorithm is $\mathcal{O}(n^3)$.

3.4. The Hidden-Paths algorithm

The Hidden-Paths algorithm [24] solves the APSP problem in the restricted case of nonnegative weighted graphs by running Dijkstra's algorithm in parallel from all nodes of the graph, in such a way that the partial results of any single-source thread can be used to reduce the work done by the other threads.

For a weighted graph (G, ω), with $G = (V, E)$, the Hidden-Paths algorithm, whose pseudocode is reported in Fig. 3, maintains a min-heap of the best path from u to v found so far, for each ordered pair of distinct nodes $u, v \in V$. The queue is arranged according to shortest-path estimate values and, for paths with the same weight, according to path length. It is initialized with all pairs of distinct nodes (u, v) (considered as paths of length 1) with weight $\omega(u, v)$, where conventionally we set $\omega(u, v) = +\infty$ for any $(u, v) \notin E$. A set *Opt* of optimal paths, initially set to the empty set, is also maintained.

At each phase, the Hidden-Paths algorithm extracts the path at the top of the heap, say $(u \rightsquigarrow v)$, and adds it to *Opt*. It turns out that the extracted path $(u \rightsquigarrow v)$ is optimal, so that the optimality property of *Opt* is not disrupted. The extracted path $(u \rightsquigarrow v)$ can then be used to construct a set of new candidate optimal paths by calling procedure UPDATE(z, u, v), for each (optimal) arc $(z, u) \in Opt \cap E$, and, provided that $(u \rightsquigarrow v)$ has length 1 (i.e., it comprises of a single arc), by also calling procedure UPDATE(u, v, z), for each path $(v \rightsquigarrow z)$ in *Opt*.

[1] In Lemma 6 we will generalize this invariant also to nodes which are reachable from the source node by a shortest path whose arcs have a nonnegative weight, with the only possible exception of the first arc.

When the heap is implemented as a Fibonacci heap, the Hidden-Paths algorithm has an $\mathcal{O}(nm^* + n^2 \log n)$-time complexity, where m^* is the number of optimal arcs in the weighted graph (G, ω).

4. A fast single-source shortest-paths algorithm in the presence of few destinations of negative arcs

In this section we present a new algorithm for the single-source shortest-path problem in the presence of few destinations of negative arcs, by generalizing Yap's approach [40]. Plainly, the case in which there are few sources of negative arcs, rather than few destinations, can be handled by working with the reverse of the graph.

As before, let $G = (V, E)$ be a directed graph with weight function $\omega : E \to \mathbb{R}$ and source $s \in V$, and let T be the set of the destinations of the negative arcs in G. Let $\ell = |T|$ and let $t_1, t_2, \ldots, t_\ell$ be the distinct elements of T.

We present now our algorithm for the SSSP problem.[2]

ALGORITHM 1

[Step 1]. Apply the reverse Dijkstra's algorithm to (G, ω) from each destination node $t_i \in T$.

Let $d_1(v, t_i)$ be the shortest-path estimate function so computed, for $v \in V$ and $t_i \in T$.

(Notice that since G may contain negative-weight arcs, in general $d_1(v, t_i) \neq \delta(v, t_i)$.)

[Step 2]. Let $\overline{G} = (T \cup \{s\}, \overline{E})$ be the directed graph with

$$\overline{E} = \left\{(v, t) \mid v \in T \cup \{s\},\ t \in T,\ v \neq t,\ d_1(v, t) \neq +\infty\right\}$$

and let $\overline{\omega}$ be the weight function on $\overline{G}$ defined by $\overline{\omega}(v, t) = d_1(v, t)$, for $(v, t) \in \overline{E}$.

Apply the Bellman–Ford–Moore algorithm to the weighted graph $(\overline{G}, \overline{\omega})$ from the source node s and let $d_2(v)$ be the distance function so computed. If negative-weight cycles reachable from s in $(\overline{G}, \overline{\omega})$ are detected, notify the presence in (G, ω) of negative-weight cycles reachable from the source node s and exit. Otherwise, go to Step 3.

(Observe that $|\overline{E}| \leqslant \ell^2$. As will be shown later, if no negative-weight cycle is detected, then $d_2(t_i) = \delta(s, t_i)$, for $i = 1, \ldots, \ell$.)

[Step 3]. Let $\tilde{G} = (V, \tilde{E})$, where $\tilde{E} = E \cup \{(s, t) \mid t \in T,\ t \neq s\}$, and let

$$\tilde{\omega}(u, v) = \begin{cases} d_2(v) & \text{if } u = s,\ v \in T,\ u \neq v \\ \omega(u, v) & \text{otherwise,} \end{cases}$$

for $(u, v) \in \tilde{E}$. (In other words, $(\tilde{G}, \tilde{\omega})$ is the weighted graph obtained from (G, ω) by augmenting G with all arcs of the form (s, t), for $t \in T \setminus \{s\}$, not already present in G, and by putting $\tilde{\omega}(s, t) = d_2(t)$, for $t \in T \setminus \{s\}$, and $\tilde{\omega}(u, v) = \omega(u, v)$, for all remaining arcs (u, v) of $\tilde{G}$.)

Apply Dijkstra's algorithm to the weighted graph $(\tilde{G}, \tilde{\omega})$ from the source node s, and return the distance function $d_3(v)$, for $v \in V$, so computed.

In Section 4.2 we will show that Algorithm 1 is correct, namely that $d_3(v) = \delta(s, v)$ holds, for every $v \in V$, provided that G contains no negative-weight cycle reachable from s.

4.1. Complexity issues

The ℓ applications of Dijkstra's algorithm in Step 1 take a total time complexity of $\mathcal{O}(\ell(m + n \log n))$, provided that the service priority queue is implemented with Fibonacci heaps. In addition, Step 2 takes $\mathcal{O}(\ell^3)$ time. Finally, the last application of Dijkstra's algorithm in Step 3 takes an additional $\mathcal{O}(m + \ell^2 + n \log n)$ time. Summing up, Algorithm 1 has an overall $\mathcal{O}(\ell(m + n \log n + \ell^2))$-time complexity.

If $\ell = o(\sqrt[3]{mn})$ (and $n = \mathcal{O}(m)$—we recall that we have assumed such relationship throughout the paper), then Algorithm 1 is asymptotically faster than the Bellman–Ford–Moore algorithm. This can be shown easily by observing that if $\ell = o(\sqrt[3]{mn})$ then we have $\ell^3 = o(mn)$ and $\ell m = o(mn)$, since $m = \mathcal{O}(n^2)$ so that $\ell = o(\sqrt[3]{n^3}) = o(n)$. We also have $\ell n \log n = o(mn)$; indeed, the assumption $n = \mathcal{O}(m)$ yields $n\sqrt[3]{mn} \log n = \mathcal{O}(n\sqrt[3]{m^2} \log m)$; in addition, as $\sqrt[3]{m^2} \log m = \mathcal{O}(m)$ we have $n\sqrt[3]{m^2} \log m = \mathcal{O}(mn)$. Thus, since $\ell n \log n = o(n\sqrt[3]{mn} \log n)$, we have $\ell n \log n = o(mn)$. The above considerations yield immediately that $\ell(m + n \log n + \ell^2) = o(mn)$, provided that $\ell = o(\sqrt[3]{mn})$.

Next we compare Algorithm 1 with Yap's algorithm [40], whose running time is $\mathcal{O}(h(m + n \log n + h^2))$, where h is the minimum between n and the number of the negative-weight arcs contained in the graph. Plainly, $\ell \leqslant h$. Therefore we have $\ell(m + n \log n + \ell^2) = \mathcal{O}(h(m + n \log n + h^2))$, i.e., Algorithm 1 is always asymptotically at least as good as Yap's one. In addition, if $\ell = o(h)$, then $\ell(m + n \log n + \ell^2) = o(h(m + n \log n + h^2))$, i.e., in this case Algorithm 1 is asymptotically faster than Yap's algorithm.

[2] A preliminary version of Algorithm 1 has been presented by the authors in the poster paper [3].

4.2. Correctness proof

For the sake of simplicity, we will limit our considerations only to the case in which no negative-weight cycle is reachable from the given source node s.

Algorithm 1 exploits the fact that even when negative-weight arcs are allowed, the standard Dijkstra's algorithm calculates correctly the distances from the source node to all *safe nodes*, which are defined as follows.

Definition 5 *(Safe nodes and paths).* Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$, and let $s, v \in V$. We say that the node v is *safe relative to s in (G, ω)* if there exists a shortest path π from s to v, called *safe path* for v relative to s, having the form $(s, u \rightsquigarrow v)$, where the subpath $(u \rightsquigarrow v)$ is purely nonnegative (thus, only the first arc of a safe path is allowed to have a negative weight).

Safe nodes are handled correctly by Dijkstra's algorithm, as proved in the following lemma.

Lemma 6. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$ containing no negative-weight cycle reachable from a given source $s \in V$. Let $v \in V$ be a safe node in (G, ω) relative to s. Then, when v is extracted from the queue during an execution of Dijkstra's algorithm on the weighted graph (G, ω) from source s, $d(v) = \delta(s, v)$ holds.*

Proof. Let $v \in V$ be a safe node in (G, ω) relative to s and assume by way of contradiction that $d(v) \neq \delta(s, v)$ holds when the node v is extracted from the queue, during the execution of Dijkstra's algorithm on (G, ω) from the source s. Then $v \neq s$, since we are assuming that no negative-weight cycle is reachable from the source s, so that $d(s) = 0 = \delta(s, s)$ must hold during the whole execution of Dijkstra's algorithm. In addition, we must also have $d(v) > \delta(s, v)$, when v is extracted from the queue, since the invariant $d(u) \geqslant \delta(s, u)$ is maintained for every node u during the execution of any shortest-path algorithm based on the labeling method (cf. [7]).

Let π be a safe path for v relative to s. Without loss of generality, we can assume that v is the closest node u to s on the path π which satisfies $d(u) \neq \delta(s, u)$, when extracted from the queue. Let S_v be the collection of nodes on the path π which have not yet been extracted from the queue when the node v is extracted (maybe just because they did not even enter the queue at that time). The set S_v is nonempty, as the predecessor v' of v in π must belong to it. Indeed, if this were not the case, then $d(v') = \delta(s, v')$ would hold when v' is extracted from the queue, so that, after the subsequent execution of $\textsc{Scan}(G, \omega; v')$, we would have $d(v) = \delta(s, v)$. Let then w be the closest node to s on the path π which belongs to S_v, and let w' be its predecessor on π. After the execution of $\textsc{Scan}(G, \omega; w')$ following the extraction of w', we have $d(w) = \delta(s, w)$, so that when the node v is extracted from the queue we have

$$d(v) > \delta(s, v) = \delta(s, w) + \omega(w \overset{\pi}{\rightsquigarrow} v) = d(w) + \omega(w \overset{\pi}{\rightsquigarrow} v),$$

where $(w \overset{\pi}{\rightsquigarrow} v)$ is the subpath of π from w to v. But since $\omega(w \overset{\pi}{\rightsquigarrow} v) \geqslant 0$, as all arcs in $(w \overset{\pi}{\rightsquigarrow} v)$ have nonnegative weight, it follows that $d(v) > d(w)$, contradicting the fact that when v is extracted from the queue it has the smallest shortest-path estimate among all the nodes in the queue.

Summing up, it follows that $d(v) = \delta(s, v)$ must hold when the node v is extracted from the queue, completing the proof of the lemma. □

For the rest of the section, let $G = (V, E)$, $\omega : E \to \mathbb{R}$, $s \in V$, and $T = \{t_1, t_2, \ldots, t_\ell\}$ be as in the opening of Section 4, and let δ be the distance function on (G, ω).

Let $v \in V$, $t_i \in T$, and let $(v \rightsquigarrow t_i)$ be a shortest path in G from v to t_i. After the execution of Step 1 of Algorithm 1, Lemma 6 implies that if no internal node of the path $(v \rightsquigarrow t_i)$ belongs to T, then $d_1(v, t_i) = \delta(v, t_i)$ holds, since the node v is safe relative to t_i in the reverse weighted graph $(G^{\mathsf{R}}, \omega^{\mathsf{R}})$ (where we recall that $d_1(v, t_i)$ denotes the distance function computed by the call to the reverse Dijkstra's algorithm from destination t_i, within Step 1). If $(v \rightsquigarrow t_i)$ has the form $(v \rightsquigarrow t_j \rightsquigarrow t_i)$, where both of its subpaths $(v \rightsquigarrow t_j)$ and $(t_j \rightsquigarrow t_i)$ contain no node in T among their internal nodes, then t_j is safe relative to t_i and v is safe relative to t_j in the reverse weighted graph $(G^{\mathsf{R}}, \omega^{\mathsf{R}})$, so that by two applications of Lemma 6 we obtain

$$\delta(v, t_i) = \delta(v, t_j) + \delta(t_j, t_i) = d_1(v, t_j) + d_1(t_j, t_i).$$

By repeating the above reasoning, it is easy to prove the following more general result:

Lemma 7. *Let $t_{i_j} \in T$, for $j = 1, \ldots, q$, and $t_{i_0} \in V$, and let*

$$(t_{i_0} \rightsquigarrow t_{i_1} \rightsquigarrow \cdots \rightsquigarrow t_{i_q})$$

be a shortest path in (G, ω) from t_{i_0} to t_{i_q} such that all of its subpaths $(t_{i_j} \rightsquigarrow t_{i_{j+1}})$, for $j = 0, 1, \ldots, q-1$, contain no node in T among their internal nodes. Then $\delta(t_{i_0}, t_{i_q}) = \sum_{j=0}^{q-1} d_1(t_{i_j}, t_{i_{j+1}})$. □

```
TWO-NEGATIVES-DIJKSTRA(G, ω; s)
  - let V be the set of nodes of G and let S be the set
    of the sources of the negative-weight arcs in (G, ω)
  INITIALIZE-SSSP(G, s)
  SCAN(G, ω; s)
  for u ∈ S \ {s} do
    SCAN(G, ω; u)
  Q := Make_Queue(V \ {s}, d)
  while Q ≠ ∅ do
    v := Extract_Min(Q)
    SCAN(G, ω; v)


REVERSE-TWO-NEGATIVES-DIJKSTRA(G, ω; t)
  TWO-NEGATIVES-DIJKSTRA(G^R, ω^R; t)
```

Fig. 4. The "two-negatives" variant of Dijkstra's algorithm and its reverse form.

In the particular case in which $t_{i_0} = s$ in the previous lemma, we have $\delta(s, t_{i_q}) = \sum_{j=0}^{q-1} d_1(t_{i_j}, t_{i_{j+1}})$. But, by Step 2 and Lemma 1,

$$\sum_{j=0}^{q} d_1(t_{i_j}, t_{i_{j+1}}) \geqslant d_2(t_{i_q}) \geqslant \delta(s, t_{i_q}),$$

so that $d_2(t_{i_q}) = \delta(s, t_{i_q})$, proving the following result[3]:

Corollary 8. *After the execution of Step* 2 *of Algorithm* 1, *we have* $d_2(t_i) = \delta(s, t_i)$, *for every* $t_i \in T$. □

Thus, after the execution of Step 2 all distances from the source s to each node in the set T have been correctly computed.

Next we show that Step 3 propagates the correct shortest-path estimate also to the remaining nodes of the graph, thus completing the proof of correctness of Algorithm 1.

Theorem 9 *(Correctness). Let* $d_3(v)$ *be the distance function computed by Algorithm* 1 *(by the last call to Dijkstra's algorithm in Step* 3*). Then* $d_3(v) = \delta(s, v)$ *holds, for each* $v \in V$.

Proof. Let $v \in V$ be a node reachable from s in G. In view of Lemmas 6 and 2, to prove that $d_3(v) = \delta(s, v)$ holds, it is enough to show that the node v is safe relative to s in $(\tilde{G}, \tilde{\omega})$.

If $v = s$, then the trivial path of length 0 from s to s is a safe path from s in (G, ω), and therefore in $(\tilde{G}, \tilde{\omega})$, as we are assuming that (G, ω) contains no negative-weight cycle reachable from s, and the same must therefore be true for $(\tilde{G}, \tilde{\omega})$.

If $v \in T \setminus \{s\}$, then, by Corollary 8, $d_2(v) = \delta(s, v)$. Thus, the arc (s, v) is optimal in (G, ω), and therefore in $(\tilde{G}, \tilde{\omega})$. Hence, the node v is safe relative to s in $(\tilde{G}, \tilde{\omega})$.

Finally, if $v \in V \setminus (T \cup \{s\})$, let $(s \rightsquigarrow v)$ be a shortest path from s to v in (G, ω). If $(s \rightsquigarrow v)$ contains no node in $T \setminus \{s\}$, then it is a safe path for v relative to s in $(\tilde{G}, \tilde{\omega})$. Otherwise, $(s \rightsquigarrow v)$ has the form $(s \rightsquigarrow t \rightsquigarrow v)$, where $t \in T$ and the subpath $(t \rightsquigarrow v)$ contains no node in T among its internal nodes. Then $(s, t \rightsquigarrow v)$ is a safe path for v relative to s in $(\tilde{G}, \tilde{\omega})$.

Having shown that v is safe relative to s in $(\tilde{G}, \tilde{\omega})$, Lemma 6 yields $d_3(v) = \delta_{\tilde{G}, \tilde{\omega}}(s, v)$. But, by Lemma 2, $\delta_{\tilde{G}, \tilde{\omega}}(s, v) = \delta(s, v)$. Hence $d_3(v) = \delta(s, v)$, completing the proof of the theorem. □

5. A fast all-pairs shortest-paths algorithm in the presence of few negative arcs

In this section we present a new algorithm for the APSP problem in the presence of few negative-weight arcs. Our algorithm is a hybridization of Dijkstra's (and a variant of it), Floyd's, and the Hidden-Paths algorithms, and is obtained by generalizing the approach for the single-source case presented in the previous section. In particular, besides the reverse Dijkstra's standard algorithm, it also uses as a subroutine the reverse form of a variant of Dijkstra's algorithm, called TWO-NEGATIVES-DIJKSTRA.

After a preliminary scan of the source node, the TWO-NEGATIVES-DIJKSTRA algorithm, whose pseudocode is reported in Fig. 4, scans also all the source nodes of the negative arcs present in the input graph, and then continues its execution

[3] We recall that d_2 is the distance function computed by the call to the Bellman–Ford–Moore algorithm within Step 2.

just as the standard Dijkstra's algorithm. It turns out, as will be shown in Section 5.2, that the Two-Negatives-Dijkstra algorithm handles correctly *2-safe nodes*, which generalize the notion of safe nodes (cf. Definition 5). We will define 2-safe nodes in Section 5.2.

As before, let $G = (V, E)$ be a directed graph, with $n = |V|$ and $m = |E|$, let $\omega : E \to \mathbb{R}$ be a weight function, and let T be the collection of the destinations of the negative arcs in G. Also, let us put $\ell = T$ and let $t_1, \ldots, t_\ell$ be the distinct nodes in T.

Our algorithm for the APSP problem is reported below.

ALGORITHM 2

[Step 1]. Apply the reverse Dijkstra's algorithm to (G, ω) from each node $t \in T$.

Let $d_1(v, t)$ be the shortest-path estimate function so computed, for $v \in V$ and $t \in T$.

(Notice that due to the possible presence of negative-weight arcs, the distances computed by Step 1 are not necessarily correct.)

[Step 2]. Let $\overline{G} = (T, \overline{E})$ be the directed graph (with no self-loops) over T, with

$$\overline{E} = \{(t, t') \mid t, t' \in T,\ t \neq t',\ d_1(t, t') \neq +\infty\},$$

and let $\overline{\omega}$ be the weight function on $\overline{G}$ defined by $\overline{\omega}(t, t') = d_1(t, t')$, for all $(t, t') \in \overline{E}$.

Apply Floyd's algorithm to the weighted graph $(\overline{G}, \overline{\omega})$, and let $d_2(t, t')$ be the distances so computed, for all distinct $t, t' \in T$.

If negative-weight cycles are detected in $(\overline{G}, \overline{\omega})$, notify the presence of negative-weight cycles in (G, ω) and stop. Otherwise, go to Step 3.

(We plainly have $|\overline{E}| \leqslant \ell(\ell - 1) < \ell^2$. As will be shown later in Lemma 14, when no negative-weight cycle is present in $(\overline{G}, \overline{\omega})$, we have $d_2(t, t') = \delta(t, t')$, for all distinct $t, t' \in T$.)

[Step 3]. Let $\tilde{G} = (V, \tilde{E})$ be the directed graph such that $\tilde{E} = E \cup \overline{E}$ and let $\tilde{\omega}$ be the weight function on $\tilde{G}$ defined by

$$\tilde{\omega}(u, v) = \begin{cases} d_2(u, v) & \text{if } (u, v) \in \overline{E} \\ \omega(u, v) & \text{otherwise,} \end{cases}$$

for $(u, v) \in \tilde{E}$ (i.e., the weighted graph $(\tilde{G}, \tilde{\omega})$ is obtained by superimposing $(\overline{G}, \overline{\omega})$ to (G, ω)).

Apply the reverse of the Two-Negatives-Dijkstra algorithm to $(\tilde{G}, \tilde{\omega})$ from each node $t \in T$.

Let $d_3(v, t)$ be the distance function so computed, for $v \in V$ and $t \in T$.

(Notice that $|E| \leqslant |\tilde{E}| < m + \ell^2$. As will be shown later in Lemma 15, the distance function d_3 computed by Step 3 is correct, in the sense that $d_3(v, t) = \delta(v, t)$, for all $v \in V$ and $t \in T$.)

[Step 4]. Let $\hat{G} = (V, \hat{E})$, where

$$\hat{E} = E \cup \{(u, t) \colon u \in V,\ t \in T,\ u \neq v\},$$

and let $\hat{\omega}$ be the weight function on $\hat{E}$ defined by

$$\hat{\omega}(u, v) = \begin{cases} d_3(u, v) & \text{if } v \in T \\ \omega(u, v) & \text{otherwise.} \end{cases}$$

Apply the Hidden-Paths algorithm to the weighted graph $(\hat{G}, \hat{\omega})$ and return the distance function d_4 so computed.

In Section 5.2 we will show that at the end of Step 4 we have $d_4(u, v) = \delta(u, v)$, for every $u, v \in V$, i.e., Algorithm 2 is correct.

5.1. Complexity analysis

We will evaluate the asymptotic behavior of Algorithm 2 under the assumption that all service priority queues are implemented with Fibonacci heaps.

To begin with, the ℓ applications of the reverse of Dijkstra's algorithm in Step 1 take a total time complexity of $\mathcal{O}(\ell(m + n \log n))$, where we recall that ℓ is the size of the set T. In addition, the execution of Floyd's algorithm in Step 2, with an input graph with ℓ nodes, takes $\mathcal{O}(\ell^3)$-time, whereas the ℓ applications of the reverse of the Two-Negatives-Dijkstra algorithm in Step 3, with an input graph of n nodes, at most $m + \ell^2$ arcs, and ℓ destinations of negative arcs, take a total time complexity of $\mathcal{O}(\ell(m + \ell^2 + \ell n + n \log n))$. Finally, the execution of the Hidden-Paths algorithm in Step 4 on the weighted graph $(\hat{G}, \hat{\omega})$ takes $\mathcal{O}(n\hat{m}^* + n^2 \log n)$-time, where $\hat{m}^*$ is the number of optimal arcs in $(\hat{G}, \hat{\omega})$. Since $|\hat{E}| \leqslant |E| + \ell n$, the weighted graph $(\hat{G}, \hat{\omega})$ can contain at most ℓn optimal arcs more than (G, ω), i.e., $\hat{m}^* \leqslant m^* + \ell n$. Thus the total time complexity for Step 4 is $\mathcal{O}(nm^* + n^2 \log n + \ell n^2)$.

In conclusion, Algorithm 2 has an overall $\mathcal{O}(nm^* + n^2 \log n + \ell n^2)$-time complexity, as $\ell^3 = \mathcal{O}(\ell n^2)$, $\ell m = \mathcal{O}(\ell n^2)$, and $\ell n \log n = \mathcal{O}(n^2 \log n)$.

Based on the results in [15,20,28], in [24] it is argued that m^* is likely to be small in practice, since $m^* = \mathcal{O}(n \log n)$ with high probability for many probability distributions on arc weights. Thus the assumptions that we will make below on m^* are quite likely to be met in practice. Notice that when $m^* = \mathcal{O}(n \log n)$, Algorithm 2 achieves an $\mathcal{O}(n^2 \log n + \ell n^2)$-time complexity. If, furthermore, $\ell = \mathcal{O}(\log n)$, then the complexity of our algorithm reduces to $\mathcal{O}(n^2 \log n)$.

Next we compare the asymptotic behavior of Algorithm 2 with that of other algorithms for the APSP problem present in the literature.

Algorithm 2 *vs. Hidden-Paths algorithm.* To begin with, we observe at once that when

$$\ell = \mathcal{O}\left(\frac{m^*}{n} + \log n\right), \tag{1}$$

Algorithm 2 achieves the same time complexity $\mathcal{O}(nm^* + n^2 \log n)$ of the Hidden-Paths algorithm, which however solves the APSP problem *only* in the case of nonnegative weighted graphs. Indeed, if (1) holds, then we have $\mathcal{O}(\ell n^2) = \mathcal{O}(nm^* + n^2 \log n)$, so that $nm^* + n^2 \log n + \ell n^2 = \mathcal{O}(nm^* + n^2 \log n)$.

Algorithm 2 *vs. Johnson's algorithm.* We recall that Johnson's algorithm has an $\mathcal{O}(mn + n^2 \log n)$-time complexity. When $\ell = \mathcal{O}(\frac{m}{n} + \log n)$, Algorithm 2 achieves the same asymptotic of Johnson's algorithm. However, if

$$m^* = o(m), \quad m = \omega(n \log n), \qquad \ell = o\left(\frac{m}{n}\right),$$

then Algorithm 2 is asymptotically faster than Johnson's one, since in this case $m^* n = \mathcal{O}(n^2 \log n)$, $n^2 \log n = o(mn)$, and $\ell n^2 = o(mn)$.

Algorithm 2 *vs. Floyd's algorithm.* When $\ell = o(n)$ and $m^* = o(n^2)$, Algorithm 2 is asymptotically faster than Floyd's algorithm.

Algorithm 2 *vs.* $\mathcal{O}(\frac{n^3}{f(n)})$*-time algorithms.* The quest for truly subcubic algorithms for the APSP problem has produced so far several algorithms characterized by an $\mathcal{O}(\frac{n^3}{f(n)})$-time complexity, with $f(n) = o(\log^2 n)$; see [4,5,9,13,17–19,36–38,41]. Up to the time of the writing of this paper, the fastest algorithm for the APSP problem is due to Han and Takaoka [19], where $f(n) = \log^2 n / \log\log n$, and it is believed that current combinatorial techniques are not able to break the $(\log^2 n)$-barrier for $f(n)$. It is an easy matter to check that when $m^* = o(\frac{n^2}{\log^2 n})$ and $\ell = o(\frac{n}{\log^2 n})$, our Algorithm 2 achieves an $o(\frac{n^3}{\log^2 n})$-time complexity.

5.2. Correctness proof

Let, as above, $G = (V, E)$ be a directed graph and let $\omega : E \to \mathbb{R}$ be a weight function on G. For the sake of simplicity, we will limit again our considerations to the case in which *no* negative-weight cycle is present in (G, ω).

The correctness of Algorithm 2 is based on the facts that when negative-weight arcs are allowed:

- Dijkstra's algorithm from a source node s calculates correctly the distances from s to all *safe* nodes relative to s (cf. Definition 5 and Lemma 6),
- the Two-Negatives-Dijkstra algorithm from a source node s calculates correctly the distances from s to all *2-safe* nodes relative to s (cf. Definition 10 and Lemma 11 below), and
- the Hidden-Paths algorithm calculates correctly the distances from any source node s to all safe nodes relative to s, as will be proved in Lemma 13.

The notion of 2-safe nodes generalizes that of safe nodes, introduced in Definition 5.

Definition 10 *(2-safe nodes and paths).* Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$, and let $s, v \in V$. We say that the node v is *2-safe relative to* s *in* (G, ω) if there exists a shortest path π from s to v, called a *2-safe path* for v relative to s, which either is safe (in the sense of Definition 5) or has the form $(s, u, u' \rightsquigarrow v)$, where the subpath $(u' \rightsquigarrow v)$ of π is purely nonnegative whereas both arcs (s, u) and (u, u') have negative weight (thus, only the first two arcs of a 2-safe path are allowed to have a negative weight, with the further restriction that if the first arc is nonnegative, so must be the second arc).

Next we prove that, as already remarked, the Two-Negatives-Dijkstra algorithm handles correctly the 2-safe nodes relative to the source node.

Lemma 11. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$ containing no negative-weight cycle reachable from a given source $s \in V$. Let $v \in V$ be a 2-safe node in (G, ω) relative to s. Then, when v is extracted from the queue during an execution of the* Two-Negatives-Dijkstra *algorithm on the weighted graph (G, ω) from source s, $d(v) = \delta(s, v)$ holds.*

Proof. Let (G, ω) and $s \in V$ be as in the statement of the lemma, and let S be the set of sources of the negative-weight arcs of (G, ω). After the initial scan of the source s and the subsequent scans of the nodes in $S \setminus \{s\}$ performed by the Two-Negatives-Dijkstra algorithm in its initial phase, we have

$$d(v) = \min_{u \in S \setminus \{s\}} \big(\omega(s, u) + \omega(u, v)\big), \tag{2}$$

for $v \in V$. Thus, the subsequent phase of the Two-Negatives-Dijkstra algorithm is equivalent to running the standard Dijkstra's algorithm (after the initial scan of the source s) on the directed graph $G' = (V, E')$, where

$$E' = E \cup \big\{(s, v) \mid v \neq s \text{ and } d(v) \neq +\infty\big\},$$

with the weight function $\omega' : E' \to \mathbb{R}$ defined by

$$\omega'(u, v) = \begin{cases} d(v) & \text{if } u = s,\ v \neq s,\ d(v) \neq +\infty \\ \omega(u, v) & \text{otherwise,} \end{cases}$$

for $(u, v) \in E'$. Since, by (2), any 2-safe node in (G, ω) relative to s is a safe node in (G', ω') relative to s, Lemma 6 yields our thesis. □

In order to prove that the Hidden-Paths algorithm calculates correctly the distances from any source node s to all safe nodes relative to s, we need to strengthen the correctness proof of the Hidden-Paths algorithm (cf. [24, Theorem 2.2]) when run in presence of negative-weight arcs.

Lemma 12. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$, containing no negative-weight cycle. Then, during the execution of the Hidden-Paths algorithm on (G, ω), when a path $(u \rightsquigarrow v)$ whose endpoints u and v can be connected by a purely nonnegative shortest path is at the top of the heap, it is optimal.*[4]

Proof. In what follows, we say that a pair (u, v) of distinct nodes of G is *good* if there exists a purely nonnegative shortest path from u to v in (G, ω).

The lemma can be proved by induction on the number of iterations of the **while**-loop during an execution of the Hidden-Paths algorithm (cf. Fig. 3) by using as inductive hypothesis the following modification of the one adopted in the proof of Theorem 2.2 in [24]:

> at the beginning of each iteration of the **while**-loop in an execution of the Hidden-Paths algorithm, with the path $(u \rightsquigarrow v)$ at the top of the heap, if the pair (u, v) of its endpoints is good, then the path $(u \rightsquigarrow v)$ is optimal (though not necessarily purely nonnegative).

For the base case, let $(u \rightsquigarrow v)$ be the path at the top of the heap in the first iteration. Plainly, $(u \rightsquigarrow v)$ is just an arc, because of the way the heap is initialized. If $\omega(u, v) < 0$, there is nothing to prove, since in this case the pair (u, v) is not good. On the other hand, if $\omega(u, v) \geqslant 0$, then, by the minimality of $\omega(u, v)$, the weight function ω would be nonnegative, so that any minimal weight arc would be optimal.

For the inductive step, let $\pi = (u \rightsquigarrow v)$ be the path at the top of the heap at the beginning of the p-th iteration of the **while**-loop of the Hidden-Paths algorithm (with $p > 0$), and assume that the inductive hypothesis has been always satisfied at the beginning of all previous iterations. If the pair (u, v) is not good, then there is nothing to prove. So, let us assume that (u, v) is good and, by way of contradiction, let us also assume that the path π is not optimal, i.e., $\omega(\pi) > \delta(u, v)$. Let $(u_0, u_1, \ldots, u_k)$ be a nonnegative shortest path from $u_0 = u$ to $u_k = v$, so that $\omega(\pi) > \omega(u_0, u_1, \ldots, u_k)$ holds. It turns out that, at the beginning of the p-th iteration, the heap contains no path of the form $(u_i \rightsquigarrow u_{i+1})$, for $i = 0, 1, \ldots, k-1$, as

$$\omega(u_i, u_{i+1}) \leqslant \omega(u_0, u_1, \ldots, u_k) < \omega(\pi).$$

But since initially the heap contained all the arcs (u_i, u_{i+1}), for $i = 0, 1, \ldots, k-1$, these must have been moved to the set *Opt*. In particular, this is the case for the arc (u_{k-1}, u_k). Let $0 \leqslant i_0 \leqslant k-1$ be the minimal index i such that the set *Opt* contains a path of the form $(u_i \rightsquigarrow u_k)$ at the beginning of the p-th iteration. Observe that $i_0 > 0$, since the path π, which has the form $(u_0 \rightsquigarrow u_k)$, is on the heap and therefore no path from u_0 to u_k can be contained in the set *Opt*. Notice also that since the pair (u_{i_0}, u_k) is good, then, by the inductive hypothesis, the path $(u_{i_0} \rightsquigarrow u_k)$ in *Opt* must be optimal. During

[4] We recall that a path is said to be purely nonnegative if all its arcs have nonnegative weight.

the iteration in which the last one among the paths $(u_{i_0} \rightsquigarrow u_k)$ and (u_{i_0-1}, u_{i_0}) entered the set *Opt*, the heap contained an optimal path of the form $(u_{i_0-1} \rightsquigarrow u_k)$. But since

$$\omega(u_{i_0-1} \rightsquigarrow u_k) \leqslant \omega(u_0, u_1, \ldots, u_k) < \omega(\pi),$$

such a path cannot be on the heap any more during the p-th iteration, and therefore it must have been moved to the set *Opt* during a previous iteration, contradicting the minimality of i_0. Thus, the path π at the top of the heap must be optimal, completing the proof of the lemma. □

We are now ready to show that safe paths are handled correctly by the Hidden-Paths algorithm.

Lemma 13. *Let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$. Then $d(u, v) = \delta(u, v)$ holds at the end of the execution of the Hidden-Paths algorithm on (G, ω), for every pair of distinct nodes $u, v \in V$, such that v is safe relative to u in (G, ω).*

Proof. Let $(u \rightsquigarrow v)$ be a safe (shortest) path for v relative to u in (G, ω). If $(u \rightsquigarrow v)$ is purely nonnegative, then the thesis follows directly from the preceding lemma. On the other hand, if the path $(u \rightsquigarrow v)$ contains an initial negative-weight arc, we distinguish two cases according to whether $(u \rightsquigarrow v)$ has length 1 or not. If $(u \rightsquigarrow v)$ has length 1, then it consists only of the negative-weight arc (u, v). In this case $d(u, v)$ is set to $\delta(u, v)$ already in the initialization phase, and its value can never change during the subsequent execution of the Hidden-Paths algorithm since (u, v) is an optimal edge. Otherwise, if the path $(u \rightsquigarrow v)$ contains more than one arc, then it has the form $(u, t \rightsquigarrow v)$, where the arc (u, t) has negative weight and the subpath $(t \rightsquigarrow v)$ is a purely nonnegative shortest path from t to v. Thus, again by the preceding lemma, when a path from t to v is at the top of the heap, it must be optimal, i.e., $d(t, v) = \delta(t, v)$ must hold. At that time the arc (u, t) must already have left the heap, as $d(u, t) < 0 \leqslant d(t, v)$, and therefore procedure UPDATE(u, t, v) will be called. After such a call we will have

$$\delta(u, v) \leqslant d(u, v) \leqslant d(u, t) + d(t, v) = \delta(u, v),$$

which implies that $d(u, v) = \delta(u, v)$ holds, and since the value $d(u, v)$ cannot further decrease, $d(u, v) = \delta(u, v)$ will hold also at the end of the execution of the Hidden-Paths algorithm. □

We are now ready to prove the correctness of Algorithm 2, by analyzing its four steps. Thus, as before, let $G = (V, E)$ be a directed graph with a weight function $\omega : E \to \mathbb{R}$ and let $t_1, t_2, \ldots, t_\ell$ be the distinct elements of the set T of the destinations of all negative-weight arcs in (G, ω).

By Lemma 6, $d_1(v, t) = \delta(v, t)$ holds after the execution of Step 1, for every $v \in V$ and $t \in T$ such that v is safe relative to t in the reverse of (G, ω). Thus, if

$$(t_{i_0} \rightsquigarrow t_{i_1} \rightsquigarrow \cdots \rightsquigarrow t_{i_q})$$

is a shortest path from t_{i_0} to t_{i_q}, where all the nodes t_{i_j} are in T and none of its subpaths $(t_{i_j} \rightsquigarrow t_{i_{j+1}})$ contains any node in the set T but its endpoints, then $d_1(t_{i_j}, t_{i_{j+1}}) = \delta(t_{i_j}, t_{i_{j+1}})$, for $j = 0, \ldots, q-1$, as $t_{i_{j+1}}$ is safe relative to t_{i_j} in the reverse weighted graph (G^R, ω^R), so that

$$d_2(t_{i_0}, t_{i_q}) \geqslant \delta(t_{i_0}, t_{i_q}) = \sum_{j=0}^{q-1} \delta(t_{i_j}, t_{i_{j+1}}) = \sum_{j=0}^{q-1} d_1(t_{i_j}, t_{i_{j+1}}) \geqslant d_2(t_{i_0}, t_{i_q})$$

(where d_2 is the distance function computed by the call to Floyd's algorithm on the graph $(\overline{G}, \overline{\omega})$ within Step 2). Hence, $d_2(t_{i_0}, t_{i_q}) = \delta(t_{i_0}, t_{i_q})$ holds.

Therefore, we have proved the following result:

Lemma 14. *After the execution of Step 2 of Algorithm 2, we have $d_2(t, t') = \delta(t, t')$, for all distinct $t, t' \in T$.*

In Step 3 we have constructed the weighted graph $(\tilde{G}, \tilde{\omega})$, where $\tilde{G}$ is obtained by adding to our input graph G all arcs in $\overline{E}$, namely all arcs (t, t') between distinct nodes t and t' of the set T such that t' is reachable from t in G, with $\tilde{\omega}(t, t') = \delta(t, t')$ (cf. Lemma 14). In the following lemma we prove that all shortest-path distances in (G, ω) to nodes in the set T are correctly computed by Step 3.

Lemma 15. *After the execution of Step 3 of Algorithm 2, we have $d_3(v, t) = \delta(v, t)$, for every $v \in V$ and $t \in T$, where we recall that d_3 is the distance function computed by Step 3.*

Proof. Since, as shown in Lemma 14, all arcs (t, t') between distinct nodes t and t' in T, such that t' is reachable from t in G, are optimal in the weighted graph $(\tilde{G}, \tilde{\omega})$ constructed in Step 3, their reverses are also optimal in the reverse weighted graph $(\tilde{G}^R, \tilde{\omega}^R)$.

Let $(v \rightsquigarrow t)$ be a shortest path in (G, ω), with $v \in V$ and $t \in T$ and such that $v \neq t$. Then, its reverse path $(t \rightsquigarrow v)$ is a shortest path in (G^R, ω^R). Let t' be the last node in $(t \rightsquigarrow v)$ which belongs to T. If $t' = v$, then (t, v) is an optimal arc of $(\tilde{G}^R, \tilde{\omega}^R)$. On the other hand, if $t' = t$, then the node v would be safe relative to t in $(\tilde{G}^R, \tilde{\omega}^R)$. Finally, if $t' \neq v$ and $t' \neq t$, then $(\tilde{G}^R, \tilde{\omega}^R)$ would contain an optimal path of the form $(t, t' \rightsquigarrow v)$, where the subpath $(t' \rightsquigarrow v)$ contains no node in T but its source. Thus, in any case it follows that the node v is 2-safe relative to t in $(\tilde{G}^R, \tilde{\omega}^R)$, so that, by Lemma 11, after the call to the reverse of Two-Negatives-Dijkstra algorithm from t within Step 3, we have $d_3(v, t) = \delta(v, t)$, completing the proof of the lemma. □

Finally, we show that Step 4 propagates the correct distance estimates also to the remaining nodes of the graph.

Theorem 16 *(Correctness). Let d_4 be the distance function computed by Algorithm* 2 *(namely, after the execution of the Hidden-Paths algorithm within Step* 4*). Then $d_4(u, v) = \delta(u, v)$, for all $u, v \in V$.*

Proof. Let $(\hat{G}, \hat{\omega})$ be the weighted graph constructed in Step 4. Observe that, for all distinct nodes $u \in V$ and $t \in T$ such that t is reachable from u in G, the graph $\hat{G}$ contains the optimal arc (u, t) (since $\hat{\omega}(u, t) = d_3(u, t) = \delta(u, t)$, by Lemma 15).

Let $u, v \in V$ be two connected nodes in our input graph G such that $u \neq v$, and let $(u \rightsquigarrow v)$ be a shortest path from u to v in (G, ω).

If the path $(u \rightsquigarrow v)$ is purely nonnegative, then, by Lemma 12, after the execution of the Hidden-Paths algorithm within Step 4 we have $d_4(u, v) = \delta(u, v)$.

On the other hand, if the path $(u \rightsquigarrow v)$ contains some negative-weight arc, we distinguish two cases according to whether $v \in T$ or not. If $v \in T$, then the graph $(\hat{G}, \hat{\omega})$ contains the optimal arc (u, v), so that v would be safe relative to u in $(\hat{G}, \hat{\omega})$. Otherwise, the path $(u \rightsquigarrow v)$ must have the form $(u \rightsquigarrow t \rightsquigarrow v)$, for some $t \in T$, where the subpath $(t \rightsquigarrow v)$ contains no node in T but its source. But then, again, v would be safe relative to u in $(\hat{G}, \hat{\omega})$, because of the safe path $(u, t \rightsquigarrow v)$ in $(\hat{G}, \hat{\omega})$. Thus, by Lemma 13, even in the case in which the path $(u \rightsquigarrow v)$ contains some negative-weight arc it follows that $d_4(u, v) = \delta(u, v)$, concluding the proof of the theorem. □

6. Conclusions

We have presented two asymptotically fast algorithms for the single-source shortest-paths and the all-pairs shortest-paths problems, respectively, in the presence of few destinations of negative-weight arcs. Obviously, by reversing the input graph, our algorithms achieve the same time complexities also in the case of graphs with few sources of negative-weight arcs.

Our algorithm for the SSSP problem is an improvement of an algorithm due to C.K. Yap for the shortest-path problem between *two* nodes in a directed graph with n nodes and m arcs in the presence of few negative-weight arcs, whose running time is $\mathcal{O}(h(m + n \log n + h^2))$, where h is the minimum between n and the number of the negative-weight arcs in the graph. Our algorithm has an $\mathcal{O}(\ell(m + n \log n + \ell^2))$-time complexity, where ℓ is the number of destinations of negative-weight arcs. Since $\ell \leqslant h$, our algorithm is always asymptotically as good as Yap's one, and it is asymptotically faster than Yap's algorithm when $\ell = o(h)$. In addition, when $\ell = o(\sqrt[3]{mn})$, our algorithm turns out to be asymptotically faster than the $\mathcal{O}(mn)$ Bellman–Ford–Moore algorithm.

In the case of the APSP problem, we have presented an $\mathcal{O}(nm^* + n^2 \log n + \ell n^2)$-time algorithm, where m^* is the number of arcs participating in shortest paths and, again, ℓ is the number of destinations of the negative-weight arcs present in the graph. When $\ell = o(n)$ and $m^* = o(n^2)$, our algorithm turns out to be asymptotically faster than Floyd's algorithm. In addition, if $\ell = \mathcal{O}(\frac{m^*}{n} + \log n)$, our algorithm achieves the same $\mathcal{O}(nm^* + n^2 \log n)$-time complexity of the Hidden-Paths algorithm which, however, works only with nonnegative weighted graphs. We also compared our algorithm with Johnson's algorithm and pointed out the cases in which they reach the same time complexity and when our algorithm outperforms Johnson's one. Finally, we showed that when $m^* = o(\frac{n^2}{\log^2 n})$ and $\ell = o(\frac{n}{\log^2 n})$, our algorithm achieves an $o(\frac{n^3}{\log^2 n})$-time complexity, outperforming the best known subcubic algorithms for the APSP problem. The above restrictions on m^* are very acceptable, as m^* is likely to be $\mathcal{O}(n \log n)$ in practice (cf. [24]).

References

[1] R. Bellman, On a routing problem, Quart. Appl. Math. 16 (1958) 87–90.
[2] D. Cantone, S. Faro, Two-Levels-Greedy: a generalization of Dijkstra's shortest path algorithm, Electron. Notes Discrete Math. 17 (2004) 81–86.
[3] D. Cantone, S. Faro, A faster algorithm for the single source shortest path problem in the presence of few sources or destinations of negative arcs, in: M. Bieliková, M. Nielsen, A. Kučera, P.B. Miltersen, C. Palamidessi, P. Tůma, F.D. Valencia (Eds.), Proc. of SOFSEM 2009, vol. II, MATFYZPRESS, Prague, 2008, pp. 125–128 (Poster paper).
[4] T.M. Chan, All-pairs shortest paths with real weights in $\mathcal{O}(n^3/\log n)$ time, in: Proceedings of the 9th International Conference on Algorithms and Data Structures, WADS'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 318–324.

[5] T.M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, in: Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing, STOC'07, ACM, New York, NY, USA, 2007, pp. 590–598.
[6] B.V. Cherkassky, A.V. Goldberg, T. Radzik, Shortest paths algorithms: Theory and experimental evaluation, in: SODA: ACM–SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 1993.
[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, McGraw–Hill Higher Education, 2001.
[8] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
[9] W. Dobosiewicz, A more efficient algorithm for the min-plus multiplication, Int. J. Comput. Math. 32 (1990) 49–60.
[10] J. Fakcharoenphol, S. Rao, Planar graphs, negative weight edges, shortest paths, and near linear time, J. Comput. System Sci. 72 (2006) 868–889.
[11] R.W. Floyd, Algorithm 97: Shortest path, Commun. ACM 5 (1962) 345.
[12] L. Ford, Network flow theory, Paper P-23, The RAND Corporation, Santa Monica, CA, USA, 1956.
[13] M.L. Fredman, New bounds on the complexity of the shortest path problem, SIAM J. Comput. 5 (1976) 83–89.
[14] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615.
[15] A.M. Frieze, G.R. Grimmett, The shortest-path problem for graphs with random arc-lengths, Discrete Appl. Math. 10 (1985) 57–77.
[16] A.V. Goldberg, T. Radzik, A heuristic improvement of the Bellman–Ford algorithm, AMLETS: Appl. Math. Lett. 6 (1993) 3–6.
[17] Y. Han, Improved algorithm for all pairs shortest paths, Inform. Process. Lett. 91 (2004) 245–250.
[18] Y. Han, An $\mathcal{O}(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest paths, in: Proceedings of the 14th Conference on Annual European Symposium, vol. 14, ESA'06, Springer-Verlag, London, UK, 2006, pp. 411–417.
[19] Y. Han, T. Takaoka, An $\mathcal{O}(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths, in: F. Fomin, P. Kaski (Eds.), Algorithm Theory, SWAT 2012, in: Lecture Notes in Computer Science, vol. 7357, Springer, Berlin, Heidelberg, 2012, pp. 131–141.
[20] R. Hassin, E. Zemel, On shortest paths in graphs with random weights, Math. Oper. Res. 10 (1985) 557–564.
[21] H. Jakobsson, Mixed-approach algorithms for transitive closure, in: Proc. of the Tenth ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, 1991, pp. 199–205.
[22] D.B. Johnson, A note on Dijkstra's shortest path algorithm, J. ACM 20 (1973) 385–388.
[23] D.B. Johnson, Efficient algorithms for shortest paths in sparse networks, J. ACM 24 (1977) 1–13.
[24] D.R. Karger, D. Koller, S.J. Phillips, Finding the hidden path: Time bounds for all-pairs shortest paths, SIAM J. Comput. 22 (1993) 1199–1217.
[25] P. Klein, S. Mozes, O. Weimann, Shortest paths in directed planar graphs with negative lengths: A linear-space $\mathcal{O}(n\log^2 n)$-time algorithm, ACM Trans. Algorithms 6 (2010) 1–18, article No. 30.
[26] P. Klein, S. Rao, M. Rauch, S. Subramanian, Faster shortest-path algorithms for planar graphs, in: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing, STOC'94, ACM, New York, NY, USA, 1994, pp. 27–37.
[27] R.J. Lipton, D.J. Rose, R.E. Tarjan, Generalized nested dissection, SIAM J. Numer. Anal. 16 (1979) 346–358.
[28] M. Luby, P. Ragde, A bidirectional shortest-path algorithm with good average case behavior, Algorithmica 4 (1989) 551–567.
[29] C.C. McGeoch, A new all-pairs shortest-path algorithm, Algorithmica 13 (1995) 426–461.
[30] E.F. Moore, The shortest path through a maze, in: Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, 1959, pp. 285–292.
[31] H. Nagamochi, T. Ibaraki, A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, Algorithmica 7 (1992) 583–596.
[32] S. Pallottino, Shortest path methods: Complexity, interrelations and new propositions, Networks 14 (1984) 257–267.
[33] U. Pape, Implementation and efficiency of Moore algorithms for the shortest root problem, Math. Program. 7 (1974) 212–222.
[34] S. Pettie, A new approach to all-pairs shortest paths on real-weighted graphs, Theoret. Comput. Sci. 312 (2004) 47–74.
[35] S. Pettie, V. Ramachandran, A shortest path algorithm for real-weighted undirected graphs, SIAM J. Comput. 34 (2005) 1398–1431.
[36] T. Takaoka, A new upper bound on the complexity of the all pairs shortest path problem, Inform. Process. Lett. 43 (1992) 195–199.
[37] T. Takaoka, A faster algorithm for the all-pairs shortest path problem and its application, in: K.Y. Chwa, J.I. Munro (Eds.), COCOON, in: Lecture Notes in Computer Science, vol. 3106, Springer, 2004, pp. 278–289.
[38] T. Takaoka, An $\mathcal{O}(n^3\log\log n/\log n)$ time algorithm for the all-pairs shortest path problem, Inform. Process. Lett. 96 (2005) 155–161.
[39] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, J. ACM 46 (1999) 362–394.
[40] C.K. Yap, A hybrid algorithm for the shortest path between two nodes in the presence of few negative arcs, Inform. Process. Lett. 16 (1983) 181–182.
[41] U. Zwick, A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths, in: Proceedings of the 15th International Conference on Algorithms and Computation, ISAAC'04, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 921–932.