All-Pairs Shortest Paths in $O(n^2)$ time with high probability

Yuval Peres [*] *Dmitry Sotnikov* [†] *Benny Sudakov* [‡] *Uri Zwick* [§]

Abstract

We present an all-pairs shortest path algorithm whose running time on a complete directed graph on n vertices whose edge weights are chosen independently and uniformly at random from $[0, 1]$ is $O(n^2)$, in expectation and with high probability. This resolves a long standing open problem. The algorithm is a variant of the dynamic all-pairs shortest paths algorithm of Demetrescu and Italiano. The analysis relies on a proof that the number of *locally shortest paths* in such randomly weighted graphs is $O(n^2)$, in expectation and with high probability. We also present a dynamic version of the algorithm that recomputes all shortest paths after a random edge update in $O(\log^2 n)$ expected time.

1 Introduction

The *All-Pairs Shortest Paths* (APSP) problem is one of the most important, and most studied, algorithmic graph problems. Given a weighted directed graph $G = (V, E, c)$, on $|V| = n$ vertices and $|E| = m$ edges, where $c : E \to \mathbb{R}^+$ is a length (or cost) function defined on its edges, we would like to compute the *distances* between all pairs of vertices in the graph and a succinct representation of all *shortest paths.* (The length of a path is the sum of the lengths of the edges participating in the path.)

The APSP problem can be solved in $O(mn + n^2 \log n)$ worst-case time by running Dijkstra's algorithm from each vertex of the graph. (See Dijkstra [11], Fredman and Tarjan [13].) A slightly better running time of $O(mn + n^2 \log\log n)$ was obtained by Pettie [29], building on techniques developed by Thorup [32]. Karger, Koller and Phillips [21] and McGeoch [23] developed algorithms that run in $O(m^*n + n^2 \log n)$ time, where m^* is the number of edges in the graph that are shortest paths.

Demetrescu and Italiano [7, 8] (see also Thorup [33]) obtained a *dynamic* APSP algorithm with an *amortized* vertex update time of $\tilde{O}(n^2)$. Thorup [34] obtained a dynamic algorithm with an $\tilde{O}(n^{2.75})$ *worst-case* vertex update time. A *vertex update* may insert, delete and change the weight of edges that touch a given vertex v. An *edge update* may only insert, delete or change the weight of a single edge. The algorithms of Demetrescu and Italiano [7, 8] and Thorup [33, 34] can be used, of course, to perform edge updates, but the updates times may still be $\tilde{O}(n^2)$ and $\tilde{O}(n^{2.75})$, respectively.

Many researchers developed APSP algorithms that work well on *random* instances, most notably complete directed graphs on n vertices with random weights on their edges. The simplest such model, on which we focus in this paper, is the one in which all edge weights are drawn independently at random from the *uniform* distribution on $[0, 1]$. Hassin and Zemel [19] and Frieze and Grimmett [16] observed that, with very high probability, only the $O(\log n)$ cheapest edges emanating from each vertex participate in shortest paths. Thus, the APSP in this setting can be solved in $O(n^2 \log n)$ expected time using the algorithms of

[*]Microsoft Research, Redmond. E-mail: peres@microsoft.com.

[†]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: dmitrysot@gmail.com.

[‡]Department of Mathematics, UCLA, Los Angeles, CA 90095, USA. E-mail: bsudakov@math.ucla.edu. Research supported in part by NSF CAREER award DMS-0812005 and by a USA-Israeli BSF grant.

[§]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: zwick@tau.ac.il. Work supported by grant no. 2006261 of the United-States - Israel Binational Science Foundation (BSF).

Karger *et al.* [21] and McGeoch [23], or by simply selecting the $O(\log n)$ cheapest edges emanating from each vertex and then running Dijkstra's algorithm from each vertex. All these results actually hold in the more general setting in which edge weights are independent identically distributed random variables with a common cumulative distribution function F that satisfies $F(0) = 0$ and $F'(0)$ exists and is strictly positive. (The uniform distribution on $[0, 1]$ with $F(x) = x$, and the *exponential* distribution, with $F(x) = 1 - \mathrm{e}^{-x}$ clearly satisfy these conditions.) Furthermore, the running time of these algorithms is $O(n^2 \log n)$ with *high probability*, i.e., probability that tends to 1 as n tends to infinity, and not just in expectation.

Spira [31] obtained an APSP algorithm with an expected running time of $O(n^2 \log^2 n)$ for complete directed graphs with edge weights drawn in an *endpoint independent* manner. More specifically, for each vertex v a sequence of n positive numbers is chosen by an arbitrary deterministic or probabilistic process. These n numbers are then assigned to the n edges emanating from v in a *random* order, with all $n!$ possible ordering being equally likely. Bloniarz [2] presented an improved algorithm with an expected running time of $O(n^2 \log n \log^* n)$. Moffat and Takaoka [27] and Mehlhorn and Priebe [24] improved the expected running time to $O(n^2 \log n)$ and showed that it also holds with high probability.

Cooper *et al.* [4] obtained an APSP algorithm with an expected running time of $O(n^2 \log n)$ in the *vertex potential* model in which edge weights may be both positive and negative.

Meyer [25], Hagerup [18] and Goldberg [17] obtained *Single-Source Shortest Paths* (SSSP) algorithms with an expected running time of $O(m)$. The m-edge input graph may be arbitrary but its edge weights are assumed to be chosen at random from a common non-negative probability distribution. When the edge weights are independent, the running time of these algorithms is $O(m)$ with high probability.

Friedrich and Hebbinghaus [14] presented an average case analysis of the dynamic APSP algorithm of Demetrescu and Italiano [7, 8] on random *undirected* graphs. The graphs in their analysis are chosen according to the $G(n, p)$ model, in which each edge of the complete graph is selected with probability p, and edges of the random graph are given i.i.d. uniform random weights. They show that the expected edge update time is at most $O(n^{4/3+\epsilon})$, for any $\epsilon > 0$. This bound is essentially tight when $p = 1/n$, i.e., at the phase transition of the random graph, when the largest component is, with high probability, of size $\Theta(n^{2/3})$. When $p \geq (1 + \epsilon')/n$, they show that the expected update time is $O(n^\epsilon/p)$, for every $\epsilon > 0$.

Non-algorithmic aspects of distances and shortest paths in randomly weighted graphs were also a subject of intensive research in probability theory. We mention here only the results that are most relevant for us. Davis and Prieditis [5] and Janson [20] showed that the expected distance of two vertices in a complete graph with random edge weights drawn independently from an *exponential* distribution with mean 1 (i.e., $F(x) = 1 - \mathrm{e}^{-x}$) is exactly $H_{n-1}/(n-1) = (\ln n)/n + O(1/n)$, where $H_k = \sum_{i=1}^{k} \frac{1}{k}$ is the k-th Harmonic number. The probability that a given edge is a shortest path between its endpoints is also exactly $H_{n-1}/(n-1)$. Exponential random variables are convenient to work with due to their *memoryless* property. The same asymptotic results hold when the edges weights are chosen independently and uniformly from $[0, 1]$. In the exponential case, the tree of shortest paths from a given vertex has the same distribution as a *random recursive tree* on n vertices obtained using the following simple process: Start with a root; add the remaining $n - 1$ vertices, each time choosing the parent of the new vertex uniformly at random among the vertices that are already in the tree. The expected *depth* of a vertex in such a tree, and hence the expected number of edges in a shortest path, is $\ln n + O(1)$. For further results regarding recursive trees and shortest paths, see Devroye [9], Smythe and Mahmoud [30] and Addario-Berry *et al.* [1].

In their survey on the algorithmic theory of random graphs, Frieze and McDiarmid [15] state the following open problem (Research Problem 22 on p. 28): "Find a $o(n^2 \log n)$ expected time algorithm for the all pairs problem under a natural class of distributions, e.g., i.i.d. uniform on $[0, 1]$." We solve this open problem by giving an $O(n^2)$ expected time algorithm for the problem, which is of course best possible. Furthermore, our algorithm runs in $O(n^2)$ time with high probability and works for both directed and undirected versions of the all-pairs shortest paths problem.

Our $O(n^2)$-time APSP algorithm is a static version of the dynamic APSP algorithm of Demetrescu and

Italiano [7, 8] (see especially Section 3.4 of [8]) with some modified data structures. The novel part of this paper is not the algorithm itself, but rather the probabilistic analysis that shows that it runs in $O(n^2)$ time, in expectation and with high probability.

We also obtain an $O(\log^2 n)$ upper bound on the expected time needed to update all shortest paths following a random edge update, i.e., an update in which a random edge of the complete directed graph is selected and given a new random edge weight drawn uniformly at random from $[0, 1]$.

The rest of the paper is organized as follows. In Section 2 we sketch the static and dynamic versions of the algorithm of Demetrescu and Italiano [7, 8] used in this paper. (Complete descriptions of these algorithms are given in Appendices A and B.) The crucial factor that determines the running time of these algorithms is the number of *locally shortest paths* in the graph. A path is a *locally* shortest path (*LSP*) if the paths obtained by deleting its first and last edge, respectively, are *shortest paths.* In Section 3 we collect some known and some new results regarding the distances between vertices in randomly weighted graphs. Using the results of Section 3, we show in Section 4 that the *expected* number of LSPs in a complete directed graph with independent uniformly distributed random weights is $O(n^2)$. In Section 5 we show that the number of LSPs is $O(n^2)$ *with high probability.* Sections 4 and 5 are the main sections of the paper. In Section 6 we show that a fairly simple *bucket* based priority queue, with a constant amortized update time, in conjunction with the fact that the number of LSPs is $O(n^2)$, in expectation and with high probability, yields the promised $O(n^2)$-time APSP algorithm. In Section 7 we consider the expected time needed to perform *random edge* updates. Interestingly, the arguments used in Sections 5 and 7 are related, as they both focus on the expected number of shortest paths that *change* when a single edge is given a new random edge weight. (The link is the Efron-Stein inequality used in Section 5.) In Section 8 we very briefly consider other random graph models. In particular, our algorithm still runs in $O(n^2)$ expected time in the directed $G(n, p)$ model, in which each edge is present with probability p, with independent uniformly distributed edge weights, at least when $p \gg (\ln n)/n$. We end in Section 8 with some concluding remarks and open problems.

2 The algorithm of Demetrescu and Italiano

Our $O(n^2)$ time bound, in expectation and with high probability, on the complexity of the solving the APSP problem on complete directed graphs with independent edge weight drawn uniformly from $[0, 1]$, and the $O(\log^2 n)$ expected time bound on the complexity of performing a random edge update are both obtained using variants of the dynamic APSP algorithm of Demetrescu and Italiano [7, 8].

As our main result is the *analysis* of these variants, and not the variants themselves, we begin by sketching the main features of the variants we use, mentioning only what the reader needs to know to understand our analysis. A complete description of the algorithms is given in Appendices A and B. (We believe that our variants are also of some interest, as they are not identical to the algorithms of [7, 8].)

Let $G = (V, E, c)$ be a weighted directed graph, where $c : E \to (0, \infty)$ is a cost function defined on its edges. (We use weights and costs interchangingly.) For simplicity, we assume that all shortest paths in G are *unique.* Under essentially all probabilistic models considered in this paper, this assumption holds with probability 1. (Non-uniqueness of shortest paths can be dealt with as in [7].) We let $u \to v$ denote the edge $(u, v) \in E$, and let $u \rightsquigarrow v$ denote the (unique) shortest path from u to v in the graph, if they exist.

The key notion behind the algorithm of Demetrescu and Italiano [7] is the notion of *locally shortest paths.*

Definition 2.1 (Locally Shortest Paths) *A path is a* locally shortest path *(or* LSP*, for short) if the path obtained by deleting its first edge, and the path obtained by deleting its last edge, are both shortest paths.*

More formally, if we let $u \to u' \rightsquigarrow v' \to v$ denote the path composed of the edge $u \to u'$, followed by the shortest path from u' to v', and then by the edge $v' \to v$, then $u \to u' \rightsquigarrow v' \to v$ is a locally shortest path if and only if $u \to u' \rightsquigarrow v'$ and $u' \rightsquigarrow v' \to v$ are both shortest paths. (If $u' = v'$, then $u' \rightsquigarrow v'$ is an empty path.) An edge is considered to be a locally shortest path. (Empty paths are considered to be shortest paths.) A shortest path is of course also a locally shortest path. A locally shortest path, however, is not necessarily a shortest path.

2.1 A static version

We begin by describing a static version of the algorithm of Demetrescu and Italiano [7, 8]. Let $G = (V, E, c)$ be a weighted directed graph. The algorithm constructs all shortest paths in G by essentially running Dijkstra's algorithm in parallel from all vertices, while only examining LSPs, as explained below.

For every $u, v \in V$, the algorithm maintains a number $dist[u, v]$ which is the length of the shortest path from u to v found so far. Initially $dist[u, v]$ is set to $c(u, v)$, if $(u, v) \in E$, or to ∞, otherwise. Each pair $(u, v) \in E$ is inserted into a heap (priority queue) Q, with $dist[u, v]$ serving as its key. The heap Q holds all pairs of vertices (u, v) such that at least one path from u to v in the graph was already discovered, but the shortest path from u to v was not yet declared.

In each iteration, the algorithm extracts a pair (u, v) with the smallest key in Q. As in Dijkstra's algorithm, $dist[u, v]$ is then the distance from u to v in G. The algorithm then examines LSPs that *extend* the shortest path $u \rightsquigarrow v$ and checks whether they are shorter than the currently best available paths between their endpoints. (An extension of a path π is a path obtained by adding an edge to its beginning or end.) To efficiently find the LSPs that extend a shortest path $u \rightsquigarrow v$, the algorithm also maintains, in addition to $dist[u, v]$, the following information for every $u, v \in V$:

$p[u, v]$ – The *second* vertex on the shortest path from u to v found so far.

$q[u, v]$ – The *penultimate* (next to last) vertex on the shortest path from u to v found so far.

$L[u, v]$ – A list of vertices w for which $w \to u \rightsquigarrow v$ is known to be a shortest path.

$R[u, v]$ – A list of vertices w for which $u \rightsquigarrow v \to w$ is known to be a shortest path.

If no path from u to v was found yet, then $p[u, v] = q[u, v] = null$. The lists $L[u, v]$ and $R[u, v]$ specify the *left* and *right* extensions of $u \rightsquigarrow v$ that are known to be shortest paths. Clearly $L[u, v]$ and $R[u, v]$ are non-empty only after the shortest path $u \rightsquigarrow v$ was identified by the algorithm.

Suppose that $u \to u' \rightsquigarrow v' \to v$, where $u' = p[u, v]$ and $v' = q[u, v]$, was just identified as a shortest path. For every $w \in L[u, v']$, $w \to u \rightsquigarrow v$ is an LSP. Similarly, for every $w \in R[u', v]$, $u \rightsquigarrow v \to w$ is an LSP. These paths are now examined by the algorithm. If, for example, a path $w \to u \rightsquigarrow v$ is found to be shorter then the currently available path from w to v, or is the first path found from w to v, then $dist[w, v]$, $p[w, v]$ and $q[w, v]$ are updated accordingly and the key of (w, v) in Q is decreased. (If (w, v) is not already in Q, it is inserted into Q.)

This is the gist of the static version of the algorithm of Demetrescu and Italiano [7, 8], which, for concreteness, we refer to as algorithm `apsp`. For a complete description and pseudo-code, see Appendix A.

As algorithm `apsp` uses a priority queue, its running time depends on the characteristics of the priority queue used. For a specific implementation, we let $T_{ins}(n), T_{dec}(n)$ and $T_{ext}(n)$ denote the (amortized) times of *inserting* an element, *decreasing* the key of a given element, and *extracting* an element of minimum key from a priority queue containing at most n elements. We next claim:

Theorem 2.2 *If all edge weights are positive and all shortest paths are unique, then algorithm* `apsp` *correctly finds all the shortest paths in the graph. Algorithm* `apsp` *runs in* $O(n^2 \cdot (T_{ins}(n^2) + T_{ext}(n^2)) + |\mathcal{LSP}| \cdot T_{dec}(n^2))$ *time, where* $|\mathcal{LSP}|$ *is the number of LSPs in the graph, and uses only* $O(n^2)$ *space.*

The proof of Theorem 2.2, which is essentially identical to the correctness proof given by Demetrescu and Italiano [7, 8], can be found in Appendix A.

If we use the Fibonacci heaps data structure (Fredman and Tarjan [13]) that supports `extract-min` operations in $O(\log n)$ amortized time, and all other operations in $O(1)$, amortized time, where n is the number of elements in the heap, we get a running time of $O(n^2 \log n + |\mathcal{LSP}|)$. There are, thus, two hurdles on our way to getting an expected $O(n^2)$-time algorithm. First, we have to show that $|\mathcal{LSP}|$ is $O(n^2)$, under natural probability distributions, in expectation and with high probability. We do that in Sections 4 and 5. Second, we have to find a faster way of implementing heaps. We do that in Section 6 using a *bucket* based implementation.

2.2 A dynamic version

The static algorithm of the previous section examines all locally shortest paths in a graph, but (implicitly) maintains only those that are currently shortest. The dynamic algorithm, on the other hand, explicitly maintains all locally shortest paths, even if they are already known not to be shortest paths.

For every path π, we let $l[\pi]$ be the path obtained by deleting the last edge of π, and $r[\pi]$ be the path obtained by deleting the first edge of π. A path π is represented by keeping its total cost, its first and last edges, and pointers to its subpaths $l[\pi]$ and $r[\pi]$. The collection of all paths maintained by the algorithm is referred to as the *path system*.

For every pair of vertices $u, v \in V$, the dynamic algorithm maintains a heap $P[u, v]$ that holds all the LSPs connecting u and v found so far. The key of each path is its cost. As in the static case, $dist[u, v]$ is the cost of the shortest path $\pi[u, v]$ from u to v found so far.

For every LSP π, the dynamic algorithm maintains four lists of left and right extensions of π. The lists $SL[\pi]$ and $SR[\pi]$ contain left and right extensions of π that are known to be shortest paths. The lists $L[\pi]$ and $R[\pi]$ contain extensions of π that are known to be LSPs.

Let E' be a set of edges whose costs are changed by an update operation. (We are mostly be interested in the case in which E' is composed of a single edge, but the description below is general.) The dynamic algorithm recomputes all shortest paths as follows. First all LSPs containing edges of E' are removed from the path system. (Note that each edge of E' is an LSP, and is thus contained in the path system. All LSPs containing edges of E' can be found by recursively following the extension lists of these edges.)

For every pair of vertices $u, v \in V$ such that the shortest path from u to v before the update passes through an edge of E', and was therefore removed from the path system, the algorithm finds the cheapest path in $P[u, v]$, if at least one such path remains, and assigns it to $\pi[u, v]$. It then inserts the pair (u, v) into a global heap Q. The key of (u, v) in Q is the cost of $\pi[u, v]$. Next, it recreates single-edge paths corresponding to the edges of E', with their new edge weights, and examines them.

The dynamic algorithm now starts to construct new shortest paths. In each iteration it extracts from Q a pair (u, v) with the smallest key. As in the static case, the path $\pi[u, v]$ is then a shortest path from u to v. LSP extensions of $\pi[u, v]$, obtained by combining $\pi[u, v]$ with paths that are already known to be shortest paths, are now generated. If such an extension is shorter than the currently shortest available path containing its endpoints u' and v', then $\pi[u', v']$ and $dist[u', v']$ are updated accordingly, and (u', v') is inserted into Q with the appropriate key. (If (u', v') is already in Q, its key is decreased.)

An important difference between the dynamic variant used in this paper and the dynamic algorithm of Demetrescu and Italiano [7, 8] is that when a path π stops being a shortest path, it, and all its extensions, are immediately removed from the path system. A similar dynamic variant was used by Friedrich and Hebbinghaus [14]. The algorithm of Demetrescu and Italiano [7, 8] keeps such paths as *historical* and *locally historical* paths. (See also Demetrescu *et al.* [6].)

The most impressive feature of the dynamic algorithm of Demetrescu and Italiano [7, 8] is that its update time is proportional to the number of shortest and locally shortest paths that are destroyed and/or created by the update operation. The algorithm does not spend time on shortest paths that remain unchanged.

Let $\mathcal{SP}^-$ and $\mathcal{LSP}^-$ be the sets of shortest and locally shortest paths destroyed by an update operation. Similarly, let $\mathcal{SP}^+$ and $\mathcal{LSP}^+$ be the sets of shortest and locally shortest paths that are created (or recreated) by an update operation. Note that $\mathcal{SP}^-$ and $\mathcal{SP}^+$, and $\mathcal{LSP}^-$ and $\mathcal{LSP}^+$, are not necessarily disjoint, as paths passing through edges of E' are first destroyed, and removed from the path system, but may then be recreated. Let Λ be an upper bound on the number of LSPs that connect any given pair of vertices before and after the update.

A complete description of the dynamic variant sketched here and its correctness proof are given in Appendix B, where the following theorem is proved. ($\texttt{update}(E', c')$ is the function that updates all shortest paths following a change in the costs of the edges of E'.)

Theorem 2.3 *The running time of* $\texttt{update}(E', c')$ *is*

$$O(\ |\mathcal{SP}^-|\cdot(T_{del}(\Lambda)+T_{min}(\Lambda)+T_{ins}(n^2))+|\mathcal{SP}^+|\cdot T_{ext}(n^2)+|\mathcal{LSP}^-|\cdot T_{del}(\Lambda)+|\mathcal{LSP}^+|\cdot(T_{ins}(\Lambda)+T_{dec}(n^2)\ ).$$

Here, $T_{ins}(n), T_{del}(n), T_{dec}(n), T_{ext}(n)$ and $T_{min}(n)$ are the (amortized) times of inserting, deleting, decreasing the key, extracting the element of minimum key, and finding the element of minimum key of a priority key containing at most n elements.

We show in Section 7 that for a random edge update we have $\mathbb{E}[\,|\mathcal{SP}^-|\,], \mathbb{E}[\,|\mathcal{SP}^+|\,] = O(\log n)$ and that $\mathbb{E}[\,|\mathcal{LSP}^-|\,], \mathbb{E}[\,|\mathcal{LSP}^+|\,] = O(\log^2 n)$. We also show that $\Lambda = O(\log n)$, with high probability. Using appropriate implementations of the priority queues, we get an expected edge update time of $O(\log^2 n)$.

3 Distances in complete randomly weighted graphs

Let $K_n = (V, E)$ be a complete directed graph on n vertices and let $a, b \in V$. We let $W(a, b)$ be the random weight attached to the edge (a, b). We assume at first that $W(a, b)$ is an *exponential* random variable with mean 1, i.e., $W(a, b) \sim EXP(1)$. Due to the memoryless property, dealing with exponentially distributed edge weights is easier than dealing directly with uniformly distributed edge weights. We later explain why all the results derived in this section for exponential edge weights also hold, asymptotically, for uniformly distributed edge weights. All $n(n-1)$ random edge weights are assumed to be independent. (Self-loops, if present, may be ignored.) Let $D(a, b)$ be the *distance* from a to b in the graph, i.e., the length (sum of weights) on the shortest path $a \rightsquigarrow b$ in the graph. (The shortest path $a \rightsquigarrow b$ is unique with probability 1.) Note that $D(a, b)$ is now also a random variable. For $k \in \{1, 2, \ldots, n-1\}$, we let $D_k(a)$ be the distance from a to the k-th closest vertex to a.

Let $H_k = \sum_{k=1}^{n} \frac{1}{k}$ be the k-th *Harmonic number*. It is known that $H_n = \ln n + \gamma + O(\frac{1}{n})$, where $\gamma = 0.57721\ldots$ is Euler's constant.

The following five lemmas can be found in Janson [20]. (The expectation of $D(a, b)$, but not the variance, can also be found in Davis and Prieditis [5]). The lemmas in [20] are stated for undirected graphs, but it is easy to check that they also hold for directed graphs.

Lemma 3.1 *Let $a \in V$ and $k \in \{1, 2, \ldots, n-1\}$. Then,*

$$D_k(a) \ = \ \sum_{i=1}^{k} \frac{X_i}{i(n-i)}\ ,$$

where $X_1, X_2, \ldots, X_k$ are i.i.d. exponential variables with mean 1.

Lemma 3.2 *Let $a \neq b \in V$. Then,*

$$D(a,b) \;=\; D_L(a) \;=\; \sum_{i=1}^{L} \frac{X_i}{i(n-i)}\ ,$$

where $X_1, X_2, \ldots, X_{n-1}$ are i.i.d. exponential variables with mean 1, and L is chosen uniformly at random from $\{1, 2, \ldots, n-1\}$.

Lemma 3.3 *Let $a \neq b \in V$. Then,*

$$\mathbb{E}[D(a,b)] \;=\; \frac{H_{n-1}}{n-1} \;=\; \frac{\ln n}{n} + O(\frac{1}{n}) \quad , \quad \mathrm{Var}[D(a,b)] \;=\; \frac{\pi^2}{2n^2} + o(\frac{1}{n^2}).$$

Lemma 3.4 *For any constant $c > 3$, we have $\mathbb{P}\left[\max_{a,b} D(a,b) \geq \frac{c \ln n}{n}\right] = O(n^{3-c} \log^2 n)$.*

Lemma 3.5 *Let $a \neq b \in V$. Then, the probability that the edge $a \to b$ is a shortest path is $\frac{H_{n-1}}{n-1} = \frac{\ln n}{n} + O(\frac{1}{n})$.*

The next two lemmas are new and might be interesting in their own right. They are used in Section 5 to show that the running time of algorithm `apsp` is $O(n^2)$ with high probability. The proof that the expected running time of `apsp` is $O(n^2)$, given in Section 4, does not rely on them.

Lemma 3.6 *Let $a \neq b \in V$. If $n^{-\alpha} < \alpha \leq 1/2$, then $\mathbb{P}[D(a,b) > (1+12\alpha)\frac{\ln n}{n}] \leq 5n^{-\alpha}$.*

Proof: Let $S_{k,\ell} = \sum_{i=k}^{\ell} \frac{X_i}{i(n-i)}$. (We allow k, ℓ to be non-integral, in which case, we have $S_{k,\ell} = S_{\lceil k \rceil, \lfloor \ell \rfloor}$.) By Lemma 3.2 we get that $D(a,b) = S_{1,L}$, where L is uniformly distributed in $\{1, 2, \ldots, n-1\}$. Let $m = n^{1-\alpha}$. We clearly have

$$\mathbb{P}[D(a,b) > S_{1,n-m}] \;=\; \mathbb{P}[L > n-m] \;\leq\; m/n \;=\; n^{-\alpha}. \tag{1}$$

We now decompose

$$S_{1,n-m} \;\leq\; \frac{X_1}{n-1} + S_{2,m} + S_{m,n/2} + S_{n/2,n-m}. \tag{2}$$

Now

$$\mathbb{P}\left[\frac{X_1}{n-1} > 2\alpha \frac{\ln n}{n}\right] \;\leq\; \mathbb{P}[X_1 > \alpha \ln n] \;=\; n^{-\alpha}. \tag{3}$$

Let $Y = \sum_{i=2}^{m} \frac{X_i}{i}$. Using our assumption that $n^{-\alpha} < \alpha$ we get that

$$S_{2,m} \;\leq\; \frac{Y}{n-m} \;=\; \frac{Y}{(1-n^{-\alpha})n} \;\leq\; \frac{Y}{(1-\alpha)n} \;\leq\; (1+2\alpha)\frac{Y}{n}.$$

Now $\mathbb{E}[\mathrm{e}^{\lambda X_i}] = (1-\lambda)^{-1}$, for $\lambda < 1$, so

$$\mathbb{E}\left[\mathrm{e}^{Y}\right] \;=\; \mathbb{E}\left[\mathrm{e}^{\sum_{i=2}^{m} X_i/i}\right] \;=\; \prod_{i=2}^{m} \mathbb{E}\left[\mathrm{e}^{X_i/i}\right] \;=\; \prod_{i=2}^{m} \left(1 - \frac{1}{i}\right)^{-1} \;=\; \prod_{i=2}^{m} \frac{i}{i-1} \;=\; m.$$

Therefore,

$$\mathbb{P}\left[S_{2,m-1} \geq (1+2\alpha)\frac{\ln n}{n}\right] \;\leq\; \mathbb{P}[Y \geq \ln n] \;=\; \mathbb{P}[\mathrm{e}^{Y} \geq n] \;\leq\; \frac{\mathbb{E}[\mathrm{e}^{Y}]}{n} \;=\; \frac{m}{n} \;=\; n^{-\alpha}. \tag{4}$$

Let $Z = \sum_{i=m}^{n/2} \frac{X_i}{i}$. Clearly, $S_{m,n/2} \le \frac{2Z}{n}$. We again have

$$\mathbb{E}[\mathrm{e}^Z] \;=\; \prod_{i=m}^{n/2}(1-\frac{1}{i})^{-1} \;\le\; \frac{n}{m} \;=\; n^{\alpha}.$$

Therefore,

$$\mathbb{P}\left[S_{m,n/2} \ge 4\alpha\frac{\ln n}{n}\right] \;\le\; \mathbb{P}[Z > 2\alpha\ln n] \;=\; \mathbb{P}[\mathrm{e}^Z \ge n^{2\alpha}] \;\le\; \frac{\mathbb{E}[\mathrm{e}^Z]}{n^{2\alpha}} \;=\; n^{-\alpha}. \tag{5}$$

As $S_{m,n/2}$ and $S_{n/2,n-m}$ have exactly the same distribution, we also get that

$$\mathbb{P}\left[S_{n/2,n-m} \ge 4\alpha\frac{\ln n}{n}\right] \le n^{-\alpha} \tag{6}$$

Using (1)-(6) together, we get that $\mathbb{P}\left[D(a,b) > (1+12\alpha)\frac{\ln n}{n}\right] \le 5n^{-\alpha}$, we required. □

No attempt to optimize the constants appearing the statement of Lemma 3.6. The condition $n^{-\alpha} < \alpha$ in the lemma is satisfied for any fixed $\alpha > 0$, when n is large enough. It also holds when, say, $\alpha = \alpha(n) = (\ln\ln n)/\ln n$.

The proof of our next lemma relies on the following large deviation theorem of Maurer [22].

Theorem 3.7 (Maurer [22]) *Let $Y_1, Y_2, \ldots, Y_n$ be non-negative independent random variables with finite first and second moments and let $S = \sum_{i=1}^n Y_i$. Let $t > 0$. Then*

$$\mathbb{P}\Big[\mathbb{E}[S] - S \ge t\Big] \le \exp\left(-\frac{t^2}{2\sum_{i=1}^n \mathbb{E}[Y_i^2]}\right).$$

For a vertex $a \in V$ and $r > 0$, let $Ball(a,r) = \{b \in V \mid D(a,b) \le r\}$ be the *ball* of radius r centered at a. We next bound the probability that $Ball(a, \alpha\frac{\ln n}{n})$ is exceptionally large.

Lemma 3.8 *For any $a \in V$, $\alpha \le 1$ and $c > 0$ we have*

$$\mathbb{P}\left[\left|Ball\left(a, \alpha\frac{\ln n}{n}\right)\right| > cn^{\alpha}\right] \;\le\; \exp\left(-\frac{\ln^2 c}{60}\right).$$

Proof: Note that $|Ball(a,r)| > k$ if and only if $D_k(a) \le r$. By Lemma 3.1 we have $D_k = D_k(a) = \sum_{i=1}^k \frac{X_i}{i(n-i)}$, where $X_1, X_2, \ldots, X_k$ are i.i.d. exponential variables with mean 1. Thus,

$$\mathbb{E}[D_k] \;=\; \sum_{i=1}^k \frac{1}{i(n-i)} \;>\; \frac{1}{n}\sum_{i=1}^k \frac{1}{i} > \frac{\ln k}{n}.$$

As $\mathbb{E}[X_i^2] = 2$, we have

$$\sum_{i=1}^k \mathbb{E}\left[\left(\frac{X_i}{i(n-i)}\right)^2\right] = \sum_{i=1}^k \frac{2}{i^2(n-i)^2} \le \sum_{i=1}^{n-1} \frac{2}{i^2(n-i)^2} \le 2\sum_{i=1}^{n/2} \frac{2}{i^2(n-i)^2} \le \frac{16}{n^2}\sum_{i=1}^{n/2}\frac{1}{i^2} \le \frac{16}{n^2}\frac{\pi^2}{6} \le \frac{30}{n^2}.$$

With $k = cn^{\alpha}$ we get that $\mathbb{E}[D_k] > \frac{\alpha\ln n}{n} + \frac{\ln c}{n}$ and by Theorem 3.7, with $Y_i = \frac{X_i}{i(n-i)}$, we have

$$\mathbb{P}\left[D_k \le \frac{\alpha\ln n}{n}\right] \;\le\; \mathbb{P}\left[\mathbb{E}[D_k] - D_k \ge \frac{\ln c}{n}\right] \;\le\; \exp\left(-\frac{\left(\frac{\ln c}{n}\right)^2}{\frac{60}{n^2}}\right) \;=\; \exp\left(-\frac{\ln^2 c}{60}\right).$$

□

As an immediate corollary, we get:

Corollary 3.9 *For any $a \in V$, $\alpha \le 1$, $\epsilon > 0$ and $c > 0$ we have*

$$\mathbb{P}\left[\left|Ball\left(a, \alpha\frac{\ln n}{n}\right)\right| > n^{\alpha+\epsilon}\right] \;=\; O(n^{-c}) \; .$$

The results of this section were derived under the assumption that the edge weights are exponential. However, as explained in detail in the beginning of Section 2 of Janson [20], the same results hold asymptotically also for the uniform distribution. For the sake of completeness we show how to deduce from Lemma 3.6 and Corollary 3.9 similar claims for uniform distributions.

Let G be a complete directed graph on n vertices with independent uniformly distributed edge weights $W(a,b)$ and let $D(a,b)$ be the distance from a to b in this graph. Define $W'(a,b) = -\ln(1 - W(a,b))$ and let G' be a complete directed graph whose edge weights are $W'(a,b)$. Denote by $D'(a,b)$ the distance from a to b in G'. Note that all the edges of G' have weights distributed as independent exponential random variables with mean 1 and that the correspondence between G and G' is a measure preserving transformation. It is easy to check that $z \le -\ln(1-z) \le z + 2z^2$ for all $0 \le z \le 1/2$.

Suppose that G has the property that $D(a,b) > (1+12\alpha)\frac{\ln n}{n}$. Since $D'(a,b) \ge D(a,b)$, each such G corresponds to a graph G' which also has $D'(a,b) > (1+12\alpha)\frac{\ln n}{n}$. Therefore by Lemma 3.6 the probability of this event is at most $5n^{-\alpha}$. Suppose that b is a vertex of G satisfying $D(a,b) \le \alpha\frac{\ln n}{n}$. Then, by the above inequality, we have that $D'(a,b) \le \alpha\frac{\ln n}{n} + 2\alpha^2\frac{\ln^2 n}{n^2} = \alpha'\frac{\ln n}{n}$ with $\alpha' = \big(1 + O(\frac{\ln n}{n})\big)\alpha$. For any $\epsilon > 0$, let $\epsilon' = \epsilon/2$. Then it is easy to check that $n^{\alpha'+\epsilon'} \le n^{\alpha+\epsilon}$. Therefore all G in which $|Ball\left(a, \alpha\frac{\ln n}{n}\right)| > n^{\alpha+\epsilon}$ correspond to instances of G' in which $|Ball\left(a, \alpha'\frac{\ln n}{n}\right)| > n^{\alpha'+\epsilon'}$. By Corollary 3.9 the probability of this event is at most $O(n^{-c})$ for any $c > 0$.

4 The expected number of locally shortest paths

Let $\mathcal{LSP}$ be the set of LSPs in K_n. Our goal in this section is to show that $\mathbb{E}[|\mathcal{LSP}|] = O(n^2)$. This would follow immediately from the following lemmas.

Lemma 4.1 *Let a, b, c be three distinct vertices. The probability that $a \to b \to c$ is an LSP is $O(\frac{\ln^2 n}{n^2})$.*

Proof: The path $a \to b \to c$ is an LSP if and only if both $a \to b$ and $b \to c$ are shortest paths. By Lemma 3.5, the probability that each one of the edges $a \to b$ and $b \to c$ is a shortest path is $\frac{\ln n}{n} + O(\frac{1}{n})$. Unfortunately, the events "$a \to b$ is a shortest path" and "$b \to c$ is a shortest path" are *not* independent (and probably positively correlated). To circumvent that, let $V_1, V_2 \subset V$ such that $V_1 \cup V_2 = V$, $a \in V_1$, $c \in V_2$, $V_1 \cap V_2 = \{b\}$ and $|V_1|, |V_2| \ge n/2$ be a fixed partition of the vertex set V. If $a \to b$ and $b \to c$ are shortest paths in G, then $a \to b$ is clearly also a shortest path in $G[V_1]$, the subgraph of G induced by V_1, and $b \to c$ is also a shortest path in $G[V_2]$. These events are now independent, as the edge sets of $G[V_1]$ and $G[V_2]$ are disjoint. The probability that $a \to b \to c$ is an LSP is thus at most $(\frac{\ln(n/2)}{n/2} + O(\frac{1}{n}))^2$. $\square$

Lemma 4.2 *Let a, b, c, d be four distinct vertices. The probability that $a \to b \rightsquigarrow c \to d$ is an LSP is $O(\frac{1}{n^2})$.*

Proof: If $a \to b \rightsquigarrow c \to d$ is an LSP, then by definition

$$W(a,b) + D(b,c) = D(a,c) \quad , \quad D(b,c) + W(c,d) = D(b,d).$$

If $a \to b \rightsquigarrow c \to d$ is an LSP, then $b \rightsquigarrow c$ does not pass through a or d. (If, for example, $b \rightsquigarrow c$ passes through a, then $a \rightsquigarrow c$ is a subpath of $b \rightsquigarrow c$, and $a \to b \rightsquigarrow c$ is therefore not a shortest path, contradicting the assumption that $a \to b \rightsquigarrow c \to d$ is an LSP.) Thus, $D(b,c) = D_{a,d}(b,c)$, where $D_{a,d}(b,c)$ is the distance from b to c when a and d are removed from the graph. We also clearly have $D(a,c) \le D_{b,d}(a,c)$ and $D(b,d) \le D_{a,c}(b,d)$.

Thus, if $a \to b \rightsquigarrow c \to d$ is an LSP, then

$$W(a,b) + D_{a,d}(b,c) \le D_{b,d}(a,c) \quad , \quad D_{a,d}(b,c) + W(c,d) \le D_{a,c}(b,d),$$

or equivalently

$$W(a,b) \le D_{b,d}(a,c) - D_{a,d}(b,c) \quad , \quad W(c,d) \le D_{a,c}(b,d) - D_{a,d}(b,c). \qquad (*)$$

It is thus sufficient to bound the probability that $(*)$ happens. For brevity, let

$$X = D_{a,d}(b,c) \quad , \quad Y = D_{b,d}(a,c) \quad , \quad Z = D_{a,c}(b,d).$$

A crucial observation now is that X, Y and Z do *not* depend on $W(a,b)$ and $W(c,d)$. This follows from the fact that in each one of these distances one of a and b, and one of c and d, is removed from the graph.

We can thus choose the random weights of the edges in two stages. First we choose the random weights of all edges *except* the two edges $a \to b$ and $c \to d$. The values of X, Y and Z are then already determined. We then choose $W(a,b)$ and $W(c,d)$, the random weights of the two remaining edges. As the choice of $W(a,b)$ and $W(c,d)$ is independent of all previous choices, and as $W(a,b)$ and $W(c,d)$ are independent and uniformly distributed in $[0,1]$, we get that

$$\mathbb{P}[(*)] \;=\; E\big[(Y-X)^+ \cdot (Z-X)^+\big] \;\le\; E\big[|Y-X||Z-X|\big],$$

where $x^+ = \max\{x, 0\}$. (Note that we are not assuming here that X, Y and Z are independent. They are in fact dependent.)

We next note that each of X, Y and Z is the distance between two given vertices in a randomly weighted complete graph on $n-2$ vertices. Thus, $\mathbb{E}[X] = \mathbb{E}[Y] = \mathbb{E}[Z]$. By Lemma 3.3, we have

$$\mathrm{Var}[X] = \mathrm{Var}[Y] = \mathrm{Var}[Z] \;=\; (1+o(1))\frac{\pi^2}{2n^2}.$$

Now,

$$\mathbb{P}[(*)] \;\le\; E\big[|Y-X||Z-X|\big] \;\le\; \frac{1}{2}(E\big[(Y-X)^2\big] + E\big[(Z-X)^2\big]),$$

using the trivial inequality $xy \le \frac{1}{2}(x^2+y^2)$. All that remains, therefore, is to bound $E\big[(Y-X)^2\big]$ and $E\big[(Z-X)^2\big]$. Let $\mu = \mathbb{E}[X] = \mathbb{E}[Y]$. Then,

$$\begin{aligned} E\big[(Y-X)^2\big] &= E\big[((Y-\mu)-(X-\mu))^2\big] \\ &\le 2(E\big[(Y-\mu)^2\big] + E\big[(X-\mu)^2\big]) \\ &= 2(\mathrm{Var}[Y] + \mathrm{Var}[X]) \\ &= (1+o(1))\tfrac{\pi^2}{n^2}, \end{aligned}$$

using the inequality $(x-y)^2 \le 2(x^2+y^2)$. Exactly the same bound applies to $E\big[(Z-X)^2\big]$. Putting everything together, we get that $\mathbb{P}[(*)] \le (1+o(1))\frac{\pi^2}{n^2}$. □

Theorem 4.3 $\mathbb{E}[|\mathcal{LSP}|] = \Theta(n^2)$

Proof: The number of LSPs of length 1 is exactly $n(n-1)$. (Every edge is an LSP of length 1.) By Lemma 4.1, the expected number of LSPs of length 2 is $O(n^3 \cdot \frac{\ln^2 n}{n^2}) = O(n \ln^2 n)$. By Lemma 4.2, the expected number of LSPs of length greater than two is $O(n^4 \cdot \frac{1}{n^2}) = O(n^2)$. □

Experiments that we have done seem to suggest that $\mathbb{E}[|\mathcal{LSP}|]$ is very close to $(\frac{\pi^2}{6}+1)n^2 \simeq 2.64n^2$.

The results of this section were stated and proved for directed graphs. It is easy to check, however, that our methods can be also used to provide an all pairs shortest paths algorithm with a quadratic running time for the complete *undirected* graphs on n vertices with uniform edge weights.

5 High probability bound on the number of locally shortest paths

Our goal in this section is to show that the number of LSPs is $O(n^2)$ asymptotically almost surely (a.a.s), i.e., that there exists a constant c such that $\mathbb{P}[|\mathcal{LSP}| < cn^2] \to 1$, as $n \to \infty$.

Let E^* be the set of edges that are shortest paths. Let Δ be the maximum outdegree in the subgraph $G^* = (V, E^*)$. (McGeoch [23] refers to $G^* = (V, E^*)$ as the *essential* subgraph.) We first show that $\Delta = O(\log n)$, with very high probability.

Lemma 5.1 *For every $c > 6$, we have $\mathbb{P}[\Delta > c \ln n] = O(n^{1-c/6})$.*

Proof: Let $G' = (V, E')$ be the subgraph of G composed of all edges of weight at most $\frac{c}{2}\frac{\ln n}{n}$, and let Δ' be the maximum outdegree in G'. The outdegree of each vertex in G' is binomially distributed with parameters n and $\frac{c}{2}\frac{\ln n}{n}$. A special case of Chernoff bound (see, e.g., [26], p. 64) states that if X is a binomial variable with $\mu = \mathbb{E}[X]$, then $\mathbb{P}[X \geq 2\mu] \leq \mathrm{e}^{-\mu/3}$. Thus, the probability that the degree of a given vertex exceeds $c \ln n$ is at most $n^{-c/6}$. Thus $\mathbb{P}[\Delta' > c \ln n] \leq n^{1-c/6}$. Now, $\Delta > \Delta'$ only if at least one distance in G is greater than $\frac{c}{2}\frac{\ln n}{n}$. By Lemma 3.4, the probability that this happens is at most $O(n^{3-c/2} \log^2 n)$. For $c > 6$ we have $1 - c/6 > 3 - c/2$. □

The following lemma is trivial and can also be found in Demetrescu and Italiano [7].

Lemma 5.2 *If all shortest paths are unique, then $|\mathcal{LSP}| \leq \Delta n^2$.*

Proof: Every LSP is obtained by appending an edge which is itself a shortest path to some shortest path. The number of shortest path in a graph is at most n^2 (assuming uniqueness) and each one of these shortest path can be extended by at most Δ edges. □

Note that Lemmas 5.1 and 5.2 imply that the number of LSPs is $O(n^2 \log n)$ with high probability. To improve this bound to $O(n^2)$ we need to work harder.

Definition 5.3 (β-short paths) *Let $\beta > 0$ be a (small) constant. We say that a shortest path π is* β-short *if and only if its length is at most $(1+\beta)\frac{\ln n}{n}$, and* β-long, *otherwise. Similarly, we say that an LSP π is* β-short *if both shortest paths $l[\pi]$ and $r[\pi]$ obtained by removing its first edge and last edge are short, and* β-long, *otherwise. Let $\mathcal{SP}^S$, $\mathcal{SP}^L$, $\mathcal{LSP}^S$, $\mathcal{LSP}^L$ be the sets of β-short and β-long shortest and locally shortest paths. (Note that these sets depend on the parameter β.)*

Clearly, $|\mathcal{LSP}| = |\mathcal{LSP}^L| + |\mathcal{LSP}^S|$. We estimate separately the number of β-long LSPs and the number of β-short LSPs. We begin by bounding the number of β-long shortest paths and locally shortest paths.

Lemma 5.4 *For every $\beta > 0$, we have $\mathbb{E}[|\mathcal{SP}^L|] = O(n^{2-\beta/12})$.*

Proof: By Lemma 3.6, with $\alpha = \beta/12$, we get that for any $a \neq b \in V$ we have

$$\mathbb{P}[D(a,b) \geq (1+\beta)\frac{\ln n}{n}] \;=\; O(n^{-\beta/12}) \;.$$

The lemma follows by the linearity of expectation. □

Lemma 5.5 *For every $\beta > 0$, we have $\mathbb{E}[|\mathcal{LSP}^L|] = O(n^{2-\beta/12}\ln n)$.*

Proof: Using the same argument used in the proof of Lemma 5.2, we get that $|\mathcal{LSP}^L| \leq \Delta|\mathcal{SP}^L|$. By Lemma 5.1 we get

$$\mathbb{E}[|\mathcal{LSP}^L|] \;\leq\; \mathbb{E}[\Delta|\mathcal{SP}^L|] \;\leq\; c\ln n \cdot \mathbb{E}[|\mathcal{SP}^L|] + n^{1-c/6}n^3 \;.$$

Letting $c = 12$ and using Lemma 5.4 we get that $\mathbb{E}[|\mathcal{LSP}^L|] = O(n^{2-\beta/12}\ln n)$, as required. □

Lemma 5.6 *For every $\beta > 0$ we have $\mathbb{P}[|\mathcal{LSP}^L| \geq n^2] = O(n^{-\beta/12}\ln n)$.*

Proof: Follows from Lemma 5.5 using Markov's inequality. □

We next show that $|\mathcal{LSP}^S| = O(n^2)$ with high probability. To do that we use the *Efron-Stein inequality* (see, e.g., Boucheron *et al.* [3]) to bound $\mathrm{Var}[|\mathcal{LSP}^S|]$.

Theorem 5.7 (Efron-Stein inequality) *Let $Z = f(X_1,\ldots,X_m)$, where $X_1, X_2, \ldots, X_m$ are independent random variables. For any $1 \leq i \leq m$, let X_i' be a random variable with the same distribution as X_i but independent from $X_1, X_2, \ldots, X_m$, and let $Z_i' = f(X_1,\ldots,X_i',\ldots,X_m)$. Then,*

$$\mathrm{Var}[Z] \;\leq\; \frac{1}{2}\sum_{i=1}^{m} E\left[\left(Z - Z_i'\right)^2\right].$$

In our case, we have $m = n(n-1)$, $X_1, X_2, \ldots, X_m$ are the random edge weights, and $Z = |\mathcal{LSP}^S|$. For every edge e, we need to compute the second moment of the random variable $|\mathcal{LSP}^S_{e,0}| - |\mathcal{LSP}^S_{e,1}|$, where $\mathcal{LSP}^S_{e,0}$ and $\mathcal{LSP}^S_{e,1}$ are the sets of β-short LSPs when all edges other than e are assigned the *same* random edge weights, while e is assigned two independent edge weights. Due to symmetry, the second moment of $|\mathcal{LSP}^S_{e,0}| - |\mathcal{LSP}^S_{e,1}|$ does not depend on e. For brevity, we write $\mathcal{LSP}^S_0$ and $\mathcal{LSP}^S_1$, instead of $\mathcal{LSP}^S_{e,0}$ and $\mathcal{LSP}^S_{e,1}$, when the e is clear from the context. We similarly define $\mathcal{SP}^S_0$ and $\mathcal{SP}^S_1$ to be the corresponding sets of β-short shortest paths.

If A and B are two sets, then $||A| - |B|| \leq |A \oplus B|$, where $A \oplus B = (A \setminus B) \cup (B \setminus A)$ is the *symmetric difference* of the two sets. We thus focus our attention on $\mathcal{LSP}^S_0 \oplus \mathcal{LSP}^S_1$. We begin by looking at $\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1$. Let $\mathcal{SP}^S_0(e)$ and $\mathcal{SP}^S_1(e)$ be the set of β-short shortest paths that *pass* through e with the two choices of the weight of e.

Lemma 5.8 *For every edge e we have $|\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1| \leq 2(|\mathcal{SP}^S_0(e)| + |\mathcal{SP}^S_1(e)|)$.*

Proof: Let $c_0(e)$ and $c_1(e)$ be the two costs of e. Suppose at first that $c_0(e) < c_1(e)$. A β-short shortest path that stops being β-short shortest path when the cost of e is increased from $c_0(e)$ to $c_1(e)$ must pass through e. Thus, $\mathcal{SP}^S_0 \setminus \mathcal{SP}^S_1 \subseteq \mathcal{SP}^S_0(e)$ and hence $|\mathcal{SP}^S_0 \setminus \mathcal{SP}^S_1| \leq |\mathcal{SP}^S_0(e)|$. The only paths in $\mathcal{SP}^S_1 \setminus \mathcal{SP}^S_0$ are paths that replace paths from $\mathcal{SP}^S_0 \setminus \mathcal{SP}^S_1$. Thus, we also have $|\mathcal{SP}^S_1 \setminus \mathcal{SP}^S_0| \leq |\mathcal{SP}^S_0(e)|$. Under the assumption $c_0(e) < c_1(e)$ we thus get $|\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1| \leq 2|\mathcal{SP}^S_0(e)|$. If $c_0(e) > c_1(e)$, we similarly get that $|\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1| \leq 2|\mathcal{SP}^S_1(e)|$. In both cases we have $|\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1| \leq 2(|\mathcal{SP}^S_0(e)| + |\mathcal{SP}^S_1(e)|)$. □

We next estimate $|\mathcal{SP}^S_0(e)|$ and $|\mathcal{SP}^S_1(e)|$. As they both have the same distribution, we omit the subscript.

Lemma 5.9 *For every $\beta > 0$, we have $\mathbb{P}[|\mathcal{SP}^S(e)| > 0] = O(\frac{\ln n}{n})$.*

Proof: The set $\mathcal{SP}^S(e)$ is non-empty only if e is a shortest path between its endpoints, which by Lemma 3.5 only happens with probability $\frac{\ln n}{n} + O(\frac{1}{n})$. $\square$

Our next goal is to show that $|\mathcal{SP}^S(e)| = O(n^{1+\beta'})$, with high probability, for any $\beta' > \beta$.

Lemma 5.10 *For every $\beta > 0$, and every $\beta' > \beta$, we have $\mathbb{P}[|\mathcal{SP}^S(e)| > n^{1+\beta'}] = O(n^{-c})$, for every $c > 0$.*

Proof: Let $e = a \to b$ be a fixed edge. Let C be the set of pairs (u, v) such that $u \rightsquigarrow a \to b \rightsquigarrow v$ is a shortest path of length at most $\frac{(1+\beta)\ln n}{n}$. Clearly $|\mathcal{SP}^S(e)| = |C|$. For a fixed integer r, and $1 \le i \le r$, let

$$A_i = \left\{ u \in V \;\middle|\; D(u,a) \le \frac{i(1+\beta)}{r}\frac{\ln n}{n} \right\} \quad , \quad B_i = \left\{ v \in V \;\middle|\; D(b,v) \le \frac{i(1+\beta)}{r}\frac{\ln n}{n} \right\}$$

be the sets of vertices of distances at most $\frac{i(1+\beta)}{r}\frac{\ln n}{n}$ to a and from b, respectively. Note that $B_i = Ball\left(b, \frac{i(1+\beta)}{r}\frac{\ln n}{n}\right)$, while $A_i = Ball\left(a, \frac{i(1+\beta)}{r}\frac{\ln n}{n}\right)$, in the graph in which all edge directions are reversed. Clearly

$$C \subseteq \bigcup_{i=1}^{r} A_i \times B_{r+1-i} \,.$$

By Corollary 3.9, we have $|A_i|, |B_i| \le n^{(1+\beta)i/r+\epsilon}$, for every i, with a probability of at least $1 - O(rn^{-c})$, for every $c > 0$. It thus follows that $|C| \le rn^{(1+\beta)(1+1/r)+2\epsilon}$, again with this very high probability. Letting r sufficiently large and ϵ sufficiently small, we get the claim of the lemma. $\square$

Lemmas 5.9 and 5.10 allow us to bound $\mathbb{E}[|\mathcal{SP}^S(e)|^2]$.

Lemma 5.11 *For every $\beta > 0$, and every $\beta' > \beta$ we have $\mathbb{E}[|\mathcal{SP}^S(e)|^2] = O(n^{2(1+\beta')-1}\ln n)$.*

Proof: For succinctness, let $X = |\mathcal{SP}^S(e)|$ and $a = n^{1+\beta'}$. We always have $X^2 \le n^4$. Using Lemma 5.10 with $c = 4$, we have

$$\mathbb{E}[X^2] \;\le\; \mathbb{P}[0 < X \le a] \cdot a^2 + \mathbb{P}[x > a] \cdot n^4 \;=\; O\left(\frac{\ln n}{n} \cdot n^{2(1+\beta')} + n^{-4} \cdot n^4\right) \;=\; O\left(n^{2(1+\beta')-1} \ln n\right).$$

$\square$

We can finally get back to estimating $\mathcal{LSP}_0^S \oplus \mathcal{LSP}_1^S$. Let Δ_0 and Δ_1 be the maximum outdegrees in the essential graph, i.e., the subgraph composed of the edges that are shortest paths, under the two independent choices of the weight of e. Let $\Delta = \max\{\Delta_0, \Delta_1\}$. By Lemma 5.1 we have $\mathbb{P}[\Delta > c\ln n] = O(n^{1-c/6})$, for every $c > 6$.

Lemma 5.12 *For every $\beta > 0$ we have $|\mathcal{LSP}_0^S \oplus \mathcal{LSP}_1^S| \le 2\Delta \cdot |\mathcal{SP}_0^S \oplus \mathcal{SP}_1^S|$.*

Proof: Suppose that $\pi \in \mathcal{LSP}_0^S \setminus \mathcal{LSP}_1^S$. Then either $l[\pi] \in \mathcal{SP}_0^S \setminus \mathcal{SP}_1^S$ or $r[\pi] \in \mathcal{SP}_0^S \setminus \mathcal{SP}_1^S$. Each shortest path in $\mathcal{SP}_0^S$ has at most Δ_0 pre-extensions and at most Δ_0 post-extensions that are locally shortest paths. Thus, $|\mathcal{LSP}_0^S \setminus \mathcal{LSP}_1^S| \le 2\Delta_0 |\mathcal{SP}_0^S \setminus \mathcal{SP}_1^S|$. Similarly, $|\mathcal{LSP}_1^S \setminus \mathcal{LSP}_0^S| \le 2\Delta_1 |\mathcal{SP}_1^S \setminus \mathcal{SP}_0^S|$, and the lemma follows. $\square$

Lemma 5.13 *For every $\beta > 0$, every $\beta' > \beta$, and every edge e we have*

$$\mathbb{E}\left[\left||\mathcal{LSP}_1^S| - |\mathcal{LSP}_0^S|\right|^2\right] = O\left(n^{2(1+\beta')-1}\ln^3 n\right).$$

Proof: By Lemmas 5.8 and 5.12 we have

$$\begin{aligned}\left||\mathcal{LSP}_1^S| - |\mathcal{LSP}_0^S|\right|^2 \ &\le\ \left|\mathcal{LSP}_0^S \oplus \mathcal{LSP}_1^S\right|^2 \ \le\ 4\Delta^2\left|\mathcal{SP}_0^S \oplus \mathcal{SP}_1^S\right|^2 \\ &\le\ 16\Delta^2\big(|\mathcal{SP}_0^S(e)| + |\mathcal{SP}_1^S(e)|\big)^2 \ \le\ 32\Delta^2(|\mathcal{SP}_0^S(e)|^2 + |\mathcal{SP}_1^S(e)|^2).\end{aligned}$$

Since both $|\mathcal{SP}_0^S(e)|, |\mathcal{SP}_1^S(e)| \le n^2$, from Lemma 5.1 with $c = 24$ we have

$$\begin{aligned}\mathbb{E}\left[\left||\mathcal{LSP}_1^S| - |\mathcal{LSP}_0^S|\right|^2\right] \ &\le\ O\left(\ln^2 n \cdot \mathbb{E}[|\mathcal{SP}^S(e)|^2] + \mathbb{P}[\Delta > 24\ln n]\cdot n^4\right) \\ &=\ O\left(\ln^2 n \cdot \mathbb{E}[|\mathcal{SP}^S(e)|^2] + n^{-3}\cdot n^4\right).\end{aligned}$$

The claim now follows from Lemma 5.11. □

Using the Efron-Stein inequality (Theorem 5.7) we thus get:

Lemma 5.14 *For every $\beta > 0$ and every $\beta' > \beta$ we have* $\mathrm{Var}\left[|\mathcal{LSP}^S|\right] = O\left(n^{2(1+\beta')+1}\ln^3 n\right)$.

Theorem 5.15 *There is a constant c such that* $\mathbb{P}[|\mathcal{LSP}| \ge cn^2] = O(n^{-1/26})$.

Proof: Let $\beta = \frac{12}{25}$. By Lemma 5.6 we get that

$$\mathbb{P}[|\mathcal{LSP}^L| \ge n^2] \ =\ O(n^{-\beta/12}\ln n) \ =\ O(n^{-1/25}\ln n)\,. \tag{7}$$

Let $\beta' = \beta + \epsilon$, where $\epsilon > 0$ is tiny. By Lemma 5.14 we get that

$$\mathrm{Var}\left[|\mathcal{LSP}^S|\right] \ =\ O\left(n^{2(1+\beta')+1}\ln^3 n\right) \ =\ O(n^{99/25+2\epsilon}\ln^3 n) \ =\ O(n^{99/25+3\epsilon}).$$

By Theorem 4.3 we have $\mathbb{E}[|\mathcal{LSP}|] = \Theta(n^2)$. By Lemma 5.5, we have $\mathbb{E}[|\mathcal{LSP}^L| = o(n^2)$. As $\mathbb{E}[|\mathcal{LSP}|] = \mathbb{E}[|\mathcal{LSP}^S|] + \mathbb{E}[|\mathcal{LSP}^L|]$, we get that $\mathbb{E}[|\mathcal{LSP}^S|] = \Theta(n^2)$.
By Chebyshev's inequality (see, e.g., [26]), for every random variable X we have

$$\mathbb{P}[X \ge 2\mathbb{E}[X]] \ \le\ \mathbb{P}[|X - \mathbb{E}[X]| \ge \mathbb{E}[X]] \ \le\ \frac{\mathrm{Var}[X]}{\mathbb{E}[X]^2}\,.$$

For $X = |\mathcal{LSP}^S|$, and using the facts that $\mathbb{E}\left[|\mathcal{LSP}^S|\right] = \Theta(n^2)$ and $\mathrm{Var}[|\mathcal{LSP}^S|] = O(n^{99/25+3\epsilon})$, we thus get

$$\mathbb{P}\left[|\mathcal{LSP}^S| \ge 2\mathbb{E}[|\mathcal{LSP}^S|]\right] \ \le\ \frac{\mathrm{Var}\left[|\mathcal{LSP}^S|\right]}{\mathbb{E}\left[|\mathcal{LSP}^S|\right]^2} \ =\ O(n^{-1/25+3\epsilon})\,. \tag{8}$$

As $|\mathcal{LSP}| = |\mathcal{LSP}^S| + |\mathcal{LSP}^L|$, combining (7) and (8) and choosing ϵ small enough, we get the claim of the Theorem. □

We believe that for every $a > 0$ there exists c such that $\mathbb{P}\left[|\mathcal{LSP}| \ge cn^2\right] = O(n^{-a})$. Proving, or disproving, this claim would require new techniques.

6 An $O(n^2)$-time implementation

In this section we describe an implementation of the algorithm of Section 2.1 (and Appendix A) that runs in $O(n^2)$ time in expectation and with high probability. This is done using a simple observation of Dinic [12] and a simple bucket-based priority queue implementation that goes back to Dial [10].

Let $\delta = \min_{(u,v)\in E} c(u,v)$ be the minimal edge weight in the graph. We claim that algorithm `apsp` of Section 2.1 remains correct if instead of requiring that the pair (u,v) extracted from the heap Q is a pair with minimal $dist(u,v)$, we only require that $dist(u,v) < dist(u',v') + \delta$ for every other pair (u',v') in Q. The proof is a simple modification of the proof of Theorem 2.2 given in Appendix A. This observation, in the context of Dijkstra's algorithm, dates back to Dinic [12]. Along with many more ideas, this observation forms the basis for the linear *worst-case* time single-source shortest paths algorithm for undirected graphs obtained by Thorup [32]. It is also used by Hagerup [18] to obtain a simple linear *expected* time algorithm for single source shortest paths, simplifying results of Meyer [25] and Goldberg [17].

In our setting, edge weights are drawn independently and uniformly at random from $[0,1]$. The probability that the minimal edge weight is smaller than $n^{-2.5}$ is clearly at most $n^{-0.5}$. If this unlikely event happens, we simply use an $O(n^2 \log n)$ time implementation based on Fibonacci heaps. This only contributes $o(n^2)$ to the expected running time of the algorithm.

We assume now that $\delta \geq n^{-2.5}$. For every $u,v \in V$, we let $dist'(u,v) = \lfloor dist(u,v)/\delta \rfloor$ and use $dist'(u,v)$, instead of $dist(u,v)$, as the key of (u,v) in Q.

We implement the heap Q as follows. (There are many possible variants. We describe the one that seems to be the most natural.) We use $L = n^2$ buckets $B_1, B_2, \ldots, B_L$. Bucket B_i, for $i < L$, is a linked list holding pairs (u,v) for which $dist'(u,v) = i$. Bucket B_L is a special *leftover* bucket that holds all pairs (u,v) for which $dist'(u,v) \geq L$. It is again implemented as a linked list. We also maintain the index k of the bucket from which the last minimal pair was extracted.

The implementation of a `heap-insert` operation is trivial. To insert a pair (u,v) into Q, we simply add (u,v) to B_i, where $i = \min\{dist'(u,v), L\}$.

A `decrease-key` operation is also simple. We simply remove (u,v) from its current bucket and move it to the appropriate bucket. (Each pair has a pointer to its position in its current bucket, so these operations take constant time.)

An `extract-min` operation is implemented as follows. We sequentially scan the buckets, starting from B_k, until we find the first non-empty bucket. If the index of this bucket is less than L, we return an arbitrary element from this bucket and update k if necessary. If the first non-empty bucket is B_L, the leftover bucket, we insert all the elements currently in B_L into a comparison-based heap and use it to process all subsequent heap operations. (We show below that in our setting, we would very rarely encounter this case.)

This implementation of the `extract-min` operation is correct as the priority queue that we need to maintain is *monotone*, in the sense that the minimal key contained in the priority queue never decreases. This follows immediately from then fact that keys of new pairs inserted into Q, or decreased keys of existing pairs in Q are always larger than the key of the last extracted pair.

The total time spent on implementing all heap operations, until all buckets $B_1, \ldots, B_{L-1}$ are empty, is clearly $O(N + L)$, where N is the number of heap operations performed. By Theorem 2.2 we have $N = O(|\mathcal{LSP}| + n^2)$. By Theorem 4.3 we have $\mathbb{E}[|\mathcal{LSP}|] = O(n^2)$. By Theorem 5.15, there is constant c such that $\mathbb{P}[|\mathcal{LSP}| \geq cn^2] = O(n^{-1/60})$. As $L = n^2$, the number of operations here is $O(n^2)$, both in expectation and with high probability.

All that remains, therefore, is to show that the probability that B_L will be the only non-empty bucket is tiny. Note that this happens if and only if there is a pair $u,v \in V$ for which $D(u,v) \geq L\delta \geq n^{-0.5}$. By Lemma 3.4, this probability is $O(n^{-c})$ for every $c > 0$. If this extremely unlikely event happens, the running time is only increased to $O(n^2 \log n)$, which has a negligible effect on the expected running time of the whole algorithm. We have thus obtained:

Theorem 6.1 *The expected running time of algorithm* `apsp`*, when implemented using a bucket-based priority queue, and when run on a complete directed graph with edge weights selected uniformly at random from* $[0,1]$ *is* $O(n^2)$. *Furthermore, there is a constant* $c > 0$ *such that the probability that the running time of the algorithm exceeds* cn^2 *is* $O(n^{-1/60})$.

7 Polylogarithmic update times

In this section we consider the expected time needed to update all shortest paths following a *random edge update*, i.e., an update operation that chooses a random edge e of the complete directed graph, uniformly at random, and assigns it a new random weight, independent of all previous weights chosen, drawn uniformly at random from $[0,1]$.

Recall that $\mathcal{SP}^-$ and $\mathcal{LSP}^-$ are the sets of shortest and locally shortest paths destroyed by an update operation, and that $\mathcal{SP}^+$ and $\mathcal{LSP}^+$ are the sets of shortest and locally shortest paths that are created (or recreated) by an update operation. More specifically, we have

$$\begin{aligned}
\mathcal{SP}^- &= \mathcal{SP}_0(e) \cup (\mathcal{SP}_0 \setminus \mathcal{SP}_1)\ , \\
\mathcal{SP}^+ &= \mathcal{SP}_1(e) \cup (\mathcal{SP}_1 \setminus \mathcal{SP}_0)\ , \\
\mathcal{LSP}^- &= \mathcal{LSP}_0(e) \cup (\mathcal{LSP}_0 \setminus \mathcal{LSP}_1)\ , \\
\mathcal{LSP}^+ &= \mathcal{LSP}_1(e) \cup (\mathcal{LSP}_1 \setminus \mathcal{LSP}_0)\ ,
\end{aligned}$$

where, as in Section 5, $\mathcal{SP}_0$ and $\mathcal{SP}_1$ are the sets of shortest paths before and after the update of e, and $\mathcal{SP}_0(e)$ and $\mathcal{SP}_1(e)$ are the sets of shortest paths, before and after the update, that pass through e. The sets $\mathcal{LSP}_0$, $\mathcal{LSP}_1$, $\mathcal{LSP}_0(e)$ and $\mathcal{LSP}_1(e)$, are the corresponding sets of locally shortest paths.

Our main goal is to bound the expected sizes of the sets $\mathcal{SP}^-$, $\mathcal{SP}^+$, $\mathcal{LSP}^-$ and $\mathcal{LSP}^+$. This, in conjunction with Theorem 2.3, would supply an upper bound on the expected update time. By symmetry, it is easy to see that $\mathbb{E}[|\mathcal{SP}^-|] = \mathbb{E}[|\mathcal{SP}^+|]$ and $\mathbb{E}[|\mathcal{LSP}^-|] = \mathbb{E}[|\mathcal{LSP}^+|]$. We can thus concentrate on estimating $\mathbb{E}[|\mathcal{SP}^-|]$ and $\mathbb{E}[|\mathcal{LSP}^-|]$.

Let e be the random edge updated by a random edge update operation. For every $u, v \in V$, let $\pi_0[u,v]$ and $\pi_1[u,v]$ be the shortest path from u to v before and after the update. Let $B_i = \{(u,v) \mid e \in \pi_i[u,v]\}$, for $i \in \{0,1\}$, be the set of pairs of vertices connected, before and after the update, by a shortest path passing through e. (Note that $|B_i| = |\mathcal{SP}_i(e)|$, for $i \in \{0,1\}$.) It is easy to see that $\pi_0[u,v] \in \mathcal{SP}^-$ if and only if $e \in \pi_0[u,v]$ or $e \in \pi_1[u,v]$. Thus, $\mathcal{SP}^- = \{\pi_0[u,v] \mid (u,v) \in B_0 \cup B_1\}$ and similarly $\mathcal{SP}^+ = \{\pi_1[u,v] \mid (u,v) \in B_0 \cup B_1\}$. In particular $|\mathcal{SP}^-| = |\mathcal{SP}^+|$. More importantly,

$$|\mathcal{SP}^-| \ \le\ |\mathcal{SP}_0(e)| + |\mathcal{SP}_1(e)|\ .$$

To bound $\mathbb{E}[|\mathcal{SP}^-|] = \mathbb{E}[|\mathcal{SP}^+|]$ it is thus enough to bound $\mathbb{E}[|\mathcal{SP}_0(e)|] = \mathbb{E}[|\mathcal{SP}_1(e)|]$.

Lemma 7.1 *The expected number of edges on a shortest path between two random vertices is* $(1+o(1))\ln n$.

Proof: When edge weights are exponential, the expected number of edges on a shortest path between two random vertices is exactly equal to the average depth of a vertex in a random recursive tree of size n. (See, e.g., Janson [20].) It is known that this average depth is $(1+o(1))\ln n$ (Moon [28]). The same asymptotic result holds also under the uniform distribution. (See Section 2 of Janson [20].) □

Lemma 7.2 *The expected number of shortest paths that pass through a random edge* e *is* $(1+o(1))\ln n$.

Proof: For every $u, v \in V$, let $\pi[u, v]$ be the shortest path from u to v, and let $|\pi[u, v]|$ be the number of edges on it. For every edge e of the complete graph, let $\mathcal{SP}(e)$ be the set of shortest paths that pass through e. By symmetry we have

$$\mathbb{E}[|\mathcal{SP}(e)|] = E\left[\frac{1}{n(n-1)} \sum_{e'} |\mathcal{SP}(e')|\right] = E\left[\frac{1}{n(n-1)} \sum_{u \neq v} |\pi[u, v]|\right] = \mathbb{E}[|\pi[u, v]|].$$

By Lemma 7.1, we get that $\mathbb{E}[|\mathcal{SP}(e)|] = (1 + o(1)) \ln n$. □

Theorem 7.3 *Following a random edge update, we have* $\mathbb{E}[|\mathcal{SP}^-|] = \mathbb{E}[|\mathcal{SP}^+|] \leq (2 + o(1)) \ln n$.

Let Δ be the maximal degree of the essential graph $G^* = (V, E^*)$ defined in the previous section. Lemma 5.1 says that with high probability $\Delta = O(\log n)$.

Theorem 7.4 *Following a random edge update we have* $\mathbb{E}[|\mathcal{LSP}^-|] = \mathbb{E}[|\mathcal{LSP}^+|] = O(\log^2 n)$.

Proof: Clearly $\pi \in \mathcal{LSP}^-$ if and only if $l[\pi] \in \mathcal{SP}^-$ or $r[\pi] \in \mathcal{SP}^-$. Each shortest path has at most 2Δ LSP extensions. Thus $|\mathcal{LSP}^-| \leq 2\Delta \cdot |\mathcal{SP}^-|$. By Lemma 5.1, we have $\mathbb{P}[\Delta > 24 \ln n] = O(n^{-3})$. As $|\mathcal{LSP}|$ is always at most n^3, we get $\mathbb{E}[|\mathcal{LSP}^-|] \leq 48 \ln n \cdot \mathbb{E}[|\mathcal{SP}^-|] + n^{-3} \cdot n^3 = O(\log^2 n)$. □

We believe that the $O(\log^2 n)$ bound in Theorem 7.4 can be improved, possibly to $O(\log n)$, and leave it as an open problem.

Theorem 7.5 *The expected running time of a random edge update, when a Fibonacci heap is used to implement the global heap, and simple linked lists are used to implement the local heaps, is* $O(\log^2 n)$.

8 Concluding remarks

We presented an algorithm that solves the APSP problem on complete directed graphs with random edges weights in $O(n^2)$ time with high probability. The expected running time of the algorithm is also $O(n^2)$. This solves an open problem of Frieze and McDiarmid [15].

We also presented a dynamic algorithm that performs random edge updates in $O(\log^2 n)$ expected time. It is an interesting open problem whether this can be improved to $O(\log n)$.

Our results also hold in the directed $G(n, p)$ model in which each edge is selected with probability p, where $p \gg (\ln n)/n$. Selected edges are again assigned independent, uniformly distributed, weights. Similarly, it is easy to see that our results apply when edge weights are *integers* chosen uniformly at random from, say, $\{1, 2, \ldots, n\}$, where n is the number of vertices.

Acknowledgment

The last author would like to thank Camil Demetrescu, Giuseppe Italiano and Mikkel Thorup for many illuminating discussions.

References

[1] L. Addario-Berry, N. Broutin, and G. Lugosi. The longest minimum-weight path in a complete graph. *Combinatorics, Probability & Computing*, 19(1):1–19, 2010.

[2] P.A. Bloniarz. A shortest-path algorithm with expected time $O(n^2 \log n \log^* n)$. *SIAM Journal on Computing*, 12(3):588–600, 1983.

[3] S. Boucheron, G. Lugosi, and O. Bousquet. Concentration inequalities. In *Advanced Lectures on Machine Learning*, pages 208–240, 2003.

[4] C. Cooper, A. Frieze, K. Mehlhorn, and V. Priebe. Average-case complexity of shortest-paths problems in the vertex-potential model. *Random Structures and Algorithms*, 16(1):33–46, 2000.

[5] R. Davis and A. Prieditis. The expected length of a shortest path. *Information Processing Letters*, 46(3):135–141, 1993.

[6] C. Demetrescu, P. Faruolo, G.F. Italiano, and M. Thorup. Does path cleaning help in dynamic all-pairs shortest paths? In *Proc. of 14th ESA*, pages 732–743, 2006.

[7] C. Demetrescu and G.F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004.

[8] C. Demetrescu and G.F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Transactions on Algorithms*, 2(4):578–601, 2006.

[9] L. Devroye. Branching processes in the analysis of the heights of trees. *Acta Informatica*, 24(3):277–298, 1987.

[10] R.B. Dial. Algorithm 360: shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

[11] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[12] E.A. Dinic. Economical algorithms for finding shortest paths in a network. In Y. Popkov and B. Shmulyian, editors, *Transportation Modeling Systems*, pages 36–?44. Institute for System Studies, Moscow, CIS, 1978.

[13] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[14] T. Friedrich and N. Hebbinghaus. Average update times for fully-dynamic all-pairs shortest paths. In *Proc. of 19th ISAAC*, pages 692–703, 2008.

[15] A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures and Algorithms*, 10(1-2):5–42, 1997.

[16] A.M. Frieze and G.R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[17] A.V. Goldberg. A practical shortest path algorithm with linear expected time. *SIAM Journal on Computing*, 37(5):1637–1655, 2008.

[18] T. Hagerup. Simpler computation of single-source shortest paths in linear average time. *Theory of Computing Systems*, 39(1):113–120, 2006.

[19] R. Hassin and E. Zemel. On shortest paths in graphs with random weights. *Mathematics of Operations Research*, 10(4):557–564, 1985.

[20] S. Janson. One, two and three times $\log n/n$ for paths in a complete graph with random weights. *Combinatorics, Probability & Computing*, 8(4):347–361, 1999.

[21] D.R. Karger, D. Koller, and S.J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22:1199–1217, 1993.

[22] A. Maurer. A bound on the deviation probability for sums of non-negative random variables. *JIPAM. Journal of Inequalities in Pure and Applied Mathematics*, 4(1):Article 15, 6 pp. (electronic), 2003.

[23] C.C. McGeoch. All-pairs shortest paths and the essential subgraph. *Algorithmica*, 13:426–441, 1995.

[24] K. Mehlhorn and V. Priebe. On the all-pairs shortest-path algorithm of Moffat and Takaoka. *Random Structures and Algorithms*, 10(1-2):205–220, 1997.

[25] U. Meyer. Average-case complexity of single-source shortest-paths algorithms: lower and upper bounds. *Journal of Algorithms*, 48(1):91–134, 2003.

[26] M. Mitzenmacher and E. Upfal. *Probability and computing, Randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

[27] A. Moffat and T. Takaoka. An all pairs shortest path algorithm with expected time $O(n^2 \log n)$. *SIAM Journal on Computing*, 16(6):1023–1031, 1987.

[28] J. W. Moon. The distance between nodes in recursive trees. In *Combinatorics (Proc. British Combinatorial Conf., Univ. Coll. Wales, Aberystwyth, 1973)*, pages 125–132. London Math. Soc. Lecture Note Ser., No. 13. Cambridge Univ. Press, London, 1974.

[29] S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.

[30] R.T. Smythe and H.M. Mahmoud. A survey of recursive trees. *Theory of Probability and Mathematical Statistics*, (51):1–29, 1995.

[31] P.M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$. *SIAM Journal on Computing*, 2(1):28–32, 1973.

[32] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.

[33] M. Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proc. of 9th SWAT*, pages 384–396, 2004.

[34] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of 37th STOC*, pages 112–119, 2005.

A The static algorithm – complete description

In this section we give a full description, and a correctness proof, of the static version of the Demetrescu and Italiano [7, 8] used in this paper. Pseudo-code of the algorithm, called `apsp`, is given in Figure 1. The input to the algorithm is a weighted directed graph $G = (V, E, c)$, where $c : E \to (0, \infty)$ assigns positive weights (or costs) to the edges of the graph. The algorithm in Figure 1 works correctly only under the assumption that all shortest paths are *unique*. Under essentially all probabilistic models considered in this paper, this assumption holds with probability 1. Algorithm `apsp` is also interesting, however, in non-probabilistic settings. For a simple way of dispensing with the uniqueness assumption, without increasing the running time of the algorithm by more than a constant factor, see Demetrescu and Italiano [7].

We next prove Theorem 2.2 of Section 2, which we repeat for the convenience of the reader.

Theorem 2.2 *If all edge weights are positive and all shortest paths are unique, then algorithm* `apsp` *correctly finds all the shortest paths in the graph. Algorithm* `apsp` *runs in* $O(n^2 \cdot (T_{ins}(n^2) + T_{ext}(n^2)) + |\mathcal{LSP}| \cdot T_{dec}(n^2))$ *time, where* $|\mathcal{LSP}|$ *is the number of LSPs in the graph, and uses only* $O(n^2)$ *space.*

Proof: It is easy to check that each stage during the operation of the algorithm, $dist[u,v]$ corresponds to some path from u to v in the graph and that this path, or an even shorter path, can be traced using the p and q fields. Thus, the distances returned by the algorithm can never be too small.

It is also easy to check that the keys of the pairs (u,v) extracted from Q form a non-decreasing sequence and that a pair (u,v) removed from Q is never inserted to Q again. Thus, the algorithm always terminates.

Assume, for the sake of contradiction, that the algorithm fails to find a shortest path $u \rightsquigarrow v$, for some $u,v \in V$. Let $u \rightsquigarrow v$ be a shortest shortest-path not found by the algorithm. (In other words, if $u' \rightsquigarrow v'$ is shorter than $u \rightsquigarrow v$, then $u' \rightsquigarrow v'$ is found by the algorithm.)

If $u \rightsquigarrow v$ is simply the edge $u \to v$, then we immediately get a contradiction, as the algorithm starts by setting $dist[u,v]$ to $c(u,v)$ (and $p[u,v]$ to v, and $q[u,v]$ to u), for every $(u,v) \in E$. Thus, the algorithm does find the shortest path $u \rightsquigarrow v = u \to v$, a contradiction.

Assume, therefore, that $u \rightsquigarrow v = u \to u' \rightsquigarrow v' \to v$ is composed of at least two edges. (If it is composed of exactly two edges, then $u' = v'$.) Clearly $u \to u' \rightsquigarrow v'$ and $u' \rightsquigarrow v' \to v$ are also shortest paths and their

```
Function apsp(G = (V, E, c))

init(G)
Q ← heap()

foreach (u, v) ∈ E do
    dist[u, v] ← c(u, v)
    p[u, v] ← v
    q[u, v] ← u
    heap-insert(Q, (u, v), dist[u, v])

while Q ≠ ∅ do
    (u, v) ← extract-min(Q)
    insert(L[p[u, v], v], u)
    insert(R[u, q[u, v]], v)
    foreach w ∈ L[u, q[u, v]] do
        examine(w, u, v)
    foreach w ∈ R[p[u, v], v] do
        examine(u, v, w)
```

```
Function init(G = (V, E, c))

foreach u, v ∈ V do
    dist[u, v] ← ∞
    p[u, v] ← null
    q[u, v] ← null
    L[u, v] ← ∅
    R[u, v] ← ∅
foreach u ∈ V do
    dist[u, u] ← 0
```

```
Function examine(u, v, w)

if dist[u, v] + dist[v, w] < dist[u, w] then
    dist[u, w] ← dist[u, v] + dist[v, w]
    if p[u, w] = null then
        heap-insert(Q, (u, w), dist[u, w])
    else
        decrease-key(Q, (u, w), dist[u, w])
    p[u, w] ← p[u, v]
    q[u, w] ← q[v, w]
```

Figure 1: A static version of the APSP algorithm of Demetrescu and Italiano [7, 8].

```
Function path(v)
    $\pi \leftarrow$ new-path()
    $l[\pi] \leftarrow null$
    $r[\pi] \leftarrow null$
    $start[\pi] \leftarrow v$
    $end[\pi] \leftarrow v$
    $first[\pi] \leftarrow null$
    $last[\pi] \leftarrow null$
    $cost[\pi] \leftarrow 0$
    $sp[\pi] \leftarrow$ true
    $L[\pi], R[\pi] \leftarrow \emptyset$
    $SL[\pi], SR[\pi] \leftarrow \emptyset$
    return $\pi$
```

```
Function path($e = (u, v)$)
    $\pi \leftarrow$ new-path()
    $l[\pi] \leftarrow p[u]$
    $r[\pi] \leftarrow p[v]$
    $start[\pi] \leftarrow u$
    $end[\pi] \leftarrow v$
    $first[\pi] \leftarrow e$
    $last[\pi] \leftarrow e$
    $cost[\pi] \leftarrow c[e]$
    $sp[\pi] \leftarrow$ false
    $L[\pi], R[\pi] \leftarrow \emptyset$
    $SL[\pi], SR[\pi] \leftarrow \emptyset$
    insert($L[p[v]], \pi$)
    insert($R[p[u]], \pi$)
    return $\pi$
```

```
Function path($\pi_1, \pi_2$)
    if $r[\pi_1] \neq l[\pi_2]$ then error
    $\pi \leftarrow$ new-path()
    $l[\pi] \leftarrow \pi_1$
    $r[\pi] \leftarrow \pi_2$
    $start[\pi] \leftarrow start[\pi_1]$
    $end[\pi] \leftarrow end[\pi_2]$
    $first[\pi] \leftarrow first[\pi_1]$
    $last[\pi] \leftarrow last[\pi_2]$
    $cost[\pi] \leftarrow c[first[\pi]] + cost[\pi_2]$
    $sp[\pi] \leftarrow$ false
    $L[\pi], R[\pi] \leftarrow \emptyset$
    $SL[\pi], SR[\pi] \leftarrow \emptyset$
    insert($L[\pi_2], \pi$)
    insert($R[\pi_1], \pi$)
    return $\pi$
```

Figure 2: Generating new paths and inserting it into the path system.

length is strictly smaller than the length of $u \rightsquigarrow v$, as $c(u, u'), c(v', v) > 0$. Thus, by the our assumptions, $u \to u' \rightsquigarrow v'$ and $u' \rightsquigarrow v' \to v$ are discovered by the algorithm. When the second of these is discovered, the algorithm examines the path $u \rightsquigarrow v = u \to u' \rightsquigarrow v' \to v$ and sets $dist[u, v]$ to its length. Also (u, v) is added to Q if it is not already there. As there is no shorter path from u to v in the graph, the values of $dist[u, v]$, $p[u, v]$ and $q[u, v]$ would never be changed again, contradicting the assumption that the algorithm does not find the shortest path from u to v.

We next analyze the running time of algorithm. Each pair (u, v) is inserted and extracted from the priority queue Q at most once. The total cost of these operations is $O(n^2(T_{ins}(n^2)+T_{ext}(n^2)))$. All paths considered by the algorithm are LSPs. The algorithm examines each LSP exactly once. For each LSP it performs a constant number of operations followed perhaps by a `decrease-key` operation. The total cost of all these operations is $O(|\mathcal{LSP}|\, T_{dec}(n^2))$. The complexity of all other operations is negligible.

Finally, to see that the algorithm uses only $O(n^2)$ space, note that the removal of a pair (u, v) from the heap Q causes the insertion of only two elements to lists $L[u', v']$ and $R[u', v']$. As each pair (u, v) is extracted at most once, the total size of all these lists is $O(n^2)$. □

B The dynamic algorithm – complete description

As explained, one of the main differences between the static and dynamic algorithms is that the dynamic algorithm explicitly maintains all LSPs in a *path system*, and does not just examine them. Paths are created by the three constructors `path`(v), `path`(e) and `path`(π_1, π_2) given in Figure 2. `path`(v) generates a path of length 0 containing the vertex v. `path`(e) generates a path composed of the edge e. `path`(π_1, π_2) takes two paths π_1 and π_2 such that $r[\pi_1] = l[\pi_2]$ and constructs a path π such that $l[\pi] = \pi_1$ and $r[\pi] = \pi_2$. The new path π is composed of the first edge of π_1 followed by π_2, or equivalently, by π_1 followed by the last edge of π_2.

Every path π has the following fields:

$l[\pi]$ - A pointer to the path obtained by removing the last edge of π.

$r[\pi]$ - A pointer to the path obtained by removing the first edge of π.

$start[\pi]$ - The first vertex on π.

$end[\pi]$ - The last vertex on π.

$first[\pi]$ - The first edge on π.

$last[\pi]$ - The last edge on π.

$cost[\pi]$ - The total cost (weighted length) of π.

$sp[\pi]$ - `true` if and only if π is known to be a shortest path.

$L[\pi]$ - List of left LSP extensions of π.

$R[\pi]$ - List of right LSP extensions of π.

$SL[\pi]$ - List of left shortest path extensions of π.

$SR[\pi]$ - List of right shortest path extensions of π.

The lists $SL[\pi]$ and $SR[\pi]$ are similar to the lists $L[u, v]$ and $R[u, v]$ used by the static algorithm. This time, however, they contain actual paths and not vertices. The lists $L[\pi]$ and $R[\pi]$ contain all LSPs, already constructed, obtained by extending π by one edge at its beginning or end, respectively.

The initialization function of the dynamic version, called `dapsp-init`, is given in Figure 3. It is similar to the static `apsp` algorithm. It too uses a global heap Q that stores pairs of vertices for which shortest paths are sought. For every $v \in V$, we let $p[v]$ be the empty path consisting of v. For every edge $e \in E$, we let $p[e]$ denote the path consisting of e. For every two vertices $u, v \in V$, the dynamic algorithm maintains the following information:

$\pi[u, v]$ - The shortest path from u to v found so far.

$cost[u, v]$ - The cost of the shortest path from u to v found so far.

$P[u, v]$ - a heap containing all the LSPs from u to v found so far.

We refer to $P[u, v]$ as the *local heap* corresponding to the pair (u, v). We refer to Q as the *global heap.*

The initialization function `dapsp-init` starts with some obvious initializations. (For every $u, v \in V$, it sets $\pi[u, v]$ to *null*, sets $dist[u, v]$ to ∞, sets $P[u, v]$ to an empty heap, etc.) For every $e \in E$ it then creates the path $p[e]$, by calling `path`(e), and then *examines* it by calling `examine`$(p[e])$, given in Figure 4.

The function `examine`(π) receives a newly created LSP connecting two vertices $u = start[\pi]$ and $v = end[\pi]$. It starts by inserting it into the heap $P[u, v]$ with key $cost[\pi]$. It then checks whether π is the first available LSP from u to v, or whether it is shorter than all existing LSPs between u and v. If π is shorter than $\pi[u, v]$, the shortest available path from u to v, then $\pi[u, v]$ is clearly not a shortest path. The algorithm thus sets $sp[\pi[u, v]]$ to `false`. It then removes all extensions of $\pi[u, v]$ from the system, if there are any. This is done by a call to `remove-exts`$(\pi[u, v],$ `false`$)$ which we discuss later. Finally, if π is currently the shortest available path from u to v, `examine` updates $\pi[u, v]$ and $dist[u, v]$ accordingly. It also inserts (u, v) into the global heap, if it is not already there, or decreases its key to $cost[\pi]$. (We assume that `heap-insert` does exactly that, i.e., inserts an item into a heap with a given key, or decreases its key, if the item is already in the heap.)

`dapsp-init` then calls `build-paths` which is also given in Figure 3. `build-paths` repeatedly removes a pair (u, v) with the smallest key from the global heap Q. The corresponding path $\pi[u, v]$ is then a shortest path. The call `new-shortest-path`(π) is then made.

The function `new-shortest-path`(π) receives a newly discovered shortest path. It sets to $sp[\pi]$ to `true`. It inserts π to the lists $SL[r[\pi]]$ and $SR[r[\pi]]$, as π is now a shortest path left extension of $r[\pi]$ and a shortest path right extension of $l[\pi]$. (Note that π is already contained in $L[r[\pi]]$ and $R[l[\pi]]$ at this stage.) Most importantly, `new-shortest-path`(π) now constructs LSPs extensions of π and examines each one of them. (These operations may add new pairs into the global heap Q.)

Using essentially the same arguments used to prove Theorem 2.2, we get that `dapsp-init` correctly finds all shortest and locally shortest paths in the graph.

```
Function dapsp-init($G = (V, E, c)$)
    $Q \leftarrow$ heap()
    foreach $u, v \in V$ do
        $\pi[u, v] \leftarrow null$
        $dist[u, v] \leftarrow \infty$
        $P[u, v] \leftarrow$ heap()
    foreach $u \in V$ do
        $p[u] \leftarrow$ path($u$)
        $\pi[u, u] \leftarrow p[u]$
        $dist[u, u] \leftarrow 0$
    foreach $e \in E$ do
        $p[e] \leftarrow$ path($e$)
        examine($p[e]$)
    build-paths()
```

```
Function update($E', c'$)
    $A \leftarrow \emptyset$
    foreach $e \in E'$ do
        remove-path($p[e]$, true)
    $Q \leftarrow$ heap()
    foreach $(u, v) \in A$ do
        replace-path($u, v$)
    foreach $e \in E'$ do
        $c[e] \leftarrow c'[e]$
        $p[e] \leftarrow$ path($e$)
        examine($p[e]$)
    build-paths()
```

```
Function build-paths()
    while $Q \neq \emptyset$ do
        $(u, v) \leftarrow$ extract-min($Q$)
        new-shortest-path($\pi[u, v]$)
```

Figure 3: Initiating and updating the dynamic all-pairs shortest paths data structure.

Updates are performed by calling `update`, also given in Figure 3. `update`(E', c') assigns the edges of E' new edges weights and recomputes all shortest paths. `update`(E', c') starts by removing all paths that pass through edges of E'. This done by calling `remove-path`$(p[e], $`true`$)$, for every $e \in E'$. (Function `remove-path` is discussed below.) These removals create a list A of pairs (u, v) that lost their shortest path. For every $(u, v) \in A$, a call is made to `replace-path`(u, v). Paths corresponding to all edges of E' are recreated, with their new costs, and these edge paths are examined. All updated shortest paths are then obtained by a call to `build-paths`.

`replace-path`(u, v), given in Figure 5, receives a pair of vertices (u, v) such that the shortest path from u to v has just been destroyed. It finds the shortest path π in $P[u, v]$, if there is one, and performs the necessary updates. (Note that π is not necessarily the shortest path from u to v. It is just the shortest path currently available.)

Finally, paths and their extensions are removed from the path system by the functions `remove-path` and `remove-exts` also given in Figure 5. To remove a path π from the path system, `remove-path`(π, rep) deletes π from $P[u, v]$, where $u = start[\pi]$ and $v = end[\pi]$ are the endpoints of π. It also deletes π from $R[l[\pi]]$ and $L[r[\pi]]$. If π is marked as a shortest path, i.e., $sp[\pi] =$ `true`, then π is also removed from $SR[l[\pi]]$ and $SL[r[\pi]]$. Finally, if $sp[\pi] =$ `true` and $rep =$ `true`, then (u, v) is inserted into a list A of pairs who lost their shortest paths. `remove-exts`(π, rep) removes all the extensions of π from the path system, by calling `remove-path`(π'), for every $\pi' \in L[\pi] \cap R[\pi]$.

Theorem 2.3 now follows by examining the operation of the algorithm. When a shortest path is destroyed it is removed from its local heap. In some cases, the shortest path in the local heap is found and a pair (u, v) is inserted into the global heap. The total cost of these operations is at most $T_{del}(\Lambda) + T_{min}(\Lambda) + T_{ins}(n^2)$, where Λ is an upper bound on the size of the local heaps. Each new shortest path is extracted from the global heap at a total cost of $T_{ext}(n^2)$. Each LSP destroyed is removed from its local heap at a cost of $T_{del}(\Lambda)$. Finally, each LSP created is inserted into the appropriate local heap and possibly causes a decrease-key operation on the global heap, a total cost of $T_{ins}(\Lambda) + T_{dec}(n^2)$.

```
Function new-shortest-path($\pi$)

  $sp[\pi] \leftarrow$ true
  insert($SL[r[\pi]], \pi$)
  insert($SR[l[\pi]], \pi$)
  foreach $\pi' \in SL[l[\pi]]$ do
      $\pi'' \leftarrow$ path($\pi', \pi$)
      examine($\pi''$)
  foreach $\pi' \in SR[r[\pi]]$ do
      $\pi'' \leftarrow$ path($\pi, \pi'$)
      examine($\pi''$)
```

```
Function examine($\pi$)

  $u \leftarrow start[\pi]$ ; $v \leftarrow end[\pi]$
  heap-insert($P[u,v], \pi, cost[\pi]$)
  if $cost[\pi] < dist[u,v]$ then
      if $\pi[u,v] \neq null$ then
          $sp[\pi[u,v]] \leftarrow$ false
          remove-exts($\pi[u,v]$, false)
      $\pi[u,v] \leftarrow \pi$
      $dist[u,v] \leftarrow cost[\pi]$
      heap-insert($Q, (u,v), cost[\pi]$)
```

Figure 4: The functions new-shortest-path and examine.

```
Function remove-path($\pi, rep$)

  $u \leftarrow start[\pi]$ ; $v \leftarrow end[\pi]$
  heap-delete($P[u,v], \pi$)
  delete($R[l[\pi]], \pi$)
  delete($L[r[\pi]], \pi$)
  if $sp[\pi] =$ true then
      if $rep =$ true then
          insert($A, (u,v)$)
      delete($SR[l[\pi]], \pi$)
      delete($SL[r[\pi]], \pi$)
  remove-exts($\pi, rep$)
```

```
Function remove-exts($\pi, rep$)

  foreach $\pi' \in L[\pi] \cup R[\pi]$ do
      remove-path($\pi', rep$)
```

```
Function replace-path($u, v$)

  if $P[u,v] \neq \emptyset$ then
      $\pi \leftarrow$ find-min($P[u,v]$)
      $\pi[u,v] \leftarrow \pi$
      $dist[u,v] \leftarrow cost[\pi]$
      heap-insert($Q, (u,v), cost[\pi]$)
  else
      $\pi[u,v] \leftarrow null$
      $dist[u,v] \leftarrow \infty$
```

Figure 5: Removing a path and its extensions from the path system.