Average Update Times for
Fully-Dynamic All-Pairs Shortest Paths

Tobias Friedrich[1,2] and Nils Hebbinghaus[1]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] International Computer Science Institute, Berkeley, CA, USA

Abstract. We study the fully-dynamic all pairs shortest path problem for graphs with arbitrary non-negative edge weights. It is known for digraphs that an update of the distance matrix costs $\tilde{\mathcal{O}}(n^{2.75})$[1] worst-case time [Thorup, STOC '05] and $\tilde{\mathcal{O}}(n^2)$ amortized time [Demetrescu and Italiano, J.ACM '04] where n is the number of vertices. We present the first average-case analysis of the undirected problem. For a random update we show that the expected time per update is bounded by $\mathcal{O}(n^{4/3+\varepsilon})$ for all $\varepsilon > 0$.

Keywords: Dynamic graph algorithms, shortest paths, average-case analysis, random graphs.

1 Introduction

Dynamic graph algorithms maintain a certain property (e. g., connectivity information) of a graph that changes (a new edge inserted or an existing edge deleted) dynamically over time. They are used in a variety of contexts, e. g., operating systems, information systems, database systems, network management, assembly planning, VLSI design and graphical applications. An algorithm is called *fully-dynamic* if both edge weight increases and edge weight decreases are allowed. While a number of fully dynamic algorithms have been obtained for various properties on undirected graphs (see [6]), the design and analysis of fully-dynamic algorithms for directed graphs has turned out to be much harder (e. g., [9, 13, 15, 16]).

In this article, we consider the fully-dynamic *all-pairs shortest path problem* (APSP) for undirected graphs, which is one of the most fundamental problems in dynamic graph algorithms. The problem has been studied intensively since the late sixties (see [4] and references therein). We are interested in algorithms that maintain a complete distance matrix as edges are inserted or deleted. The static directed APSP problem can be solved in $\mathcal{O}(mn + n^2 \log n)$ time [8] where n is the number of vertices and m is the number of edges. This gives $\mathcal{O}(n^3)$ per update in the worst-case for a static recomputation from scratch. The first major improvement that is provably faster than this only worked on digraphs with small integer weights. King [10] presented a fully-dynamic APSP algorithm

---

[1] Throughout the paper, we use $\tilde{\mathcal{O}}(f(n))$ to denote $\mathcal{O}(f(n) \operatorname{polylog}(n))$.

for general directed graphs with positive integer weights less than C that supported updates in $\mathcal{O}(n^{2.5}\sqrt{C \log n})$. In the remainder of the paper, we will only consider non-negative real-valued edge weights. Demetrescu and Italiano pursued this problem in a series of papers and showed that it can be solved in $\mathcal{O}(n^2 \log^3 n)$ amortized time per update. [4]. This has been slightly improved to $\mathcal{O}(n^2(\log n + \log^2((m+n)/n)))$ amortized time per update by Thorup [17]. In [18], Thorup showed a worst-case update time of $\tilde{\mathcal{O}}(n^{2.75})$.

We are interested in *expected update times*. The only known result for this is for the undirected, unweighted, decremental, randomized, and approximate version of the APSP problem. Roditty and Zwick [14] showed for this setting an expected amortized time of $\tilde{\mathcal{O}}(n)$. For our setting of the problem on *undirected graphs with arbitrary non-negative edge weights*, there is nothing known about the average-case update times. We analyze a variant of Demetrescu and Italiano's algorithm described in Section 3. Let $\mathcal{R}(p)$ denote the expected runtime of our algorithm for a single random edge update of a random graph $G \in \mathcal{G}(n,p)$. Let $\varepsilon, \varepsilon' > 0$. For arbitrary p, we can show $\mathcal{R}(p) = \mathcal{O}(n^{4/3+\varepsilon})$. However, for most p we can prove that the runtime is actually much smaller. The above bound is best only at the phase transition around $pn = 1$, i.e., when the size of the largest component rapidly grows from $\Theta(\log n)$ to $\Theta(n)$. When the graph is sparser, our algorithm is much faster. In this case, we can show $\mathcal{R}(p) = \mathcal{O}(n^{2/3+\varepsilon})$ for $pn \leq 1 - n^{-1/3}$ and $\mathcal{R}(p) = \mathcal{O}(n^{\varepsilon})$ for $pn < 1/2$. Similarly, the algorithm becomes faster when the graph has passed the critical window. We show $\mathcal{R}(p) = \mathcal{O}(n^{\varepsilon}/p)$ for $pn \geq 1 + \varepsilon'$. The final result is given in Theorem 11. Additionally to these asymptotic upper bounds on the expected runtime, we also examined the empirical average runtime. Interestingly, this also shows that the update costs are first increasing and later decreasing when more edges are inserted. This corresponds well with the above phase distinction for the asymptotic bounds.

The remainder of this paper is organized as follows. The next section presents all necessary graph theoretical notations. In Section 3 we present our algorithm. In Section 4 we prove a number of random graph properties which are then used in Section 5 to show the asymptotic bounds. The last section presents some empirical results.

2   Preliminaries

Demetrescu and Italiano [5] performed several experiments on directed random graphs. We want to bound the expected runtime of random updates on a random graph of a very similar algorithm. We utilize the random graph model $\mathcal{G}(n,p)$ introduced and popularized by Erdős and Rényi [7]. The $\mathcal{G}(n,p)$ model consists of a graph with n vertices in which each edge is chosen independently with probability p. In our model, a random update first chooses two vertices x and y ($x \neq y$). Then, with (fixed) probability δ, it inserts the edge (x,y) with a random weight $w \in [0,1]$. If the edge was already in the graph, it changes its weight to w. Otherwise, with probability $1-\delta$, a random update deletes the edge (x,y) if (x,y) is in the graph (otherwise, it does nothing). Note that T random updates on a

random graph $G \in \mathcal{G}(n,p)$ lead to a graph with edges present with probability $p' = \delta + (p - \delta)(1 - 1/\binom{n}{2})^T$, but not necessarily mutually independent.

Throughout the paper, we use the following notations:

- $G = (V, E)$ is an undirected graph with arbitrary non-negative edge weights.
- $\Delta := \max_{v \in V} \deg(v)$ is the maximum degree of the graph G.
- $\mathrm{dist}(x, y)$ (distance) is the length of the shortest path from x to y.
- $\mathrm{diam}(G)$ (diameter) is the greatest distance between any two vertices of one component.
- $C(x)$ denotes the component that contains the vertex x.
- $C(G)$ denotes the largest component of G.
- w_{xy} denotes the weight of an edge (x, y).
- $\pi_{xy} = \langle u_0, u_1, \ldots, u_k \rangle$ is a path from vertex $x = u_0$ to vertex $y = u_k$, i.e., a sequence of vertices such that with $(u_i, u_{i+1}) \in E$ for each $0 \leq i < k$ (no repeated edges).
- $w(\pi_{xy}) = \sum_{i=0}^{k-1} w_{u_i u_{i+1}}$ is the weight of a path.
- $\pi_{xy} \circ \pi_{yz}$ denotes the concatenation of two paths π_{xy} and π_{yz}.
- $\ell(\pi_{xy})$ denotes the subpath π_{xa} of π_{xy} such that $\pi_{xy} = \pi_{xa} \circ \langle a, y \rangle$.
- $r(\pi_{xy})$ denotes the subpath π_{by} of π_{xy} such that $\pi_{xy} = \langle x, b \rangle \circ \pi_{by}$.

We assume without loss of generality that there is only one shortest path between each pair of vertices in G. Otherwise, ties can be broken as discussed in Section 3.4 of [4].

3   Algorithm

We will now describe our algorithm. It is a slight modification of the algorithm of Demetrescu and Italiano [5] as our aim is an average-case analysis of the undirected problem while they were interested in the amortized costs for directed graphs.

The main tool Demetrescu and Italiano [4] very cleverly introduced and applied is the concept of "locally shortest paths". A path π_{xy} is *locally shortest* if every proper subpath is a shortest path or it consists of only a single vertex. The algorithm maintains the following data structures:

- w_{xy} weight of edge (x, y)
- P_{xy} priority queue of the locally shortest paths from x to y (priority $w(\pi_{xy})$)
- P_{xy}^* shortest path from x to y
- $L(\pi_{xy})$ set of left-extensions $\langle x', x \rangle \circ \pi_{xy}$ of π_{xy} that are locally shortest paths
- $L^*(\pi_{xy})$ set of left-extensions $\langle x', x \rangle \circ \pi_{xy}$ of π_{xy} that are shortest paths
- $R(\pi_{xy})$ set of right-extensions $\pi_{xy} \circ \langle y, y' \rangle$ of π_{xy} that are locally shortest paths
- $R^*(\pi_{xy})$ set of right-extensions $\pi_{xy} \circ \langle y, y' \rangle$ of π_{xy} that are shortest paths

Note that $P_{xy}^* \subseteq P_{xy}$ and that every minimum weight path in P_{xy} is also a shortest path. Each path $\pi_{xy} \in P_{xy}$ is stored implicitly with constant space by just storing two pointers to the subpaths $\ell(\pi_{xy})$ and $r(\pi_{xy})$.

UPDATE(u, v, w)
    ▷ Phase 1: Delete edge (u, v) and all paths containing edge (u, v),
                store pairs of vertices affected by the update in list A
1   **if** $(u, v) \in P_{uv}$ **then**
2   $Q \leftarrow \{(u, v)\}$
3   **while** $Q \neq \emptyset$ **do**
4   extract any π_{xy} from Q
5   remove π_{xy} from P_{xy}, $L(r(\pi_{xy}))$, and $R(\ell(\pi_{xy}))$
6   **if** $\pi_{xy} \in P_{xy}^*$ **then**
7   remove π_{xy} from P_{xy}^*, $L^*(r(\pi_{xy}))$, and $R^*(\ell(\pi_{xy}))$
8   add (x, y) to A
9   add all paths in $L(\pi_{xy})$ to Q
10   add all paths in $R(\pi_{xy})$ to Q
    ▷ Phase 2: Insert edge (u, v) with weight w
11 **if** $w < \infty$ **then**
12   add (u, v) to A
13   add (u, v) to P_{uv}, $L(\pi_{vv})$, and $R(\pi_{uu})$
    ▷ Phase 3: Scan pairs in A
14 **while** $A \neq \emptyset$ **do**
15   extract any pair (x, y) from A
16   add π_{xy} with minimum $w(\pi_{xy})$ to H (if any)
    ▷ Phase 4: Propagation loop
17 **while** $H \neq \emptyset$ **do**
18   extract path π_{xy} with minimum $w(\pi_{xy})$ from H
19   **if** $w(\pi_{xy})$ is larger than the smallest weight in P_{xy} **then continue**
20   add P_{xy}^* to Q
21   add π_{xy} to P_{xy}^*, $L^*(r(\pi_{xy}))$, and $R^*(\ell(\pi_{xy}))$
22   **for each** $\pi_{x'b} \in L^*(\ell(\pi_{xy}))$ **do**
23   **if** $(x', x) \circ \pi_{xy} \in L(\pi_{xy})$ **then continue**
24        $\pi_{x'y} \leftarrow (x', x) \circ \pi_{xy}$
25        $w(\pi_{x'y}) \leftarrow w_{x'x} + d_{xy}$
26        $\ell(\pi_{x'y}) \leftarrow \pi_{x'b}, r(\pi_{x'y}) \leftarrow \pi_{xy}$
27   add $\pi_{x'y}$ to $P_{x'y}$, $L(\pi_{xy})$, $R(\pi_{x'b})$, and H
28   **for each** $\pi_{ay'} \in R^*(r(\pi_{xy}))$ **do**
29   **if** $\pi_{xy} \circ (y, y') \in R(\pi_{xy})$ **then continue**
30        $\pi_{xy'} \leftarrow \pi_{xy} \circ (y, y')$
31        $w(\pi_{xy'}) \leftarrow d_{xy} + w_{yy'}$
32        $\ell(\pi_{xy'}) \leftarrow \pi_{xy}, r(\pi_{xy'}) \leftarrow \pi_{ay'}$
33   add $\pi_{xy'}$ to $P_{xy'}$, $L(\pi_{ay'})$, $R(\pi_{xy})$, and H
    ▷ Phase 5: Delete all LSPs π that stopped being LSP
                because $\ell(\pi)$ or $r(\pi)$ stopped being SP
34 **while** $Q \neq \emptyset$ **do**
35   extract any π_{xy} from Q
36   **for each** $\pi_{x'y} \in L(\pi_{xy})$ **do**
37   remove $\pi_{x'y}$ from $R((x', x) \circ \ell(\pi_{x,y}))$ and $L(\pi_{x,y})$
38   **for each** $\pi_{xy'} \in R(\pi_{xy})$ **do**
39   remove $\pi_{xy'}$ from $L(r(\pi_{x,y}) \circ (y, y'))$ and $R(\pi_{x,y})$

Fig. 1. The slightly modified APSP algorithm of Demetrescu and Italiano [5]

The pseudo-code of our algorithm is given in Figure 1. The first four phases are equivalent to [5]. We will just describe them briefly. A detailed description can be found in [4]. In the first phase, the algorithm deletes from the data structure all the paths that would stop being locally shortest if we deleted the edge (u, v). In doing so it stores the pairs of the endpoints of the affected paths in the temporary list A. In the following phase it adds the edge (if it is an insert or update operation) to the data structures. The third phase initializes the heap H with the minimum weight paths π_{xy} for all $(x, y) \in A$. In the fourth phase the algorithm repeatedly extracts the cheapest path π_{xy} from H. The first extracted path for each pair (x, y) must be a shortest path. If this is the case, the path is stored in the data structures. To propagate this information, also its left- and right-extensions are updated and added to H to find all further extensions.

The amortized number of new locally shortest paths can be $\Omega(n^3)$ per update. To allow a better worst-case performance, Demetrescu and Italiano [4] had to delay the update of the data structure in a very clever way. Their data structure can contain paths in P_{xy} which are not locally shortest anymore. We avoid this with the fifth phase. There, all locally shortest paths which stopped being locally shortest because one of their two subpaths stopped being shortest path are detected and deleted.

We analyze the expected time for the algorithm to insert a randomly chosen edge e in the graph $G \in \mathcal{G}(n, p)$ and maintain the sets of shortest path and locally shortest path. The weights of e and of the edges in G are chosen uniformly at random from the set $[0, 1]$.

4   Random Graph Properties

To bound the runtime of our algorithm in the next section, we first provide some properties of random graphs $G \in \mathcal{G}(n, p)$. The main result of this section will be Theorem 9. It bounds the quantity $\mu(p)$ which we define as the expected number of locally shortest paths and shortest paths passing a fixed edge of G. Let LSP denote the set of all locally shortest paths and SP the set of all shortest paths in G.

The following four lemmas are well-known.

Lemma 1 (Bollobás [2]). *Let $G \in \mathcal{G}(n, p)$ with $pn < 1/2$. Then,* $\mathbf{Pr}\left[|C(G)| \le 20 \log n\right] = 1 - \mathcal{O}(n^{-2})$.

Lemma 2 (Bollobás [2]). *For every $\alpha > 0$ and $G \in \mathcal{G}(n, p)$ with $pn = \alpha \log n$,* $\mathbf{Pr}\left[G \text{ is connected}\right] = 1 - \mathcal{O}(n^{1-2\alpha})$.

Lemma 3 (Chung and Lu [3]). *For every $\varepsilon > 0$ and $G \in \mathcal{G}(n, p)$ with $pn = 1 + \varepsilon$,* $\mathbf{Pr}\left[\text{diam}(G) \le 2 \log n\right] = 1 - o(n^{-1})$.

Lemma 4 (Nachmias and Peres [12]). *Let $x \in G$ and $G \in \mathcal{G}(n, p)$ with $pn \le 1 + n^{-1/3}$. Then,* $\mathbf{Pr}\left[|C(x)| > 2n^{2/3}\right] = \mathcal{O}(n^{-1/3})$.

The following lemma gives a general upper bound on the expected diameter of a random graph $G \in \mathcal{G}(n,p)$ for arbitrary p. Recall that we defined the diameter of a disconnected graph as the maximum diameter of its components.

Lemma 5. *Let $G \in \mathcal{G}(n,p)$. Then, $\mathbf{E}[\mathrm{diam}(G)] = \mathcal{O}(n^{1/3})$.*

Proof. Let G be a complete graph on n vertices with edge weights uniformly distributed at random in $[0,1]$. Let $G_{\leq p} = (V, E_{\leq p})$ be the subgraph of G containing all vertices but only those edges with weight less or equal p. Then $G_{\leq p}$ is a $\mathcal{G}(n,p)$-graph. We apply Kruskal's algorithm [11] for the construction of a minimum spanning forest of G, i.e., we look at the edges in increasing weight order and integrate every edge that does not introduce a cycle in the current edge set. We stop this process if the current edge has weight greater than p and denote the obtained subset of edges $E_{\mathrm{Kruskal}, \leq p}$. Let us also denote the edge set of the spanning forest which is returned from the completed Kruskal algorithm by E_{Kruskal}. By Addario-Berry, Broutin, and Reed [1], the expected diameter of $G_{\mathrm{Kruskal}} := (V, E_{\mathrm{Kruskal}})$ is of order $\Theta(n^{1/3})$. Clearly, $E_{\mathrm{Kruskal}, \leq p} \subseteq E_{\mathrm{Kruskal}} \cap E_{\leq p}$. As $G_{\mathrm{Kruskal}, \leq p} := (V, E_{\mathrm{Kruskal}, \leq p})$ is a minimum spanning forest of $G_{\leq p}$, we get

$$\mathbf{E}[\mathrm{diam}(G_{\leq p})] \leq \mathbf{E}[\mathrm{diam}(G_{\mathrm{Kruskal}, \leq p})] \leq \mathbf{E}[\mathrm{diam}(G_{\mathrm{Kruskal}})] = \mathcal{O}(n^{1/3}). \quad \square$$

To prove the desired bound on $\mu(p)$ we also need the following three technical lemmas.

Lemma 6. *Let $G \in \mathcal{G}(n,p)$ with $pn \geq 4 \log n$. Then every shortest path in G has weight $\mathcal{O}(\frac{\log^2 n}{n})$ with probability $1 - \mathcal{O}(n^{-2})$.*

Proof. Let us consider two random graphs $G_1, G_2 \in \mathcal{G}(n, \frac{2 \log n}{n})$ on the same set of vertices and let $G_{\cup}$ be the union of G_1 and G_2 (union of the edge sets). Then we get $G_{\cup} \in \mathcal{G}(n, \frac{4 \log n}{n} - \frac{4 \log^2 n}{n^2})$. By Lemmas 2 and 3, G_i is connected and and $\mathrm{diam}(G_i) \leq 2 \log n$ with probability $1 - \mathcal{O}(n^{-1})$ for $i = 1, 2$. As the two random graphs are chosen independently, at least one of them is connected and has diameter $\mathcal{O}(\log n)$ with probability $1 - \mathcal{O}(n^{-2})$. By construction, this also holds for $G_{\cup}$. Therefore all $G \in \mathcal{G}(n,p)$ with $pn \geq 4 \log n$ are connected and have a diameter of order $\mathcal{O}(\log n)$ with probability $1 - \mathcal{O}(n^{-2})$.

We now prove that every shortest path in G has a total weight of $\mathcal{O}(\frac{\log^2 n}{pn})$ with probability $1 - \mathcal{O}(n^{-2})$. Let us consider the subgraph $G_0 = (V, E_{\leq \frac{4 \log n}{pn}})$ consisting of all vertices but only those edges of G with weight at most $\frac{4 \log n}{pn}$. This is a random graph $\mathcal{G}(n, \frac{4 \log n}{n})$ with weights chosen uniformly at random from $[0, \frac{4 \log n}{pn}]$. As we have shown above, G_0 is connected and has diameter of order $\mathcal{O}(\log n)$ with probability $1 - \mathcal{O}(n^{-2})$. This implies that every shortest path in G_0 has a total weight of $\mathcal{O}(\frac{\log^2 n}{pn})$ with probability $1 - \mathcal{O}(n^{-2})$. This upper bound also holds with the same probability for all shortest paths in G. $\quad \square$

Lemma 7. *Let $G \in \mathcal{G}(n, p)$. The subgraph $G_{\mathrm{SP}} = (V, E_{\mathrm{SP}})$ of all edges that are shortest paths in G fulfills $\Delta(G_{\mathrm{SP}}) \leq n^\varepsilon$ with probability $1 - \mathcal{O}(n^{-2})$. In particular, $|\mathrm{LSP}| \leq |\mathrm{SP}| n^\varepsilon$ with probability $1 - \mathcal{O}(n^{-2})$.*

Proof. Let us first consider the case $pn \geq 4 \log n$. We know from Lemma 6 that all elements of E_{SP} have weight $\mathcal{O}(\frac{\log^2 n}{pn})$ with probability $1 - \mathcal{O}(n^{-2})$. Thus, G_{SP} is a subgraph of a random graph in $\mathcal{G}(n, \frac{\log^2 n}{n})$. Therefore, we can prove the first claim of the lemma by bounding $\Delta(G')$ for $G' \in \mathcal{G}(n, \frac{\log^2 n}{n})$. Using Stirling's formula, the probability for a vertex in G' to have a degree greater or equal n^ε is at most

$$\binom{n}{n^\varepsilon} \left(\frac{\log^2 n}{n} \right)^{n^\varepsilon} \leq \frac{(\log^2 n)^{n^\varepsilon}}{\sqrt{2\pi n^\varepsilon} \left(\frac{n^\varepsilon}{e} \right)^{n^\varepsilon}} \leq n^{-\varepsilon n^{\varepsilon/2}}$$

for n large enough. Hence, the probability for $\Delta(G')$ to be greater or equal n^ε is at most

$$1 - \left(1 - n^{-\varepsilon n^{\varepsilon/2}} \right)^n \leq 1 - \left(\left(1 - n^{-\varepsilon n^{\varepsilon/2}} \right)^{n^{\varepsilon n^{\varepsilon/2}} - 1} \right)^{2n^{1 - \varepsilon n^{\varepsilon/2}}}$$

$$\leq 1 - e^{-2n^{1 - \varepsilon n^{\varepsilon/2}}} \leq n^{-n^{\varepsilon/3}},$$

where we used $2n^{1 - \varepsilon n^{\varepsilon/2}} \leq n^{-n^{\varepsilon/3}}$ for n large enough and $1 + x \leq e^x$ for all $x \in \mathbb{R}$. This proves $\Delta(G_{\mathrm{SP}}) \leq n^\varepsilon$ with probability $1 - \mathcal{O}(n^{-2})$. The second claim is a consequence of the fact, that each locally shortest path from vertex x to vertex y is uniquely determined by its first and also by its last edge. Moreover, every locally shortest path with at least 2 edges starts and ends with edges that are shortest paths themselves. Thus, there are at most $\mathcal{O}(n^\varepsilon)$ locally shortest paths for each shortest path with probability at least $1 - \mathcal{O}(n^{-2})$, which proves the lemma for $pn \geq 4 \log n$.

Let $pn < 4 \log n$. We consider $G' \in \mathcal{G}(n, \frac{4 \log n}{n})$ with edge weights chosen randomly in $[0, \frac{4 \log n}{pn}]$. Although the weights of this graph are scaled up by the factor $\frac{4 \log n}{pn}$, we get $\Delta(G'_{\mathrm{SP}}) \leq n^\varepsilon$ with probability $1 - \mathcal{O}(n^{-2})$, since the scaling has no effect on the subgraph G'_{SP} of all shortest path edges in G'. Now the subgraph G of all edges of G' with weight less or equal 1 is a $\mathcal{G}(n, p)$-graph with edge weights chosen uniformly at random from $[0, \frac{4 \log n}{pn}]$. Hence, every edge that is a shortest path in G is also a shortest path in G'. With this we get $\Delta(G_{\mathrm{SP}}) \leq n^\varepsilon$ with probability $1 - \mathcal{O}(n^{-2})$. The second claim follows with the same arguments as in the case $pn \geq 4 \log n$. $\square$

Lemma 8. *Let $G \in \mathcal{G}(n, p)$ with $pn \geq 1/2$. For all $\varepsilon > 0$, $\mu(p) = \mathcal{O}\big(\mathbf{E}\left[\mathrm{diam}(G)\right] |\mathrm{SP}| n^{\varepsilon - 2} / p \big)$.*

Proof. We consider the subgraph $G' = (V, E_{<1/(2pn)})$ containing all vertices of G but only those edges with weight less than $1/(2pn)$. Then by Lemma 1 $G \in \mathcal{G}(n, 1/(2n))$ and the largest component of G' is of order $\mathcal{O}(\log n)$ with

probability $1 - \mathcal{O}(n^{-2})$. Thus, every path in G' contains $\mathcal{O}(\log n)$ edges with probability $1 - \mathcal{O}(n^{-2})$. The expected weight of the heaviest element of LSP is at most $\mathbf{E}\left[\mathrm{diam}(G)\right]$. Moreover, in the case $p \geq \frac{4 \log n}{n}$ the expected weight of the heaviest element of LSP is at most $\mathcal{O}(\frac{\log^2 n}{pn})$ as shown in Lemma 6. Thus, in expectation the largest number of edges with a weight greater or equal $1/(2pn)$ in an element of LSP is of order $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]pn)$ and $\mathcal{O}(\log^2 n)$ if $p \geq \frac{4 \log n}{n}$. This implies an upper bound of $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]\log^2 n)$ for the maximal number of edges with weight greater or equal $1/(2pn)$ in locally shortest paths of G in expectation for all $p \geq 1/(2pn)$.

By Lemma 7 we know that the bound $|\mathrm{LSP}| = \mathcal{O}(|\mathrm{SP}|n^{\varepsilon/2})$ is violated with probability $\mathcal{O}(n^{-2})$. In this case we can estimate the number of locally shortest paths and shortest paths in G by $\mathcal{O}(n^3)$ (the first edge and the other endpoint of a locally shortest path determines the path uniquely) and the length of this paths trivially by $n - 1$. Since the probability for this event is $\mathcal{O}(n^{-2})$, the contribution to the expected number of edges in the multiset of all edges of all (locally) shortest paths is $\mathcal{O}(n^2)$. If $|\mathrm{LSP}| = \mathcal{O}(|\mathrm{SP}|n^{\varepsilon/2})$, the maximal number of edges with weight greater or equal $1/(2pn)$ in locally shortest paths of G is $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]\log^2 n)$ in expectation. Now in every (locally) shortest path in G there can only be consecutive parts of edges of G' of order $\mathcal{O}(\log n)$ and they must be followed by an edge with weight greater or equal $1/(2pn)$. Since there can only be $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]\log^2 n)$ of these edges in the path in expectation, the total number of edges in the longest of all (locally) shortest paths is $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]\log^3 n)$ in expectation. Thus, the multiset of all edges of all (locally) shortest paths contains $\mathcal{O}(|\mathrm{SP}|\mathbf{E}\left[\mathrm{diam}\,G\right]n^{\varepsilon/2}\log^3 n)$ edges. By Chernoff bounds G has $\Theta(pn^2)$ edges. Therefore, the average number of (locally) shortest paths through a fixed edge is $\mathcal{O}(\mathbf{E}\left[\mathrm{diam}\,G\right]|\mathrm{SP}|n^{\varepsilon-2}/p)$. $\qquad\square$

We are now well-prepared to prove the main theorem of this section. It bounds $\mu(p)$ which is the expected number of locally shortest paths and shortest paths passing a fixed edge.

Theorem 9. *Let $G \in \mathcal{G}(n, p)$. For all $\varepsilon, \varepsilon' > 0$,*

    (i) $\mu(p) = \mathcal{O}(1)$ *for $pn < 1/2$,*
   (ii) $\mu(p) = \mathcal{O}(n^{2/3})$ *for $1/2 \leq pn \leq 1 - n^{-1/3}$,*
  (iii) $\mu(p) = \mathcal{O}(n^{1+\varepsilon})$ *for $1 - n^{-1/3} \leq pn \leq 1 + n^{-1/3}$,*
  (iv) $\mu(p) = \mathcal{O}(n^{4/3+\varepsilon})$ *for $1 + n^{-1/3} \leq pn \leq 1 + \varepsilon'$,*
   (v) $\mu(p) = \mathcal{O}(n^{\varepsilon}/p)$ *for $1 + \varepsilon' \leq pn$.*

Proof. (i) We bound $\mu(p)$ by the total number of paths passing a fixed edge. Let us first estimate the expected number of paths of a fixed length k in G going through a fixed edge. There are k possible positions of the fixed edge in a path of length k. Furthermore, we can choose the remaining $k - 1$ vertices of such a path in $\prod_{i=1}^{k-1}(n - i)$ different ways. Since every $e \in \binom{[n]}{2}$ is an edge of G with probability p, the expected number of paths in G that go through a fixed edge is bounded above by

$$\sum_{k=1}^{n-1} kp^{k-1} \prod_{i=1}^{k-1}(n-i) \leq \sum_{k=1}^{n-1} k(pn)^{k-1} \leq (pn)^{-1} \sum_{k=1}^{n} k(pn)^{k}.$$

Thus, the expected number of paths in G going through a fixed edge is at most

$$(pn)^{-1} \sum_{k=1}^{n} \sum_{i=k}^{n}(pn)^{k} = \sum_{k=1}^{n} \frac{(pn)^{k-1}-(pn)^{n}}{1-pn} \leq \frac{1-(pn)^{n}}{(1-(pn))^{2}} \leq \frac{1}{(1-pn)^{2}}.$$

Thus, $\mu(p) = \mathcal{O}(1)$ for $pn \leq 1/2$.

(ii) Using the bound in (i), we get $\mu(p) = \mathcal{O}(1/(1-pn)^2) = \mathcal{O}(n^{2/3})$ for $pn \leq 1 - n^{-1/3}$.

(iii) Applying Lemma 4, we get $|\text{SP}| = \mathcal{O}(n^{5/3})$ with probability $1 - O(n^{-1/3})$ and $|SP| = \mathcal{O}(n^2)$ otherwise. Combining this with Lemma 8 and Lemma 5 gives $\mu(p) = O(n^{1+\varepsilon})$.

(iv) By Lemma 5, the expected diameter of G is $\mathcal{O}(n^{1/3})$. Thus, Lemma 8 yields $\mu(p) = \mathcal{O}(n^{4/3+\varepsilon})$.

(v) By Lemma 3, we get $\mathbf{E}\left[\text{diam}\, G\right] = O(\log n)$. Now Lemma 8 yields $\mu(p) = \mathcal{O}(n^{\varepsilon}/p)$. $\qquad\square$

5   Runtime Analysis

In this section we describe the runtime of our algorithm in terms of the parameter $\mu(p)$. With this, the main result Theorem 11 is an immediate corollary of the bounds on $\mu(p)$ from the previous section.

Theorem 10. *Let $G \in \mathcal{G}(n,p)$. The expected runtime of our algorithm for a random edge update on G is $\mathcal{O}(\mu(p)\, n^{\varepsilon})$.*

Proof. To bound the runtime, we will use the quantity $\mu(p)$ which is the expected number of locally shortest paths and shortest paths through a fixed edge e of G. If the algorithm performs the deletion of the edge e, this is exactly the number of paths that stop being (locally) shortest. In the case of the insertion, we get (almost) the same picture by making a backwards analysis. Instead of the insertion of e to $G = (V, E)$ we can also investigate the deletion of e from the graph $G' = (V, E \cup \{e\})$. Therefore the quantity $\mu(p)$ is also the expected number of paths in G' that start being (locally) shortest. The slight modification that G' contains one edge more than G has no consequence for the order of $\mu(p)$.

We bound the runtime of all five phases separately. The algorithm is running through the first phase, only if the considered edge e is already in the graph and has to be deleted or updated. If this is the case, the algorithm goes through the while loop for every locally shortest path of G that contains the edge e at most twice, since a locally shortest path can be added to Q as a left- and as a right-extension. The only part of the while loop with more than constant runtime is the removing of the path π_{xy} from the lists P_{xy}, $L(r(\pi_{xy}))$, and $R(\ell(\pi_{xy}))$. Since every locally shortest path is uniquely determined by the first (respectively the last edge), we can bound the runtime of phase 1 by $\mathcal{O}(\mu(p)n^{\varepsilon})$ using Lemma 7.
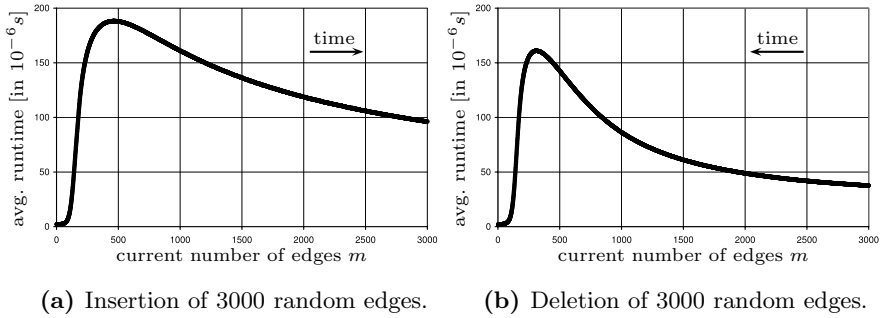
(a) Insertion of 3000 random edges.   **(b)** Deletion of 3000 random edges.

Fig. 2. Experimental results for the algorithm of Demetrescu and Italiano [5]. We start with an empty graph with $n = 100$ vertices and insert 3000 random edges. (a) shows the measured runtimes depending on the number of inserted edges. Analogously, (b) shows the measured runtimes for the deletion of 3000 edges in a random order till the empty graph is obtained again. The horizontal axes describe the current number of edges m. The vertical axes show the measured runtimes averaged over three million runs.

The runtime of the second phase is constant. The while loop in the third phase has an expected length of $\mathcal{O}(\mu(p))$. Since adding the path π_{xy} to the priority queue H costs $\mathcal{O}(\log n)$, the runtime of phase 3 is $\mathcal{O}(\mu(p) \log n)$.

For the analysis of the runtime of phase 4 it is crucial to observe that every line in the for loops as well as every other line is executed $\mathcal{O}(\mu(p))$ times in expectation. Moreover, the algorithm has to add the extended paths $\pi_{x'y}$ and $\pi_{xy'}$ to lists of locally shortest path and the priority queue H which is done in time $\mathcal{O}(n^{\varepsilon})$ in every execution. Thus, the runtime of the algorithm in phase 4 is $\mathcal{O}(\mu(p)n^{\varepsilon})$.

If the algorithm performs an insertion or an update, the set Q in phase 5 contains all shortest paths of G that stop being shortest. If the algorithm performs a deletion, Q is empty. Thus, the algorithm is running through the while loop $\mathcal{O}(\mu(p))$ times in expectation. The for loops are both performed $\mathcal{O}(n^{\varepsilon/2})$ times using Lemma 7 in the same way as in the beginning of this proof but with $\varepsilon/2$ instead of ε. In the same way, we can bound the expected runtime of the lines in the for loops by $\mathcal{O}(n^{\varepsilon/2})$. Altogether this gives an expected runtime of $\mathcal{O}(\mu(p)n^{\varepsilon})$ in phase 5. □

With this we can now conclude our main result.

Theorem 11. *Let $\mathcal{R}(p)$ denote the expected runtime for an edge update in a graph $G \in \mathcal{G}(n,p)$. For all $\varepsilon, \varepsilon' > 0$ we have shown that*

(i) $\mathcal{R}(p) = \mathcal{O}(n^{\varepsilon})$ *for $pn < 1/2$,*
(ii) $\mathcal{R}(p) = \mathcal{O}(n^{2/3+\varepsilon})$ *for $pn \leq 1 - n^{-1/3}$,*
(iii) $\mathcal{R}(p) = \mathcal{O}(n^{1+\varepsilon})$ *for $pn \leq 1 + n^{-1/3}$,*
(iv) $\mathcal{R}(p) = \mathcal{O}(n^{4/3+\varepsilon})$ *for $pn \leq 1 + \varepsilon'$,*
(v) $\mathcal{R}(p) = \mathcal{O}(n^{\varepsilon}/p)$ *for $pn \geq 1 + \varepsilon'$.*

Let us give an intuition how the properties of $\mathcal{G}(n, p)$ change when more and more edges are inserted and how this affects $\mathcal{R}(p)$. In the early stage (i) of the random graph process, the graph consists of many small components of size $O(\log n)$ which are trees or unicyclic. There, it is very fast to update edges. Soon after in stage (ii), the components become larger and it becomes likely for a new edge to connect two of them. Therefore, the expected number of new (locally) shortest paths increases significantly. In stage (iii) and (iv) a giant component grows and the algorithm has to update many (locally) shortest paths whenever the giant component catches other components of the graph. In (v) the last isolated vertex joins the giant component and the graph becomes connected. As the process evolves, the minimum degree and the connectivity grows and it becomes less and less likely that an inserted edge is a shortest path. Thus, also the expected insertion costs are going down.

6   Empirical Observations

To show that the theoretically observed behavior indeed occurs in practice, we also performed some experiments. For this, we used the original algorithm of Demetrescu and Italiano [5] available from www.dis.uniroma1.it/∼demetres/experim/dsp/^{2}. As the number of locally shortest paths between any pair of nodes has been reported to be very small [5], we assume that the experimental performance of our algorithm described in Section 3 should be similar to that of Demetrescu and Italiano.

We start with an empty graph with $n = 100$ vertices and add 3000 edges in a random order. Figure 2(a) shows the measured runtimes per insertion averaged over three million runs. Afterwards, we examine the opposite direction and remove all edges in a random order. The measured average runtimes per deletion are shown in Figure 2(b). Note that as predicted in Theorem 11, both charts identify the largest update complexity shortly after the critical window.

Acknowledgements

References

[1] Addario-Berry, L., Broutin, N., Reed, B.: The diameter of the minimum spanning tree of a complete graph. In: Proc. Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities (2006)

---

^{2} Analogous to the experiments of Demetrescu and Italiano [5], we used the `D-LHP` code without smoothing. All experiments are conducted on 2.4 GHz Opteron machines running Debian GNU/Linux.

[2] Bollobás, B.: Random Graphs. Cambridge Univ. Press, Cambridge (2001)
[3] Chung, F., Lu, L.: The diameter of sparse random graphs. Advances in Applied Mathematics 26, 257–279 (2001)
[4] Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. ACM 51, 968–992 (2004)
[5] Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans. Algorithms 2, 578–601 (2006)
[6] Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Algorithms and Theory of Computation Handbook, ch. 8 (1999)
[7] Erdős, P., Rényi, A.: On random graphs. Publ. Math. Debrecen. 6, 290–297 (1959)
[8] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615 (1987)
[9] Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic shortest paths and negative cycles detection on digraphs with arbitrary arc weights. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 320–331. Springer, Heidelberg (1998)
[10] King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), pp. 81–91 (1999)
[11] Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. of the AMS 7, 48–50 (1956)
[12] Nachmias, A., Peres, Y.: The critical random graph, with martingales. Israel Journal of Mathematics (2008) arXiv:math/0512201
[13] Ramalingam, G., Reps, T.W.: On the computational complexity of dynamic graph problems. Theor. Comput. Sci. 158, 233–277 (1996)
[14] Roditty, L., Zwick, U.: Dynamic approximate all-pairs shortest paths in undirected graphs. In: Proc. 45th Annual Symposium on Foundations of Computer Science (FOCS 2004), pp. 499–508 (2004a)
[15] Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proc. 36th Annual ACM Symposium on Theory of Computing (STOC 2004), pp. 184–191 (2004b)
[16] Roditty, L., Zwick, U.: On dynamic shortest paths problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 580–591. Springer, Heidelberg (2004)
[17] Thorup, M.: Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 384–396. Springer, Heidelberg (2004)
[18] Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proc. 37th Annual ACM Symposium on Theory of Computing (STOC 2005), pp. 112–119. ACM Press, New York (2005)