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

Yuval Peres
Microsoft Research, Redmond
peres@microsoft.com

Dmitry Sotnikov
Tel Aviv University
dmitrysot@gmail.com

Benny Sudakov
UCLA
bsudakov@math.ucla.edu

Uri Zwick
Tel Aviv University
zwick@tau.ac.il

***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.**

***Keywords*-graph algorithms; shortest paths;**

I. 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 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 [1], Johnson [2], Fredman and Tarjan [3].) A slightly better running time of $O(mn + n^2 \log\log n)$ was obtained by Pettie [4], building on techniques developed by Thorup [5]. Karger, Koller and Phillips [6] and McGeoch [7] 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 [8], [9] (see also Thorup [10]) obtained a *dynamic* APSP algorithm with an *amortized* vertex update time of $\tilde{O}(n^2)$. Thorup [11] 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 weights 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 [8], [9] and Thorup [10], [11] 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 [12] and Frieze and Grimmett 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 Karger *et al.* [6] and McGeoch [7], or simply by first 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 [14] 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 [15] presented an improved algorithm with an expected running time of $O(n^2 \log n \log^* n)$. Moffat and Takaoka [16] and Mehlhorn and Priebe [17] improved the expected running time to $O(n^2 \log n)$ and showed that it also holds with high probability.

Cooper *et al.* [18] 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 [19], Hagerup [20] and Goldberg [21] obtained *Single-Source Shortest Paths* (SSSP) algorithms with an expected running time of $O(m)$. The m-edge input graph

---

Work of Uri Zwick supported by grant no. 2006261 of the United-States - Israel Binational Science Foundation (BSF).

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 [22] presented an average case analysis of the dynamic APSP algorithm of Demetrescu and Italiano [8], [9] 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 [23] and Janson [24] 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 [25], Smythe and Mahmoud [26] and Addario-Berry *et al.* [27].

In their survey on the algorithmic theory of random graphs, Frieze and McDiarmid [28] 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.

Our $O(n^2)$-time APSP algorithm is a static version of the dynamic APSP algorithm of Demetrescu and Italiano [8], [9] (see especially Section 3.4 of [9]) 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 II we sketch the static and dynamic versions of the algorithm of Demetrescu and Italiano [8], [9] used in this paper. 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 III we show that the *expected* number of LSPs in a complete directed graph with independent uniformly distributed random weights is $O(n^2)$. In Section IV we show that the number of such LSPs is $O(n^2)$ *with high probability*. Sections III and IV are the main sections of the paper. In Section V 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 VI we consider the expected time needed to perform *random edge* updates. Interestingly, the arguments used in Sections IV and VI are related, as they both concentrate 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 IV.) In Section VII 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 VIII with some concluding remarks and open problems.

II. THE ALGORITHM OF DEMETRESCU AND ITALIANO

Our $O(n^2)$ expected and high probability time bound 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 a random edge update are both obtained using variants of the dynamic APSP algorithm of Demetrescu and Italiano [8], [9].

As our main results are the *analysis* of these variants, and not the variants themselves, and as space is limited, we briefly describe 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 we use will appear in the full version of the paper.

Let $G = (V, E, c)$ be a weighted directed graph, where $c : E \to (0, \infty)$ is a cost function defined on its edges. For simplicity, we assume that 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 [8].) We let $u \to v$ denote the edge $(u, v) \in E$, and $u \rightsquigarrow v$ to 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 [8] 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 paths obtained by deleting its first and last edge, respectively, are 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.) A shortest path is of course also a locally shortest path. A locally shortest path, however, is not necessarily a shortest path. Note that each edge of the graph forms a locally shortest path. (Empty paths are considered to be shortest paths.)

A. A static version

We begin by describing a static version of the algorithm of Demetrescu and Italiano [8], [9]. 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) is inserted into a heap (priority queue) Q, with $dist[u, v]$ serving as its key.

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.

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 $u \rightsquigarrow v$ 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 then 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, then $dist[w, v]$, $p[w, v]$ and $q[w, v]$ are updated accordingly and the key of (w, v) in Q is decreased.

This is the gist of the static version of the algorithm of Demetrescu and Italiano [8], [9], which, for concreteness, we refer to as algorithm `apsp`. A complete description of the algorithm will appear in the full version of the paper.

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. The following theorem is not hard to prove. (Proof will appear in the full version.)

Theorem 2.2: Algorithm `apsp` correctly finds all shortest paths. It 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. It uses only $O(n^2)$ space.

If we use the Fibonacci heaps data structure (Fredman and Tarjan [3]) 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 $O(n^2 \log n + |\mathcal{LSP}|)$. The are, thus, two hurdles on our way to getting an expected $O(n^2)$-time algorithm. First, we have to show that $E[|\mathcal{LSP}|] = O(n^2)$, under natural probability distributions. We do that in Section III. Second, we have to find a faster way of implementing a heap. We do that in Section V using *bucket-based* priority queues.

B. A dynamic version

The static algorithm of the previous section examines all locally shortest paths in a graph, but maintains only those that are currently shortest. The dynamic algorithm, on the other hand, explicitly maintains all locally shortest paths.

For every path π, we let $\ell[\pi]$ and $r[\pi]$ be the paths obtained by deleting the last and first edge, respectively, of π. A path π is represented by keeping its total cost, its first and last edge, and pointers to its subpaths $\ell[\pi]$ 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 $P[u,v]$, 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 $\pi[u,v]$. Finally, 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. All LSP extensions of $\pi[u,v]$ are 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 [8], [9] 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 is used by Friedrich and Hebbinghaus [22]. The algorithm of Demetrescu and Italiano [8], [9] keeps such paths as *historical* and *locally historical* paths. (See also Demetrescu *et al.* [29].)

The most impressive feature of the dynamic algorithm of Demetrescu and Italiano [8], [9] 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 Δ 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, as well as a proof of the following theorem, will be given in the full version of the paper. We let $\texttt{update}(E', c')$ be 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

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

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

III. EXPECTED NUMBER OF LOCALLY SHORTEST PATHS

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 in this section that $W(a,b)$ is uniformly distributed in $[0,1]$. 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. We let $D_c(a,b)$ be the distance from a to b when c is removed from the graph. Let $\mathcal{LSP}$ be the set of LSPs in K_n. Our goal in this section is to show that $E[|\mathcal{LSP}|] = O(n^2)$.

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 lemmas can be found in Janson [24]. (The expectation of $D(a,b)$, but not the variance, can also be found in Davis and Prieditis [23]).

(The lemmas in [24] are stated for undirected graphs, but it is easy to check that they also hold for directed graphs.)

Lemma 3.1: For every two vertices $a \neq b \in V$ we have

$$E[D(a,b)] \;=\; \frac{H_{n-1}}{n-1} \;=\; \frac{\ln n}{n} + O(\frac{1}{n})$$

$$Var[D(a,b)] \;=\; \frac{\pi^2}{2n^2} + o(\frac{1}{n^2}).$$

Lemma 3.2: For any constant $c > 3$,

$$\Pr\left[\max_{a,b} D(a,b) \geq \frac{c \ln n}{n}\right] \;=\; O(n^{3-c} \log^2 n).$$

The fact that the expected number of LSPs is $O(n^2)$ follows almost immediately from the following lemmas.

Lemma 3.3: 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 probability that an edge $a \to b$ 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). We use, therefore, the following crude upper bound which is sufficient for our purposes.

As $D(a,b) \leq W(a,b)$, and $D(b,c) \leq W(b,c)$, the event "$a \to b$ and $b \to c$ are both shortest paths" is clearly contained in the union of the following two events: (i) $W(a,b) < \frac{5 \ln n}{n}$ and $W(b,c) < \frac{5 \ln n}{n}$. (ii) $D(a,b) \geq \frac{5 \ln n}{n}$ or $D(b,c) \geq \frac{5 \ln n}{n}$. The probability of the first event is exactly $(\frac{5 \ln n}{n})^2$, as $W(a,b)$ and $W(b,c)$ are independent. The probability of the second event is $O(\frac{\ln^2 n}{n^2})$ using Lemma 3.2 with $c = 5$. ■

Lemma 3.4: 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

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

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, $D(b,c) = D_a(b,c) = D_d(b,c)$. We also clearly have $D(a,c) \leq D_b(a,c)$ and $D(b,d) \leq D_c(b,d)$.

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

$$\begin{aligned} W(a,b) + D_a(b,c) \;&\leq\; D_b(a,c), \\ D_a(b,c) + W(c,d) \;&\leq\; D_c(b,d), \end{aligned}$$

or equivalently

$$\begin{aligned} W(a,b) \;&\leq\; D_b(a,c) - D_a(b,c), \\ W(c,d) \;&\leq\; D_c(b,d) - D_a(b,c). \end{aligned} \qquad (*)$$

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

$$X = D_a(b,c) \quad , \quad Y = D_b(a,c) \quad , \quad Z = D_c(b,d).$$

A crucial observation now is that X, Y and Z do *not* depend on $W(a,b)$ and $W(c,d)$. Indeed, $X = D_a(b,c)$ does not depend on $W(a,b)$ as $D_a(b,c)$ is the length of the shortest path from b to c that *avoids* a. Similarly, $Y = D_b(a,c)$ does not depend on $W(a,b)$ as it is the length of the shortest path that *avoids* b. Finally $Z = D_c(b,d)$ does not depend on $W(a,b)$ as the length of a shortest path from b to d, avoiding c, does not depend on the weights of edges *entering* b. A very similar reasoning shows that X, Y and Z also do not depend on $W(c,d)$.

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

$$\Pr[(*)] \;=\; E\big[(Y-X)^+ \cdot (Z-X)^+\big] \;\leq\; E\big[|Y-X||Z-X|\big],$$

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 $n-1$ vertices. Thus, $E[X] = E[Y] = E[Z]$. By Lemma 3.1, we have

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

Now,

$$\begin{aligned} \Pr[(*)] \;&\leq\; E\big[|Y-X||Z-X|\big] \\ &\leq\; \tfrac{1}{2}(E\big[(Y-X)^2\big] + E\big[(Z-X)^2\big]), \end{aligned}$$

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

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

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

Theorem 3.5: $E[|\mathcal{LSP}|] = O(n^2)$

Proof: The number of LSPs of length 1 is $n(n-1)$. (Every edge is an LSP of length 1.) By Lemma 3.3, 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 3.4, 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 $E[|\mathcal{LSP}|]$ is very close to $(\frac{\pi^2}{6} + 1)n^2 \simeq 2.64n^2$.

IV. 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 $\Pr[|\mathcal{LSP}| < cn^2] \to 1$, 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 [7] refers to $G^* = (V, E^*)$ as the *essential* subgraph.) We first show that $\Delta = O(\log n)$, with very high probability.

Lemma 4.1: For every $c > 6$, we have

$$\Pr[\Delta > c \ln n] \;=\; O(n^{1-c/6}).$$

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., [30], p. 64) states that if X is a binomial variable with $\mu = E[X]$, then $\Pr[X \geq 2\mu] \leq \mathrm{e}^{-\mu/3}$. Thus, the probability that the degree of a given vertex exceeds $c \ln n$ is at most $n^{-c/6}$. Thus $\Pr[\Delta' > c \ln n] \leq n^{1-c/6}$. Now, $\Delta > \Delta'$ only if at least one distance in G is greater than $\frac{c}{2}\frac{\ln n}{n}$. By Lemma 3.2, 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 [8].

Lemma 4.2: In any graph we have $|\mathcal{LSP}| \leq \Delta n^2$.

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 4.1 and 4.2 imply that the number of LSPs is $O(n^2 \log n)$ with high probability.

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

Lemma 4.3: For every $\beta > 0$, we have

$$E[|\mathcal{SP}^L|] = O(\tfrac{n^2}{\log^2 n}).$$

Proof: Let a, b be two distinct vertices and let $D(a,b)$ be the length of the shortest path connecting them. We know that $E[D(a,b)] = (1+o(1))\frac{\ln n}{n}$ and that $Var[D(a,b)] = (\frac{\pi^2}{2} + o(1))\frac{1}{n^2}$. By Chebyshev's inequality, we get that $\Pr[D(a,b) > (1+\beta)\frac{\ln n}{n}] = O(\frac{1}{(\beta \ln n)^2})$. The lemma follows by the linearity of expectation. ■

Using a more involved analysis, we can actually show that $E[|\mathcal{SP}^L|] = O(n^{2-c\beta})$, for some $c > 0$. The improved bound will appear in the full version of the paper.

Lemma 4.4: For every $\beta > 0$, we have

$$E[|\mathcal{LSP}^L|] = O(\tfrac{n^2}{\log n}).$$

Proof: Follows from Lemmas 4.3 and 4.1. ■

Lemma 4.5: For every $\beta > 0$ we have

$$\Pr[|\mathcal{LSP}^L| \geq n^2] = O(\tfrac{1}{\log n}).$$

Proof: Follows from Lemma 4.5 using Markov's inequality. ■

We now show that $|\mathcal{LSP}^S| = O(n^2)$ with high probability. To do that we use the *Efron-Stein inequality* to bound $Var[|\mathcal{LSP}^S|]$.

Let $f(X_1, \ldots, X_m)$ be a function of m independent random variables. Let X_i' be a random variable with the same distribution as X_i but independent from $X_1, X_2, \ldots, X_m$. We then have (see, e.g., [31]):

Theorem 4.6 (Efron-Stein inequality):

$$Var\big[f(X_1,\ldots,X_m)\big] \;\leq\; \tfrac{1}{2}\sum_{i=1}^{m} E\Big[\big(f(X_1,\ldots,X_i,\ldots,X_m) - f(X_1,\ldots,X_i',\ldots,X_m)\big)^2\Big].$$

In our case, we have $m = n(n-1)$, $X_1, X_2, \ldots, X_m$ are the random edge weights, and $f(X_1, \ldots, X_m) = |\mathcal{LSP}^S|$. For every edge e, we need to compute the second moment of the random variable $|\mathcal{LSP}^S_1| - |\mathcal{LSP}^S_0|$, where $\mathcal{LSP}^S_1$ and $\mathcal{LSP}^S_0$ are the sets of short LSPs when all edges other than e are assigned the *same* random edge weights, while e is assigned two independent edge weights.

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 $\mathcal{LSP}^S_1 \oplus \mathcal{LSP}^S_0$. We begin by looking at $\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1$.

For every $u, v \in V$, let $\pi_0[u,v]$ and $\pi_1[u,v]$ be the shortest path from u to v with the two choices of the weight of e. The following lemma is immediate:

Lemma 4.7: If $\pi_0[u,v] \neq \pi_1[u,v]$, then $e \in \pi_0[u,v]$ or $e \in \pi_1[u,v]$.

Proof: Let $c_0(e)$ and $c_1(e)$ be the two costs of e and suppose that $\pi_0[u,v] \neq \pi_1[u,v]$. If $c_0(e) < c_1(e)$, then $\pi_0[u,v]$ must pass through e. If $c_0(e) > c_1(e)$, then $\pi_1[u,v]$ must pass through e. ■

Let $\mathcal{SP}^S_0(e)$ and $\mathcal{SP}^S_1(e)$ be the set of short shortest paths that pass through e with the two choices of the weight of e. By Lemma 4.7 we get that $|\mathcal{SP}^S_0 \oplus \mathcal{SP}^S_1| \leq |\mathcal{SP}^S_0(e)| + |\mathcal{SP}^S_1(e)|$. We next estimate $|\mathcal{SP}^S_0(e)|$ and $|\mathcal{SP}^S_0(e)|$. As they both have the same distribution, we omit the subscript.

Lemma 4.8: For every $\beta > 0$, we have $\Pr[|\mathcal{SP}^S(e)| > 0] = O(\frac{\ln n}{n})$.

Proof: $\mathcal{SP}^S(e)$ is non-empty only if e is a shortest path between its endpoints, which only happens with probability $O(\frac{\ln n}{n})$. ■

Our next goal is to show that $|\mathcal{SP}^S(e)| = O(n^{(1+\beta)^2})$, with high probability. The proof of this result relies on the following large deviation theorem of Maurer [32].

Theorem 4.9: Let $X_1, X_2, \ldots, X_n$ be non-negative independent random variables with finite first and second moments and let $S = \sum_{i=1}^{n} X_i$. Let $t > 0$. Then

$$\Pr\big[E[S] - S \geq t\big] \leq \exp\left(\frac{-t^2}{2\sum_{i=1}^{n} E[X_i^2]}\right).$$

Lemma 4.10: For any $\alpha \leq 1$ and $\epsilon > 0$, the probability that the number of vertices at distance at most $\alpha\frac{\ln n}{n}$ from a given vertex is greater than $n^{\alpha+\epsilon}$ is $O(n^{-c})$, for any $c > 0$.

Proof: We may assume that $\alpha + \epsilon < 1$, as otherwise the claim is obvious. Let a be a fixed vertex. Let us bound the probability that the distance of the k-th closest vertex to a, where $k \leq n$, is at most $\frac{\alpha \ln n}{n}$. It is known (see, e.g., Janson [24]), that the distance to the k-th vertex from a is $D_k = \sum_{i=1}^{k-1} X_i$, where $X_i = EXP(\frac{1}{i(n-i)})$ is an exponential random variable with mean $\frac{1}{i(n-i)}$. Now

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

As $E[X_i^2] = \frac{2}{i^2(n-i)^2}$, we have

$$\begin{aligned} &\textstyle\sum_{i=1}^{k} E[X_i^2] = \sum_{i=1}^{k-1} \frac{2}{i^2(n-i)^2} \leq \sum_{i=1}^{n-1} \frac{2}{i^2(n-i)^2} \\ &= 2\textstyle\sum_{i=1}^{n/2} \frac{2}{i^2(n-i)^2} \leq \frac{8}{n^2}\sum_{i=1}^{n/2} \frac{1}{i^2} \leq \frac{8\pi^2}{6}\frac{1}{n^2} < \frac{14}{n^2}. \end{aligned}$$

With $k = n^{\alpha+\epsilon}$ we get that $E[D_k] > \frac{(\alpha+\epsilon)\ln n}{n}$ and by Theorem 4.9 we have

$$\begin{aligned} \Pr\big[D_k \leq \tfrac{\alpha \ln n}{n}\big] \;&\leq\; \Pr\big[E[D_k] - D_k \geq \tfrac{\epsilon \ln n}{n}\big] \\ \leq\; \exp\left(\frac{-\epsilon^2\big(\frac{\ln n}{n}\big)^2}{\frac{28}{n^2}}\right) \;&=\; \exp\big(-(\epsilon \ln n)^2/28\big). \end{aligned}$$

Thus, for every $\epsilon > 0$, the probability that the number of vertices at distance at most $\frac{\alpha \ln n}{n}$ is more than $n^{\alpha+\epsilon}$ is $O(n^{-c})$, for any $c > 0$. ■

Lemma 4.11: For every $\beta > 0$, $\Pr[|\mathcal{SP}^S(e)| > n^{(1+\beta)^2}] = O(n^{-c})$, for every $c > 0$.

Proof: Let $e = (a, b)$ 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, let

$$\begin{aligned} A_i \;&=\; \left\{u \in V \mid D(u, a) \leq \tfrac{i(1+\beta)}{r}\tfrac{\ln n}{n}\right\}, \\ B_i \;&=\; \left\{v \in V \mid D(b, v) \leq \tfrac{i(1+\beta)}{r}\tfrac{\ln n}{n}\right\}, \end{aligned}$$

be the sets of vertices of distances at most $\frac{i(1+\beta)}{r}\frac{\ln n}{n}$ to a and from b, respectively. Clearly

$$C \;\subseteq\; \bigcup_{i=0}^{r} A_i \times B_{r+1-i}.$$

By Lemma 4.10, we have $|A_i|, |B_i| \leq n^{(1+\beta)i/r+\epsilon}$, for every i, with a probability of at least $1 - O(rn^{-c})$, for every $c > 0$. It thus follows that $|C| \leq (r+1)n^{(1+\beta)(1+1/r)+\epsilon}$, again with this very high probability. Letting $r = \lceil\frac{2}{\beta}\rceil$ and ϵ sufficiently small, we get the claim of the lemma. ■

Lemmas 4.8 and 4.11 allow us to bound $E[|\mathcal{SP}^S(e)|^2]$.

Lemma 4.12: For every $\beta > 0$, we have $E[|\mathcal{SP}^S(e)|^2] = O(n^{2(1+\beta)^2-1}\log n)$.

Proof: For succinctness, let $X = |\mathcal{SP}^S(e)|$ and $a = n^{(1+\beta)^2}$. We always have $X^2 \leq n^4$. Using Lemma 4.11 with $c = 4$, we have

$$\begin{aligned} E[X^2] \;&\leq\; \Pr[0 < X \leq a] \cdot a^2 + \Pr[x > a] \cdot n^4 \\ =\; O(\tfrac{\ln n}{n} \cdot n^{2(1+\beta)^2} + n^{-4} \cdot n^4) \;&=\; O(n^{2(1+\beta)^2-1}\log n). \end{aligned}$$

■

We can finally get back to estimating $\mathcal{LSP}_1^S \oplus \mathcal{LSP}_0^S$.

Lemma 4.13: For every $\beta > 0$ we have $|\mathcal{LSP}_1^S \oplus \mathcal{LSP}_0^S| \leq 2\Delta \cdot |\mathcal{SP}_1^S \oplus \mathcal{SP}_0^S|$.

Proof: Suppose that $\pi \in \mathcal{LSP}_1^S \oplus \mathcal{LSP}_0^S$. Thus means that either $\ell[\pi] \in \mathcal{SP}_1^S \oplus \mathcal{SP}_0^S$ or $r[\pi] \in \mathcal{SP}_1^S \oplus \mathcal{SP}_0^S$. Each shortest path has at most Δ pre-extensions and at most Δ post-extensions. ■

Lemma 4.14: For every $\beta > 0$ and every edge e we have $E[\big||\mathcal{LSP}_1^S| - |\mathcal{LSP}_0^S|\big|^2] = O(n^{2(1+\beta)^2-1}\log^3 n)$.

Proof: By Lemmas 4.7 and 4.13 we have

$$\begin{aligned} &\big||\mathcal{LSP}_1^S| - |\mathcal{LSP}_0^S|\big|^2 \leq \big|\mathcal{LSP}_0^S \oplus \mathcal{LSP}_1^S\big|^2 \leq 4\Delta^2\big|\mathcal{SP}_0^S \oplus \mathcal{SP}_1^S\big|^2 \\ &\leq 4\Delta^2\big(|\mathcal{SP}_0^S(e)| + |\mathcal{SP}_1^S(e)|\big)^2 \leq 8\Delta^2(|\mathcal{SP}_0^S(e)|^2 + |\mathcal{SP}_1^S(e)|^2). \end{aligned}$$

The claim now follows from Lemmas 4.1 and 4.12. ■

Using the Efron-Stein inequality (Theorem 4.6) we get:

Lemma 4.15: For every $\beta > 0$ we have $Var[|\mathcal{LSP}^S|] = O(n^{2(1+\beta)^2+1}\log^3 n)$.

We now choose β small enough such that, say, $Var[|\mathcal{LSP}^S|] \leq n^{3.5}$. By Theorem 3.5, we have $E[|\mathcal{LSP}^S|] \leq E[|\mathcal{LSP}|] = O(n^2)$. By Chebyshev's inequality, we get that $\Pr[|\mathcal{LSP}^S| \geq 2E[|\mathcal{LSP}^S|]] = O(n^{-1/2})$. Putting everything together, we get

Theorem 4.16: There is a constant c such that $\Pr[|\mathcal{LSP}| \geq cn^2] = O(\frac{1}{\log n})$.

Using the improvement mentioned after Lemma 4.3, we can improve the bound of Theorem 4.16 from $O(1/\log n)$ to $O(n^{-a})$, for some $a > 0$. More details will be given in the full version of the paper.

V. AN $O(n^2)$-TIME IMPLEMENTATION

In this section we describe an implementation of the algorithm of Section II-A that runs in $O(n^2)$ time, in expectation and with high probability. This is done using a simple observation of Dinic [33] and a simple bucket-based priority queue implementation that goes back to Dial [34].

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 II-A 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. This observation, in the context of Dijkstra's algorithm, dates back to Dinic [33]. 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 [5]. It is also used by Hagerup [20] to obtain a simple linear *expected* time algorithm for single source shortest paths, simplifying results of Meyer [19] and Goldberg [21].

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 $1/(n^2 \ln n)$ is clearly at most $1/\ln n$. If this (unlikely but not extremely 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 $\delta \geq 1/(n^2 \ln n)$. 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 $L = \lceil 4n \ln^2 n \rceil$ 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 3.5 we have $E[|\mathcal{LSP}|] = O(n^2)$. By Theorem 4.16, there is a constant c such that $\Pr[|\mathcal{LSP}| \geq cn^2] = O(1/\log n)$. Thus, the number of operations here is $O(n^2)$, both in expectation and with high probability. (Note that $L = \lceil 4n \ln^2 n \rceil \ll n^2$. We could have used many more buckets if we wanted to.)

All that remains, therefore, is to show that the probability that B_L will be the only non-empty bucket is fairly small. Note that this happens if and only if there is a pair (u,v) for which $D(u,v) \geq L\delta \geq \frac{4 \ln n}{n}$. Using Lemma 3.2, with $c = 4$, we get that the probability that we will end up with B_L being non-empty is $O(\ln^2 n/n)$. As the running time in this case is $O(n^2 \log n)$, the contribution of this event to the expected running time of the whole algorithm is negligible. We have thus obtained:

Theorem 5.1: The 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 $O(n^2)$, in expectation and with high probability.

VI. 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 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. Our main goal is to bound the expected sizes of these sets. This, in conjunction with Theorem 2.3, would supply an upper bound on the expected update time.

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. By examining the behavior of the dynamic algorithm of Section II-B, we get that $\mathcal{SP}^- = \{\pi_0[u,v] \mid (u,v) \in B_0 \cup B_1\}$ and $\mathcal{SP}^+ = \{\pi_1[u,v] \mid (u,v) \in B_0 \cup B_1\}$. In particular $|\mathcal{SP}^-| = |\mathcal{SP}^+|$.

To bound $E[|\mathcal{SP}^-|] = E[|\mathcal{SP}^+|]$ it is thus enough to bound $E[|B_0|]$ and $E[|B_1|]$. Although B_0 and B_1 are not always of the same size, we have $E[|B_0|] = E[|B_1|]$, as they are both equal to the number of shortest paths passing through a randomly selected edge e, when all edge weights are random.

Lemma 6.1: 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 [24].) It is known that this average depth is $(1 + o(1)) \ln n$ (Moon [35]). The same asymptotic result holds also under the uniform distribution. (See Section 2 of Janson [24].) ■

Lemma 6.2: 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

$$\begin{aligned} E[|\mathcal{SP}(e)|] &= E\left[\tfrac{1}{n(n-1)} \textstyle\sum_{e'} |\mathcal{SP}(e')|\right] \\ &= E\left[\tfrac{1}{n(n-1)} \textstyle\sum_{u \neq v} |\pi[u, v]|\right] = E[|\pi[u, v]|]. \end{aligned}$$

By Lemma 6.1, we get that $E[|\mathcal{SP}(e)|] = (1 + o(1)) \ln n$. ■

Theorem 6.3: Following a random edge update, we have $E[|\mathcal{SP}^-|] = E[|\mathcal{SP}^+|] \leq (2 + o(1)) \ln n$.

Let Δ be the maximal degree of the essential graph $G^* = (V, E^*)$ defined in the previous section. Lemma 4.1 says that with high probability $\Delta = O(\log n)$.

Theorem 6.4: Following a random edge update we have $E[|\mathcal{LSP}^-|] = E[|\mathcal{LSP}^+|] = O(\log^2 n)$.

Proof: Clearly $\pi \in \mathcal{LSP}^-$ if and only if $\ell[\pi] \in \mathcal{SP}^-$ 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 4.1, we have $\Pr[\Delta > 24 \ln n] = O(n^{-3})$. As $|\mathcal{LSP}|$ is always at most n^3, we get $E[|\mathcal{LSP}^-|] \leq 48 \ln n \cdot E[|\mathcal{SP}^-|] + n^{-3} \cdot n^3 = O(\log^2 n)$. ■

We believe that the $O(\log^2 n)$ bound in Theorem 6.4 can be improved, possibly to $O(\log n)$, and leave it as an open problem.

Theorem 6.5: 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 $O(\log^2 n)$.

VII. OTHER GRAPH MODELS

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. Details will be give in the full version of the paper. 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.

VIII. 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 [28].

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 $O(\log n)$.

REFERENCES

[1] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] D. Johnson, “Efficient algorithms for shortest paths in sparse graphs,” *Journal of the ACM*, vol. 24, pp. 1–13, 1977.

[3] M. Fredman and R. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[4] S. Pettie, “A new approach to all-pairs shortest paths on real-weighted graphs,” *Theoretical Computer Science*, vol. 312, no. 1, pp. 47–74, 2004.

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

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

[7] C. McGeoch, “All-pairs shortest paths and the essential subgraph,” *Algorithmica*, vol. 13, pp. 426–441, 1995.

[8] C. Demetrescu and G. Italiano, “A new approach to dynamic all pairs shortest paths,” *Journal of the ACM*, vol. 51, no. 6, pp. 968–992, 2004.

[9] ——, “Experimental analysis of dynamic all pairs shortest path algorithms,” *ACM Transactions on Algorithms*, vol. 2, no. 4, pp. 578–601, 2006.

[10] M. Thorup, “Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles,” in *Proc. of 9th SWAT*, 2004, pp. 384–396.

[11] ——, “Worst-case update times for fully-dynamic all-pairs shortest paths,” in *Proc. of 37th STOC*, 2005, pp. 112–119.

[12] R. Hassin and E. Zemel, “On shortest paths in graphs with random weights,” *Mathematics of Operations Research*, vol. 10, no. 4, pp. 557–564, 1985.

[13] A. Frieze and G. Grimmett, “The shortest-path problem for graphs with random arc-lengths,” *Discrete Applied Mathematics*, vol. 10, pp. 57–77, 1985.

[14] P. 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*, vol. 2, no. 1, pp. 28–32, 1973.

[15] P. Bloniarz, "A shortest-path algorithm with expected time $O(n^2 \log n \log^* n)$," *SIAM Journal on Computing*, vol. 12, no. 3, pp. 588–600, 1983.

[16] A. Moffat and T. Takaoka, "An all pairs shortest path algorithm with expected time $O(n^2 \log n)$," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 1023–1031, 1987.

[17] K. Mehlhorn and V. Priebe, "On the all-pairs shortest-path algorithm of Moffat and Takaoka," *Random Structures and Algorithms*, vol. 10, no. 1-2, pp. 205–220, 1997.

[18] 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*, vol. 16, no. 1, pp. 33–46, 2000.

[19] U. Meyer, "Average-case complexity of single-source shortest-paths algorithms: lower and upper bounds," *Journal of Algorithms*, vol. 48, no. 1, pp. 91–134, 2003.

[20] T. Hagerup, "Simpler computation of single-source shortest paths in linear average time," *Theory of Computing Systems*, vol. 39, no. 1, pp. 113–120, 2006.

[21] A. Goldberg, "A practical shortest path algorithm with linear expected time," *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1637–1655, 2008.

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

[23] R. Davis and A. Prieditis, "The expected length of a shortest path," *Information Processing Letters*, vol. 46, no. 3, pp. 135–141, 1993.

[24] S. Janson, "One, two and three times $\log n/n$ for paths in a complete graph with random weights," *Combinatorics, Probability & Computing*, vol. 8, no. 4, pp. 347–361, 1999.

[25] L. Devroye, "Branching processes in the analysis of the heights of trees," *Acta Informatica*, vol. 24, no. 3, pp. 277–298, 1987.

[26] R. Smythe and H. Mahmoud, "A survey of recursive trees," *Theory of Probability and Mathematical Statistics*, no. 51, pp. 1–29, 1995.

[27] L. Addario-Berry, N. Broutin, and G. Lugosi, "The longest minimum-weight path in a complete graph," *Combinatorics, Probability & Computing*, vol. 19, no. 1, pp. 1–19, 2010.

[28] A. Frieze and C. McDiarmid, "Algorithmic theory of random graphs," *Random Structures and Algorithms*, vol. 10, no. 1-2, pp. 5–42, 1997.

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

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

[31] S. Boucheron, G. Lugosi, and O. Bousquet, "Concentration inequalities," in *Advanced Lectures on Machine Learning*, 2003, pp. 208–240.

[32] A. Maurer, "A bound on the deviation probability for sums of non-negative random variables," *JIPAM. Journal of Inequalities in Pure and Applied Mathematics*, vol. 4, no. 1, pp. Article 15, 6 pp. (electronic), 2003.

[33] E. Dinic, "Economical algorithms for finding shortest paths in a network," in *Transportation Modeling Systems*, Y. Popkov and B. Shmulyian, Eds. Institute for System Studies, Moscow, CIS, 1978, pp. 36–44.

[34] R. Dial, "Algorithm 360: shortest-path forest with topological ordering," *Communications of the ACM*, vol. 12, no. 11, pp. 632–633, 1969.

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