A New Approach to Dynamic All Pairs Shortest Paths

CAMIL DEMETRESCU

Università di Roma "La Sapienza", Rome, Italy

AND

GIUSEPPE F. ITALIANO

Università di Roma "Tor Vergata", Rome, Italy

Abstract. We study novel combinatorial properties of graphs that allow us to devise a completely new approach to dynamic all pairs shortest paths problems. Our approach yields a fully dynamic algorithm for general directed graphs with non-negative real-valued edge weights that supports any sequence of operations in $O(n^2 \log^3 n)$ amortized time per update and unit worst-case time per distance query, where n is the number of vertices. We can also report shortest paths in optimal worst-case time. These bounds improve substantially over previous results and solve a long-standing open problem. Our algorithm is deterministic, uses simple data structures, and appears to be very fast in practice.

Categories and Subject Descriptors: E.1 [**Data Structures**]: *Graphs and networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computation on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms: Algorithms

Additional Key Words and Phrases: Dynamic graph algorithms, shortest paths

---

This work has been partially supported by the Sixth Framework Programme of the EU under contract number 507613 (Network of Excellence "EuroNGI: Designing and Engineering of the Next Generation Internet"), by the IST Programme of the EU under contract n. IST-1999-14186 (ALCOM-FT), by the HPRN Programme of the EU under contract n. HPRN-CT-1999-00104 (AMORE), and by the Italian Ministry of University and Research (Project "ALINWEB: Algorithmics for Internet and the Web").

A preliminary version of this paper was presented at the 35th Annual ACM Symposium on Theory of Computing (STOC'03).

The final remarks of the conference version claimed that an $O(n^2 \log^2 n)$ update bound could be achieved for fully dynamic all pairs shortest paths; however, this claim was supported with inaccurate arguments.

Authors' addresses: C. Demetrescu, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy, e-mail: demetres@dis.uniroma1.it. URL: http://www.dis.uniroma1.it/∼demetres; G. F. Italiano, Dipartimento di Informatica Sistemi e Produzione, Università di Roma "Tor Vergata", Roma, Italy and Centro "Vito Volterra", e-mail: italiano@disp.uniroma2.it, URL: http:// www.disp.uniroma2.it/users/italiano/.

1. *Introduction*

In this article, we present fully dynamic algorithms for maintaining all pairs shortest paths (APSP) in directed graphs with *real-valued* edge weights. A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a weighted graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. Note that edge deletions and edge insertions can be easily formulated as edge weight updates, by setting to $+\infty$ the weight of edges not in the graph. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and must perform update operations faster than recomputing the solution from scratch, as carried out by the fastest static algorithm. We say that an algorithm is *fully dynamic* if it can handle both edge weight increases and edge weight decreases. A *partially dynamic* algorithm can handle either edge weight increases or decreases, but not both.

1.1. THE PROBLEM. In the fully dynamic APSP problem that we consider, we wish to maintain a directed graph with real-valued edge weights under an intermixed sequence of the following operations:

`update`(v, w'):	update the weights of all edges incident to vertex v according to weight function w'.
`distance`(x, y):	return the distance from vertex x to vertex y.
`path`(x, y):	report a shortest path from vertex x to vertex y, if any.

Notice that, in this article, we consider a generalized version of the dynamic all pairs shortest path problem where the weights of all edges incident to a given vertex can be changed with just one update operation. In the following, we will call such operation *vertex update*. We recall that, in this setting, edge deletions can be modeled by raising edge weights to $+\infty$, while edge insertions can be realized by decreasing edge weights from $+\infty$ to a finite value. Throughout the article, we denote by n the number of vertices in the graph and by m the number of edges with weights $< +\infty$ in the graph.

1.2. PREVIOUS WORK. The dynamic maintenance of shortest paths has a remarkably long history, as the first articles date back to over 35 years ago [Loubal 1967; Murchland 1967; Rodionov 1968]. After that, many dynamic shortest paths algorithms have been proposed (see e.g., Even and Gazit [1985], Frigioni et al. [1998, 2000], Ramalingam and Reps [1996a, 1996b], and Rohnert [1985]), but their running times in the worst case were comparable to recomputing APSP from scratch.

The first dynamic shortest path algorithms that are provably faster than recomputing APSP from scratch only worked on graphs with small integer weights. In particular, Ausiello et al. [1991] proposed a decrease-only shortest path algorithm for directed graphs having positive integer weights less than C: the amortized running time of their algorithm is $O(Cn \log n)$ per edge insertion. Henzinger et al. [1997] designed a fully dynamic algorithm for APSP on planar graphs with integer weights, with a running time of $O(n^{4/3} \log(nC))$ per operation. Recently, Fakcharoemphol and Rao [2001] designed a fully dynamic algorithm for maintaining single-source shortest paths in planar directed graphs that supports both queries and edge weight updates in $O(n^{4/5} \log^{13/5} n)$ amortized time per edge operation.

The first big step on general graphs and integer weights was made by King [1999], who presented a fully dynamic algorithm for maintaining all pairs shortest

paths in directed graphs with positive integer weights less than C: the running time of her algorithm is $O(n^{2.5}\sqrt{C \log n})$ per update. In previous work, [Demetrescu and Italiano 2001, 2002], we have presented fully dynamic APSP on general directed graphs with real weights. In particular, given a directed graph G, subject to dynamic operations, and such that each edge weight can take at most S different *real* values, we proposed a deterministic algorithm that supports each update in $O(n^{2.5}\sqrt{S \log^3 n})$ amortized time and each query in $O(1)$ worst-case time. Other deletions-only algorithms for APSP, in the simpler case of unweighted graphs, are presented in Baswana et al. [2002].

1.3. OUR RESULTS. We study novel combinatorial properties of graphs that allow us to devise a completely new approach to dynamic all pairs shortest paths. This approach yields a fully dynamic algorithm for APSP on general directed graphs with nonnegative real-valued edge weights that achieves the following time bounds: it supports any sequence of operations in $O(n^2 \log^3 n)$ amortized time per update and one look-up in the worst case per distance query; it can also report shortest paths in optimal worst-case time. We remark that our algorithm improves substantially over previous bounds [Demetrescu and Italiano 2001, 2002; King 1999]. Furthermore, and unlike all the previous approaches, it solves fully dynamic APSP in its generality. Indeed, it runs on (nonnegative) real weights, and each weight has no limit on the number of different values it can take. Finally, we note that when the distance matrix has to be maintained explicitly, that is, distance queries have to be answered with exactly one look-up, as many as $\Omega(n^2)$ entries of the distance matrix can change during each update. Thus, in this model our algorithm is only a polylogarithmic factor away from the best possible bound. In the special case of increase-only update sequences, our techniques yield a faster update algorithm that runs in $O(n^2 \log n)$ amortized time per operation. Similarly to the fully dynamic case, no previous general solution was known for this problem.

Another interesting feature of our techniques is that both weight increases and weight decreases can be supported with exactly the same code. Surprisingly, our algorithms are rather simple and thus amenable to efficient implementations: indeed, in accordance with a recent experimental study [Demetrescu et al. 2004], the techniques described in this article are not only asymptotically efficient, but can yield very fast implementations in many practical scenarios.

1.4. NOTATION. Let $G = (V, E)$ be a directed graph with real edge weights and no negative-length cycles. A path $\pi_{xy} = \langle x_0, x_1, \ldots, x_k \rangle$ from vertex x to vertex y in G is a sequence of vertices such that $x_0 = x$, $x_k = y$, and $(x_i, x_{i+1}) \in E$, for each i, $0 \leq i < k$. Let (u, v) be an edge in E: we denote by w_{uv} the weight of edge (u, v). Let $\pi_{xy} = \langle x_0, x_1, \ldots, x_k \rangle$ be a path in G: we denote by

$$w(\pi_{xy}) = \sum_{i=0}^{k-1} w_{x_i x_{i+1}}$$

the weight of π_{xy}. We assume that as a special case, $\pi_{xx} = \langle x \rangle$ is a path of weight zero.

Given $\pi_{xv} = \langle x, \ldots, x', v \rangle$ and $\pi_{vy} = \langle v, y', \ldots, y \rangle$, we denote by $\pi_{xv} \cdot \pi_{vy}$ the path $\langle x, \ldots, x', v, y', \ldots, y \rangle$ obtained by concatenating π_{xv} and π_{vy} at v. Moreover, we denote by $\ell(\pi_{xy})$ the path π_{xb} such that $\pi_{xy} = \pi_{xb} \cdot \langle b, y \rangle$. Similarly, we denote by $r(\pi_{xy})$ the path π_{ay} such that $\pi_{xy} = \langle x, a \rangle \cdot \pi_{ay}$.

TABLE I. NOTATION USED IN THE ARTICLE

$G = (V, E)$	weighted directed graph with vertex set V and edge set E
w_{uv}	weight of edge (u, v)
$\pi_{xy} = \langle x, \ldots, y \rangle$	path from vertex x to vertex y
$w(\pi_{xy})$	weight of path π_{xy} (sum of the weights of the edges in π_{xy})
$\ell(\pi_{xy})$	subpath π_{xb} of π_{xy} such that $\pi_{xy} = \pi_{xb} \cdot \langle b, y \rangle$
$r(\pi_{xy})$	subpath π_{ay} of π_{xy} such that $\pi_{xy} = \langle x, a \rangle \cdot \pi_{ay}$
$\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$	sequence of update operations
t_σ	time at which update σ occurs
v_σ	vertex affected by update σ

Let the graph G be subject to a sequence of (vertex) updates $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$. We denote by t_{σ_i} the time immediately after update σ_i, with $t_{\sigma_i} = i$ for any i, $1 \leq i \leq k$. We also denote by v_σ the vertex affected by the update σ: that is, if $\sigma = \texttt{update}(x, w')$, then $v_\sigma = x$. The notation used in this article is summarized in Table I.

Throughout the article, we assume that there is only one shortest path between each pair of vertices in G. This is, without loss of generality, since ties can be broken consistently as we will discuss in Section 3.4.

1.5. ORGANIZATION OF THE ARTICLE. The remainder of this article is organized as follows. Section 2 studies some properties of special classes of paths in a graph, while Section 3 shows how to exploit them to devise the first general increase-only update algorithm for all pairs shortest paths. To deal with fully dynamic sequences, Section 4 addresses more path properties, which are next used in Section 5 to devise the first general algorithm for fully dynamic all pairs shortest paths. We remark that, while Section 2 and Section 4 contain combinatorial results on graphs, algorithmic aspects are treated in Section 3 and Section 5. We conclude the article in Section 6 with some remarks and directions for further research.

2. *Locally Shortest Paths*

In this section, we study the properties of a class of paths in a graph that we call *locally shortest paths*. Using this notion, in Section 3, we will show how to maintain efficiently all pairs shortest paths in a graph subject to partially dynamic edge weight updates. Locally shortest paths are defined as follows:

Definition 2.1. A path π_{xy} is LOCALLY SHORTEST[1] in G if either:

(i) π_{xy} consists of a single vertex, or

(ii) every proper subpath of π_{xy} is a shortest path in G.

This definition is inspired by the optimal-substructure property of shortest paths: all subpaths of a shortest path are shortest paths. Here, we relax this property by considering only *proper* subpaths. Indeed, in a locally shortest path, all proper subpaths are shortest paths: however, the path itself may not necessarily be shortest. We assume that every trivial path formed by a single vertex or a single edge is locally

[1] Since every internal vertex of a locally shortest path has the same sum of distances to the endpoints, in an earlier version of this article we called such paths *uniform paths*.

shortest. Notice that, by the optimal substructure of shortest paths, it is possible to check whether a nontrivial path π_{xy} is locally shortest by just verifying that $\ell(\pi_{xy})$ and $r(\pi_{xy})$ are shortest paths in G. In the remainder of this section we discuss some properties of locally shortest paths.

LEMMA 2.2. *If we denote by SP and LSP respectively the sets of shortest paths and locally shortest paths in G, then SP* $\subseteq$ *LSP.*

PROOF. Every subpath of a shortest path is a shortest path itself. Thus, every shortest path is trivially a locally shortest path. □

LEMMA 2.3. *If shortest paths are unique in G, then for each pair of vertices x and y, the locally shortest paths connecting x and y in G are internally vertex-disjoint, that is, except for the endpoints, they use different vertices.*

PROOF. Suppose by contradiction that there exist two distinct locally shortest paths π_{xy}^1 and π_{xy}^2 that are not internally vertex-disjoint. This means that there is some vertex v, with $v \neq x$ and $v \neq y$, that belongs to both π_{xy}^1 and π_{xy}^2. Since shortest paths are unique, then there is only one shortest path π_{xv} from x to v, and only one shortest path π_{vy} from v to y. Since every proper subpath of π_{xy}^1 and π_{xy}^2 is shortest, then π_{xv} and π_{vy} are necessarily subpaths of both π_{xy}^1 and π_{xy}^2. Thus, $\pi_{xy}^1 = \pi_{xy}^2$, which contradicts our initial assumption. □

LEMMA 2.4. *If shortest paths are unique in G, then there can be at most mn locally shortest paths in G.*

PROOF. Fix an edge (x, v) and a vertex y in G. We first note that there can be at most one locally shortest path $\pi_{xy} = \langle x, v, \ldots, y\rangle$ starting from edge (x, v). This derives from the fact that every proper subpath of π_{xy} must be shortest (Definition 2.1), and from the uniqueness of shortest paths. Since the first edge (x, v) can be chosen in m different ways and the destination vertex y can be chosen among n different vertices, at any time there can be at most mn locally shortest paths in G. □

We now study how the set of locally shortest paths changes in a graph subject to partially dynamic updates such as vertex increases, that is, operations that increase the weights of all edges incident to it, or vertex decreases, that is, operations that decrease the weights of all edges incident to it.

LEMMA 2.5. *Let G be a graph subject to a sequence* Σ *of vertex updates. If shortest paths are unique in G, then in the worst case at most* $O(n^2)$ *paths can* stop *being locally shortest due to a vertex increase.*

PROOF. We observe that a path can stop being locally shortest only if any of its proper subpaths stops being shortest. In case of increases, this can happen only if that subpath contains the updated vertex, say vertex v. By Lemma 2.3, there can be at most $O(n^2)$ locally shortest paths that contain v as an internal vertex. Furthermore, there can be at most $O(n^2)$ locally shortest paths starting or ending in v. This yields a total of at most $O(n^2)$ paths that can stop being locally shortest because of a weight increase in v. □

THEOREM 2.6. *Let G be a graph subject to a sequence* Σ *of increase-only vertex update operations and let m be the maximum number of edges in G throughout*

sequence Σ. *If shortest paths are unique in* G, *then the number of paths that* start *being locally shortest after each update is:*

(1) $O(mn)$ *in the worst case.*

(2) $O(n^2)$ *amortized over* $\Omega(m/n)$ *update operations.*

PROOF. Claim (1) follows immediately from Lemma 2.4. To prove Claim (2), we assign a debit to each locally shortest path in the graph. The debit is paid for by the operation that makes it stop being locally shortest. By Lemma 2.5, at most $O(n^2)$ paths can stop being locally shortest at each update. Moreover, by Lemma 2.4 there can be at most mn locally shortest paths in a graph at any time, so the total unpaid debit at the end of the sequence never exceeds mn. Thus, the amortized number of paths that start being locally shortest after each update in any sequence of $\Omega(m/n)$ operations can be at most $O(n^2)$. □

Notice that both the statements and the proofs of Lemma 2.5 and Theorem 2.6 hold symmetrically if we replace "increase" by "decrease" and swap "start" with "stop". This can be intuitively explained by observing that, if we replay a decrease-only sequence backwards starting from the final graph, each vertex decrease in the forward sequence corresponds to a symmetric vertex increase in the backward sequence that undoes edge weights back to their previous values. In this scenario, a path starts being locally shortest during a decrease in the forward sequence if and only if it stops being locally shortest during the corresponding increase in the backward sequence, and thus the counting argument holds in both directions.

3. *Partially Dynamic Shortest Paths*

We now show how to exploit the properties of locally shortest paths discussed in Section 2 to devise an increase-only update algorithm for all pairs shortest paths that runs in $O(n^2 \log n)$ amortized time per operation. To the best of our knowledge, this is the first general result for increase-only all pairs shortest paths that is faster than recomputing the solution from scratch after each update. This is rather surprising compared to the decrease-only case, where an $O(n^2)$ bound can be immediately obtained by just running a single-source computation from the updated vertex v to every other vertex, and a single-sink computation from every vertex to v; by doing so, for each pair of vertices x and y, we find a shortest path π^*_{xv} from x to v, and a shortest path π^*_{vy} from v to y: if $\pi'_{xy} = \pi^*_{xv} \cdot \pi^*_{vy}$ is shorter than the previous shortest path π_{xy} from x to y, we simply replace π_{xy} with π'_{xy}.

Although the update algorithm that we describe below works also for decreases, for the sake of simplicity in this section we analyze the algorithm in the case of increases only. The approach is very simple: we maintain all the locally shortest paths of the underlying graph. By Theorem 2.6, changes in the data structure will be $O(n^2)$ per update in an increase-only sequence of $\Omega(m/n)$ operations. We will maintain locally shortest paths in priority queues, and thus we will pay $O(\log n)$ for each path that starts/stops being locally shortest. This will yield an $O(n^2 \log n)$ amortized time per update. Even though the combinatorial results discussed in Section 2 impose no restrictions on the edge weights (provided that shortest paths are unique) our algorithm requires that all the edge weights in the graph are non-negative.

TABLE II. NOTATION INTRODUCED IN SECTION 3.1

P_{xy}	set of locally shortest paths from x to y in G
P^*_{xy}	set of shortest paths from x to y in G
$L(\pi_{xy})$	set of pre-extensions $\langle x', x\rangle \cdot \pi_{xy}$ of π_{xy} that are locally shortest paths in G
$L^*(\pi_{xy})$	set of pre-extensions $\langle x', x\rangle \cdot \pi_{xy}$ of π_{xy} that are shortest paths in G
$R(\pi_{xy})$	set of post-extensions $\pi_{xy} \cdot \langle y, y'\rangle$ of π_{xy} that are locally shortest paths in G
$R^*(\pi_{xy})$	set of post-extensions $\pi_{xy} \cdot \langle y, y'\rangle$ of π_{xy} that are shortest paths in G

```
    distance(x, y):
1.      if P_xy = ∅ then return +∞
2.      else return the weight of a minimum weight path in P_xy
    path(x, y):
1.      if P_xy = ∅ then return ∅
2.      else return a minimum weight path in P_xy
```

FIG. 1. Implementation of `distance` and `path` operation.

3.1. DATA STRUCTURE. For each pair of vertices x and y in G, we maintain the weight $w_{xy} \geq 0$ of edge (x, y) (or $+\infty$ if no such edge exists) and the following two data structures:

$$P_{xy} = \{\pi_{xy} : \pi_{xy} \text{ is a locally shortest path in } G\}$$
$$P^*_{xy} = \{\pi_{xy} : \pi_{xy} \text{ is a shortest path in } G\}.$$

We maintain each P_{xy} as a priority queue where item $\pi_{xy} \in P_{xy}$ has priority $w(\pi_{xy})$. We note that, if shortest paths are unique, $|P^*_{xy}| \leq 1$. Furthermore, since by Lemma 2.2 any shortest path is locally shortest, then for each pair of vertices x and y, $P^*_{xy} \subseteq P_{xy}$. Therefore, a minimum weight path in P_{xy} is a shortest path. We also observe that each path π_{xy} in P_{xy} and P^*_{xy} can be represented implicitly with constant space by just storing two pointers to the subpaths $\ell(\pi_{xy})$ and $r(\pi_{xy})$. This is correct by the optimal-substructure property of locally shortest paths and by the assumption of uniqueness of shortest paths. Finally, for each path π_{xy} stored in P_{xy} we maintain $w(\pi_{xy})$ and the following four lists:

$$\begin{aligned}
L(\pi_{xy}) &= \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} : (x', x) \in E \text{ and } \pi_{x'y} \text{ is a loc. shortest path in } G\}\\
L^*(\pi_{xy}) &= \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} : (x', x) \in E \text{ and } \pi_{x'y} \text{ is a shortest path in } G\}\\
R(\pi_{xy}) &= \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle : (y, y') \in E \text{ and } \pi_{xy'} \text{ is a loc. shortest path in } G\}\\
R^*(\pi_{xy}) &= \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle : (y, y') \in E \text{ and } \pi_{xy'} \text{ is a shortest path in } G\}.
\end{aligned}$$

In other words, $L(\pi_{xy})$ and $L^*(\pi_{xy})$ represent pre-extensions of π_{xy}, while $R(\pi_{xy})$ and $R^*(\pi_{xy})$ represent post-extensions of π_{xy}. Once again, by Lemma 2.2, any shortest path is locally shortest, and thus for each path π_{xy} stored in P_{xy}, $L^*(\pi_{xy}) \subseteq L(\pi_{xy})$ and $R^*(\pi_{xy}) \subseteq R(\pi_{xy})$. For the sake of simplicity, in the following, we will sometimes write P or P^* instead of P_{xy} or P^*_{xy} whenever the meaning is clear from the context. The notation introduced in this section is summarized in Table II.

3.2. IMPLEMENTATION OF OPERATIONS. The `distance`(x, y) and `path`(x, y) operations can be implemented as shown in Figure 1, by simply accessing the minimum weight path in P_{xy}. Since each P_{xy} is a subset of the paths in G, the correctness of query operations follows directly from Lemma 2.2.

The implementation of the `update`(v, w') operation is shown in Figure 2. The update works in two steps: `cleanup` and `fixup`. To simplify the description, we

```
update(v, w′):
1.    cleanup(v)
2.    fixup(v, w′)
```

```
cleanup(v):
1.    Q ← {⟨v⟩}
2.    while Q ≠ ∅ do
3.       extract any π from Q
4.       for each π_xy ∈ L(π) ∪ R(π) do
5.          add π_xy to Q
6.          remove π_xy from P_xy, L(r(π_xy)), and R(ℓ(π_xy))
7.          if π_xy ∈ P*_xy then remove π_xy from P*_xy, L*(r(π_xy)), and R*(ℓ(π_xy))
```

```
fixup(v, w′):
1.    for each u ≠ v do                                              {Phase 1}
2.       w_uv ← w′_uv;  w_vu ← w′_vu
3.       if w_uv < +∞ then
4.          w(⟨u, v⟩) ← w_uv;  ℓ(⟨u, v⟩) ← ⟨u⟩;  r(⟨u, v⟩) ← ⟨v⟩
5.          add ⟨u, v⟩ to P_uv, L(⟨v⟩), and R(⟨u⟩)
6.       if w_vu < +∞ then
7.          w(⟨v, u⟩) ← w_vu;  ℓ(⟨v, u⟩) ← ⟨v⟩;  r(⟨v, u⟩) ← ⟨u⟩
8.          add ⟨v, u⟩ to P_vu, L(⟨u⟩), and R(⟨v⟩)
9.    H ← ∅                                                          {Phase 2}
10.   for each (x, y) do
11.      add π_xy ∈ P_xy with minimum w(π_xy) to H
12.   while H ≠ ∅ do                                                 {Phase 3}
13.      extract π_xy from H with minimum w(π_xy)
14.      if π_xy is the first extracted path for pair (x, y) then
15.         if π_xy ∉ P*_xy then
16.            add π_xy to P*_xy, L*(r(π_xy)), and R*(ℓ(π_xy))
17.            for each π_x′b ∈ L*(ℓ(π_xy)) do
18.               π_x′y ← ⟨x′, x⟩ · π_xy
19.               w(π_x′y) ← w_x′x + w(π_xy);  ℓ(π_x′y) ← π_x′b;  r(π_x′y) ← π_xy
20.               add π_x′y to P_x′y, L(π_xy), R(π_x′b), and H
21.            for each π_ay′ ∈ R*(r(π_xy)) do
22.               π_xy′ ← π_xy · ⟨y, y′⟩
23.               w(π_xy′) ← w(π_xy) + w_yy′;  ℓ(π_xy′) ← π_xy;  r(π_xy′) ← π_ay′
24.               add π_xy′ to P_xy′, L(π_ay′), R(π_xy), and H
```

FIG. 2. Implementation of the `update` operation.

say that a path that is shortest (resp., locally shortest) after the update is *new* either if it was not shortest (resp., locally shortest) before the update, or if it contains the updated vertex v. We now describe procedures `cleanup` and `fixup` in more detail; pseudo-code is given in Figure 2.

`cleanup`(v)

The procedure removes every path π_{xy} containing vertex v from P_{xy}, P^*_{xy}, $L(r(\pi_{xy}))$, $L^*(r(\pi_{xy}))$, $R(\ell(\pi_{xy}))$, and $R^*(\ell(\pi_{xy}))$. Namely, we remove from the data structure all the paths that would stop being locally shortest if we deleted v from the graph. This task can be accomplished iteratively by first removing paths of the form $\langle u, v\rangle$ and $\langle v, u\rangle$, and then by removing all paths listed in $L(\pi_{xy})$ and $R(\pi_{xy})$ for each path π_{xy} removed in the previous iterations.

`fixup`(v, w')

The procedure adds to the data structure all the new shortest and locally shortest paths. It works in three phases, as follows.

Phase 1. Sets the weight of every edge (u, v) entering v to the new value w'_{uv} (line 2). If $w'_{uv} < +\infty$, then the trivial path $\pi_{uv} = \langle u, v\rangle$ is added to P_{uv}, $L(r(\pi_{uv}))$, and $R(\ell(\pi_{uv}))$ (lines 3–5). Similar steps are performed for every edge of the form (v, u) (line 2 and lines 6–8). Thus, all the new locally shortest paths formed by one edge are added to the data structure. Longer new paths will be added in *Phase* 3.

Phase 2. Initializes a priority queue H with the minimum weight path $\pi_{xy} \in P_{xy}$ for each pair of vertices (x, y) (lines 9–11);

Phase 3. Repeatedly extracts paths π_{xy} from H in increasing weight order (line 13). The first extracted path for each pair (x, y) is a shortest path between x and y (see Invariant 3.1 below), while paths extracted later on for the same pair are ignored (line 14). Whenever a shortest path π_{xy} is extracted, we check whether π_{xy} is already in P^*_{xy} (line 15). If not, we add π_{xy} to P^*_{xy}, $L^*(r(\pi_{xy}))$, and $R^*(\ell(\pi_{xy}))$ (line 16), and we combine it with existing shortest paths to form new locally shortest paths (lines 17–24). This is done by scanning all paths $\pi_{x'b}$ listed in $L^*(\ell(\pi_{xy}))$ (line 17) and all paths $\pi_{ay'}$ listed in $R^*(r(\pi_{xy}))$ (line 21), forming the new locally shortest paths $\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy}$ (lines 18–19) and $\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle$ (lines 22–23). Each new locally shortest path π_{ij} is added to P_{ij}, $L(r(\pi_{ij}))$, $R(\ell(\pi_{ij}))$, and H as soon as it is discovered (line 20 and line 24).

3.3. ANALYSIS. To prove the correctness of `update`, we assume that P and P^* are correct before the operation, and we show that they are also correct afterwards. Operations on the other data structures are simple bookkeeping operations and their correctness can be easily checked. We first discuss an invariant maintained by procedure `fixup`.

INVARIANT 3.1. *If shortest paths are unique and edge weights are non-negative, then for each pair of vertices x and y in G, the first path connecting x and y extracted from H in* Phase 3 *of* `fixup` *is a shortest path.*

PROOF. Suppose by contradiction that at some extraction the invariant is violated, and the first path $\widehat{\pi}_{xy}$ extracted for some pair (x, y) is not a shortest path. Consider the earliest of these events, and let π_{xy} be the unique shortest path between x and y, with $w(\pi_{xy}) < w(\widehat{\pi}_{xy})$. Clearly, $\pi_{xy} \notin H$, otherwise it would have been extracted in place of $\widehat{\pi}_{xy}$. Moreover, $\pi_{xy} \notin P_{xy}$, otherwise it would have been inserted in H in *Phase* 2, being a shortest path. Therefore, π_{xy} has to be necessarily a new locally shortest path, but since, in *Phase* 1, we add to P all the edges incident to v,

it cannot be one of them. This implies that π_{xy} contains at least two edges and either one of $\ell(\pi_{xy})$ or $r(\pi_{xy})$ is a new shortest path and was not in P^* at the beginning of `fixup`. As edge weights are non-negative, then $w(\ell(\pi_{xy})) \leq w(\pi_{xy}) < w(\widehat{\pi}_{xy})$ and $w(r(\pi_{xy})) \leq w(\pi_{xy}) < w(\widehat{\pi}_{xy})$. Since paths are extracted from H in increasing weight order, shortest paths are unique, and all the extractions before the wrong extraction are correct, then either $\ell(\pi_{xy})$ or $r(\pi_{xy})$ should have been extracted from H and added to P^* before the wrong extraction. This implies that π_{xy} should have been formed and inserted in H at the extraction time of either one of $\ell(\pi_{xy})$ or $r(\pi_{xy})$. So π_{xy} should have been extracted before $\widehat{\pi}_{xy}$, contradicting the assumption that the invariant is violated. □

THEOREM 3.2. *If the operation is an increase, shortest paths are unique, and edge weights are non-negative, then algorithm* `update` *correctly updates* P_{xy} *and* P^*_{xy} *for each pair of vertices* x *and* y.

PROOF. We first observe that `cleanup`(v) removes from the data structures every locally shortest path that contains the updated vertex v. Thus, since only paths containing v can stop being shortest after an increase, then every path π_{xy} that is no longer locally shortest after the update is removed from P_{xy}, and possibly from P^*_{xy}.

We now show that every new locally shortest path π_{xy} is added to P_{xy} by `fixup`(v, w'). Paths π_{uv} and π_{vu} made of one edge are correctly added to P_{uv} and P_{vu} in *Phase* 1: therefore, we only focus on paths π_{xy} with at least two edges. We recall that each such path π_{xy} has to be added to P_{xy} if both $\ell(\pi_{xy})$ and $r(\pi_{xy})$ are shortest paths after the update, but at least one of them (say $\ell(\pi_{xy})$) was not in P^* before `fixup`. By Invariant 3.1, the first extracted path for each pair is shortest, and then $\ell(\pi_{xy})$ is certainly extracted and added to P^* at some iteration in *Phase* 3. A that time, $\ell(\pi_{xy})$ is correctly combined with $r(\pi_{xy}) \in R^*(r(\ell(\pi_{xy})))$ to form π_{xy}, which is added to P_{xy}.

To conclude the proof, we observe that by Invariant 3.1 all shortest paths are extracted at some iteration and are added to P^*, if they are not already there. □

To conclude the analysis, we address the running time and the space usage of our implementation.

THEOREM 3.3. *In an increase-only sequence of* $\Omega(m/n)$ *operations, if shortest paths are unique and edge weights are non-negative, our data structure supports each* `update` *operation in* $O(n^2 \log n)$ *amortized time, and each* `distance` *and* `path` *query in optimal time. The space used is* $O(mn)$.

PROOF. The bounds for queries are immediate, since the minimum weight path in the priority queue P_{xy} can be retrieved in optimal time.

Since each iteration of `cleanup` removes a path π_{xy} from a constant number of lists and priority queues, it requires $O(\log n)$ time in the worst case. By Lemma 2.3, at most $O(n^2)$ locally shortest paths can go through the updated vertex in the worst case, and thus `cleanup` will perform at most $O(n^2)$ iterations, thus spending at most $O(n^2 \log n)$ time.

Next, it is easy to see that *Phase* 1 of `fixup` requires $O(n \log n)$ time in the worst case. To bound the time required by *Phase* 2 of `fixup`, we consider that finding the minimum in P_{xy} takes constant time and each insertion in H takes $O(\log n)$ time. This gives a total bound of $O(n^2 \log n)$ for this phase. Finally,

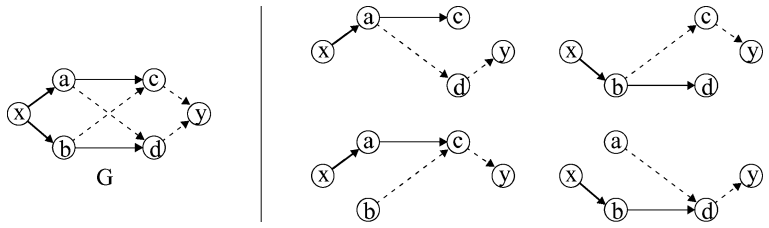

FIG. 3. If shortest paths are not unique, using an arbitrary tie-breaking strategy may lead to pairs of vertices connected in G, but disconnected in P^*. In this example, if P^* contains $\langle x, a, c\rangle$, $\langle x, b, d\rangle$, $\langle a, d, y\rangle$, and $\langle b, c, y\rangle$ as the unique shortest paths of length 2, then no locally shortest path between x and y can be obtained as the union of any two of them.

we note that a path π_{xy} is a new locally shortest path if and only if both $\ell(\pi_{xy}) = \langle x, \ldots, b\rangle \in P^*_{xb}$ and $r(\pi_{xy}) = \langle a, \ldots, y\rangle \in P^*_{ay}$ *after* the update, and either $\ell(\pi_{xy}) \notin P^*_{xb}$ or $r(\pi_{xy}) \notin P^*_{ay}$ *before* calling `fixup`. Thus, since in *Phase* 3 we process a shortest path π_{xy} extracted from H only if it was not already in P^*_{xy} before calling `fixup`, then we spend time only for the new locally shortest paths. This implies that *Phase* 3 requires $O(n^2 \log n)$ time for extracting the $O(n^2)$ pairs which are initially in H, plus $O(\log n)$ time to insert/delete each new locally shortest path in H and P: let p the number of such paths. By Theorem 2.6, there can be $O(n^2)$ paths that start being locally shortest after each update, amortized over an increase-only sequence of $\Omega(m/n)$ operations. Moreover, by Lemma 2.3, there can be at most $O(n^2)$ locally shortest paths that were removed by `cleanup` since they contain the updated vertex, and are added back by `fixup`: thus, $p = O(n^2)$ when amortized over $\Omega(m/n)$ operations and `update` requires $O(n^2 \log n)$ overall time.

At any time, each locally shortest path π_{xy} can only appear in a constant number of data structures: namely, in H, P_{xy}, $L(r(\pi_{xy}))$, $R(\ell(\pi_{xy}))$, and possibly in P^*_{xy}, $L^*(r(\pi_{xy}))$, $R^*(\ell(\pi_{xy}))$. Thus, the space usage of our data structure is bounded by the number of locally shortest paths in a graph, which is $O(mn)$ in the worst case by Lemma 2.4. □

By Theorem 3.2, algorithm `update` maintains correctly the locally shortest paths of the graph, and thus the shortest paths, under increase-only update sequences. As we will see in Section 5, the above algorithm is indeed correct also in case of decreases, even if it does not maintain P and P^* as they were defined in Section 3.1: specifically, in case of decreases, paths that stop being shortest may not be removed from the data structure by `cleanup`, and thus P^* might contain paths that were shortest in the past, even if they are no longer shortest. In Section 4, we will call such "old" paths *historical* and show that they can play a crucial role in dealing with fully dynamic sequences.

3.4. BREAKING TIES. If shortest paths in the graph are not unique, the properties of locally shortest paths studied in Section 2 do not hold. Furthermore, using an arbitrary tie-breaking strategy may lead to incorrect results. Consider the example of Figure 3, where all edges have weight 1, and suppose that at some point the unique shortest paths of length 2 in P^* are: $\langle x, a, c\rangle$, $\langle x, b, d\rangle$, $\langle a, d, y\rangle$, and $\langle b, c, y\rangle$. Notice that we can find no path of length 3 such that every subpath of length 2 is in P^*: therefore, no path of length 3 can be formed by `update`, leaving the pair (x, y) incorrectly disconnected in P^*. In order `update` to be correct, we thus need

a tie-breaking strategy that both ensures uniqueness and leads to a complete set of shortest paths closed under subpaths.

How can we break ties consistently? A first simple approach is to add some tiny random fraction to the weight of each edge: in this way, we can make the probability of finding two paths with the same weight arbitrarily small.

A more robust deterministic tie-breaking approach is the following. Without loss of generality, assume that $V = \{1, 2, \ldots, n\}$. We assign to each edge (u, v) a unique number $ID(u, v)$ (e.g., $ID(u, v) = u + nv$), and for each path π we define $ID(\pi)$ as the maximum ID of its edges. This allows us to define the *extended weight* $\mathrm{ew}(\pi)$ of a path π as follows.

Definition 3.4. For any path π with at least one edge we define the extended weight of π as $\mathrm{ew}(\pi) = \langle w(\pi),\ ID(\pi)\rangle$, where $w(\pi)$ is the real weight of π in the graph and $ID(\pi)$ is defined as follows:

$$ID(\pi) = \begin{cases} u + nv & \text{if } \pi = \langle u, v\rangle \\ \max\{ID(\ell(\pi)),\ ID(r(\pi))\} & \text{otherwise.} \end{cases}$$

The extended weights of two paths π_1 and π_2 can be compared "lexicographically" in constant time by assuming that:

$$\mathrm{ew}(\pi_1) \leq \mathrm{ew}(\pi_2) \Leftrightarrow w(\pi_1) < w(\pi_2)\ \textit{or}\ w(\pi_1) = w(\pi_2)\ \textit{and}\ ID(\pi_1) \leq ID(\pi_2).$$

We now show that using extended weights we can break ties consistently so that exactly one shortest path (if any) between each pair of vertices can be identified in a deterministic way. To do so, we define the following set of paths in a graph:

Definition 3.5. Let G be a graph with real-valued edge weights and let ew be the extended weight function of Definition 3.4. We define S_{ew} as follows:

$$S_{\mathrm{ew}} = \{\pi \in G\ :\ \forall \pi_{xy} \subseteq \pi,\ \forall \pi'_{xy} \in G,\ \mathrm{ew}(\pi_{xy}) \leq \mathrm{ew}(\pi'_{xy})\}.$$

According to this definition, S_{ew} contains every path in G that has minimum extended weight and whose subpaths have minimum extended weight as well (optimal-substructure property). As a consequence, S_{ew} is closed under subpaths, that is, every subpath of a path in S_{ew} belongs to S_{ew}. It is easy to prove that there can be only one path in S_{ew} between each pair of vertices.

LEMMA 3.6. *For each pair of vertices x and y in G, there can be at most one path $\pi_{xy} \in S_{\mathrm{ew}}$ connecting them.*

PROOF. The proof is by induction on the number of edges in the paths. The basis is trivially satisfied, since there can be only one path formed by one edge between each pair of vertices. Now, assume by induction that the claim holds for every path that uses at most k edges, and suppose by contradiction that there are two different paths $\pi_{xy} \in S_{\mathrm{ew}}$ and $\pi'_{xy} \in S_{\mathrm{ew}}$ that use at most $k+1$ edges. Since their extended weights are equal, then both π_{xy} and π'_{xy} must contain the edge (u, v) such that $ID(u, v) = ID(\pi_{xy}) = ID(\pi'_{xy})$. Since every subpath of π_{xy} and π'_{xy} belongs to S_{ew} and uses at most k edges, then π_{xy} and π'_{xy} must have the same subpaths from x to u and from v to y. Thus, π_{xy} and π'_{xy} must be equal, contradicting the assumption that they are different. □

Clearly, if $\pi_{xy} \in S_{\text{ew}}$, then π_{xy} is a shortest path in G. We now show that, if there is a shortest path from x to y in G, then there must be a path $\pi_{xy} \in S_{\text{ew}}$.

LEMMA 3.7. *For each pair of connected vertices x and y in G, there is a path $\pi_{xy} \in S_{\text{ew}}$.*

PROOF. Assume that y is reachable from x in G, $x \neq y$. Then, we show how to construct a path π^*_{xy} that satisfies Definition 3.5. We start with an empty π^*_{xy} and construct it one edge at the time. Let π_{xy} be a path from x to y with minimum extended weight in G and let (u, v) be the edge with maximum ID in π_{xy}, that is, such that $ID(u, v) = ID(\pi_{xy})$. We put (u, v) in π^*_{xy} and build recursively the subpath π^*_{xu} of π^*_{xy} if $x \neq u$ and the subpath π^*_{vy} of π^*_{xy} if $v \neq y$.

We now show that the above procedure constructs paths that belong to S_{ew}. We prove this claim proceeding by induction on the path length. Let k be the number of edges in π^*_{xy}. The base for $k = 1$ ($\pi^*_{xy} = \langle x, y \rangle$) is trivial. Assume by induction that the claim holds for every path containing less than k edges and assume by contradiction that there is a subpath $\pi_{ab} \in \pi^*_{xy}$ that is not of minimum extended weight. Three cases are possible:

(1) both vertices a and b are in π^*_{xu};
(2) both vertices a and b are in π^*_{vy};
(3) vertex a is in π^*_{xu} and vertex b is in π^*_{vy}.

Case (1) and (2) are not possible, since π^*_{xu} and π^*_{vy} have strictly less than k edges, and thus by induction are in S_{ew}. Assume that we are in case (3). Since π_{ab} is not of minimum extended weight, there must be a path π'_{ab} such that $\text{ew}(\pi'_{ab}) < \text{ew}(\pi_{ab})$. Note that, by construction π^*_{xy} contains only edges that lie on some shortest path from x to y in G, and thus $w(\pi_{ab}) \leq w(\pi'_{ab})$. Consequently, it must be $w(\pi_{ab}) = w(\pi'_{ab})$ and $ID(\pi'_{ab}) < ID(\pi_{ab})$. This implies that there is a shortest path from x to y avoiding edge (u, v), with an ID smaller than $ID(u, v)$. This contradicts the fact that $ID(u, v)$ is the ID of a path with minimum extended weight from x to y. □

By Lemma 3.6 and Lemma 3.7, S_{ew} contains exactly one representative shortest path between each pair of vertices connected in G. Thus, we can conclude that $|S_{\text{ew}}| \leq n^2$ and S_{ew} is a complete set of unique shortest paths in G. For this reason, in the remainder of this article we will assume that we can always ensure uniqueness of shortest paths by dealing with paths in S_{ew}.

We now show that using the deterministic approach described above we can break ties consistently and ensure both completeness and uniqueness of shortest paths, avoiding inconsistencies like the one shown in Figure 3. As a matter of fact, we can easily modify the code of the `update` operation given in Figure 2 so that at any time during a sequence of updates, $P^* = S_{\text{ew}}$. Specifically, we keep an additional field $ID(\pi)$ for each path π stored in the data structure, and perform the following additional steps in `fixup` to maintain it as in Definition 3.4:

Line 4	$ID(\langle u, v \rangle) \leftarrow u + nv$
Line 7	$ID(\langle v, u \rangle) \leftarrow v + nu$
Line 19	$ID(\pi_{x'y}) \leftarrow \max\{ID(\pi_{x'b}),\ ID(\pi_{xy})\}$
Line 23	$ID(\pi_{xy'}) \leftarrow \max\{ID(\pi_{xy}),\ ID(\pi_{ay'})\}$.

Furthermore, we let the priority of any path π in P and H be $\text{ew}(\pi)$. Thus, in line 11 and line 13 of `fixup` we pick paths with minimum ew, instead of

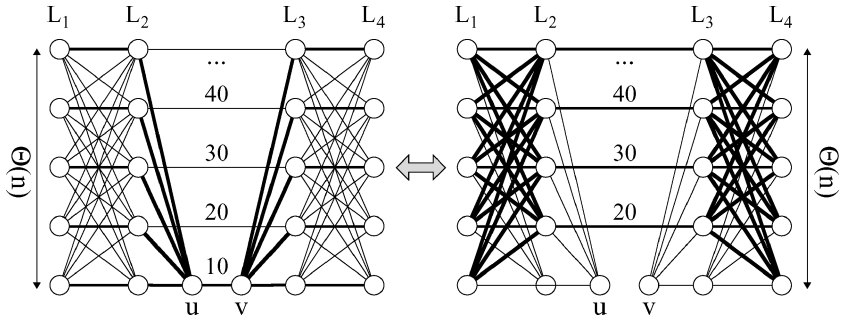

FIG. 4. Pathological update sequence consisting of repeated deletion and re-insertion of edge (u, v). Each insertion/deletion causes $\Theta(n^3)$ changes in the set of locally shortest paths. Locally shortest paths connecting vertices in layer L_1 to vertices in layer L_4 are highlighted as thick lines. Edges are directed from left to right and, unless explicitly shown, their weights are small values close to zero.

minimum w. To check the correctness of this approach, we prove the following theorem.

THEOREM 3.8. *If we compare paths according to the extended weight function* ew *given in Definition* 3.4*, then* $P^* = S_{\text{ew}}$ *after each execution of* `update`.

PROOF. We first observe that for any path π, if edge weights in the graph are non-negative reals, the extended weight function ew given in Definition 3.4 satisfies the monotonic property $\text{ew}(\ell(\pi)) \leq \text{ew}(\pi)$ and $\text{ew}(r(\pi)) \leq \text{ew}(\pi)$. Moreover, by Lemma 3.6 and Lemma 3.7 there is exactly one path in S_{ew} between each pair of connected vertices. Therefore, it is easy to check that Invariant 3.1 continues to hold if we compare paths according to ew instead of w: this implies that each path added to P^* has minimum extended weight.

Furthermore, by construction, paths in P^* are closed under subpaths, that is, every subpath of a path in P^* is also in P^*: thus, $P^* = S_{\text{ew}}$ as in Definition 3.5. □

We conclude this section by observing that each increase (respectively decrease) of an edge weight implies an increase (respectively decrease) of the corresponding extended weight. Furthermore, we notice that comparing extended weights still takes constant time. Thus, both Theorem 3.2 and Theorem 3.3 continue to hold using extended weights instead of the original weights in the graph.

4. *Historical and Locally Historical Paths*

Figure 4 shows a graph and a pathological fully dynamic update sequence such that each operation causes $\Theta(n^3)$ changes in the set of locally shortest paths of the graph. This proves that maintaining locally shortest paths in a fully dynamic setting can require as much as $\Omega(n^3)$ time per operation. To deal with fully dynamic update sequences efficiently, in this section, we introduce other two classes of paths that we call *historical paths* and *locally historical paths*: historical paths will play the role of shortest paths, while locally historical paths will play the role of locally shortest paths. Unlike shortest paths and locally shortest paths, which only depend on the graph at a given time, these classes of paths capture the history of the changes in the graph, encompassing the time dimension of the problem. We define them as follows:

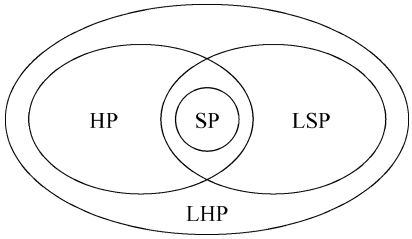

FIG. 5. Inclusions between shortest paths (SP), locally shortest paths (LSP), historical paths (HP), and locally historical paths (LHP).

Definition 4.1. Let π_{xy} be a path in G at time t, and let $t' \leq t$ be the time of the latest vertex update on π_{xy}. We say that π_{xy} is HISTORICAL at time t if it has been a shortest path at least once in the time interval $[t', t]$.

In accordance with this definition, a shortest path is also a historical path. Although it may stop being a shortest path at some point, it keeps on being a historical path until a vertex update occurs on it.

Definition 4.2. We say that a path π_{xy} is LOCALLY HISTORICAL[2] in G at time t if either:

(i) π_{xy} consists of a single vertex, or
(ii) every proper subpath of π_{xy} is a historical path in G at time t.

Unlike locally shortest paths, as defined in Definition 2.1, proper subpaths do not have to be shortest paths necessarily at the same time. Indeed, in a locally historical path, all proper subpaths have been shortest paths in the past; however, the path itself may have never been locally shortest.

The next lemma addresses the relationship between shortest paths, historical paths, and locally historical paths:

LEMMA 4.3. *If we denote by SP, HP, and LHP respectively the sets of shortest paths, historical paths, and locally historical paths in G at any time in a sequence of updates, then the following inclusions hold at that time: SP* $\subseteq$ *HP* $\subseteq$ *LHP.*

PROOF. The inclusion *SP* $\subseteq$ *HP* is straightforward from Definition 4.1. Let π be a historical path at time t and let $t' \leq t$ be the time of the latest vertex update on π. For every proper subpath $\widehat{\pi} \subset \pi$, let $\hat{t} \leq t' \leq t$ be the time of the latest vertex update on $\widehat{\pi}$. By Definition 4.1, π has been a shortest path in the time interval $[t', t]$. By the optimal-substructure property of shortest paths, $\widehat{\pi}$ has been a shortest path in the time interval $[\hat{t}, t] \supseteq [t', t]$, and thus it is historical at time t. Consequently, *HP* $\subseteq$ *LHP*. □

Figure 5 summarizes the relationship between shortest paths, locally shortest paths, historical paths, and locally historical paths. As we will see later on, the reason why we are considering locally historical paths is that they can be maintained very efficiently in a sequence of fully dynamic updates. Since by Lemma 4.3 locally

[2] In an earlier version of this article, we used terminology *potentially uniform paths* instead of *locally historical paths*.

historical paths include shortest paths, this implies that we can maintain information about dynamic shortest paths very efficiently. Historical paths, on the other hand, are only used as a tool for defining locally historical paths.

4.1. PROPERTIES OF LOCALLY HISTORICAL PATHS. In this section, we study combinatorial properties of locally historical paths, addressing their dependence on the number of historical paths in the graph. These properties will be crucial in Section 5 to designing our efficient algorithm for fully dynamic shortest paths. We start by analyzing the maximum number of paths that can be locally historical at each time during a sequence of updates.

LEMMA 4.4. *Let G be a graph subject to a sequence Σ of update operations. If at any time throughout the sequence Σ there are at most z historical paths between each pair of vertices and m edges in G, then at that time there can be at most zmn locally historical paths in G.*

PROOF. We follow the same argument given in the proof of Lemma 2.4. To build a locally historical path $\pi_{xy} = \langle x, u, \ldots, y \rangle$, the first edge (x, u) can be chosen in m different ways, while the destination vertex y can be chosen among n different vertices. Since every proper subpath of π_{xy} has to be historical and there are at most z historical paths between each pair of vertices, there can be at most z historical subpaths between u and y. Thus, there can be at most zmn locally historical paths in G. □

Another interesting question is how many locally historical paths can go through any given vertex in the graph. Since a path can stop being locally historical only when it is touched by a vertex update, this yields a worst-case bound on the number of paths that can stop being locally historical at each update. To address this, we first need to prove that, given any two distinct historical paths between any pair of vertices x and y, their latest updated vertices must be different.

LEMMA 4.5. *Let G be a graph subject to a sequence Σ of update operations and let x, y be any two vertices in G. If shortest paths are unique and we denote by $\sigma(\pi)$ the latest vertex update on path π, then for any two distinct historical paths π_{xy} and π'_{xy} in G at any time t, $v_{\sigma(\pi_{xy})} \neq v_{\sigma(\pi'_{xy})}$.*

PROOF. Assume by contradiction that $v_{\sigma(\pi_{xy})} = v_{\sigma(\pi'_{xy})}$, and thus $t_{\sigma(\pi_{xy})} = t_{\sigma(\pi'_{xy})}$. By Definition 4.1, both π_{xy} and π'_{xy} have to be shortest paths at least once in the time interval $[t_{\sigma(\pi_{xy})}, t]$, but this is impossible since neither of them is touched in the time interval $(t_{\sigma(\pi_{xy})}, t]$, and at each time there can be only one shortest path between x and y in G. □

LEMMA 4.6. *Let G be a graph subject to a sequence Σ of update operations and let v be any vertex in G. If shortest paths are unique and at any time t throughout the sequence Σ, there are at most z historical paths between each pair of vertices, then at time t there can be at most $O(zn^2)$ locally historical paths that contain v.*

PROOF. We first show that there can be at most $O(zn^2)$ locally historical paths having v as endpoint. Indeed, to form a locally historical path $\pi_{vy} = \langle v, x, \ldots, y \rangle$, we can choose x and y in at most $O(n^2)$ different ways; since every proper subpath of π_{vy} has to be historical and there are at most z historical paths between each

pair of vertices, there can be at most z historical subpaths between x and y. The argument for paths ending in v is completely analogous.

To complete the proof, it suffices to show that for each pair of vertices x, y, with $x \neq v$ and $y \neq v$, there can be at most $O(z)$ locally historical paths of the form $\langle x, \ldots, v, \ldots, y \rangle$, that is, having v as an internal vertex. Denote by P the set of such paths, and let $Q = \{\pi_1, \ldots, \pi_q\}$ be the subset of P such that for each $\pi_i \in Q$, v follows (or coincides with) v_i in π_i, where v_i is the latest updated vertex in $\pi_i - \{x, y\}$.

Consider now the set of subpaths $\widehat{Q} = \{\widehat{\pi}_i = \langle x, \ldots, v_i, \ldots, v \rangle \subset \pi_i, \pi_i \in Q\}$. We first claim that $|Q| = |\widehat{Q}|$. To prove this claim, we show that any two distinct paths π_i and π_j in Q must give rise to distinct subpaths $\widehat{\pi}_i \subset \pi_i$ and $\widehat{\pi}_j \subset \pi_j$ in $\widehat{Q}$. Assume by contradiction that this is not the case, that is, $\pi_i \neq \pi_j$ and $\widehat{\pi}_i = \widehat{\pi}_j$, which implies that $v_i = v_j$. Consider the subpaths of π_i and π_j from v_i to y: those paths must be historical since π_i and π_j are locally historical and $v_i \neq x$, and must share the same latest updated vertex, which can be either v_i or y. By Lemma 4.5, they must be equal, which implies $\pi_i = \pi_j$, clearly a contradiction.

Thus, it must be $|Q| = |\widehat{Q}|$. To conclude the proof of the lemma, suppose by contradiction that $|P| > 2z$, and assume without loss of generality that $|Q| \geq |P|/2$, and thus $|Q| > z$. By Definition 4.2, paths in $\widehat{Q}$ are historical at time t. Since $|Q| = |\widehat{Q}| > z$, we can conclude that at time t there are more than z historical paths between x and v in G, which contradicts the assumptions of the lemma. □

The next corollary, whose proof follows immediately from Definition 4.1, Definition 4.2, and Lemma 4.6, bounds the number of paths that stop being locally historical at each time during a sequence of updates:

COROLLARY 4.7. *Let G be a graph subject to a sequence Σ of update operations. If shortest paths are unique and at any time t throughout the sequence Σ, there are at most z historical paths between each pair of vertices, then the number of paths that stop being locally historical at each update is $O(zn^2)$ in the worst case.*

We finally bound the number of paths that become locally historical at each time during a sequence of updates:

THEOREM 4.8. *Let G be a graph subject to a sequence Σ of update operations and let m be the maximum number of edges in G throughout sequence Σ. If shortest paths are unique and at any time throughout the sequence Σ there are at most z historical paths between each pair of vertices, then the number of paths that become locally historical after each update is:*

(1) *$O(zmn)$ in the worst case.*
(2) *$O(zn^2)$ amortized over $\Omega(m/n)$ update operations.*

PROOF. Analogous to the proof of Lemma 2.6. Indeed, Claim (1) follows immediately from Lemma 4.4. To prove Claim (2), we assign a debit to each locally historical path in the graph. The debit is paid for by the operation that makes it stop being locally historical. By Corollary 4.7, at most $O(zn^2)$ paths can stop being locally historical at each update. Moreover, by Lemma 4.4, there can be at most zmn locally historical paths in a graph at any time, so the total unpaid debit at the end of the sequence never exceeds zmn. Thus, the amortized number of paths that

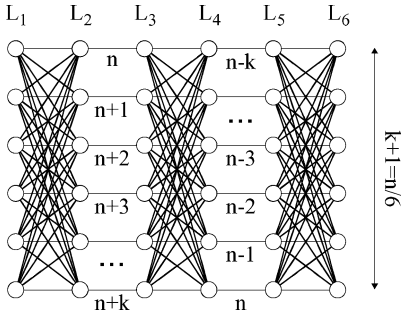

FIG. 6. Worst-case instance with $\Omega(n^3)$ historical paths and an amortized number of $\Omega(n^3)$ new locally historical paths per update.

start being locally shortest after each update in any sequence of $\Omega(m/n)$ operations can be at most $O(zn^2)$. □

4.2. A WORST-CASE SCENARIO. We now show that there exists a graph and an update sequence that produces $\Omega(n^3)$ historical paths. This implies by Theorem 4.8 an amortized number of $\Omega(n^3)$ new locally historical paths per update. Consider the graph in Figure 6, where $L_1 \ldots L_6$ are layers containing $n/6$ vertices each. The graphs induced by $L_1 \cup L_2$, $L_3 \cup L_4$ and $L_5 \cup L_6$ are complete bipartite graphs with edge weights equal to 1, while the graphs induced by $L_2 \cup L_3$ and $L_4 \cup L_5$ have weights shown below the corresponding edges. The update sequence is as follows:

Phase I. Perform the following sequence of $k = (n/6) - 1$ decreases on edges in $L_2 \cup L_3$:

1)	$n+1$	$\rightarrow$	$n-1$
2)	$n+2$	$\rightarrow$	$n-2$
3)	$n+3$	$\rightarrow$	$n-3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k)	$n+k$	$\rightarrow$	$n-k$

Phase II. Perform the following sequence of $k = (n/6) - 1$ increases on edges in $L_4 \cup L_5$:

1)	$n-k$	$\rightarrow$	$n+k$
2)	$n-(k-1)$	$\rightarrow$	$n+(k-1)$
3)	$n-(k-2)$	$\rightarrow$	$n+(k-2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k)	$n-1$	$\rightarrow$	$n+1$

During Phase I, $(k+1)^2$ paths connecting L_1 to L_5 become historical at each decrease, and thus at the end of Phase I there are $\Theta(k^3)$ historical paths in total. During Phase II, at each increase, k paths connecting each pair of vertices in L_1 and L_6 become locally historical in the graph. Since $k = (n/6) - 1$, each increase takes $\Omega(n^3)$ time. Notice that this update sequence can be made arbitrarily long by repeating Phase I and Phase II back and forth many times.

The example considered here suggests that keeping too much information about the history of changes in a dynamic graph could be prohibitive, that is, maintaining all the locally historical paths can lead to $\Omega(n^3)$ time per update. In contrast, in the example considered in Figure 4 we have shown that keeping no information about the history of changes, that is, keeping only locally shortest paths, can lead to cubic update times as well. In the next section, we will discuss an effective compromise between those two extreme solutions.

4.3. KEEPING HISTORICAL PATHS UNDER CONTROL: SMOOTHING. In this section, we show that it is possible to transform on-line every update sequence into another sequence so that the total number of historical paths in the graph is always very close to n^2. To guarantee that this transformation does not alter the input update sequence by affecting the result of shortest path queries, we make sure that it produces exactly the same changes in the graph as the input sequence. In the following, we will call such a transformation SMOOTHING. Furthermore, we will call *optimal* a historical path that is also a shortest path.

We first observe that at any time t, if shortest paths are unique, then there can be at most n^2 optimal historical paths. Any other historical path must have been optimal strictly before time t, and is no longer optimal at time t. By Definition 4.1, if we update a vertex, every nonoptimal historical path containing it will immediately stop being historical. To become again historical, it has to be a shortest path once again. Therefore, to reduce the number of historical paths, it is sufficient to add to the input update sequence suitable operations whose only effect is to clean up nonoptimal historical paths. For this reason, we will call these additional operations CLEANUP UPDATES. To guarantee that cleanup updates do not alter the input update sequence by affecting the result of shortest path queries, we simply let them overwrite edge weights without changing their values. We now define how our smoothing strategy works:

Definition 4.9. Let $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ be an update sequence and let

$$F(\Sigma) = \left\langle \sigma_1, \delta_1^1, \ldots, \delta_1^{q_1}, \sigma_2, \delta_2^1, \ldots, \delta_2^{q_2}, \ldots, \sigma_k, \delta_k^1, \ldots, \delta_k^{q_k} \right\rangle$$

be another sequence derived from Σ such that each δ_i^j is a cleanup update, and such that, if a vertex is updated by operation σ_t in Σ, it is touched by cleanup update δ_i^j in $F(\Sigma)$ when $i = t+1, t+2, t+4, t+8, \ldots, t+2^p, \ldots$, for some j. We call $F(\Sigma)$ a SMOOTHED SEQUENCE derived from Σ, we refer to $\langle \delta_i^1, \ldots, \delta_i^{q_i} \rangle$ as the BURST OF CLEANUP UPDATES AFTER σ_i, and we say that cleanup update $\delta_{t+2^p}^j$ is TRIGGERED BY THE ORIGINAL UPDATE σ_t.

Note that a smoothed sequence is obtained by squeezing a burst of cleanup updates between each pair of consecutive updates in the original sequence. Throughout the article, we assume that the burst of cleanup updates $\langle \delta_i^1, \ldots, \delta_i^{q_i} \rangle$ occurs exactly at the same time as update σ_i, that is, $t_{\delta_i^j} = t_{\sigma_i}$ for any j. We now show that a smoothed sequence is operationally equivalent to the sequence it is derived from, and produces a smaller number of historical paths.

THEOREM 4.10. *Let $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ be an update sequence of length k. Then the smoothed sequence*

$$F(\Sigma) = \left\langle \sigma_1, \delta_1^1, \ldots, \delta_1^{q_1}, \sigma_2, \delta_2^1, \ldots, \delta_2^{q_2}, \ldots, \sigma_k, \delta_k^1, \ldots, \delta_k^{q_k} \right\rangle$$

of Definition 4.9 *has the following properties:*

(1) *For $1 \le i \le k$, $q_i = O(\log k)$;*
(2) *For $1 \le i \le k$, after performing operation σ_i, the graph produced by Σ and the graph produced by $F(\Sigma)$ are equal;*
(3) *At any time there are at most $O(\log k)$ historical paths between each pair of vertices in the graph produced by $F(\Sigma)$.*

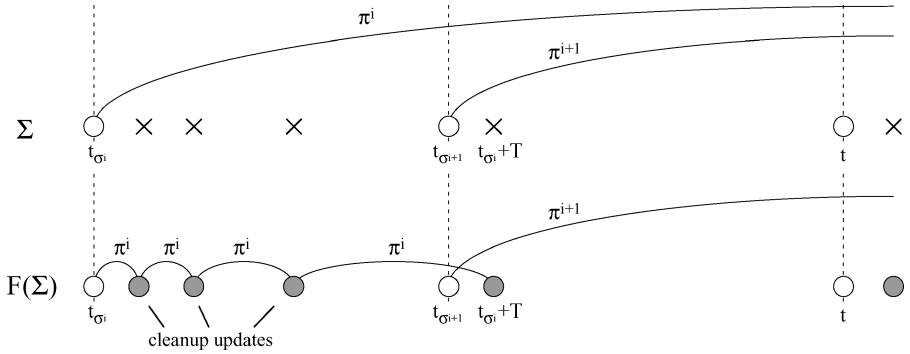

FIG. 7. Graphs of historical paths H_{xy} for an update sequence Σ and for the smoothed updated sequence $F(\Sigma)$. If $t_{\sigma^{i+1}} < t_{\sigma^i} + T$, then π^i cannot be historical at time t in the graph produced by $F(\Sigma)$.

PROOF. Properties 1 and 2 follow directly from Definition 4.9. To prove Property 3, we first note that a cleanup update on a vertex v does not create new historical paths: its only effect is to remove nonoptimal historical paths containing vertex v. For each pair of vertices (x, y) in G, consider the *graph of historical paths* H_{xy}. There is a vertex in H_{xy} for each update operation τ in $F(\Sigma)$ and there is an edge (τ_a, τ_b) in H_{xy}, with $t_{\tau_a} < t_{\tau_b}$, for each path π_{xy} in G having the following three properties:

(i) π_{xy} contains both v_{τ_a} and v_{τ_b} (possibly $v_{\tau_a} = v_{\tau_b}$);

(ii) no vertex in π_{xy} is updated in the time interval (t_{τ_a}, t_{τ_b});

(iii) π_{xy} becomes a shortest path at some time $t \in [t_{\tau_a}, t_{\tau_b})$.

It is easy to see that at any time at most one path π_{xy} in G can satisfy Properties (i), (ii) and (iii) above.

Let $\pi^1, \ldots, \pi^z$ be the historical paths between vertices x and y in the graph G obtained right after performing update σ_t in $F(\Sigma)$, that is, at time t. Let $(\tau^i, \hat{\tau}^i)$ be the edge in H_{xy} corresponding to π^i, $1 \le i \le z$. Since π^i is historical at time t, it must be $t_{\tau^i} \le t < t_{\hat{\tau}^i}$. If τ^i is a cleanup update, let σ^i be the original update that triggered τ^i, $t_{\sigma^i} < t_{\tau^i}$. Thus, to each historical path π^i we can associate an original update σ^i, with $t_{\sigma^1} < t_{\sigma^2} < \cdots < t_{\sigma^z} \le t$. Let σ^i and σ^{i+1} be two such updates, and let T be the largest power of 2 smaller than $t - t_{\sigma^i}$, that is, $T = 2^{\lfloor \log_2(t - t_{\sigma^i}) \rfloor}$. We claim that it must be $t_{\sigma^i} + T \le t_{\sigma^{i+1}}$.

Indeed, assume by contradiction that $t_{\sigma^{i+1}} < t_{\sigma^i} + T$ (see Figure 7). Since $T = 2^{\lfloor \log_2(t - t_{\sigma^i}) \rfloor}$ is a power of 2, by Definition 4.9, there is a cleanup update triggered by σ^i that touches a vertex of π^i at time $t_{\sigma^i} + T$: thus π^i cannot be historical at time $t_{\sigma^i} + T$ by Definition 4.1. Notice that, since π^{i+1} must be a shortest path between x and y after π^i, and no vertex of π^i and π^{i+1} is updated in the time interval $[t_{\sigma^{i+1}}, t]$, π^i cannot become again a shortest path between x and y in the time interval $[t_{\sigma^{i+1}}, t]$. Thus π^i cannot become historical again in the time interval $[t_{\sigma^i} + T, t]$, contradicting the assumption that π^i was historical at time t.

We have thus shown that if $\pi^1, \ldots, \pi^z$ are historical paths at time t, there must be z updates $\sigma^1, \ldots, \sigma^z$ in the original sequence Σ such that $t_{\sigma^i} + 2^{\lfloor \log_2(t - t_{\sigma^i}) \rfloor} \le t_{\sigma^{i+1}} \le t$, for $1 \le i \le z - 1$. This implies $z = O(\log t) = O(\log k)$. □

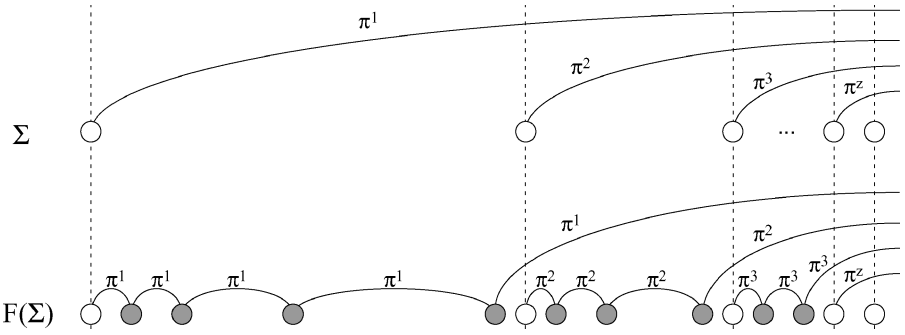

FIG. 8. Worst case instance of Σ such that at time k there are $\Theta(\log k)$ historical paths between pair x, y in the graph produced by $F(\Sigma)$.

We note that $\Theta(\log k)$ historical paths between a pair of vertices in G can be actually realized, as illustrated in Figure 8.

5. *Fully Dynamic Shortest Paths*

In this section, we show that the properties of locally historical paths given in Section 4 allow us to use the update algorithm of Section 3 in a fully dynamic setting, and to achieve $\tilde{O}(n^2)$ amortized time per update. This is the first general algorithm for fully dynamic all pairs shortest path that is faster than recomputing the solution from scratch after each update. The approach we follow here differs in just two points from Section 3:

(1) We maintain locally *historical* paths instead of locally *shortest* paths. Since by Lemma 4.3 shortest paths are locally historical, this guarantees that we are still maintaining information about shortest paths.
(2) We provide a new front-end `fully-update` for the update operation, which calls the `update` procedure of Figure 2, implementing the smoothing strategy of Section 4.3.

We now describe in more detail how to achieve these modifications, and then discuss correctness and time/space requirements of the new procedure `fully-update`.

5.1. DATA STRUCTURE. We maintain exactly the same data structure described in Section 3.1, with just one difference: we replace "shortest" with "historical".

$$
\begin{array}{lcll}
P_{xy} & = & \{\pi_{xy} & : \ \pi_{xy} \text{ is a locally historical path in } G\} \\
P^*_{xy} & = & \{\pi_{xy} & : \ \pi_{xy} \text{ is a historical path in } G\} \\
L(\pi_{xy}) & = & \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} & : \ (x', x) \in E \text{ and } \pi_{x'y} \text{ is a loc. hist. path in } G\} \\
L^*(\pi_{xy}) & = & \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} & : \ (x', x) \in E \text{ and } \pi_{x'y} \text{ is a hist. path in } G\} \\
R(\pi_{xy}) & = & \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle & : \ (y, y') \in E \text{ and } \pi_{xy'} \text{ is a loc. hist. path in } G\} \\
R^*(\pi_{xy}) & = & \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle & : \ (y, y') \in E \text{ and } \pi_{xy'} \text{ is a hist. path in } G\}
\end{array}
$$

In order to support the smoothing operation, we also keep for each vertex v the time t_v of its latest update and a counter *time* of the number of `fully-update` operations performed. The notation introduced in this section is summarized in Table III.

TABLE III. NOTATION INTRODUCED IN SECTION 5.1.

P_{xy}	set of locally historical paths from x to y in G
P^*_{xy}	set of historical paths from x to y in G
$L(\pi_{xy})$	set of pre-extensions $\langle x', x\rangle \cdot \pi_{xy}$ of π_{xy} that are locally historical paths in G
$L^*(\pi_{xy})$	set of pre-extensions $\langle x', x\rangle \cdot \pi_{xy}$ of π_{xy} that are historical paths in G
$R(\pi_{xy})$	set of post-extensions $\pi_{xy} \cdot \langle y, y'\rangle$ of π_{xy} that are locally historical paths in G
$R^*(\pi_{xy})$	set of post-extensions $\pi_{xy} \cdot \langle y, y'\rangle$ of π_{xy} that are historical paths in G

```
    fully-update(v, w'):
1.      time ← time + 1
2.      t_v ← time
3.      update(v, w')                                          {original update σ}
4.      for each u ∈ V :  time − t_u = 2^⌈log2(time−t_u)⌉ do    {smoothing}
5.          update(u, w)                                       {cleanup update δ}
```

FIG. 9. Implementation of the `fully-update` operation.

5.2. IMPLEMENTATION OF OPERATIONS. The operations are exactly the same as described in Section 3.2. The only difference is that we do not call directly the `update` procedure shown in Figure 2, but rather provide a new front-end procedure named `fully-update` and shown in Figure 9, which implements the smoothing strategy of Section 4.3 by calling the `update` procedure of Figure 2 for both original and cleanup updates.

5.3. ANALYSIS. The correctness of query operations follows directly from Lemma 4.3. To prove the correctness of `fully-update`(v, w'), we observe that it calls `update`(v, w') to update the data structure after changing the weights of the edges incident to v to the new values w' (line 3). Then, it repeatedly calls `update`(v, w) on vertices that have to be smoothed (lines 4–5): notice that the call overwrites the edge weights without changing their values, and the only effect of this operation is to remove non-optimal historical paths from the data structure as discussed in Section 4.3. Therefore, it is sufficient to prove the correctness of `update`. To this aim, it is easy to check that both Invariant 3.1 and its proof continue to hold using the new data structure of Section 5.1 instead of the data structure of Section 3.1. We can thus prove the following theorem:

THEOREM 5.1. *If shortest paths are unique and edge weights are nonnegative, then algorithm* `update` *correctly updates* P_{xy} *and* P^*_{xy} *for each pair of vertices* x *and* y.

PROOF. The proof is exactly the same as the proof of Theorem 3.2, provided that we suitably replace "shortest" with "historical" and "increase" with "update". □

We conclude our analysis of the fully dynamic algorithm by discussing time and space requirements of our implementation. To this aim, we first analyze procedure `update` in a fully dynamic setting.

LEMMA 5.2. *In a fully dynamic sequence of* $\Omega(m/n)$ *operations, if shortest paths are unique, edge weights are nonnegative, and at any time there are at most* z *historical paths between each pair of vertices, our data structure supports each* `update` *operation in* $O(zn^2 \log n)$ *amortized time. The space used is* $O(zmn)$.

PROOF. We rephrase the proof of Theorem 3.3, by suitably replacing "shortest" with "historical", and by using Lemma 4.6, Theorem 4.8, and Lemma 4.4 instead of Lemma 2.3, Theorem 2.6, and Lemma 2.4, respectively.

The bounds for queries are immediate, since the minimum weight path in the priority queue P_{xy} can be retrieved in optimal time.

Since each iteration of `cleanup` removes a path π_{xy} from a constant number of lists and priority queues, it requires $O(\log n)$ time in the worst case. By Lemma 4.6, at most $O(zn^2)$ locally historical paths can go through the updated vertex in the worst case, and thus `cleanup` will perform at most $O(zn^2)$ iterations, thus spending at most $O(zn^2 \log n)$ time.

Next, it is easy to see that *Phase* 1 of `fixup` requires $O(n \log n)$ time in the worst case. To bound the time required by *Phase* 2 of `fixup`, we consider that finding the minimum in P_{xy} takes constant time and each insertion in H takes $O(\log n)$ time. This gives a total bound of $O(n^2 \log n)$ for this phase. Finally, we note that a path π_{xy} is a new locally historical path if and only if both $\ell(\pi_{xy}) = \langle x, \ldots, b\rangle \in P^*_{xb}$ and $r(\pi_{xy}) = \langle a, \ldots, y\rangle \in P^*_{ay}$ *after* the update, and either $\ell(\pi_{xy}) \notin P^*_{xb}$ or $r(\pi_{xy}) \notin P^*_{ay}$ *before* calling `fixup`. Thus, since in *Phase* 3 we process a shortest path π_{xy} extracted from H only if it was not already in P^*_{xy} before calling `fixup`, then we spend time only for the new locally historical paths. This implies that *Phase* 3 requires $O(n^2 \log n)$ time for extracting the $O(n^2)$ pairs which are initially in H, plus $O(\log n)$ time to insert/delete each new locally historical path in H and P: let p the number of such paths. By Theorem 4.8, there can be $O(zn^2)$ paths that start being locally historical after each update, amortized over a sequence of $\Omega(m/n)$ operations. Moreover, by Lemma 4.6, there can be at most $O(zn^2)$ locally historical paths that were removed by `cleanup` since they contain the updated vertex, and are added back by `fixup`: thus, $p = O(zn^2)$ and `update` requires $O(zn^2 \log n)$ overall time.

At any time, each locally historical path π_{xy} can only appear in a constant number of data structures: namely, in H, P_{xy}, $L(r(\pi_{xy}))$, $R(\ell(\pi_{xy}))$, and possibly in P^*_{xy}, $L^*(r(\pi_{xy}))$, $R^*(\ell(\pi_{xy}))$. Thus, the space usage of our data structure is bounded by the number of locally historical paths in a graph, which is $O(zmn)$ in the worst case by Lemma 4.4. □

THEOREM 5.3. *In a fully dynamic sequence of* $\Omega(m/n)$ *operations, if shortest paths are unique and edge weights are non-negative, our data structure supports each* `fully-update` *operation in* $O(n^2 \log^3 n)$ *amortized time, and each* `distance` *and* `path` *query in optimal time. The space used is* $O(mn \log n)$.

PROOF. As in the proof of Theorem 3.3, the bounds for queries are immediate. To prove our update bound, we observe that procedure `fully-update` generates the smoothed sequence of Definition 4.9: thus, by Theorem 4.10, at any time in a sequence of k updates, there are at most $z = O(\log k)$ historical paths between each pair of vertices in G.

By Lemma 5.2, each call to `update` requires $O(n^2 \log n \log k)$ amortized time. Since each `fully-update` causes as many as $O(\log k)$ `update` operations, then the amortized running time of `fully-update` is $O(n^2 \log n \log^2 k) = O(n^2 \log^3 n)$ on any sequence of length k, with k polynomial in n. To get rid of this assumption, we can simply restart our data structure from scratch every $\Theta(n^2)$ operations. The space bound follows immediately from Lemma 5.2. □

6. *Conclusions*

In this article, we have presented a new algorithm that achieves nearly-quadratic update bounds for fully dynamic all-pairs shortest paths on graphs with nonnegative real edge weights. The algorithm is deterministic and uses simple data structures. In accordance with a recent computational study [Demetrescu et al. 2004], the techniques described in this article are not only asymptotically efficient, but can yield very fast implementations in many practical scenarios. The interested reader can download the C package used in the experiments at the URL: `http://www.dis.uniroma1.it/~demetres/experim/dsp/` .

The algorithms presented in this article hinge upon the novel notion of locally defined path properties. Using this approach, we have considered *locally shortest paths*, that is, paths whose proper subpaths are shortest paths. Since at most $O(n^2)$ such paths (amortized) can appear or disappear in the graph at each update in a partially dynamic setting, maintaining them in a data structure can be done in $O(n^2 \log n)$ amortized time per update. In a fully dynamic setting though, changes in locally shortest paths may be much higher. To overcome this problem, we have used again the locality approach, considering *locally historical paths*, that is, paths whose proper subpaths are historical, thus achieving an $O(n^2 \log^3 n)$ amortized time per update with a space usage of $O(mn \log n)$. Using our approach, but with a different smoothing strategy, Thorup [2004] has shown how to achieve $O(n^2(\log n + \log^2(m/n)))$ amortized time per update and $O(mn)$ space. His algorithm works with negative weights as well.

There are several issues that seem worth of further investigation. First, can we reduce the space usage to $O(n^2)$? Second, and perhaps more importantly, can we solve efficiently fully dynamic *single-source* shortest paths on general graphs?

ACKNOWLEDGMENTS. We are indebted to Valerie King, Mikkel Thorup, Uri Zwick, and the anonymous reviewers for many useful comments and suggestions on this work.

REFERENCES

AUSIELLO, G., ITALIANO, G., MARCHETTI-SPACCAMELA, A., AND NANNI, U. 1991. Incremental algorithms for minimal length paths. *J. Algorithms 12*, 4, 615–38.

BASWANA, S., HARIHARAN, R., AND SEN, S. 2002. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC'02)*. ACM, New York, pp. 117–123.

DEMETRESCU, C., EMILIOZZI, S., AND ITALIANO, G. 2004. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*. ACM, New York, pp. 362–371.

DEMETRESCU, C., AND ITALIANO, G. 2001. Fully dynamic all pairs shortest paths with real edge weights. In *Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01)* (Las Vegas, Nevada). IEEE Computer Society Press, Los Alamitos, Calif., pp. 260–267.

DEMETRESCU, C. AND ITALIANO, G. 2002. Improved bounds and new trade-offs for dynamic all pairs shortest paths. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP'02)* (Málaga, Spain). 633–643.

EVEN, S., AND GAZIT, H. 1985. Updating distances in dynamic graphs. *Meth. Oper. Res. 49*, 371–387.

FAKCHAROEMPHOL, J., AND RAO, S. 2001. Planar graphs, negative weight edges, shortest paths, and near linear time. In *Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01)* (Las Vegas, Nevada). IEEE Computer Society Press, Los Alamitos, Calif., pp. 232–241.

FRIGIONI, D., MARCHETTI-SPACCAMELA, A., AND NANNI, U. 1998. Semi-dynamic algorithms for maintaining single source shortest paths trees. *Algorithmica 22*, 3, 250–274.

FRIGIONI, D., MARCHETTI-SPACCAMELA, A., AND NANNI, U. 2000. Fully dynamic algorithms for maintaining shortest paths trees. *J. Algorithms 34*, 351–381.

HENZINGER, M., KLEIN, P., RAO, S., AND SUBRAMANIAN, S. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci. 55*, 1 (Aug.), 3–23.

KING, V. 1999. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS'99)*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 81–99.

LOUBAL, P. 1967. A network evaluation procedure. *Highway Research Record 205*, 96–109.

MURCHLAND, J. 1967. The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Tech. rep., LBS-TNT-26, London Business School, Transport Network Theory Unit, London, UK.

RAMALINGAM, G., AND REPS, T. 1996a. An incremental algorithm for a generalization of the shortest path problem. *J. Algorithms 21*, 267–305.

RAMALINGAM, G., AND REPS, T. 1996b. On the computational complexity of dynamic graph problems. *Theoret. Comput. Sci. 158*, 233–277.

RODIONOV, V. 1968. The parametric problem of shortest distances. *U.S.S.R. Comput. Math. Math. Phys. 8*, 5, 336–343.

ROHNERT, H. 1985. A dynamization of the all-pairs least cost problem. In *Proceedings of the 2nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'85)*. Lecture Notes in Computer Science, vol. 182, Springer-Verlag, New York, pp. 279–286.

THORUP, M. 2004. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04)*. 384–396.

RECEIVED DECEMBER 2003, SEPTEMBER 2004; ACCEPTED SEPTEMBER 2004