Theoretical Computer Science 411 (2010) 1013–1037

Contents lists available at ScienceDirect

Theoretical Computer Science

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

Partially dynamic efficient algorithms for distributed shortest paths☆

Serafino Cicerone, Gianlorenzo D'Angelo*, Gabriele Di Stefano, Daniele Frigioni

Dipartimento di Ingegneria Elettrica e dell'Informazione, Università degli studi dell'Aquila, I-67040 Monteluco di Roio, L'Aquila, Italy

ARTICLE INFO

Article history:
Received 12 January 2007
Received in revised form 29 September 2009
Accepted 7 November 2009
Communicated by G. Italiano

Keywords:
Dynamic algorithm
Distributed algorithm
Shortest paths

ABSTRACT

We study the dynamic version of the *distributed all-pairs shortest paths* problem. Most of the solutions given in the literature for this problem, either (i) work under the assumption that before dealing with an edge operation, the algorithm for the previous operation has to be terminated, that is, they are not able to update shortest paths *concurrently*, or (ii) concurrently update shortest paths, but their convergence can be very slow (possibly infinite) due to the looping and counting infinity phenomena. In this paper, we propose partially dynamic algorithms that are able to concurrently update shortest paths. We experimentally analyze the effectiveness and efficiency of our algorithms by comparing them against several implementations of the well-known Bellman–Ford algorithm.

1. Introduction

We consider the distributed *all-pairs shortest paths* problem in a network whose topology dynamically changes over the time, in the sense that communication links can change status during the lifetime of the network. This problem arises naturally in practical applications. For instance, the *OSPF* protocol, widely used in the Internet (e.g., see [16]), basically updates shortest paths after a network change by distributing the network topology to all processors and using centralized Dijkstra's algorithm for shortest paths on every node.

If the topology of a network is represented as a weighted graph, where nodes represent processors, edges represent links between processors, and edge weights represent costs of communication among processors, then the typical update operations on a dynamic network can be modelled as insertions and deletions of edges and edge weight changes. When arbitrary sequences of the above operations are allowed, we refer to the *fully dynamic problem*; if only *insert* and *weight decrease* (*delete* and *weight increase*) operations are allowed, then we refer to the *incremental* (*decremental*) problem. Incremental and decremental problems are usually called *partially dynamic*.

In many crucial routing applications the worst case complexity of the adopted protocols is never better than recomputing the shortest paths from scratch after each change to the network. Therefore, it is important to find efficient dynamic distributed algorithms for shortest paths, since the recomputation from scratch could result very expensive in practice.

The efficiency of a distributed algorithm is evaluated in terms of *message* and *space* complexity (e.g., see [2]). The *message complexity* is the total number of messages sent over the edges. The *space complexity* is the space usage per node.

In this paper we consider a dynamic network in which a change can occur while another change is under processing. A processor v could be affected by both these changes. As a consequence, v could be involved in the *concurrent* executions related to both the changes.

☆ An extended abstract of this work appeared in the proceedings of the International Conference on Computing: Theory and Application (ICCTA'07), March 5–7, Kolkata, India, 2007 [8]. Work partially supported by the Future and Emerging Technologies Unit of EC (IST priority — 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

* Corresponding author. Tel.: +39 0862 434471.
E-mail addresses: serafino.cicerone@univaq.it (S. Cicerone), gianlorenzo.dangelo@univaq.it (G. D'Angelo), gabriele.distefano@univaq.it (G. Di Stefano), daniele.frigioni@univaq.it (D. Frigioni).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.11.008

Previous works. Given a weighted graph G with n nodes and m edges, many solutions have been proposed in the literature to find and update shortest paths in the *sequential case* on graphs with non-negative real edge weights. The state of the art is that no efficient fully dynamic solution is known for general graphs that is faster than recomputing single-source shortest paths from scratch after each update. Actually, only *output bounded* fully dynamic solutions are known on general graphs [10,18]. In the case of all-pairs shortest paths the best fully dynamic solution has been proposed in [9] and works in $O(n^2 \log^3 n)$ amortized time per update.

A number of dynamic solutions for the shortest paths problem have been proposed in the literature also in the *distributed case* (see [4,7,11,13,17,19]). Some of these solutions rely on the classical Bellman–Ford method, whose distributed version has been originally introduced in the Arpanet [15]. This algorithm, and a number of its variations, has been shown to converge to the correct distances if the edge weights stabilize and all cycles have positive lengths (e.g., see [5]). However, the convergence can be very slow in the case of *weight increase* operations (possibly infinite). This is due to the well-known *looping* and *counting infinity* phenomena (see, e.g., [21]) and is a major drawback of the Bellman–Ford algorithm that is avoided in many protocols by broadcasting the whole topology of the network to all nodes [20]. Furthermore, if the network is asynchronous and static, the message complexity of the Bellman–Ford method can be exponential in the size of the network (see, [3]). In [11] Humblet proposes a variation of the Bellman–Ford algorithm, that has the same message and space complexity, and, under certain conditions, avoids the looping phenomenon thus converging in a finite number of steps.

In [13], an incremental solution has been proposed for the distributed all-pairs shortest paths problem, requiring $O(n \log(nW))$ amortized number of messages, and the difficulty of dealing with edge deletions has been addressed. Here, W is the largest positive *integer* edge weight. In [4], a general technique is proposed that allows us to update the all-pairs shortest paths in a distributed network in $\Theta(n)$ amortized number of messages, by using $O(n^2)$ space per node. In [19], algorithms are given for both finding and updating shortest paths distributively. In particular, the authors propose a distributed algorithm for finding single-source shortest paths (all-pairs shortest paths) of a network with positive real edge weights requiring $\Theta(n^2)$ ($O(n^3)$) messages and $O(n)$ space per node. Furthermore, they propose a distributed incremental algorithm requiring $O(n^2)$ messages for updating all-pairs shortest paths. Finally, they give fully dynamic algorithms for single-source (all-pairs) shortest paths that work in $O(n^2)$ ($O(n^3)$) messages, and show that, in the worst case, updating shortest paths is as difficult as computing shortest paths.

In [7] a solution for the fully dynamic distributed all-pairs shortest paths problem is presented whose message complexity is evaluated in terms of *output complexity* (see [10,18]). Output complexity allows us to evaluate the cost of dynamic algorithms in terms of the *intrinsic cost* of the problem on hand, i.e., in terms of the number of updates to the output information of the problem that are needed after any input change. The algorithm in [7] is able to update only the distances and the shortest paths that actually change after an edge modification σ. It requires in the worst case $O(maxdeg \cdot \Delta_\sigma)$ messages per operation and $O(n)$ space per node. Here, *maxdeg* is the maximum degree of the nodes in the network and Δ_σ is the number of pairs of nodes affected by σ. On one hand, if $\Delta_\sigma = o(n^2)$, then these bounds compare favorably with respect to those in [19]. On the other hand, the algorithm for *weight increase* operations is not robust because it works in three phases and requires that a phase is terminated before the execution of the subsequent one.

Summarizing, we can conclude that most of the algorithms known in the literature falls in one of the following two categories:

- algorithms which are not able to *concurrently* update shortest paths when multiple edge changes occur in the network, as those in [4,7,13,19]. In particular, algorithms that work under the assumption that before dealing with an edge operation, the algorithm for the previous operation has to be terminated. This is a limitation in real networks, where changes can occur in an unpredictable way;
- algorithms which are able to *concurrently* update shortest paths as those in [11,15], but (i) either they suffer of the looping and count to infinity phenomena, or (ii) their convergence can be very slow in the case of *weight increase* operations (possibly infinite).

Results of the paper. In this paper we provide a *decremental* and an *incremental* solution that are able to concurrently update shortest paths. The details of our contribution can be summarized as follows:

1. We propose a new decremental algorithm that is robust since it works in one phase (thus avoiding the main drawback of [7]). Furthermore, it is able to *concurrently* update shortest paths in the case of multiple *weight increase* and *delete* operations. The algorithm requires $O(maxdeg \cdot n)$ space per node and can suffer of the looping phenomenon. However, our solution has been shown to be experimentally efficient when compared with two different implementations of the classical Bellman–Ford method.
2. We propose an extension of the incremental algorithm given in [7] for *weight decrease* and *insert* operations that works also in the *concurrent* case, within the same bounds of [7], that is $O(maxdeg \cdot \Delta)$ messages per operation and $O(n)$ space per node. Here, Δ is the number of nodes affected by a set of *weight decrease*/*insert* operations. This is only a factor *maxdeg* far from the optimal incremental solution. Besides being theoretically efficient, this algorithm has been shown to be also experimentally fast.

2. Preliminaries

We consider a network made of processors linked through communication channels. Each processor can send messages only to its neighbors. We assume that messages are delivered to their destination within a finite delay but they might be delivered out of order. We consider an asynchronous system, that is, a sender of a message does not wait for the receiver to be ready to receive the message. There is no shared memory, that is, each processor has its own storage system and the other processors cannot access it.

We represent the network by an undirected weighted graph $G = (V, E, w)$, where V is a finite set of n nodes, one for each processor; E is a finite set of m edges, one for each communication channel; and w is a weight function $w : E \to \mathbb{R}^+$. If $v \in V$, $N(v)$ denotes the set of neighbors of v and $\deg(v)$ the degree of v. The maximum degree of G is denoted by *maxdeg*.

An edge $e \in E$ that links the pair of nodes $u, v \in V$ is denoted as $u \to v$. A path in G between nodes u and v is denoted as $P = u \rightsquigarrow v$. We define the *length* of P as the number of edges of P and denote it by $\ell(P)$, and define the *weight* of P as the sum of the weights of the edges in P and denote it by $weight(P)$. A *shortest path* between nodes u and v is a path from u to v with the minimum weight. The *distance* between u and v is the weight of a shortest path from u to v, and is denoted as $d(u, v)$.

Given a pair of nodes $u, v \in V$, the *via* from u to v is the set of neighbors of u that belong to a shortest path from u to v. Formally:

$$via(u, v) \equiv \{z \in N(u) \mid d(u, v) = w(u, z) + d(z, v)\}.$$

Given a graph $G = (V, E, w)$, we suppose that k operations $\sigma_1, \sigma_2, \ldots, \sigma_k$ are performed on edges $(x_i, y_i) \in E$, $i \in \{1, 2, \ldots, k\}$, at times $t_1, t_2, \ldots, t_k$, respectively. The operation σ_i modifies the weight $w(x_i, y_i)$ by a quantity $\epsilon_i > 0$, $i \in \{1, 2, \ldots, k\}$. Without loss of generality, we assume that $t_1 < t_2 < \cdots < t_k$.

Assuming $G \equiv G^0$, we denote as G^i the graph obtained at time t_i by applying the edge modification σ_i. Notations $d^i()$ and $via^i()$ are used to denote the distance and the via over G^i, $0 \leq i \leq k$, respectively.

Asynchronous model. Given an asynchronous system, the model summarized below is based on that proposed in [2]. The *state* of a processor v is the content of the data structure at node v. The *network state* is the set of states of all the processors in the network plus the network topology and the edge weights. An *event* is the reception of a message by a processor or a change to the network state. When a processor p sends a message m to a processor q, m is stored in a buffer in q. When q reads m from its buffer and processes it, the event "reception of m" occurs.

An *execution* is an alternate sequence (possibly infinite) of network states and events. A non-negative real number is associated to each event, the *time* at which that event occurs. The time is a *global* parameter and is not accessible to the processors of the network. The times must be nondecreasing and must increase without bound if the execution is infinite. Events are ordered according to the times at which they occur. Several events can happen at the same time as long as they do not occur at the same processor. This implies that the times related to a single processor are strictly increasing.

Concurrent executions. In this paper we consider a dynamic network in which a change can occur while another change is under processing. A processor v could be affected by both these changes. As a consequence, v could be involved in the executions related to both the changes. Hence, according to the asynchronous model we need to define the notion of *concurrent* executions as follows.

Let us consider an algorithm A that maintains shortest paths on G after a weight change operation. Given two operations σ_i and σ_j we denote as:

- t_i and t_j the times at which σ_i and σ_j occur, respectively.
- $\mathcal{A}_i$ ($\mathcal{A}_j$) the execution of A related to σ_i (σ_j).
- $t_{\mathcal{A}_i}$ the time when $\mathcal{A}_i$ terminates.

If $t_i < t_j$ and $t_{\mathcal{A}_i} \geq t_j$, then $\mathcal{A}_i$ and $\mathcal{A}_j$ are *concurrent*, otherwise they are *sequential*.

3. The decremental algorithm

In this section we describe our new decremental solution for the concurrent update of distributed all-pairs shortest paths in the case of multiple operations. We consider the algorithm to handle k *weight increase* operations $\sigma_1, \sigma_2, \ldots, \sigma_k$, since the extension to *delete* operations is straightforward. In fact, deleting an edge (x, y) is equivalent to increase $w(x, y)$ to $+\infty$.

Data structures. A node knows the identity of each node of the graph, the identity of all its neighbors and the weight of the edges incident to it.

The information on the shortest paths in G are stored in a data structure called *routing table* RT distributed over all nodes. Each node v maintains its own routing table $\mathtt{RT}_v[\cdot]$; this table has one entry $\mathtt{RT}_v[s]$ for each $s \in V$. The entry $\mathtt{RT}_v[s]$ consists of two fields:

- $\mathtt{RT}_v[s].\mathtt{d}$, that stores the estimated distance between nodes v and s in G.
- $\mathtt{RT}_v[s].\mathtt{via} \equiv \{v_i \in N(v) \mid \mathtt{RT}_v[s].\mathtt{d} = w(v, v_i) + \mathtt{RT}_{v_i}[s].\mathtt{d}\}$, that stores the estimated *via* from v to s.

For the sake of simplicity, we write $\mathtt{d}[v, s]$ and $\mathtt{via}[v, s]$ instead of $\mathtt{RT}_v[s].\mathtt{d}$ and $\mathtt{RT}_v[s].\mathtt{via}$, respectively.

In what follows we denote as $\mathtt{d}_t[v, s]$ and $\mathtt{via}_t[v, s]$ the value of the data structures at time t; we simply write $\mathtt{d}[v, s]$ and $\mathtt{via}[v, s]$ when time is clear by the context.

Algorithm. The subsequent Fact 3.1 shows that there exist topological properties that allow the algorithm to propagate messages only among affected nodes. Such properties can be easily formulated on the basis of the following definitions.

- a node v is *white* with respect to s if v does not change both its distance and its via to s. Formally:
  - $d^0(v, s) = d^k(v, s)$ and
  - $via^0(v, s) \equiv via^k(v, s)$.
- a node v is *gray* with respect to s if v does not change its distance from s, but it changes its via to s. Formally:
  - $d^0(v, s) = d^k(v, s)$ and
  - $via^0(v, s) \not\equiv via^k(v, s)$.

  Notice that, in the case of weight increase operations, $via^0(v, s) \supsetneq via^k(v, s)$.
- a node v is *black* with respect to s if v changes its distance from s. Formally:
  - $d^0(v, s) \neq d^k(v, s)$.

  Notice that, in the case of weight increase operations, $d^0(v, s) < d^k(v, s)$.

Fact 3.1. *The following properties trivially hold:*

1. *If v is* gray *or* black *with respect to s, then there exists a node $u \in via^0(v, s)$ such that either u is* black *with respect to s or it is an endpoint x_i of a modified edge (x_i, y_i), $v \equiv y_i$;*
2. *If v is* black *with respect to s, then each node in $via^0(v, s)$ is either a* black *node with respect to s or an endpoint x_i of a modified edge (x_i, y_i), $v \equiv y_i$;*
3. *If v is* white *or* gray *with respect to s, then, each node z such that $v \in via^0(z, s)$ is* white *or* gray *with respect to s;*
4. *If v is* white *or* gray *with respect to s, then each node in $via^k(v, s)$ is* white *or* gray *with respect to s.*

Before the decremental algorithm starts, we assume that $\mathtt{d}_t[v, s]$ and $\mathtt{via}_t[v, s]$ are correct, for each $v, s \in V$ and for each $t < t_1$. The decremental algorithm starts at each t_i, $i \in \{1, 2, \ldots, k\}$. For instance, a *weight increase* operation on the edge (x_i, y_i) represents an event that is detected only by nodes x_i and y_i; as a consequence:

1. x_i sends the message *increase*$(x_i, s, \mathtt{d}_{t_i}[x_i, s])$ to y_i, for each $s \in V$;
2. y_i sends the message *increase*$(y_i, s, \mathtt{d}_{t_i}[y_i, s])$ to x_i, for each $s \in V$.

If the operation on edge (x_i, y_i) is a *delete* operation, then y_i (x_i, respectively) cannot receive any message by x_i (y_i). In this case, y_i simulates the reception of message *increase*$(x_i, s, \mathtt{d}_{t_i}[x_i, s])$, where $\mathtt{d}_{t_i}[x_i, s] = +\infty$, by x_i. Analogously, x_i simulates the reception of message *increase*$(y_i, s, \mathtt{d}_{t_i}[y_i, s])$, where $\mathtt{d}_{t_i}[y_i, s] = +\infty$, by y_i.

When x_i receives *increase*$(y_i, s, \mathtt{d}_{t_i}[y_i, s])$ by y_i, x_i executes procedure Increase (see Fig. 1). This procedure is responsible for checking if it is necessary to update $\mathtt{RT}_{x_i}[s]$ and, consequently, to propagate the decremental algorithm. The behavior of y_i (when y_i receives the message *increase*$(x_i, s, \mathtt{d}_{t_i}[x_i, s])$) is symmetric. At most one between x_i and y_i will propagate the decremental algorithm. In fact, at most one of the following conditions is true:

1. $x_i \in \mathtt{via}[y_i, s]$
2. $y_i \in \mathtt{via}[x_i, s]$.

If none of these conditions is true, then the tests performed by x_i and y_i at Lines 1, 15 and 23 are false and the algorithm simply stops its execution. In this section, we assume that $\mathtt{d}_{t_i}[s, x_i] \leq \mathtt{d}_{t_i}[s, y_i]$. Under this hypothesis, only Condition 2 above can be true. In the affirmative case y_i, if it is necessary, updates $\mathtt{RT}_{y_i}[s]$ at a certain time t and, in order to propagate the decremental algorithm, sends the message *increase*$(y_i, s, \mathtt{d}_t[y_i, s])$ to its neighbors.

Let us now analyze Procedure Increase with respect to a generic node v that receives the message *increase*$(u, s, \mathtt{d}_{\tilde{t}}[u, s])$, $\tilde{t} < t$, by a neighbor u. In order to this update $\mathtt{RT}_v[s]$, v may need to know the estimated distances of its neighbors from s, that is, $\mathtt{d}_t[v_i, s]$ for each $v_i \in N(v)$. Hence, v sends messages *get-dist*(v, s); when v_i receives such message, it performs procedure Dist (see Fig. 2).

Notice that, in our model, multiple *increase* messages received by a node are stored and processed in a certain order, while each message *get-dist* is processed immediately.

Now we provide an informal description of the algorithm. The purpose of this description is to give an intuition of the behavior of the algorithm; the formal proof of correctness is given in the next section. The description is focused on the execution of the algorithm by a generic node v with respect to a source s, and uses the scenario for node v depicted in Fig. 3 as a representative case.

Node v in Fig. 3 is *black* with respect to s since, according to Property 2 of Fact 3.1, each node in $via^0(v, s) \equiv \{u_1, u_2, u_3\}$ is *black*. As a consequence, v surely receives messages *increase*$(u_i, s, \mathtt{d}[u_i, s])$, $1 \leq i \leq 3$, in some order. This implies that v performs three times procedure Increase. In this procedure, Line 1 and 4 tests if v satisfies Property 1 and 2 of Fact 3.1 respectively. Hence, the first two executions simply perform phase reduce-via, while the third one performs reduce-via and build-table.

Let us suppose that the third execution is related to u_3. During the execution of build-table, node v sends the message *get-dist*(v, s) to each node $v_i \in N(v) \setminus \{u_3\}$. We assume that this message is received by v_i at time $\tilde{t}_{1,i}$. In this phase, let us

Event: node v receives the message $increase(u, s, \mathtt{d}[u, s])$ by u

Procedure INCREASE

1. **if** $u \in \mathtt{via}[v, s]$ **then**
2. **begin**
3. $\mathtt{via}[v, s] := \mathtt{via}[v, s] \setminus \{u\}$ Line 3: phase REDUCE-VIA
4. **if** $\mathtt{via}[v, s] \equiv \emptyset$ **then**
5. **begin** Lines 5-12: phase BUILD-TABLE
6. **for each** $v_i \in N(v)$ **do**
7. send $get\text{-}dist(v, s)$ to v_i
8. $\mathtt{d}[v, s] := \min_{v_i \in N(v)} \{w(v, v_i) + \mathtt{d}[v_i, s]\}$
9. $\mathtt{via}[v, s] := \{v_i \in N(v) \mid w(v, v_i) + \mathtt{d}[v_i, s] = \mathtt{d}[v, s]\}$
10. **for each** $v_i \in N(v)$ **do** Lines 10-11: phase PROPAGATE_1
11. send $increase(v, s, \mathtt{d}[v, s])$ to v_i
12. **end**
13. **end**
14. **else**
15. **if** $\mathtt{d}[v, s] > w(v, u) + \mathtt{d}[u, s]$ **then**
16. **begin** Lines 16-21: phase IMPROVE-TABLE
17. $\mathtt{d}[v, s] := w(v, u) + \mathtt{d}[u, s]$
18. $\mathtt{via}[v, s] := \{u\}$
19. **for each** $v_i \in N(v)$ **do** Lines 19-20: phase PROPAGATE_2
20. send $increase(v, s, \mathtt{d}[v, s])$ to v_i
21. **end**
22. **else**
23. **if** $\mathtt{d}[v, s] = w(v, u) + \mathtt{d}[u, s]$ **then**
24. $\mathtt{via}[v, s] := \mathtt{via}[v, s] \cup \{u\}$ Line 24: phase EXTEND-VIA

Fig. 1. The Increase procedure.

Event: node v receives the message $get\text{-}dist(u, s)$ by u

Procedure DIST

1. **if** ($\mathtt{via}[v, s] \equiv \{u\}$) **or** ($v$ is performing phase BUILD-TABLE or phase IMPROVE-TABLE of procedure INCREASE with respect to source s)
2. **then** send $+\infty$ to u
3. **else** send $\mathtt{d}[v, s]$ to u

Fig. 2. The Dist procedure.

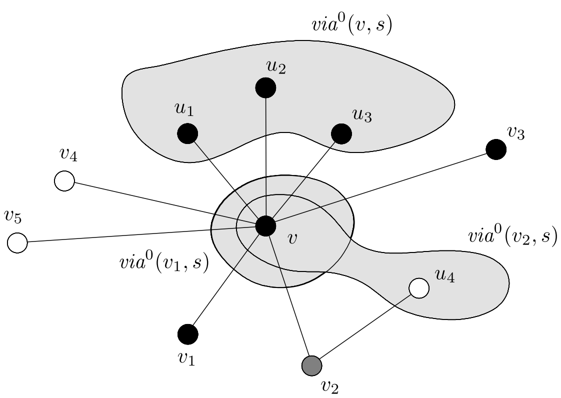

Fig. 3. The scenario used to describe the decremental algorithm.

assume that the following conditions hold for nodes v_1 and v_3, respectively:

(a) $\mathtt{via}_{\tilde{t}_{1,1}}[v_1, s] \equiv \{v\}$

(b) at time $\tilde{t}_{1,3}$, node v_3 is performing either BUILD-TABLE or IMPROVE-TABLE phases of procedure INCREASE with respect to source s.

According to these conditions and to test at line 1 of procedure DIST, nodes v_3 and v_1 send $+\infty$ to v.

By using the collected information, v performs the instructions

$$\mathtt{d}[v, s] := \min_{v_i \in N(v)} \{w(v, v_i) + \mathtt{d}[v_i, s]\}$$

and

$$\mathtt{via}[v, s] := \{v_i \in N(v) \mid w(v, v_i) + \mathtt{d}[v_i, s] = \mathtt{d}[v, s]\}.$$

Let us assume that now $\mathtt{via}_{\tilde{t}_2}[v, s] = \{v_2\}$. Notice that, since v has received partial information, the content of $\mathtt{RT}_v[s]$ at time $\tilde{t}_2$ could be not correct. Now, two relevant observations have to be remarked:

(i) since nodes v_1 and v_3 sent $+\infty$ to v, then v does not consider such nodes as possible new elements of $\mathtt{via}$.

(ii) in the subsequent Items 1 and 2 we show that nodes v_1 and v_3 will eventually send $\mathtt{d}[v_1, s]$ and $\mathtt{d}[v_3, s]$ to v.

The BUILD-TABLE phase of v is completed by the PROPAGATE_1 phase. In this phase v broadcast to $N(v)$ the message *increase*$(v, s, \mathtt{d}_{\tilde{t}_2}[v, s])$; it may seem useless to send the message to nodes u_i, $1 \le i \le 3$, (the old via of v) and to node v_2 (the new via of v). The former will be explained later (last paragraph of Item 1), while the latter is due to the fact that $v \in via^0(v_2, s)$, and hence v_2 has to perform the REDUCE-VIA phase.

Let us now analyze what happens to the nodes v_1, v_3, v_4 and v_5.

1. node v_1 receives message *increase*$(v, s, \mathtt{d}_{\tilde{t}_2}[v, s])$ at time $\tilde{t}_3 > \tilde{t}_2$, and it executes INCREASE. Since $\mathtt{via}_{\tilde{t}_{1,1}}[v_1, s] \equiv \mathtt{via}_{\tilde{t}_3}[v_1, s] \equiv \{v\}$, v_1 performs the BUILD-TABLE phase. At the end of this phase, at time $\tilde{t}_4 > \tilde{t}_3$, v_1 updates $\mathtt{RT}_{v_1}[s]$. Now, two major cases may occur:
   - v is in $\mathtt{via}_{\tilde{t}_4}[v_1, s]$;
   - v is not in $\mathtt{via}_{\tilde{t}_4}[v_1, s]$. This means that v_1 now uses a new via to s.

   In both cases, at the end of the BUILD-TABLE phase, v_1 broadcast the message *increase*$(v_1, s, \mathtt{d}_{\tilde{t}_4}[v_1, s])$ to $N(v_1)$, and hence to v also (see Item (ii) above).

   In the first case, v performs tests at lines 1, 15 and 23 of INCREASE. All such tests return false, and hence, node v terminates INCREASE without modifying its routing tables and without propagating the decremental algorithm.

   In the second case, one of the tests performed by v at lines 15 and 23 may return true. If test at line 15 returns true, then v has to perform the IMPROVE-TABLE phase to rebuild $\mathtt{RT}_v[s]$. If test at line 23 returns true, then v has to perform the EXTEND-VIA phase to add v_1 to $\mathtt{via}[v, s]$.

   Notice that the behavior of v after receiving message *increase*$(v_1, s, \mathtt{d}_{\tilde{t}_4}[v_1, s])$ is essentially the same of nodes u_i, $1 \le i \le 3$, after receiving message *increase*$(v, s, \mathtt{d}_{\tilde{t}_2}[v, s])$.
2. node v_3, once terminated the execution of phase BUILD-TABLE or phase IMPROVE-TABLE of procedure INCREASE with respect to source s (see Item 3 above), executes phase PROPAGATE_1 or phase PROPAGATE_2. This implies that node v restarts INCREASE now using the current estimated distance from v_3 to s (see Item (ii) above).
3. since nodes v_4 and v_5 are *white*, once received message *increase*$(v, s, \mathtt{d}_{\tilde{t}_2}[v, s])$ they perform tests at lines 1, 15 and 23 of procedure INCREASE. All such tests return false, and hence nodes v_4 and v_5 terminate INCREASE without modifying their routing tables and without propagating the decremental algorithm.

Correctness analysis.

Before the algorithm starts, we assume that the routing tables are correct, that is for each node v, for each source s, and for each time $t \le t_1$ the information stored by v in its routing table are:

$$\mathtt{d}_t[v, s] = d^0(v, s)$$
$$\mathtt{via}_t[v, s] = via^0(v, s)$$

Lemma 3.2. *For each node v, for each source s and for each time t the inequality $\mathtt{d}_t[v, s] \ge d^0(v, s)$ holds.*

Proof. By contradiction, let us suppose that v is the first node to fail to update its routing table, that is, there exists a minimum time t_v such that $\mathtt{d}_{t_v}[v, s] < d^0(v, s)$. v updates its routing table as a consequence of the reception of a message *increase*$(z, s, d_{t_z}[z, s])$, with $t_z < t_v$, from a node $z \in N(v)$. The updating is performed either in BUILD-TABLE or in IMPROVE-TABLE phase. In any case, $\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s]$. Since v is the first node to fail, then $\mathtt{d}_{t_z}[z, s] \ge d^0(z, s)$. Thus,

$$d^0(v, s) > \mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] \ge w(v, z) + d^0(z, s),$$

a contradiction. $\square$

Lemma 3.3. *For each source s, a node v is* black *with respect to s if and only if it performs phase* PROPAGATE_1*. If v performs phase* PROPAGATE_2*, then v is* black *with respect to s.*

Proof. We can summarize the thesis as follows:

v is *black* with respect to $s \Leftarrow v$ performs phase PROPAGATE_1 *or* PROPAGATE_2
v is *black* with respect to $s \Rightarrow v$ performs phase PROPAGATE_1

"$\Leftarrow$": by contradiction, we show that if v is *nonblack* with respect to s, then it does not perform neither phase PROPAGATE_1 nor phase PROPAGATE_2.

The proof is by induction on the number:

$$L_s(v) = \max\{\ell(P) \mid P \equiv v \rightsquigarrow s \text{ is a shortest path in } G^0\}.$$

Inductive basis ($L_s(v) = 0$): $L_s(v) = 0$ if and only if $v \equiv s$. For each time $t \le t_1$, s is *white* with respect to s and in the routing table stored by s we have:

$$\mathtt{d}_t[s, s] = d^0(s, s) = 0$$
$$\mathtt{via}_t[s, s] \equiv via^0(s, s) \equiv \emptyset.$$

Node s can perform phases PROPAGATE_1 or PROPAGATE_2 only after receiving an *increase* message. Let t_m be the time when s receives the first *increase* message $m = increase(z, s, \mathtt{d}_{t_z}[z, s])$ where z is a node in $N(s)$ and t_z is a time such that $t_z < t_m$.

Since $\mathtt{via}_{t_m}[s, s] \equiv \emptyset$, the condition in Line 1 of Procedure Increase is false and hence s does not perform phase PROPAGATE_1.

By Lemma 3.2, $\mathtt{d}_{t_z}[z, s] \ge d^0(z, s)$, hence

$$w(s, z) + \mathtt{d}_{t_z}[z, s] \ge w(s, z) + d^0(z, s) > \mathtt{d}_{t_m}[s, s] = 0.$$

Thus the conditions in lines 15 and 23 of Procedure Increase are false and s does not perform phase PROPAGATE_2.

In any case, s does not change $\mathtt{RT}_s[s]$ thus, if s receives further *increase* messages, the same arguments can be used to prove the statement.

Inductive step: by inductive hypothesis each *nonblack* nodes v with respect to s such that $L_s(v) \le l - 1$ does not perform neither phase PROPAGATE_1 nor phase PROPAGATE_2; this implies that such nonblack nodes do not send *increase* messages.

Let v be a *nonblack* node such that $L_s(v) = l$. Node v can perform phases PROPAGATE_1 or PROPAGATE_2 only after receiving an *increase* message. Let t_m be the time when v receives the first *increase* message $m = increase(u, s, \mathtt{d}_{t_u}[u, s])$, $t_u < t_m$. For each time $t \le t_m$ we have $\mathtt{d}_t[v, s] = d^0(v, s)$ and $\mathtt{via}_t[v, s] = via^0(v, s)$.

If $u \notin \mathtt{via}_{t_m}[v, s]$, then the condition in line 1 of Procedure Increase is false and v does not perform phase PROPAGATE_1. By Lemma 3.2, $\mathtt{d}_{t_u}[u, s] \ge d^0(u, s)$, and hence

$$w(v, u) + \mathtt{d}_{t_u}[u, s] \ge w(v, u) + d^0(u, s) \ge d^0(v, s) = \mathtt{d}_{t_m}[v, s].$$

Thus the condition in Line 15 of Procedure Increase is false and v does not perform phase PROPAGATE_2.

If $u \in \mathtt{via}_{t_m}[v, s]$, then the condition in Line 1 of procedure Increase is true and v does not perform phase PROPAGATE_2. By inductive hypothesis, *nonblack* nodes z such that $L_s(z) \le l - 1$ do not send *increase* messages; since $u \in \mathtt{via}_{t_m}[v, s] \equiv via^0(v, s)$, then $L_s(u) \le l-1$, and hence u is *black* with respect to s. u black and Property 1 of Fact 3.1 imply that v is *gray* with respect to s. By definition of gray nodes, it follows that there exists a *nonblack* node $z \in via^0(v, s)$ such that $L_s(z) \le l - 1$. By inductive hypothesis, z did not send any *increase* message, then $z \in \mathtt{via}_{t_m}[v, s]$. Hence, $\mathtt{via}_{t_m}[v, s]$ is not empty and the condition in Line 4 of Procedure Increase is false. Thus, v does not perform phase PROPAGATE_1.

In any case, v does not change the value of $\mathtt{d}[v, s]$, thus, if v receives further *increase* messages, the same arguments can be used to prove the statement.

"$\Rightarrow$": We first recall that y_i, $i \in \{1, 2, \ldots, k\}$, is used to denote a *black* node that is an endpoint of the modified edge (x_i, y_i) and $d^0(x_i, s) \le d^0(y_i, s)$. Then, we introduce the following definitions with respect to G^0:

$v \overset{y}{\rightsquigarrow} s$: shortest path from v to s containing y
$\mathcal{P}_s(v, y) = \{P = v \rightsquigarrow y \mid \exists P' = v \overset{y}{\rightsquigarrow} s : P \subseteq P' \text{ and } \forall v_i \in P\ v_i \text{ is } \textit{black} \text{ wrt } s\}$
$$L_s(v, y) = \max_{P \in \mathcal{P}_s(v,y)} \ell(P)$$
$$L_s(v) = \max_{y_i, i \in \{1,2,\ldots,k\}} L_s(v, y_i).$$

The proof is by induction on $L_s(v)$.

Inductive basis ($L_s(v) = 0$): Let v be a *black* node such that $L_s(v) = 0$. This implies that $v \equiv y_i$, for some $i \in \{1, 2, \ldots, k\}$. Moreover, $L_s(y_i) = 0$ if and only if $via^0(y_i, s) \equiv \{x_{i_1}, x_{i_2}, \ldots, x_{i_h}\}$, where $\{i_1, i_2, \ldots, i_h\} \subseteq \{1, 2, \ldots, k\}$, $i_1 \le i_2 \le \cdots \le i_h$. Let y_i be a node satisfying such condition.

y_i is involved in the *weight increase* operations $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_h}$, thus it performs Procedure Increase h times, one for each operation σ_{i_j}, $1 \le j \le h$. Each operation σ_{i_j} starts a local execution of Procedure Increase that removes the node x_{i_j} from $\mathtt{via}[y_i, s]$ (see phase REDUCE-VIA).

Let t be a time such that $t_1 \le t \le t_{i_h}$, no nodes are added to $\mathtt{via}_t[y_i, s]$. In fact, each node $z \in N(y_i)$ that does not belong to $via^0(y_i, s)$ satisfies $d^0(y_i, s) < w(y_i, z) + d^0(z, s)$. Let $\tilde{t}$ be a time such that $t_1 \le \tilde{t} \le t_{i_h}$, by Lemma 3.2, $\mathtt{d}_{\tilde{t}}[z, s] \ge d^0(z, s)$, then $d^0(y_i, s) < w(y_i, z) + \mathtt{d}_{\tilde{t}}[z, s]$. At time t, v has not yet performed phase PROPAGATE_1, then $\mathtt{d}_t[y_i, s] = d^0(y_i, s)$. It follows that $\mathtt{d}_t[y_i, s] < w(y_i, z) + \mathtt{d}_{\tilde{t}}[z, s]$, hence conditions at Lines 15 and 23 of Procedure INCREASE are false.

Thus, $\mathtt{via}_{t_{i_h}}[y_i, s]$ is empty. Then, at time t_{i_h}, the condition at Line 4 is true and y_i performs phase PROPAGATE_1.

Inductive step: the inductive hypothesis is: each *black* node u, such that $L_s(u) \le l - 1$ performs PROPAGATE_1.

Let v be a *black* node with respect to s such that $L_s(v) = l$. By Property 2 of Fact 3.1, since v is *black*, each node u in $via^0(v, s)$ is either (i) a *black* node with respect to s or (ii) an endpoint of a modified edge (u, v). In case (i), black nodes satisfy $L_s(u) \le l - 1$, then, by inductive hypothesis, they send *increase* messages to v. In case (ii), u sends an increase message to v as a consequence of the operation occurred on (u, v).

As a consequence of these messages, v performs $\left|via^0(v, s)\right|$ times the Procedure INCREASE and removes all the elements of $via^0(v, s)$. By the same arguments used in the inductive basis, v does not add elements in $\mathtt{via}[v, s]$.

Thus, at the end of the $\left|via^0(v, s)\right|$ local executions of Procedure INCREASE, the set $\mathtt{via}[v, s]$ is empty. Then the condition in line 4 is true and v performs phase PROPAGATE_1. □

Corollary 3.4. *For each node s, for each* nonblack *node v with respect to s and for each time t, the following equality holds:* $\mathtt{d}_t[v, s] = d^0(v, s) = d^k(v, s)$.

Proof. For each time $t \le t_1$ the thesis is true. Furthermore, by Lemma 3.3, if v is a *nonblack* nodes with respect to s, it does not perform neither phase PROPAGATE_1 nor phase PROPAGATE_2. Hence v does not perform the instructions that change the value of $\mathtt{d}_{t_1}[v, s]$. □

Lemma 3.5. *For each node s, if a node v is* white *with respect to s, then v does not perform neither phase* REDUCE-VIA *nor phase* BUILD-TABLE.

Proof. By contradiction, let us suppose that a node v white with respect to s performs phase REDUCE-VIA or phase BUILD-TABLE. Let t be the minimum time such that v performs phase REDUCE-VIA or phase BUILD-TABLE after receiving the message *increase*$(u, s, \mathtt{d}[u, s])$ sent by a node u. This implies $u \in \mathtt{via}_t[v, s]$. Since u delivers this message, then it performed phase PROPAGATE_1 or phase PROPAGATE_2; hence, by Lemma 3.3, u is a *black* node with respect to s.

In the remainder, we show that $\mathtt{via}_t[v, s] \equiv via^0(v, s)$; this is a contradiction with respect to Property 4 of Fact 3.1 because u is a *black* node with respect to s.

Recall that $\mathtt{via}_{t_1}[v, s] \equiv via^0(v, s)$. By contradiction hypothesis, v did not perform phase REDUCE-VIA before time t. Moreover, by Lemma 3.3, v never performs neither phase PROPAGATE_1 nor phase PROPAGATE_2. Hence, at time $\tilde{t}$, $t_1 \le \tilde{t} \le t$, $\mathtt{via}_{\tilde{t}}[v, s]$ can be modified only by performing phase EXTEND-VIA. Hence, to get the above contradiction, it is sufficient to show that v does not perform EXTEND-VIA at time $\tilde{t}$ as a consequence of *increase* messages from nodes in $via^0(v, s)$.

Let $z \in N(v)$ be a node such that $z \notin via^0(v, s)$, and let $\bar{t}$ be a time such that $t_1 \le \bar{t} \le \tilde{t}$. We get the following relationships:

$$\begin{aligned} \mathtt{d}_{\tilde{t}}[v, s] &= d^0(v, s) && \text{(by Corollary 3.4)} \\ &< w(v, z) + d^0(z, s) && (\text{since } z \notin via^0(v, s)) \\ &\le w(v, z) + \mathtt{d}_{\bar{t}}[z, s] && \text{(by Lemma 3.2)}. \end{aligned}$$

It follows that $\mathtt{d}_{\tilde{t}}[v, s] < w(v, z) + \mathtt{d}_{\bar{t}}[z, s]$. Hence, at time $\tilde{t}$, condition at Line 23 of Procedure INCREASE is false and then v does not performs phase EXTEND-VIA as a consequence of *increase* messages from z. Thus, $\mathtt{via}_t[v, s] \equiv via^0(v, s)$. □

In the remainder we will use the further following notations for each pair of nodes v and s:

- $Exe_f(v, s)$ the first local execution by v of phase BUILD-TABLE with respect to s;
- $t_f(v, s)$ denotes the time when $Exe_f(v, s)$ updates $\mathrm{RT}_v[s]$.
- $Exe_l(v, s)$ denotes the last local execution by v of phases BUILD-TABLE or IMPROVE-TABLE with respect to s;
- $t_l(v, s)$ denotes the time when $Exe_l(v, s)$ updates $\mathrm{RT}_v[s]$;

Lemma 3.6. *For each source s, for each* black *node v with respect to s, $t_f(v, s)$ and $t_l(v, s)$ are defined.*

Proof. Let v be a *black* node v with respect to s. To prove that $t_f(v, s)$ is defined it is sufficient to observe that, by Lemma 3.3, v performs phase PROPAGATE_1. To prove that $t_f(v, s)$ is defined we show that v performs finitely many executions of BUILD-TABLE and IMPROVE-TABLE phases with respect to source s. To this aim, we introduce the following notations:

- $\mathcal{P}(v, s)$ is a set of walks (i.e. paths which can contain nondistinct nodes) P from v to s with the following property:
  - if P contains a cycle C and u is a node in C such that the subwalk P_u from u to s of P has minimum weight, then C is contained ℓ times in P, where ℓ is a finite number such that $\ell \cdot w(C) < w(P_u)$.
- Let $P_1 \equiv v_1 \to v_2 \to \cdots \to v_{l_1}$ and $P_2 \equiv u_1 \to u_2 \to \cdots \to v_{l_2}$ be two walks in $\mathcal{P}(v, s)$. $P_1 \equiv P_2$ if and only if:
  - $l_1 = l_2$ and
  - $v_i \equiv u_i$, $1 \le i \le l_1$, and
  - $w(v_i, v_{i+1}) = w(u_i, u_{i+1})$, $1 \le i \le l_1 - 1$.
- If $P_1 \not\equiv P_2$ we say that P_1 and P_2 are different.

Note that, the sets $\mathcal{P}(v, s)$ have finite sizes (see subsequent Example 3.11 for a visualization).

We associate each local execution of BUILD-TABLE or IMPROVE-TABLE phases to a walk in $\mathcal{P}(v, s)$. In particular, if a node v performs a local execution $Exe(v, s)$ of BUILD-TABLE or IMPROVE-TABLE phase as a consequence of an *increase* message m, then the walk associated to $Exe(v, s)$ is $x_i \to y_i \equiv u_1 \to u_2 \rightsquigarrow u_j \equiv v, j \geq 1$. Informally, m is due to the weight increase operation σ_i on the edge (x_i, y_i), and to the propagation of the decremental algorithm through nodes $u_1, u_2, \ldots, u_j$. In what follows, we show that two different local executions $Exe'(v, s)$ and $Exe''(v, s)$ of BUILD-TABLE or IMPROVE-TABLE phases are associated to two different walks in $\mathcal{P}(v, s)$. Since the number of such walks is bounded, then the number of local executions is bounded.

Let us now denote as $Exe_t(v, s, u)$ a local execution of BUILD-TABLE or IMPROVE-TABLE phases performed by v with respect to s at time t as a consequence of a *increase* message sent by $u \in N(v)$.

By contradiction, let v be the first node such that two local executions $Exe_{t_1}(v, s, u_1)$ and $Exe_{t_2}(v, s, u_2)$, $t_1 \neq t_2$, of BUILD-TABLE or IMPROVE-TABLE phases are associated to $P_{t_1}(v, s), P_{t_2}(v, s) \in \mathcal{P}(v, s)$ such that $P_{t_1}(v, s) \equiv P_{t_2}(v, s)$.

If $u_1 \neq u_2$, then it is straightforward to see that $P_{t_1}(v, s)$ and $P_{t_2}(v, s)$ are different. Hence, let us now consider local executions of BUILD-TABLE or IMPROVE-TABLE phases performed by v as a consequence of messages sent by the same node $u \in N(v)$.

Let us suppose that v performs $Exe_{t_1}(v, s, u)$ and $Exe_{t_2}(v, s, u)$ as a consequence of two different *increase* messages, m_1 and m_2, sent by u. Node u sends m_1 and m_2 as a consequence of one of the following:

1. a weight increase operation on edge $u \to v$;
2. a local execution performed by u of BUILD-TABLE or IMPROVE-TABLE phases with respect to source s.

If m_1 or m_2 is related to a weight increase operation on edge $u \to v$, then $P_{t_1}(v, s)$ and $P_{t_2}(v, s)$ are different since the weight of the edge $u \to v$ in $P_{t_1}(v, s)$ and $P_{t_2}(v, s)$ is different. Hence, let us suppose that both m_1 and m_2 are sent during the local executions of BUILD-TABLE or IMPROVE-TABLE phases $Exe_{t'_1}(u, s, w_1)$ and $Exe_{t'_2}(u, s, w_2)$ performed by u as a consequence of two messages sent by nodes $w_1 \in N(u)$ and $w_2 \in N(u)$, respectively. According to the contradiction hypothesis, the walks $P_{t'_1}(u, s)$ and $P_{t'_2}(u, s)$ associated to $Exe_{t'_1}(u, s, w_1)$ and $Exe_{t'_2}(u, s, w_2)$, respectively, are different. Hence, $P_{t_1}(v, s) \equiv P_{t'_1}(u, s) \to v$ and $P_{t_2}(v, s) \equiv P_{t'_2}(u, s) \to v$ are different.

Hence, each local execution of phases BUILD-TABLE and IMPROVE-TABLE is associated to a different walk in $\mathcal{P}(v, s)$. □

Lemma 3.7. *For each source s, for each* black *node v with respect to s and for each time $t \geq t_f(v, s)$, $\mathtt{d}_t[v, s] > d^0(v, s)$.*

Proof. By contradiction let us suppose that v is the first *black* node with respect to s failing to update its routing table. Let $t_v \geq t_f(v, s)$ be the smallest time such that $\mathtt{d}_{t_v}[v, s] \leq d^0(v, s)$. Such hypothesis along with Lemma 3.2 imply that $\mathtt{d}_{t_v}[v, s] = d^0(v, s)$.

Let z be a node in $N(v)$ that belongs to $\mathtt{via}_{t_v}[v, s]$. In this case, $\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] = d^0(v, s)$, where $t_z < t_v$. Now we analyze different cases according to $via^0(v, s)$. For each of such cases we obtain a contradiction.

- if $z \notin via^0(v, s)$ then $w(v, z) + d^0(z, s) > d^0(v, s)$. Furthermore, by Lemma 3.2, $\mathtt{d}_{t_z}[z, s] \geq d^0(z, s)$. Thus, we have $\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] > d^0(v, s)$;
- if $z \in via^0(v, s)$, by Property 2 of Fact 3.1 z is either a *black* node with respect to s or an endpoint of a modified edge (v, z). In the first case, since v is the first node to fail and $t_z \geq t_f(z, s)$, $\mathtt{d}_{t_z}[z, s] > d^0(z, s)$. Thus, $\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] > w(v, z) + d^0(z, s) \geq d^0(v, s)$. In the second case, by Lemma 3.2, we have $\mathtt{d}_{t_v}[v, s] \geq w(v, z) + d^0(z, s)$ and as a consequence of the increment of $w(z, v)$, $w(v, z) + d^0(z, s) > d^0(v, s)$.

In both cases we obtained a contradiction to the hypothesis $\mathtt{d}_{t_v}[v, s] = d^0(v, s)$. □

Corollary 3.8. *If a node v performs phase* EXTEND-VIA *with respect to source s, then v is* black *with respect to s.*

Proof. We prove that if v is a *nonblack* node with respect to s, then it does not perform phase EXTEND-VIA with respect to source s.

Let v be a *nonblack* node with respect to s that receives the message $m = increase(u, s, \mathtt{d}_{t_u}[u, s])$ at time t_m, where $t_f(u, s) \leq t_u < t_m$. By Lemma 3.3, u is *black* with respect to s. Let us consider the local execution at node v related to message m. Since u is *black*, by Lemma 3.7, $\mathtt{d}_{t_u}[u, s] > d^0(u, s)$. Hence

$$w(v, u) + \mathtt{d}[u, s] > w(v, u) + d^0(u, s) \geq d^0(v, s).$$

By Corollary 3.4, $d^0(v, s) = \mathtt{d}_{t_m}[v, s]$. Thus, $w(v, u) + \mathtt{d}_{t_u}[u, s] > \mathtt{d}_{t_m}[v, s]$ and then the condition in Line 23 of Procedure INCREASE is false. □

Lemma 3.9. *For each pair of nodes v and s and for each time $t \geq t_l(v, s)$, $\mathtt{d}_t[v, s] \geq d^k(v, s)$.*

Proof. By contradiction, let us suppose that v is the first node failing to update its routing table, and let $t_v \geq t_l(v, s)$ be the smallest time such that

$$\mathtt{d}_{t_v}[v, s] < d^k(v, s). \tag{1}$$

For each $z \in \mathtt{via}_{t_v}[v, s]$:

$$\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] < d^k(v, s), \quad t_z < t_v.$$

If there exists a node $z \in \mathtt{via}_{t_v}[v, s]$ which is *nonblack* with respect to s, then, by Corollary 3.4, $\mathtt{d}_{t_z}[z, s] = d^k(z, s)$. Hence, $\mathtt{d}_{t_v}[v, s] = w(v, z) + d^k(z, s) \geq d^k(v, s)$, a contradiction with respect to Eq. (1).

In what follows, we suppose that each node $z \in \mathtt{via}_{t_v}[v, s]$ is *black* with respect to s. By Lemma 3.6, $t_l(z, s)$ is defined. If there exists a node $z \in \mathtt{via}_{t_v}[v, s]$ such that $t_z \geq t_l(z, s)$, since v is the first node to fail, then $\mathtt{d}_{t_z}[z, s] \geq d^k(z, s)$. Thus,

$$\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] \geq w(v, z) + d^k(z, s) \geq d^k(v, s),$$

and, again, we obtain a contradiction with respect to Eq. (1). Hence, let us suppose that $t_z < t_l(z, s)$, for each node $z \in \mathtt{via}_{t_v}[v, s]$.

Let m'_z and m''_z be the messages sent to v by z at times t_z and $t_l(z, s)$ respectively and let $t_{m''_z}$ and $t_{m''_z}$ be the times when they are received by v. Since two events cannot occur on one processor at the same time, then $t_{m'_z} \neq t_{m''_z}$.

If there exists a node $z \in \mathtt{via}_{t_v}[v, s]$ such that $t_{m'_z} > t_{m''_z}$, since v is the first node to fail, we have $\mathtt{d}_{t_l(z,s)}[z, s] \geq d^k(z, s)$. Thus,

$$\mathtt{d}_{\tilde{t}}[v, s] = w(v, z) + \mathtt{d}_{t_l(z,s)}[z, s] \geq w(v, z) + d^k(z, s) \geq d^k(v, s)$$

where $\tilde{t}$ is the time when v updates $\mathtt{d}[v, s]$ as a consequence of m''_z. Note that $t_{m''_z} < \tilde{t} < t_{m'_z} < t_v$. After $t_l(z, s)$, $\mathtt{d}[z, s]$ is no longer updated. This implies that $\mathtt{d}_t[v, s] \geq d^k(v, s)$, for each $t \geq \tilde{t}$. Hence

$$\mathtt{d}_{t_v}[v, s] \geq d^k(v, s),$$

and, again, we obtain a contradiction with respect to Eq. (1). Hence, let us suppose that $t_{m'_z} < t_{m''_z}$, for each node $z \in \mathtt{via}_{t_v}[v, s]$. It follows that $t_{m'_z} < t_v < t_{m''_z}$.

Since $t_v \geq t_l(v, s)$, after t_v, v can only perform phases REDUCE-VIA and EXTEND-VIA. Let $\mathrm{Ext}(v, s)$ be the set of nodes added to $\mathtt{via}[v, s]$ after t_v as a consequence of an EXTEND-VIA phase performed by v. We can assume that each node z in $\mathrm{Ext}(v, s)$ fulfills the same properties of nodes in $\mathtt{via}_{t_v}[v, s]$, that is:

1. z is *black* with respect to s,
2. $t_z < t_l(z, s)$,
3. $t_{m'_z} < t_{m''_z}$.

Let $t_{\max} = \max\{t_{m''_z} \mid z \in \mathtt{via}_{t_v}[v, s] \cup \mathrm{Ext}(v, s)\}$. Informally, $t_{\max}$ is the time when v receives the last *increase* message from nodes in $\mathtt{via}_{t_v}[v, s] \cup \mathrm{Ext}(v, s)$. It follows that, at time $t_{\max}$, v performs Procedure INCREASE and tests at Lines 1 and 4 return true. Then, v performs phase BUILD-TABLE at time $t_{\max} > t_v \geq t_l(v, s)$, a contradiction with respect to the definition of $t_l(v, s)$. □

The following theorem shows the correctness of the decremental algorithm.

Theorem 3.10. *There exists t_F such that, for each pair of nodes $v, s \in V$ and for each time $t \geq t_F$:*

$$\mathtt{d}_t[v, s] = d^k(v, s)$$
$$\mathtt{via}_t[v, s] \equiv via^k(v, s).$$

Proof. The correctness of the algorithm is shown with respect to a fixed source s. The correctness for all pairs of nodes is a straightforward consequence. In fact, since procedures INCREASE and DIST always refer to the record of the routing table related to a single source, then the two executions of the decremental algorithm related to two different sources cannot access the same record of the routing table.

Let us denote as $t_F(v, s)$ the time when the statement is true for v and s. If there exists $t_F(v, s)$ for each $v, s \in V$, then $t_F = \max_{v \in V}(t_F(v, s))$. Now we show that $t_F(v, s)$ exists for a generic pair (v, s). We consider *white*, *gray* and *black* nodes with respect to s separately.

Let v be a *white* node with respect to s. By Lemmas 3.3 and 3.5 and Corollary 3.8, v does not perform none of the following phases: IMPROVE-TABLE, REDUCE-VIA, BUILD-TABLE and EXTEND-VIA. Thus v never changes the values of $\mathtt{RT}_v[s]$. Hence, $t_F(v, s) = t_1$.

Let v be a *gray* node with respect to s. We have to show that there exists a time $t_F(v, s)$ such that, for each $t \geq t_F(v, s)$:

$$\mathtt{d}_t[v, s] = d^k(v, s) = d^0(v, s)$$
$$\mathtt{via}_t[v, s] \equiv via^k(v, s) \subsetneq via^0(v, s).$$

Concerning the distances, Corollary 3.4 directly implies $\mathtt{d}_t(v, s) = d^0(v, s)$, for each t. To prove that the via information are correctly updated, we first observe that $via^k(v, s)$ can be alternatively defined as follows:

$$via^k(v, s) \equiv \{u \in N(v) \mid d^0(v, s) = w(v, u) + d^0(u, s) \text{ and } d^k(u, s) = d^0(u, s)\}.$$

Moreover, at time t_1:

$$\mathtt{via}_{t_1}[v, s] \equiv via^0(v, s) \equiv \{u \in N(v) \mid d^0(v, s) = w(v, u) + d^0(u, s)\}.$$

Hence, it remains to be shown that v removes from $\mathtt{via}_{t_1}[v, s]$ each node u such that $d^0(u, s) \neq d^k(u, s)$ (that is, each *black* node with respect to s) and does not add further nodes to $\mathtt{via}_{t_1}[v, s]$.

By Lemma 3.3 and Corollary 3.8, v does not perform any of the following phases: BUILD-TABLE, IMPROVE-TABLE, EXTEND-VIA. Hence v does not add nodes to $\mathtt{via}[v, s]$.

By Lemma 3.3, each *black* node u in $\mathtt{via}_{t_1}[v, s]$ performs phase PROPAGATE_1, then it sends an *increase* message to v. Since each u belongs to $\mathtt{via}_{t_1}[v, s]$, the local executions related to these messages perform phase REDUCE-VIA at time t_u. Hence, at a time $\tilde{t} = \max\{t_u\}$, all *black* nodes with respect to s in $\mathtt{via}_{t_1}[v, s]$ are removed from $\mathtt{via}[v, s]$. Thus, $t_F(v, s) = \tilde{t}$.

Let v be a *black* node with respect to s. We have to show that there exists a time $t_F(v, s)$ such that, for each $t \geq t_F(v, s)$:

$$\mathtt{d}_t[v, s] = d^k(v, s) > d^0(v, s)$$
$$\mathtt{via}_t[v, s] \equiv via^k(v, s) \not\equiv via^0(v, s).$$

Putting together the definition of node colors and the definition of via, it is easy to see that the set $via^k(v, s)$ can be alternatively defined the union of two disjoint sets:

- $via^k(v, s) \equiv B_s(v) \cup NB_s(v)$;
- $B_s(v) \equiv \{u \in N(v)$ *black* with respect to $s \mid d^k(v, s) = w(v, u) + d^k(u, s)\}$;
- $NB_s(v) \equiv \{u \in N(v)$ *nonblack* with respect to $s \mid d^k(v, s) = w(v, u) + d^0(u, s)\}$.

Now we show that each shortest path P in G^k from v to s has the following structure:

$$P = v \equiv v_1 \rightsquigarrow v_{j-1} \rightarrow v_j \rightsquigarrow v_h \equiv s \quad \text{where:}$$
$$v_1, v_2, \ldots, v_{j-1}: \quad \text{are } \textit{black} \text{ nodes with respect to } s$$
$$v_j, v_{j+1}, \ldots, v_h: \quad \text{are } \textit{nonblack} \text{ nodes with respect to } s.$$

In fact, v is *black*, and, since s is *white*, then there exists a *nonblack* node v_j, $2 \leq j \leq h$. By Property 4 of Fact 3.1, the existence of v_j *nonblack* implies that nodes $v_{j+1}, \ldots, v_h$ are all *nonblack*.

Let us define the set of subpaths $v \rightsquigarrow v_j$ as follows:

$$\mathcal{P}_s(v) = \{P' \equiv v \rightsquigarrow v_j \mid \quad P' \subseteq P, P = v \rightsquigarrow s \text{ is a shortest path in } G^k, \text{ and } \forall v_i \not\equiv v_j \text{ in } P', v_i \text{ is } \textit{black} \text{ wrt } s \text{ and } v_j \text{ is } \textit{nonblack} \text{ wrt } s\}.$$

We define $L_s(v)$ as the maximum length among all paths in $\mathcal{P}_s(v)$

$$L_s(v) = \max_{P' \in \mathcal{P}_s(v)} \{\ell(P')\},$$

and give the proof by induction on $L_s(v)$.

Inductive basis ($L_s(v) = 1$): let v be a node such that $L_s(v) = 1$ that is, for each shortest path $P = v \equiv v_1 \rightarrow v_2 \rightsquigarrow v_h \equiv s$ in G^k, nodes $v_2, \ldots, v_h$ are all *white* or *gray* with respect to s. Notice that, in this case, $B_s(v) \equiv \emptyset$; this means that each node in $viatop^k(v, s)$ is *nonblack*.

First of all, we show the correctness of $\mathtt{d}[v, s]$.

From one side, by Lemma 3.3, v performs phase BUILD-TABLE. Let z be a node such that $z \in NB_s(v)$. The first local execution of these instructions assigns a value of $\mathtt{d}_{t_f(v,s)}[v, s]$ such that:

$$\begin{aligned}
\mathtt{d}_{t_f(v,s)}[v, s] &= \min_{v_i \in N(v)} \{w(v, v_i) + \mathtt{d}[v_i, s]\} \\
&\leq w(v, z) + \mathtt{d}_{t_1}[z, s] \\
&= w(v, z) + d^0(z, s) \\
&= d^k(v, s).
\end{aligned}$$

Hence,

$$\mathtt{d}_{t_f(v,s)}[v, s] \leq d^k(v, s). \tag{2}$$

On the other hand, by Lemma 3.9,

$$\mathtt{d}_{\tilde{t}}[v, s] \geq d^k(v, s), \quad \text{for each } \tilde{t} \geq t_l(v, s). \tag{3}$$

Now, let us consider a local execution by v of Procedure INCREASE at time t' such that $t_f(v, s) \leq t' \leq t_l(v, s)$. Trivially, if v performs phase IMPROVE-TABLE or phase EXTEND-VIA, then the value of $\mathtt{d}[v, s]$ can only decrease. Now, assume that v performs phase BUILD-TABLE. In this case, v recomputes $\mathtt{d}[v, s]$ by using values obtained from its neighbors (see Line 8). Among these neighbors there are *nonblack* nodes z such that $\mathtt{d}_t[z, s] = d^k(z, s)$ for each t. Hence, also in the case of BUILD-TABLE execution, the value of $\mathtt{d}[v, s]$ can only decrease. This observation, along with Eq. (2), implies that at time t',

$$\mathtt{d}_{t'}[v, s] \leq \mathtt{d}_{t_f(v,s)}(v, s) \leq d^k(v, s). \tag{4}$$

In conclusion, by using Eqs. (3) and (4), we get

$$\mathtt{d}_t[v, s] = d^k(v, s), \quad \text{for each } t \geq t_l(v, s).$$

In order to show the correctness of $\mathtt{via}[v, s]$, we show that, if $L_s(v) = 1$, $Exe_l(v, s)$ performs phase BUILD-TABLE. If we assume, by contradiction, that $Exe_l(v, s)$ performs phase IMPROVE-TABLE, then, the following facts hold:

1. phase IMPROVE-TABLE is performed by v as a consequence of the message $m = \text{INCREASE}(u, s, \mathtt{d}_{t_u}[u, s])$, where $u \in N(v)$ and $t_u < t_l(v, s)$;
2. m is received by v at time t' such that $t_u < t' < t_l(v, s)$;

3. since v executes phase IMPROVE-TABLE, the condition at Line 15 is true. Hence,

$$w(v,u) + \mathtt{d}_{t_u}[u,s] < \mathtt{d}_{t'}[v,s];$$

4. by Eq. (4), $\mathtt{d}_{t'}[v,s] \leq \mathtt{d}_{t_f(v,s)}(v,s) \leq d^k(v,s)$. Hence, $w(v,u) + \mathtt{d}_{t_u}[u,s] < d^k(v,s)$;
5. v performs the instruction at Line 17, then

$$\mathtt{d}_{t_l(v,s)}[v,s] = w(v,u) + \mathtt{d}_{t_u}[u,s] < d^k(v,s). \tag{5}$$

Eq. (5) represents a contradiction for Lemma 3.9. This prove that $Exe_l(v,s)$ performs phase BUILD-TABLE.

During $Exe_l(v,s)$, each node in $NB_s(v)$, is added to $\mathtt{via}_{t_l(v,s)}[v,s]$ at Line 9 of Procedure INCREASE. Furthermore, there can exist a *black* node u' in $\mathtt{via}_{t_l(v,s)}[v,s]$. But u', at time $t_l(u',s)$, will send an *increase* message to v. The local execution related to m performs phase REDUCE-VIA at time $t_{u'}$. Hence, $t_F(v,s) = \max\{t_{u'}, t_l(v,s)\}$.

Inductive step: the inductive hypothesis is: each node v such that $L_s(v) \leq l-1$ correctly assigns $\mathtt{d}_{t_F(v,s)}[v,s]$ and $\mathtt{via}_{t_F(v,s)}[v,s]$.

Let v be a node such that $L_s(v) = l$. Each node $z \in via^k(v,s)$ satisfies $L_s(z) \leq l-1$. Then, by inductive hypothesis, z correctly updates $\mathtt{d}[z,s]$ and $\mathtt{via}[z,s]$ at time $t_l(z,s)$ and, consequently, it sends the message *increase*$(z,s,d^k(z,s))$ to v. Notice that, this message is due to the propagation of the algorithm after a weight increase operation occurred "far from" v. Furthermore, z could send another *increase* message as a consequence of a weight increase operation that occurs "locally" to v, that is on the edge (z,v) at time t_i, $1 \leq i \leq k$. Let $t_z = \max\{t_l(z,s), t_i\}$, and let t'_z be the time when the *increase* message sent by z at time t_z is received by v. Now, let $m =$ *increase*$(\bar{z},s,d^k(\bar{z},s))$ be the message received by v at time $t_{\bar{z}} = \min\{t'_u \mid u \in via^k(v,s)\}$ sent by $\bar{z} \in via^k(v,s)$.

When v receives m, it performs the Procedure INCREASE. Let $\bar{t}$ be the time when this execution terminates. Then,

$$\begin{aligned}\mathtt{d}_{\bar{t}}[v,s] &\leq w(v,\bar{z}) + \mathtt{d}_{t_{\bar{z}}}[\bar{z},s]\\ &= w(v,\bar{z}) + d^k(\bar{z},s)\\ &= d^k(v,s).\end{aligned}$$

Hence,

$$\mathtt{d}_{\bar{t}}[v,s] \leq d^k(v,s). \tag{6}$$

Furthermore, by Lemma 3.9,

$$\mathtt{d}_{\tilde{t}}[v,s] \geq d^k(v,s), \quad \text{for each } \tilde{t} \geq t_l(v,s). \tag{7}$$

Moreover, if t' is a time such that $\bar{t} \leq t' \leq t_l(v,s)$, then we get the following relationship:

$$\mathtt{d}_{t'}[v,s] \geq \mathtt{d}_{\bar{t}}[v,s] \leq d^k(v,s). \tag{8}$$

To show that Eq. (8) holds, we can use the same arguments used to show Eq. (4) in the inductive basis. In particular,

- Eqs. (6) and (7) play the same role of Eqs. (2) and (3);
- node $\bar{z}$ plays the same role of nodes in $NB_s(v)$. In fact, $\mathtt{d}_t[\bar{z},s] = d^k(\bar{z},s)$, for each $t \geq t_{\bar{z}}$.

In conclusion, by using Eqs. (7) and (8), we get

$$\mathtt{d}_t[v,s] = d^k(v,s), \quad \text{for each } t \geq t_l(v,s).$$

Regarding the values stored in $\mathtt{via}[v,s]$, note that $\bar{z} \in \mathtt{via}_t[v,s]$ for each $t \geq t_l(v,s)$. Node v receives messages *increase*$(z,s,d^k(z,s))$, $z \neq \bar{z}$, at times $t_z > t_{\bar{z}}$. As a consequence of these messages, v performs phase IMPROVE-TABLE or EXTEND-VIA, and then all nodes z are added to $\mathtt{via}[v,s]$. As in the inductive basis, there can exist nodes u' such that $u' \notin via^k(v,s)$ but $u' \in \mathtt{via}_t[v,s]$ for a time $t \geq t_l(v,s)$. For the same arguments used in the inductive basis, these nodes will be removed from $\mathtt{via}[v,s]$ at time $t_{u'}$. Hence $t_F(v,s) = \max\{t_{u'}, t_l(v,s), t_z\}$. □

Complexity issues.

In the remainder of the section, we show by an example that the message complexity of the decremental algorithm cannot be bounded by a worst case analysis. In fact, the number of messages in the example can be arbitrarily large and does not depend by any topological parameter of the graph.

However, we show by another example that in practical cases, the number of messages sent by concurrent executions of the INCREASE algorithm can be smaller than in the sequential cases. The practical efficiency of the algorithm is analyzed in Section 5.

Example 3.11. Let us consider a graph $G = (V,E,w)$ with the following properties:

- a subset of nodes $V_R \subsetneq V$ is a cycle, $V_R = \{v \equiv v_0, v_1, v_2, \ldots, v_{\ell-1}, v_\ell\}$ with $w(v_i, v_{i+1}) = \epsilon$ for $i = 0,1,2,\ldots,\ell-1$ and $w(v, v_\ell) = \ell \cdot \epsilon$;
- the only connection between nodes in V_R and $V \setminus V_R$ is (u, v).

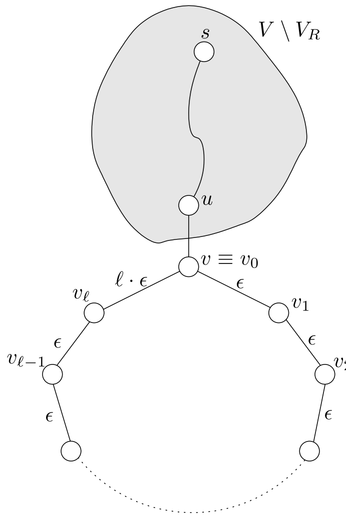

Fig. 4. A scenario which shows that the number of messages sent by the decremental algorithm cannot be bounded by worst case analysis.

For a visualization of graph G, see Fig. 4. For a given source s, we have:

- $\mathtt{via}_0[v, s] = \{u\}$;
- $\mathtt{via}_0[v_i, s] = \{v_{i-1}\}$, $\mathtt{d}_0[v_i, s] = \mathtt{d}_0[v, s] + i\epsilon$, for $i = 1, 2, \ldots, \ell - 1$;
- $\mathtt{via}_0[v_\ell, s] = \{v, v_{\ell-1}\}$, $\mathtt{d}_0[v_\ell, s] = \mathtt{d}_0[v, s] + \ell\epsilon$.

Let us suppose that, as a consequence of *weight increase* operation, node v receives a message *increase*$(u, v, \mathtt{d}[u, s])$ form u, and $\mathtt{d}[u, s] + w(u, v) > \mathtt{d}_0[v, s] + 2\ell\epsilon$. In the BUILD-TABLE phase, v chooses v_ℓ as new via and sets $\mathtt{via}[v, s] = \{v_\ell\}$ and $\mathtt{d}_1[v, s] = \mathtt{d}_0[v, s] + 2\ell\epsilon$. Then v sends *increase* messages to its neighbors in phase PROPAGATE_1. When node v_ℓ receives the *increase* message form v , it removes v from $\mathtt{via}[v_\ell, s]$. When node v_1 receives the *increase* message form v, it sets $\mathtt{d}_1[v_1, s] = \mathtt{d}_0[v_1, s] + 2\ell\epsilon$. Then, in turn, each node v_i, $i = 2, 3, \ldots, \ell$ sets $\mathtt{d}_1[v_i, s] = \mathtt{d}_0[v_i, s] + 2\ell\epsilon$.

Therefore, node v_ℓ sends message *increase*$(v_\ell, s, \mathtt{d}_1[v_\ell, s])$ to node v which updates its distance to s and sets $\mathtt{d}_2[v, s] = \mathtt{d}_1[v, s] + 2\ell\epsilon = \mathtt{d}_0[v, s] + 4\ell\epsilon$, and a new round is started along nodes in V_R. After T rounds of updates along the cycle V_R, where T is the minimal number such that $\mathtt{d}[u, s]+w(u, v) > \mathtt{d}_0[v, s]+T\ell\epsilon$, node v sets the correct values of $\mathtt{via}_T[v, s] = \{u\}$ and $\mathtt{d}_T[v, s] = d[u, s] + w(u, v)$ and the last round of updates along nodes in V_R takes place. Note that the value T can be made arbitrarily large by choosing an appropriate value of ϵ. Hence the number of messages sent does not depend by any topology parameter of G.

In the next example, we show that in some cases the concurrent executions of the algorithms for two *weight increase* operations allows us to deliver a number of messages that is smaller than the number of messages delivered in the sequential case.

Example 3.12. The scenario for the following example is depicted in Fig. 5. Let $G = (V, E, w)$ be a weighted undirected graph on which the following two *weight increase* operations are performed:

1. σ_1 that involves edge $x_1 \to y_1$ whose weight is increased by a quantity $\epsilon_1 = 1$
2. σ_2 that involves edge $x_2 \to y_2$ whose weight is increased by a quantity $\epsilon_2 = 100$.

Let $s \in V$ be a source node such that $\delta_{\sigma_1,s} \cap \delta_{\sigma_2,s} \neq \emptyset$. This means that there exists at least one node v such that each shortest paths from v to s contains the edges (x_1, y_1) and (x_2, y_2). The propagation of messages related to source s can change depending on the order in which the *increase* messages are delivered to the nodes in $\delta_{\sigma_1,s} \cap \delta_{\sigma_2,s}$. In other words, the messages exchange depends on the two executions of the algorithm. Let us consider the node v in Fig. 5 and let m_1 and m_2 be the two messages received by v related to the operations σ_1 and σ_2, respectively:

$$m_1 = \mathit{increase}(u, s, \mathtt{d}_{\tilde{t}_1}[u, s])$$
$$m_2 = \mathit{increase}(u, s, \mathtt{d}_{\tilde{t}_2}[u, s]).$$

We suppose that:

- t_{m_1} and t_{m_2}, $t_{m_2} < t_{m_1}$, are the times when v receives m_1 and m_2, respectively;
- $\mathtt{d}_t[u, s] = 10$ for any time $t \leq t_1$;
- $\mathtt{d}_t[z, s] = 30$ for any time t.
- $\mathtt{via}_{t_{m_1}}[v, s] \equiv \{u\}$ and $\mathtt{d}_{t_{m_1}}[v, s] = w(v, u) + \mathtt{d}_t[u, s] = 11$ for a certain $t \leq t_1$;

In other words, before any *weight increase* operation u is the only via from v to s and both u and v are black nodes while z is *white*.

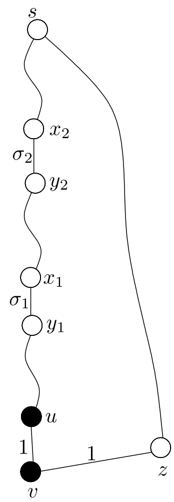

Fig. 5. A scenario where the concurrent executions of the algorithm allow us to deliver a number of message that is smaller than the sequential case.

Since $t_{m_2} < t_{m_1}$, v performs the following operations in the order in which they are written:

1. **Event**: v receives message m_2.
   line 1 : since $u \in \mathtt{via}_{t_{m_1}}[v, s]$, the condition is true.
   REDUCE-VIA : $\mathtt{via}[v, s] := \{u\} \setminus \{u\} = \emptyset$.
   line 4 : since $\mathtt{via}[v, s] = \emptyset$, the condition is true.
   BUILD-TABLE : Since m_2 is related to σ_2, then $\mathtt{d}_{\tilde{t}_2}[u, s] \geq \mathtt{d}_{\tilde{t}_1}[u, s] + \epsilon_2$, and hence v performs the following operations:
   - $\mathtt{d}[v, s] := w(v, z) + \mathtt{d}[z, s] = 31$
   - $\mathtt{via}[v, s] := \{z\}$.

   PROPAGATE_1 : v sends $\deg(v)$ times the message *increase*$(v, s, \mathtt{d}[v, s])$.
2. **Event**: v receives message m_1.
   line 1 : since $u \notin \mathtt{via}_{t_{m_1}}(v, s)$, the condition is false.
   REDUCE-VIA : not performed.
   line 4 : not performed.
   BUILD-TABLE : not performed.
   PROPAGATE_1 : not performed, then v does not send the *increase* message.

Let us now analyze the sequential case. Notice that the node z is such that for any time t the following inequalities hold:

$$w(v, u) + \mathtt{d}_{t_1}[u, s] + \epsilon_2 > w(v, z) + \mathtt{d}_t[z, s] > w(v, u) + \mathtt{d}_{t_1}[u, s] + \epsilon_1.$$

Hence, v performs two times the BUILD-TABLE phase:

1. as a consequence of σ_1, v performs:
   - $\mathtt{d}[v, s] := w(v, u) + \mathtt{d}_{t_1}[u, s] + \epsilon_1 = 11$
   - $\mathtt{via}[v,] := \{u\}$
2. as a consequence of σ_2, v performs:
   - $\mathtt{d}[v, s] := w(v, z) + \mathtt{d}[z, s] = 31$
   - $\mathtt{via}[v, s] := \{z\}$.

Then v sends $2\deg(v)$ messages as a consequence of each *weight increase* operation ($\deg(v)$ *increase* messages and $\deg(v)$ *get-dist* messages). Thus, in the concurrent framework, we have saved at least $2\deg(v)$ messages.

4. The incremental algorithm

In this section we describe our new incremental solution for the concurrent update of distributed all-pairs shortest paths in the case of multiple operations. The algorithm proposed in this section is an extension of the incremental algorithm proposed in [7]. The incremental algorithm of [7] has been shown to work only in the sequential case. We will show that our extension works correctly also in the concurrent case. Our solution differs from that in [7] in how the algorithm starts and in the message delivering policy between neighbors. In particular, we force the messages between two neighbors to be delivered in a FIFO order. Furthermore, we will show that the incremental algorithm in [7] is not able to work in the case of multiple concurrent weight decrease operations if the channels are not FIFO.

Event: node v receives the message $init(u, s, \mathtt{d}[u, s])$.

Procedure INIT

```
1.  if d[v,s] > w(v,u) + d[u,s] then
2.    begin
3.      d[v,s] := w(v,u) + d[u,s]
4.      via[v,s] := u
5.      for each v_i ∈ N(v) \ {u} do
6.        send decrease(v, s, d[v,s], v) to v_i
7.    end
```

Fig. 6. The INIT procedure.

Event: node v receives the message $decrease(u, s, \mathtt{d}[u, s], y)$.

Procedure DECREASE

```
1.  if via[v,y] = u then
2.    begin
3.      if d[v,s] > w(v,u) + d[u,s] then
4.        begin
5.          d[v,s] := w(v,u) + d[u,s]
6.          via[v,s] := u
7.          for each v_i ∈ N(v) \ {u} do
8.            send decrease(v, s, d[v,s], y) to v_i
9.        end
10.   end
```

Fig. 7. The DECREASE procedure.

We consider the algorithm to handle k *weight decrease* operations $\sigma_1, \sigma_2, \ldots, \sigma_k$, since the extension to *insert* operations is straightforward. In fact, inserting a new edge $x \to y$ with weight w is equivalent to decrease $w(x, y)$ from $+\infty$ to w.

Data structures. The data structures used in the incremental case are almost identical to those of the decremental case. The only difference stays in the field $\mathtt{RT}_v[s].\mathtt{via}$ in the routing table of v which is defined as follows:

$$\mathtt{RT}_v[s].\mathtt{via} \in \{v_i \in N(v) \mid \mathtt{RT}_v[s].\mathtt{d} = w(v, v_i) + \mathtt{RT}_{v_i}[s].\mathtt{d}\}.$$

This implies that this field stores only *the* neighbor of v used to determine the estimated distance. This means that in the incremental case, for each node v we store only one shortest path to each source s.

Algorithm.

Before the incremental algorithm starts, we assume that $\mathtt{d}_t[v, s]$ and $\mathtt{via}_t[v, s]$ are correct, for each $v, s \in V$ and for each $t < t_1$. The algorithm starts at each t_i, $i \in \{1, 2, \ldots, k\}$. For instance, the *weight decrease* operation σ_i represents an event that is detected, at time t_i, only by nodes x_i and y_i; as a consequence:

- y_i sends the message $init(y_i, s, \mathtt{d}_{t_i}[y_i, s])$ to x_i, for each $s \in V$;
- x_i sends the message $init(x_i, s, \mathtt{d}_{t_i}[x_i, s])$ to y_i, for each $s \in V$.

When x_i receives $init(y_i, s, \mathtt{d}_{t_i}[y_i, s])$ by y_i, x_i executes procedure INIT (see Fig. 6). This procedure is responsible for checking if it is necessary to start the incremental algorithm. In the affirmative case (see line 1), x_i updates $\mathtt{RT}_{x_i}[s]$ at a certain time t and, in order to propagate the incremental algorithm, sends the message $decrease(x_i, s, \mathtt{d}_t[x_i, s], x_i)$ to its neighbors (line 6). The first three arguments of the message have the same meaning as in *init*, while the fourth argument is one of the endpoints of the edge changed by σ_i, that is, either x_i or y_i.

The behavior of y_i (when y_i receives the message $init(x_i, s, \mathtt{d}_{t_i}[x_i, s])$) is symmetric. At most one between x_i and y_i will propagate the incremental algorithm. In fact, if we assume, without loss of generality, that $\mathtt{d}_{t_i}[s, x_i] \leq \mathtt{d}_{t_i}[s, y_i]$, then the test performed by x_i at Line 1 of procedure INIT is false. Thus, x_i does not update $\mathtt{RT}_{x_i}[s]$ and does not propagate the *decrease* message to its neighbors.

Conversely, under the same assumptions, y_i may improve its distance from s. In this case y_i updates $\mathtt{RT}_{y_i}[s]$ at a certain time t and, in order to propagate the incremental algorithm, sends the message $decrease(y_i, s, \mathtt{d}_t[y_i, s], y_i)$ to its neighbors. When a node v receives the message $decrease(u, s, \mathtt{d}_{\tilde{t}}[u, s], y_i)$, $\tilde{t} \geq t$, from a node u, it performs procedure DECREASE (see Fig. 7).

Remember that, in our model, multiple messages *init* and *decrease* received by a node are stored and processed in FIFO order.

Procedure Decrease differs from the classical distributed Bellman–Ford algorithm (e.g., see [5]) in the way in which messages are propagated. In the Bellman–Ford algorithms messages containing the estimated distances are sent to all the nodes in the graph. In the algorithm described in this section these messages are sent only to the nodes that change the shortest path with respect to at least one source as a consequence of an edge modification. To better explain this characteristic it is convenient to formalize the notion of "nodes affected by an edge modification".

Given a node v and a *weight decrease* operation σ_i, if there exists a node s such that v decreases its distance from s as a consequence of σ_i, then we say that v is *affected* by σ_i. Hence, $\delta_{\sigma_i,s}$ is defined as follows:

$$\delta_{\sigma_i,s} = \{v \in V \mid d^i(v, s) < d^{i-1}(v, s)\}.$$

When a node v receives the message *decrease*$(u, s, \mathtt{d}[u, s], y)$ from a node $u \in N(v)$, it performs procedure Decrease. Before testing at Line 3 whether $\mathtt{d}[u, s]$ contributes to give a better estimated distance from v to s, Procedure Decrease verifies at Line 1 if $u \in \mathtt{via}[v, y]$. This test is due to the following fact.

Fact 4.1. *The following properties hold:*

1. *if $v \in \delta_{\sigma_i,s}$, then each shortest path from s to v in G^i contains edge $x_i \to y_i$. Hence, such shortest paths are in the form*

   $$s \rightsquigarrow x_i \to y_i \rightsquigarrow v$$

   where the subpaths $s \rightsquigarrow x_i$ and $y_i \rightsquigarrow v$ are shortest paths in G^{i-1}. We remark that the subpaths $s \rightsquigarrow x_i$ and $y_i \rightsquigarrow v$ may be empty, that is, $s \equiv x_i$ and $y_i \equiv v$.
2. *if $v \in \cup_{i=1}^{k} \delta_{\sigma_i,s}$, then each shortest path from s to v in G^k contains at least a modified edge $x_i \to y_i$. Hence, such shortest paths are in the form*

   $$s \rightsquigarrow x_{i_1} \to y_{i_1} \rightsquigarrow x_{i_2} \to y_{i_2} \rightsquigarrow \cdots \rightsquigarrow x_{i_h} \to y_{i_h} \rightsquigarrow v$$

   where $\{i_1, i_2, \ldots, i_h\} \subseteq \{1, 2, \ldots, k\}$, the subpaths $s \rightsquigarrow x_{i_1}$, $y_{i_j} \rightsquigarrow x_{i_{j+1}}$ with $1 \le j \le h-1$, and $y_{i_h} \rightsquigarrow v$ are shortest path in G^i, $i = 0, 1, \ldots, k$. As in the previous case, we remark that such subpaths may be empty.

If only the weight decrease operation on the edge $x_i \to y_i$ occurs, then the messages necessary to update v with respect to source s are delivered, according to Property 1 of Fact 4.1, only along the path $y_i \rightsquigarrow v$. To achieve this, the algorithm performs the test at Line 1 of procedure Decrease. If there are many weight decrease operations, the messages are delivered according to Property 2 of Fact 4.1.

Correctness analysis. The following lemma and the subsequent theorem show the correctness of the algorithm.

Lemma 4.2. *For each node v, for each source s and for each time t the inequality $\mathtt{d}_t[v, s] \ge d^k(v, s)$ holds.*

Proof. By contradiction, let us suppose that v is the first node to fail to update its routing table, that is, there exists a minimum time t_v such that $\mathtt{d}_{t_v}[v, s] < d^k(v, s)$. v updates its routing table as a consequence of the reception of a message *decrease*$(z, s, d_{t_z}[z, s], y)$ or *init*$(z, s, d_{t_z}[z, s])$, with $t_z < t_v$, from a node $z \in N(v)$. The updating is performed at Line 5 of Procedure Decrease or at Line 3 of Procedure Init. In any case, $\mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s]$. Since v is the first node to fail, then $\mathtt{d}_{t_z}[z, s] \ge d^k(z, s)$. Thus,

$$d^k(v, s) > \mathtt{d}_{t_v}[v, s] = w(v, z) + \mathtt{d}_{t_z}[z, s] \ge w(v, z) + d^k(z, s)$$

a contradiction. □

The following theorem shows that the incremental algorithm works also in the concurrent case under the hypothesis that the messages are delivered, on each edge, in a FIFO order. In the next section we will show how to implement a FIFO order in the actual model without getting worse the complexity bounds.

Theorem 4.3. *There exists t_F such that, for each pair of nodes $v, s \in V$ and for each time $t \ge t_F$:*

$$\mathtt{d}_t[v, s] = d^k(v, s);$$
$$\mathtt{via}_t[v, s] \in via^k(v, s).$$

Proof. Let us denote as $t_F(v, s)$ the time when the statement is true for nodes v and s. If there exists $t_F(v, s)$ for each $v, s \in V$, then $t_F = \max_{(v,s)\in V\times V}\{t_F(v, s)\}$.

Let v, s be a pair of nodes in G, each shortest path from s to v in G^k is in the form

$$P = s \rightsquigarrow x_{i_1} \to y_{i_1} \rightsquigarrow x_{i_2} \to y_{i_2} \rightsquigarrow \cdots \rightsquigarrow x_{i_h} \to y_{i_h} \rightsquigarrow v,$$

such that $0 \le h \le k$, $\{i_1, i_2, \ldots, i_h\} \subseteq \{1, 2, \ldots, k\}$. Note that, if $h = 0$, then $v \notin \cup_{i=1}^{k} \delta_{\sigma_i,s}$, hence $P = s \rightsquigarrow v$ is a shortest path in G^i, $i = 0, 1, \ldots, k$, that does not contains any modified edge. Otherwise, the subpaths $s \rightsquigarrow x_{i_1}$, $y_{i_j} \rightsquigarrow x_{i_{j+1}}$ with $1 \le j \le h-1$, and $y_{i_h} \rightsquigarrow v$ are shortest paths in G^i, $i = 0, 1, \ldots, k$. Moreover, if $P' = a \rightsquigarrow b$ represents one of such subpaths, then we assume that P' has the following property: it is the path induced by the values of the fields $\mathtt{RT}_u[a].\mathtt{via}$ before time t_1, for each node $u \not\equiv a$ belonging to P'. We denote by $\mathcal{P}(v, s)$ the set containing all the shortest paths from v to s in G^k having the above property, and we set $len(v, s) = \max_{P\in\mathcal{P}(v,s)}\{\ell(P)\}$.

The proof is by induction on $len(v, s)$ for each pair of nodes v and s in G.
Inductive basis ($len(v, s) = 0$): a pair of nodes v, s is such that $len(v, s) = 0$ if and only if $v \equiv s$. In this case, at time t_1 we have:

$$\mathtt{d}_{t_1}[s, s] = 0$$
$$\mathtt{via}_{t_1}[s, s] = s.$$

Node s updates its routing table only after receiving *init* or *decrease* messages. Let t_m be the time when s receives the first message m from a node $z \in N(s)$. Let $\mathtt{d}_{t_z}[z, s]$ be the distance estimate contained in m, where $t_z < t_m$.
By Lemma 4.2, $\mathtt{d}_{t_z}[z, s] \geq d^k(z, s)$. Hence

$$w(s, z) + \mathtt{d}_{t_z}[z, s] \geq w(s, z) + d^k(z, s) \geq 0.$$

Since m is the first message received by s, $\mathtt{d}_{t_1}[s, s] = \mathtt{d}_{t_m}[s, s] = 0$. Thus the conditions in lines 1 and 3 of Procedure Init and Decrease respectively are false and s does not update its routing table.

If s receives further messages, the same arguments can be used to show that s never updates $\mathtt{RT}_s[s]$. Then $t_F(s, s) = t_1$.
Inductive step: by inductive hypothesis, for each pair of nodes s, v such that $len(v, s) \leq l - 1$, v (s, resp.) correctly updates $\mathtt{RT}_v[s]$ ($\mathtt{RT}_s[v]$, resp.) at time $t_F(v, s)$ ($t_F(s, v)$, resp.). We now show that the theorem holds for pair of nodes (v, s) such that $len(v, s) = l$.

Let v and s be two nodes such that $len(v, s) = l$. If there exists a path $P \in \mathcal{P}(v, s)$ such that P is a shortest path in G^0, then P does not contains any modified edge. Since $\mathtt{RT}_v[s]$ is correct before t_1, at time t_1 we have

$$\mathtt{d}_{t_1}[v, s] = d^0(v, s) = d^k(v, s);$$
$$\mathtt{via}_{t_1}[v, s] \in via^0(v, s) \subseteq via^k(v, s).$$

Hence, we have to show that v does not update $\mathtt{RT}_v[s]$. v updates $\mathtt{RT}_s[v]$ only after receiving *init* or *decrease* messages. Let t_m be the time when v receives the first message m from a node $z \in N(v)$. Let $\mathtt{d}_{t_z}[z, s]$ be the distance estimate contained in m, where $t_z < t_m$.
By Lemma 4.2, $\mathtt{d}_{t_z}[z, s] \geq d^k(z, s)$. Hence

$$w(v, z) + \mathtt{d}_{t_z}[z, s] \geq w(v, z) + d^k(z, s) \geq d^k(v, s).$$

Since m is the first message received by s, $\mathtt{d}_{t_m}[v, s] = \mathtt{d}_{t_1}[v, s] = d^k(v, s)$. Thus the conditions in lines 1 and 3 of Procedure Init and Decrease respectively are false and v does not update its routing table.

If s receives further messages, the same arguments can be used to show that v never updates $\mathtt{RT}_v[s]$. Then $t_F(v, s) = t_1$.

Let us now analyze the case in which each shortest path from v to s contains at least a modified edge. v correctly updates $\mathtt{RT}_v$ only if, at a certain time t_u, it receives from a node u in $via^k(v, s)$ one of the following messages:

1. $init(u, s, d^k(u, s))$
2. $decrease(u, s, d^k(u, s), y_u)$ and $\mathtt{via}_{t_u}[v, y_u] = u$.

Let I be the set of messages $init(u, s, d^k(u, s))$ such that $u \in via^k(v, s)$ and let D be the set of messages $m = decrease(u, s, d^k(u, s), y_u)$ such that $u \in via^k(v, s)$ and $\mathtt{via}_{t_u}[v, y_u] = u$, where t_u is the time when v receives m. We now show that the set $I \cup D$ is not empty.

If $I \not\equiv \emptyset$ then $I \cup D$ is clearly not empty. Let us assume that $I \equiv \emptyset$, we have to show that in this case $D \not\equiv \emptyset$. Since each shortest path from v to s contains a modified edge, by inductive hypothesis, each node u in $via^k(v, s)$ correctly updates $\mathtt{RT}_u[s]$ and sends to v a message containing $d^k(u, s)$. Since $I \equiv \emptyset$, these messages are *decrease* messages. Let $m = decrease(u, s, d^k(u, s), y_u)$ be the message sent by u and t_u be the time when v receives m. We have to show that there exists a message such that $\mathtt{via}_{t_u}[v, s] = u$. Note that u sends m only if $\mathtt{RT}_u[y_u]$ has already a correct value when u performs procedure Init or Decrease, that is $t_F(u, y_u) < t_F(u, s)$. Furthermore may exist $u_1 \not\equiv u_2 \in via^k(v, s)$ such that $y_{u_1} \equiv y_{u_2}$. Hence we define the following sets:

$$Y = \{y_u \mid u \in via^k(v, s) \text{ and } u \text{ sends } decrease(u, s, d^k(u, s), y_u)\}$$

for each $y \in Y$

$$U_y = \{u \mid u \in via^k(v, s) \text{ and } u \text{ sends } decrease(u, s, d^k(u, s), y)\}.$$

The sets U_y define a partition of $via^k(v, s)$, that is, for each $y_1 \not\equiv y_2 \in Y$, $U_{y_1} \cap U_{y_2} \not\equiv \emptyset$ and $\bigcup_{y \in Y} U_y \equiv via^k(v, s)$.

For each $y \in Y$, two cases may occur:

- there exists a node u in U_y such that $\mathtt{RT}_u[y]$ has never been changed since time t_1 and then $t_F(u, y) = t_1$. Then, there exists a shortest path from u to y that does not contains any modified edge. In fact, if by contradiction, each path in $\mathcal{P}(u, y)$ contains a modified edge, then $\mathtt{d}_{t_1}[u, y] = d^0(u, y) > d^k(u, y)$. But, since $t_F(u, y) = t_1$, then $\mathtt{d}_{t_1}[u, y] = d^k(u, y)$, a contradiction. Furthermore, since $I \equiv \emptyset$, the edge (u, v) does not change and then there exists a shortest path from v to y that does not contains any modified edge. As a consequence, for each $u \in U_y$, $t_F(v, y) = t_1 < t_u$ and then $\mathtt{via}_{t_u}[v, y] = u' \in U_y$. Then, at time $t_{u'}$, $\mathtt{via}_{t_{u'}}[v, y] = u'$.

- each node u in U_y has correctly update $\mathtt{RT}_u[y]$ at time $t_F(u, y) > t_1$. Hence, each u in U_y has sent to v the message $m_u = \mathit{decrease}(u, y, d^k(u, y), \cdot)$ at time $t_F(u, y)$. By inductive hypothesis, v correctly updates $\mathtt{RT}_v[y]$ as a consequence of $m_{u'}$, for a certain $u' \in U_y$, at time $t_F(v, y)$. Since, for each $u \in U_y$, $t_F(u, y) < t_F(u, s)$, by the FIFO assumption on the messages delivering, v receives m_u before $m = \mathit{decrease}(u, s, d^k(u, s), y)$. Hence, for each $u \in U_y$, $t_F(v, y) < t_u$ and then $\mathtt{via}_{t_u}[v, y] = u' \in U_y$. Then, at time $t_{u'}$, $\mathtt{via}_{t_{u'}}[v, y] = u'$.

Hence, for each set U_y, there exists a node $u \in \mathit{via}^k(v, s)$ such that v receives, at time t_u, a message $\mathit{decrease}(u, s, d^k(u, s), y)$ and $\mathtt{via}_{t_u}[v, y] = u$. Hence $D \not\equiv \emptyset$.

Thus, $I \cup D \not\equiv \emptyset$. Let m be the first message in $I \cup D$ received by v, let u be the sender of m and let t_m be the time when v receives m. We now show that $\mathtt{d}_{t_m}[v, s] > w(v, u) + d^k(u, s)$. By Lemma 4.2, we have $\mathtt{d}_{t_m}[v, s] \geq d^k(v, s) = w(u, v) + d^k(u, s)$. Since m is the first message in $I \cup D$, any other messages $\bar{m}$, sent by nodes in $N(v)$ before m, are such that, if $\bar{t}$ is the time when v updates $\mathtt{RT}_v[s]$ as a consequence of $\bar{m}$, $\mathtt{d}_{\bar{t}}[v, s] > d^k(v, s)$. Hence the condition $\mathtt{d}_{t_m}[v, s] > w(v, u) + d^k(u, s)$ holds and the test at Line 3 (resp. Line 1) of Procedure Decrease (resp. Init) is true. Then v correctly updates $\mathtt{RT}_v[s]$ as a consequence of m at time t_l.

Furthermore, if v receives messages $\mathit{decrease}(z, s, \mathtt{d}_{\tilde{t}}[z, s], \cdot)$ (resp. $\mathit{init}(z, s, \mathtt{d}_{\tilde{t}}[z, s])$) at time $t_z > t_l$, by Lemma 4.2, $\mathtt{d}_{\tilde{t}}[z, s] \geq d^k(z, s)$. Thus,

$$w(v, z) + \mathtt{d}_{\tilde{t}}[z, s] \geq w(v, z) + d^k(z, s) \geq d^k(v, s)$$

and then the test at Line 3 (resp. Line 1) of Procedure Decrease (resp. Init) is false. Hence, v does not update $\mathtt{RT}_v[s]$ after t_l. It follows that $t_F(v, s) = t_l$.

To complete the proof we have to show that the concurrent update for two pair of nodes (v, s) and (v', s') such that $\mathit{len}(v, s) = \mathit{len}(v', s') = l$ does not lead to conflicts.

Let us consider two pair of nodes (v, s) and (v', s') such that $s \not\equiv s'$ and $\mathit{len}(v, s) = \mathit{len}(v', s') = l$. If $v \not\equiv v'$, then the executions related to pairs (v, s) and (v', s') cannot conflicts each other because v and v' write on two separate data structures ($\mathtt{RT}_v$ and $\mathtt{RT}_{v'}$ respectively). If $v \equiv v'$, then the executions performed by v of procedures Init and Decrease wrt s and s' cannot conflicts each other because, since $\mathit{len}(v, s) = \mathit{len}(v, s')$, $s' \notin \mathcal{P}(v, s)$ and $s \notin \mathcal{P}(v, s')$. $\square$

Complexity analysis.

In what follows we give the complexity bounds of the algorithm in Figs. 6 and 7 in the concurrent case.

These bounds are given in terms of the number of affected nodes. More precisely, given a weighted undirected graph G, a set of k weight decreases $\sigma_1, \sigma_2, \ldots, \sigma_k$ and a source node s, we denote as $\delta_{\sigma_i,s}$ the set of nodes that decrease the distance to s as a consequence of σ_i. Formally:

$$\delta_{\sigma_i,s} = \{v \in V \mid d^i(v, s) \neq d^{i-1}(v, s)\}.$$

If $v \in \cup_{s \in V} \delta_{\sigma_i,s}$ we say that v is *affected* by σ_i. The total number of times that nodes of G are affected by the k weight decrease operations is exactly $\Delta = \sum_{i=1}^{k} \sum_{s \in V} \left|\delta_{\sigma_i,s}\right|$.

Theorem 4.4. *The concurrent update of all-pairs shortest paths over a graph G with n nodes and positive real edges weights, after a set of* weight decrease *operations, requires* $O(\mathit{maxdeg} \cdot \Delta)$ *messages and* $O(n)$ *space per node.*

Proof. Given a source s and a *weight decrease* operation σ_i, a node v can update $\mathtt{RT}_v[s]$ at most one time. Each time that v updates $\mathtt{RT}_v[s]$, it sends $\deg(v)$ messages. Hence, v sends at most *maxdeg* messages. Since there are $\left|\delta_{\sigma_i,s}\right|$ nodes that change their distance from s as a consequence of σ_i, the number of messages related to the source s sent as a consequence of operation σ_i is $\mathit{maxdeg} \cdot \left|\delta_{\sigma_i,s}\right|$. The sum of this value over all sources $s \in V$ and *weight decrease* operations σ_i, $i \in \{1, 2, \ldots, k\}$ is:

$$\sum_{i=1}^{k} \sum_{s \in V} \left(\mathit{maxdeg} \cdot \left|\delta_{\sigma_i,s}\right|\right) = \mathit{maxdeg} \cdot \Delta.$$

Thus, the message complexity is $O(\mathit{maxdeg} \cdot \Delta)$.

The space complexity is $O(n)$ per node because a node stores only $\mathtt{RT}_v[\cdot]$. $\square$

On the message delivering policy. In this section we explain how to implement the FIFO channels and provide an example showing that they are necessary to the correctness of the algorithm.

During the execution of the incremental algorithm, each message sent by a node u is progressively numbered by integer values. We may assume that the first message is numbered by 1. Messages sent to v and not yet processed are stored in a buffer local to v. A message m stored in the buffer of v, numbered $\mathit{num}(m)$, and sent by $u \in N(v)$ is processed by v if and only if the last message processed by v and sent by u is numbered $\mathit{num}(m) - 1$. It is easy to see that implementing the FIFO channels does not affect the complexity bounds provided in the previous section.

In the following we provide an example to show that non-FIFO channels affect the correctness of the algorithm. This example is based on the graph G represented in Fig. 8; G is modified by means of two weight decrease operations. The figure also shows some real values of distance and via related to the node v after each modification.

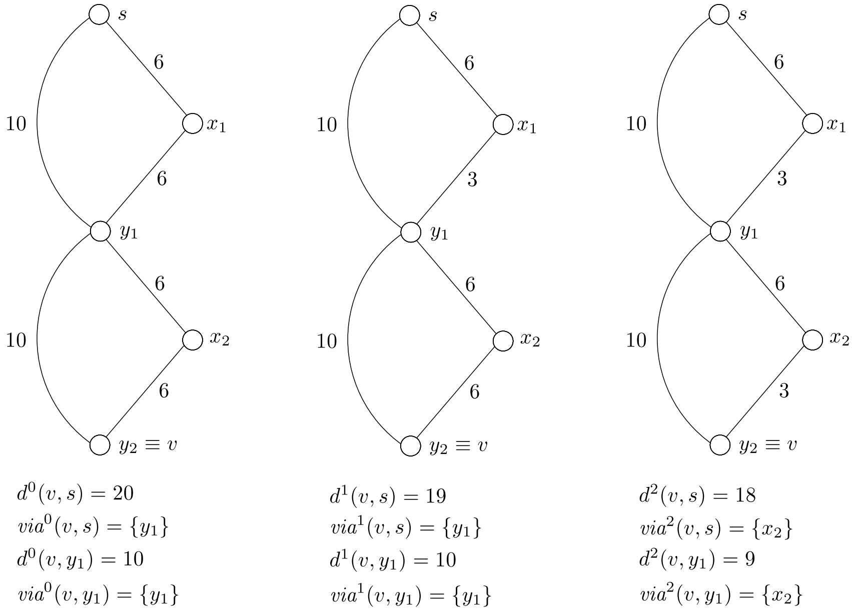

Fig. 8. Graphs G^0, G^1, and G^2. G^1 and G^2 are obtained from G^0 by applying two edge modification on edges $x_1 \to y_1$ and $x_2 \to y_2$, respectively.

According to the hypothesis, at time t_1:

- $\mathtt{d}_{t_1}[v, s] = 20$, $\mathtt{via}_{t_1}[v, s] = y_1$
- $\mathtt{d}_{t_1}[v, y_1] = 10$, $\mathtt{via}_{t_1}[v, y_1] = y_1$.

We assume that the first four messages received by v, ordered according to the time in which they are received, are listed in the following. After each message we show the content of $\mathtt{RT}_v[s]$ and $\mathtt{RT}_v[y_1]$.

1. *init*$(x_2, s, 16)$:
   This message is sent by x_2; we assume that x_2 not yet updated its routing table according to σ_1. After the message processing, v changes its estimated distance to s. These are the new values:
   - $\mathtt{d}[v, s] = 19$, $\mathtt{via}[v, s] = x_2$
   - $\mathtt{d}[v, y_1] = 10$, $\mathtt{via}[v, y_1] = y_1$
2. *decrease*$(y_1, s, 9, y_1)$:
   This message is sent by y_1; we assume that y_1 has already updated its routing table according to σ_1 (hence, now $\mathtt{d}[y_1, s] = 9$). After the message processing, v does not change its estimated distance to s since test at Line 3 of Procedure Decrease returns false.
   - $\mathtt{d}[v, s] = 19$, $\mathtt{via}[v, s] = x_2$
   - $\mathtt{d}[v, y_1] = 10$, $\mathtt{via}[v, y_1] = y_1$
3. *decrease*$(x_2, s, 15, y_1)$:
   This message is sent by x_2. As in the previous case, we assume that x_2 has already updated its routing table according to σ_1 (hence, now $\mathtt{d}[x_2, s] = 15$). Test at Line 1 of Procedure Decrease returns false, hence v does not update its routing table.
   - $\mathtt{d}[v, s] = 19$, $\mathtt{via}[v, s] = x_2$
   - $\mathtt{d}[v, y_1] = 10$, $\mathtt{via}[v, y_1] = y_1$
4. *init*$(x_2, y_1, 6)$:
   This message is sent by x_2. Test at Line 1 of Procedure Init returns true, hence v updates $\mathtt{RT}_v[y_1]$.
   - $\mathtt{d}[v, s] = 19$, $\mathtt{via}[v, s] = x_2$
   - $\mathtt{d}[v, y_1] = 9$, $\mathtt{via}[v, y_1] = x_2$.

Notice that v does not receive further messages with respect to source s. Hence, after the termination of the algorithm it results that $\mathtt{d}[v, s] = 19$ versus $d^2(v, s) = 18$. FIFO channels prevents this drawback since the delivering ordering of the example cannot occurs. In particular, FIFO channels avoid that v processes the *decrease* messages 2 and 3 before processing the *init* message 4.

5. Experimental evaluation

In this section we describe the experiments we performed to check the effectiveness of our algorithms also in the practical case.

Experimental environment. All the experiments have been carried out on a workstation equipped with a 2,66 GHz processor (Intel Core2 Duo E6700 Box) and a 8Gb RAM (PC6400 PRO Series, 800 MHz). The experiments consist of simulations within the OMNeT++ environment, version 3.3p1 [1].

OMNeT++ is an object-oriented modular discrete event network simulator, useful to model protocols, telecommunication networks, multiprocessors and other distributed systems. It also provides facilities to evaluate performance aspects of complex software systems where the discrete event approach is suitable.

An OMNeT++ model consists of hierarchically nested modules, that communicate through message passing. Modules and messages can have their own parameters, stored in arbitrarily complex data structures, that can be used to customize specific behaviors or topologies.

In our model, we defined a basic module *node* to represent a node in the network. A node v has a communication *gate* with each node in $N(v)$. Each node can send messages to a destination node through a *channel* which is a module that connects gates of different nodes (both gate and channel are OMNeT++ predefined modules). In our model, a channel connects exactly two gates and represents an edge between two nodes. We associate two parameters per channel: a *weight* and a *delay*. The former represents the cost of the edge in the graph, and the latter simulates a finite but not null transmission time.

Implemented algorithms. We implemented the algorithms described in Sections 3 and 4, that in the remainder we denote as DECR and INCR, respectively. In order to compare their performances with respect to known algorithms in the literature, we also implemented two different versions of the well-known Bellman–Ford algorithm [5,14]. They are denoted as BF.1 and BF.2 and briefly described in what follows.

BF.1 This version is described in [5], a node v updates its estimated distance to a node s, by simply executing the iteration

$$\mathtt{d}[v,s] := \min_{u\in N(v)} \{w(v,u) + \mathtt{d}[u,s]\},$$

using the last estimated distance $\mathtt{d}[u,s]$ received from a neighbor $u \in N(v)$ and the latest status of its links. Eventually, node v transmits the new estimated distance to its neighbors. It requires $O(n \cdot maxdeg)$ space per node to store the last estimated distance vector $\{\mathtt{d}[u,s] \mid s \in V\}$ received from each neighbor $u \in N(v)$.

BF.2 This version is described in [14]. It assumes that each node v initially overestimates the distance with the remaining nodes in the network. Then, for each new $\mathtt{d}[u,s]$ received from a neighbor $u \in N(v)$, it first checks whether its estimated distance to s can be improved, and, in the affirmative case, it sends the new estimated distance to each neighbor but u. It requires $O(n)$ space per node.

Algorithm BF.1 and BF.2 have the same message complexity but BF.2 does not require to store the last estimated distance vector $\{\mathtt{d}[u,s] \mid s \in V\}$ received from each neighbor $u \in N(v)$, hence BF.2 is more space efficient than BF.1. However, BF.2 cannot be used when edge weights increase as it assumes that the routing tables initially contain overestimated distances.

Thus, we experimentally compared the performances of DECR against those of BF.1 and the performances of INCR against those of BF.2.

Input data and executed tests. For our tests we use both real world and artificial instances of the problem. In particular, we use CAIDA IPv4 topology dataset [12] and Erdös–Rényi random graphs [6].

CAIDA (Cooperative Association for Internet Data Analysis), is an association which provides data and tools for the analysis of the Internet infrastructure.

The CAIDA dataset is collected by a globally distributed set of monitors. The monitors collect data by sending probe messages continuously to destination IP addresses. Destinations are selected randomly from each routed IPv4 /24 prefix on the Internet such that a random address in each prefix is probed approximately every 48 hours (one probing cycle). The current prefix list includes approximately 7.4 million prefixes. In the current configuration, probes are made by sending ICMP packets. For each destination selected, the path from the source monitor to the destination is collected, in particular, data collected for each path probed includes:

- the set of IP addresses of the hops which form the path;
- the Round Trip Times (RTT), of both intermediate hops and the destination.

We parsed the files provided by CAIDA in order to obtain a weighted undirected graph G_{IP} where a node represents an IP address contained in the dataset (both source/destination hosts and intermediate hops), edges represent links among hops and weights are given by RTTs.

As the graph G_{IP} consists of $n \approx$ 50e+03 nodes, we cannot use it for the experiments. In fact, the amount of memory required to store the routing tables of all the nodes is $O(n^2)$. Hence, we performed our tests in subgraphs of G_{IP} induced by the settled nodes of a breadth first search starting from a node taken at random.

We generated a set of different tests, where a test consists of a dynamic graph characterized by: a subgraph of G_{IP} of 1000 nodes, a set of k concurrent edge updates, where k assumes values in $\{5, 10, 15, 20\}$. For the decremental tests, an edge update consists of increasing the edge weight of a random selected edge by a percentage value randomly chosen in [110%, 150%], while for the incremental tests weights are decreased by a percentage value randomly chosen in [50%, 90%]. For each test configuration – i.e. a subgraph of G_{IP} and a set of k modification – we performed 5 different experiments and we report average values.

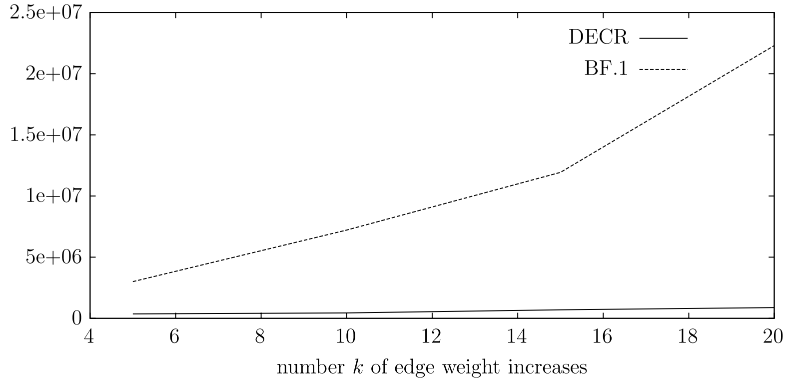

Fig. 9. Number of messages sent by DECR and BF.1 on subgraphs of G_{IP}.

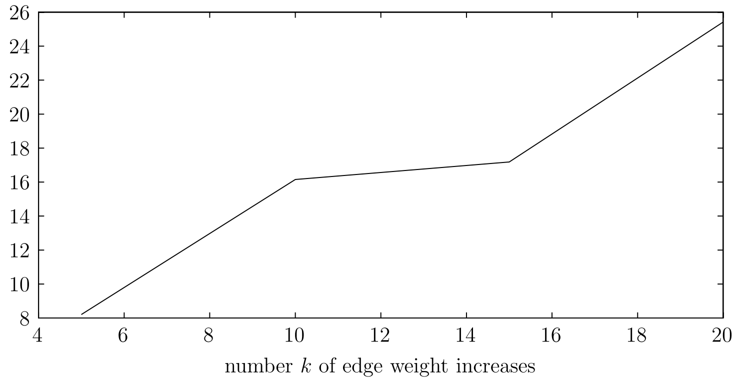

Fig. 10. Ratio between the number of messages sent by BF.1 and DECR on subgraphs of G_{IP}.

The graph G_{IP} turns out to be very sparse (i.e. $m/n \approx 1.5$), so it is worth analyzing denser graphs. To this aim we generated Erdös–Rényi random graphs. In detail, we randomly generated a set of different tests, where a test consists of a dynamic graph characterized by:

- an Erdös–Rényi random graphs G_{random} of 1000 nodes;
- *dens*, the density of the graph. It is computed as the ratio between m and the number of the edges of the n-complete graph;
- k, the number of edge update operations.

We chosen different values of *dens* ranging from 0.01 to 0.41. The number k assumes values in $\{30, 100\}$. Edge weights are non-negative real numbers randomly chosen in $[1, 10e+03]$. Edge updates are randomly chosen as in the CAIDA tests. For each test configuration – i.e. a graph G_{random}, a value of density *dens*, and a set of k modification – we performed 5 different experiments and we report average values.

Decremental algorithm. In Fig. 9, we report the number of messages sent by algorithms DECR and BF.1 on subgraphs of graph G_{IP} with 1000 nodes and an average value of 1411 edges. Fig. 9 shows that DECR always performs better than BF.1. In particular, it always sends less messages than BF.1 and, according to Example 3.12 in Section 3, the gap increases with k due to concurrent executions of the algorithms.

Fig. 10 shows the same results as Fig. 9 from a different point of view, that is, it shows the ratio between the number of messages sent by BF.1 and DECR in the same settings as Fig. 9. It is worth noting that the ratio is more than 8 in the worst cases, i.e. when $k = 5$, and it increases with k reaching the value of 25.5 in the cases when $k = 20$.

To conclude our analysis, we give the space occupancy per node of DECR and BF.1. BF.1 requires a node v to store, for each destination, the estimated distance given by each of its neighbors, while DECR only needs the estimated distance of v and the set `via`, for each destination. Since in these sparse graphs it is not common to have more than one via to a destination, the size to store the routing table for DECR is much smaller than the size required by BF.1.

In particular, DECR requires in average 8000 bytes per node and 8020 bytes per node in the worst case. BF.1 requires in average 9644 bytes per node and $740e+03$ bytes per node in the worst case. This implies that DECR is in average 1.20 times more space efficient than BF.1 and it is 92.27 times more space efficient than BF.1 in the worst case.

The good performances of DECR are mainly due to the sparsity of the graphs used. In fact, DECR uses two kind of messages: *increase* and *get-dist*. The former is sent only when a node v changes its routing table and it is used to propagate this changing, while the latter is just used by v in order to know the estimated distances of its neighbors. Hence, the number of *get-dist*

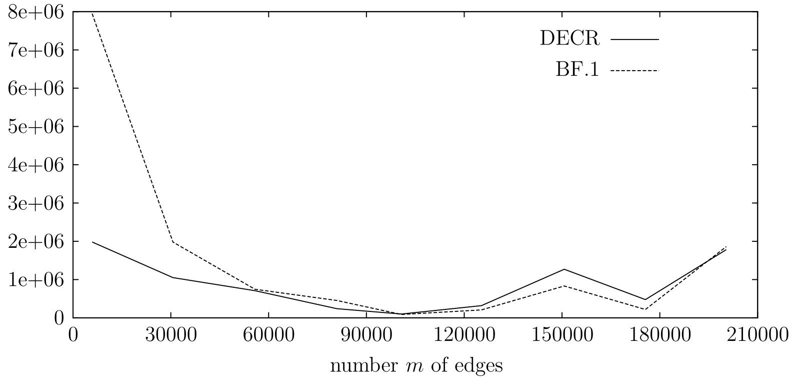

Fig. 11. Number of messages sent by DECR and BF.1 on graphs G_{random}.

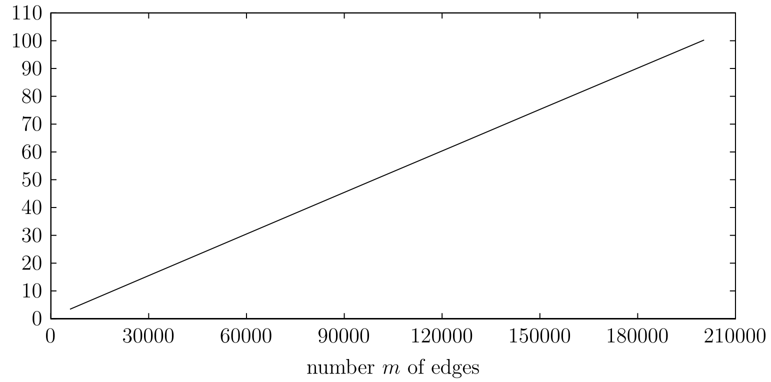

Fig. 12. Ratio between the space required by BF.1 and DECR in the average case on graphs G_{random}.

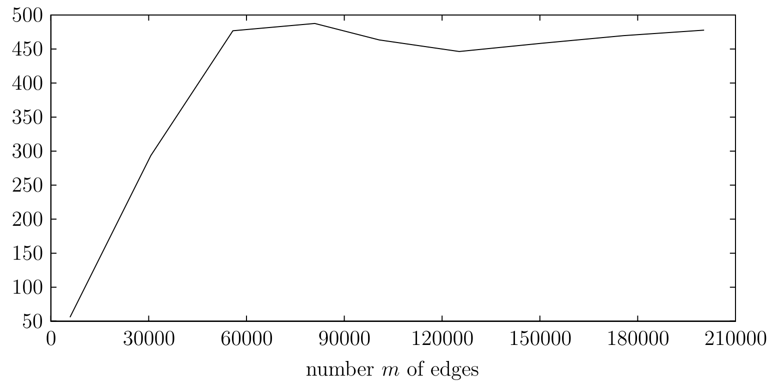

Fig. 13. Ratio between the space required by BF.1 and DECR in the worst case on graphs G_{random}.

messages is proportional to the average node degree of a graph. Note that, BF.1 does not need to use *get-dist* messages as it stores, for each node, the estimated distances of its neighbors. Hence, in sparse graphs, where the average degree of a graph is small, the number of *get-dist* messages sent by DECR is also small and this implies that, in these cases, DECR sends less messages than BF.1.

By the above discussion, it is worth investigating how the two algorithms perform when the graph is denser. To this aim, Fig. 11 shows the number of messages sent by algorithms DECR and BF.1 on Erdös-Rényi random dynamic graphs with 1000 nodes, 30 edge weight increases and *dens* ranging from 0.01 to 0.41 which leads to a number *m* of edges which ranges from about 5000 to about 200e+03. The number of messages sent by DECR is less than the number of messages sent by BF.1 when the number of edges is less than 100e+03. In most of the cases when the number of edges is more than 100e+03, BF.1 is slightly better than DECR. This is due to the fact that DECR does not require a node to store the estimated distances of its neighbors but it sends a *get-dist* message to each neighbor (see Line 7 of Procedure Increase). Hence, the number of *get-dist* messages is high when the average number of neighbors is high. Contrarily, BF.1 does not need to send such messages as it stores for each node v the estimated distances of each neighbor of v. This implies an increase in the space occupancy of BF.1 as highlighted by Figs. 12 and 13. In detail, Fig. 12 shows the ratio between the average space occupancy per node required

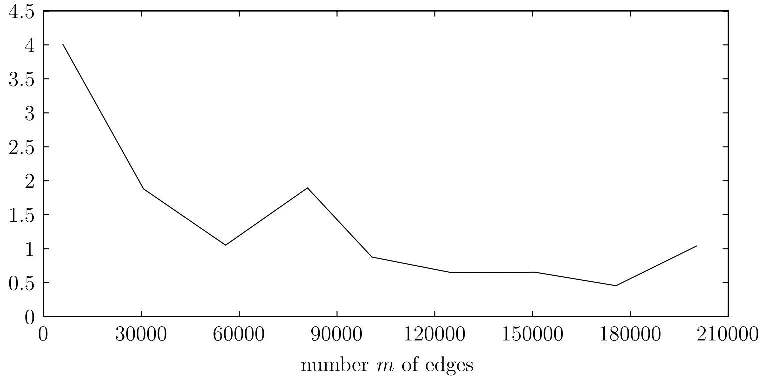

Fig. 14. Ratio between the number of messages sent by BF.1 and DECR on graphs G_{random}.

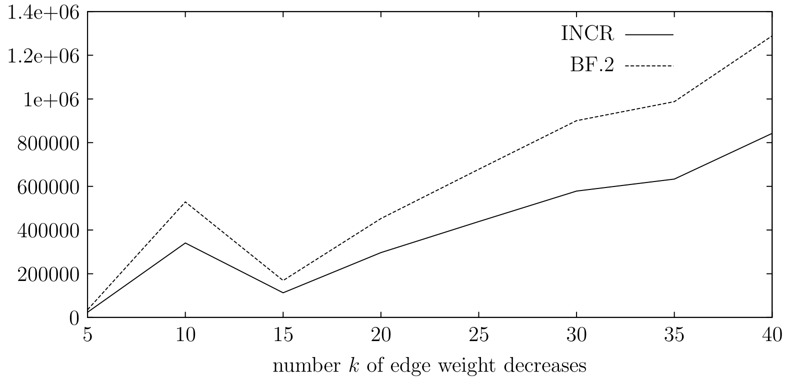

Fig. 15. Number of messages sent by INCR and BF.2 on subgraphs of G_{IP}.

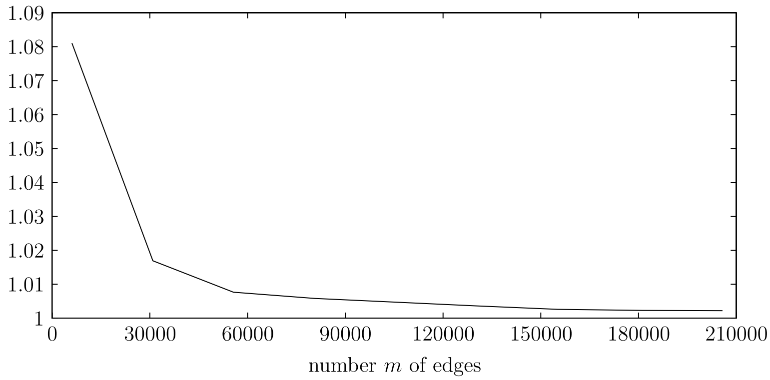

Fig. 16. Ratio between the number of messages sent by BF.2 and INCR on graphs G_{random}.

by BF.1 and DECR while Fig. 13 shows the ratio between the worst case space occupancy per node required by BF.1 and DECR. The average space occupancy ratio grows linearly with the number of edges as the space occupancy of DECR remains almost constant while the space occupancy of BF.1 is proportional to the average node degree. The worst case space occupancy of BF.1 grows very fast as in the executed tests where *dens* > 0.10 there exists at least a node v such that $\deg(v) = n - 1$.

A different point of view is given in Fig. 14 which shows the ratio between the number of messages sent by BF.1 and DECR in the same settings as Fig. 11. Note that, the ratio is about 4 in the sparse graphs and it decreases until it assumes approximately the value of 1 for dense graphs.

Figs. 11–14 refer to the case when $k = 30$, the case when $k = 100$ is similar and hence it is not reported.

Incremental algorithm. The space required by INCR and BF.2 is the same, then we focus only on the number of messages sent by the two algorithms.

Figs. 15 and 16 show the performances of INCR and BF.2 in subgraphs of G_{IP} and in Erdös-Rényi random graphs, respectively.

In Fig. 15 we can see that the number of messages sent by INCR is always smaller than the number of messages sent by BF.2. In particular, the number of messages sent by BF.2 is between 1.47 and 1.55 times greater than the number of messages sent by INCR.

The same behavior can be observed for Erdös–Rényi random graphs but in this case the ratio between the number of messages sent by BF.2 and INCR is smaller when the graph is denser as we can see in Fig. 16 where such ratio is about 1.08 in the best cases and it tends to be 1 for dense graphs. Fig. 16 refers to the case when $k = 30$, the case when $k = 100$ is similar and hence it is not reported.

6. Conclusions and future work

Most of the solutions known in the literature for the dynamic distributed *all-pairs shortest paths* problem suffer of two main drawbacks:

- they are not able to update shortest paths *concurrently* when multiple edge changes occur in the network. In fact, many algorithms work under the assumption that before dealing with an edge operation, the algorithm for the previous operation has to be terminated. This is a limitation in real networks, where edge changes can occur in an unpredictable way;
- they are able to *concurrently* update shortest paths but, (i) either they suffer of the looping and counting phenomenons, or (ii) their convergence can be very slow in the case of *weight increase* operations (possibly infinite).

In this paper we have provided *partially dynamic* solutions that are able to concurrently update shortest paths. In detail:

1. We have proposed a new robust decremental algorithm which is able to concurrently update shortest paths in the case of multiple *weight increase* and *delete* operations. The algorithm requires $O(maxdeg \cdot n)$ space per node and can suffer of the looping phenomenon. However, this algorithm has been shown to be experimentally efficient when compared with two different implementations of the classical Bellman–Ford method.
2. We have proposed an extension of the incremental algorithm given in [7] for *weight decrease* and *insert* operations that works also in the *concurrent* case, within the same bounds of [7], that is $O(maxdeg \cdot \Delta)$ messages per operation and $O(n)$ space per node. Here, Δ is the number of nodes affected by a set of *weight decrease/insert* operations. This is only a factor *maxdeg* far from the optimal incremental solution. Besides being theoretically efficient, this algorithm has been shown to be also experimentally faster than two different implementations of the classical Bellman–Ford method.

Furthermore, in real cases the concurrent executions of the algorithms of this paper for two (or more) *weight decrease/insert* or *weight increase/delete* operations allows us to deliver a number of messages that is much smaller than the number of messages delivered in the sequential case. An example is given for the decremental algorithm in Section 3, an analogous example could be easily provided for the incremental algorithm. This considerations have been confirmed by our experimental study.

The main future research direction is that of finding efficient and practical solutions for the more realistic fully dynamic case of the problem at hand. Along this line we are working on the extension to the fully dynamic case of the partially dynamic solutions given in this paper.

Acknowledgements

Support for the IPv4 Routed/24 Topology Dataset is provided by the National Science Foundation, the US Department of Homeland Security, the WIDE Project, Cisco Systems, and CAIDA Members.

We also wish to thank an anonymous referee for the useful comments that contributed to improve the paper, and for pointing out Example 3.11 and a mistake in a previous version of this paper.

References

[1] OMNeT++: The discrete event simulation environment. http://www.omnetpp.org/.
[2] H. Attiya, J. Welch, Distributed Computing, John Wiley and Sons, 2004.
[3] B. Awerbuch, A. Bar-Noy, M. Gopal, Approximate distributed Bellman–Ford algorithms, IEEE Transactions on Communications 42 (8) (1994) 2515–2517.
[4] B. Awerbuch, I. Cidon, S. Kutten, Communications-optimal maintenance of replicated information, in: Proc. IEEE Symposium on Foundation of Computer Science, 1990, pp. 492–502.
[5] D. Bertsekas, R. Gallager, Data Networks, Prentice Hall International, 1992.
[6] B. Bollobás, Random Graphs, Cambridge University Press, 2001.
[7] S. Cicerone, G.Di. Stefano, D. Frigioni, U. Nanni, A fully dynamic algorithm for distributed shortest paths, Theoretical Computer Science 297 (1–3) (2003) 83–102.
[8] G. D'Angelo, S. Cicerone, G.Di. Stefano, D. Frigioni, Partially dynamic concurrent update of distributed shortest paths, in: Proc. of the Int. Conference on Computing: Theory and Application, IEEE Press, 2007.
[9] C. Demetrescu, G.F. Italiano, A new approach to dynamic all pairs shortest paths, Journal of ACM 51 (6) (2004) 968–992.
[10] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni, Fully dynamic algorithms for maintaining shortest paths trees, Journal of Algorithms 34 (2) (2000) 251–281.
[11] P.A. Humblet, Another adaptive distributed shortest path algorithm, IEEE Transactions on Communications 39 (6) (1991) 995–1002.
[12] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, K.C. Claffy, The CAIDA IPv4 routed /24 topology dataset. http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml.

[13] G.F. Italiano, Distributed algorithms for updating shortest paths, in: Proceedings Int. Workshop on Distributed Algorithms, in: Lecture Notes in Computer Science, vol. 579, 1991, pp. 200–211.
[14] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, 1996.
[15] J. McQuillan, Adaptive routing algorithms for distributed computer networks, Technical Report BBN Report 2831, Cambridge, MA 1974.
[16] J.T. Moy, Ospf — Anatomy of an Internet Routing Protocol, Addison-Wesley, 1998.
[17] A. Orda, R. Rom, Distributed shortest-path and minimum-delay protocols in networks with time-dependent edge-length, Distributed Computing 10 (1996) 49–62.
[18] G. Ramalingam, T. Reps, On the computational complexity of dynamic graph problem, Theoretical Computer Science 158 (1996) 233–277.
[19] K.V.S. Ramarao, S. Venkatesan, On finding and updating shortest paths distributively, Journal of Algorithms 13 (1992) 235–257.
[20] E.C. Rosen, The updating protocol of arpanet's new routing algorithm, Computer Networks 4 (1980) 11–19.
[21] A.S. Tanenbaum, Computer Networks, Prentice Hall, 1996.