# All-Pairs Shortest Paths with a Sublinear Additive Error

Liam Roditty [*] Asaf Shapira [†]

**Abstract**

We show that for every $0 \leq p \leq 1$ there is an $O(n^{2.575-p/(7.4-2.3p)})$ time algorithm that given a directed graph with small positive integer weights, estimates the length of the shortest path between every pair of vertices $u, v$ in the graph to within an *additive* error $\delta^p(u, v)$, where $\delta(u, v)$ is the exact length of the shortest path between $u$ and $v$. This algorithm runs faster than the fastest algorithm for computing exact shortest paths for any $0 < p \leq 1$.

Previously the only way to "bit" the running time of the exact shortest path algorithms was by applying an algorithm of Zwick [FOCS '98] that approximates the shortest path distances within a *multiplicative* error of $(1 + \epsilon)$. Our algorithm thus gives a smooth qualitative and quantitative transition between the fastest *exact* shortest paths algorithm, and the fastest *approximation* algorithm with a linear additive error. In fact, the main ingredient we need in order to obtain the above result, which is also interesting in its own right, is an algorithm for computing $(1 + \epsilon)$ multiplicative approximations for the shortest paths, whose running time is faster than the running time of Zwick's approximation algorithm when $\epsilon \ll 1$ and the graph has small integer weights.

[*]Weizmann Institute. E–mail: liam.roditty@weizmann.ac.il

[†]Microsoft Research. E–mail: asafico@tau.ac.il

## 1 Introduction

Computing all-pairs shortest paths (APSP) in graphs is without a doubt one of the most well-studied problems both in practical and theoretical computer-science (see [25] for a recent survey). For arbitrarily dense graphs with real weighted edges, the best algorithm is essentially the classical $O(n^3)$ time algorithm of Floyd-Warshall (see [7]). The first improvement over this algorithm was obtained by Fredman [12] who gave an $O(n^3/(\frac{\log n}{\log\log n})^{1/3})$ time algorithm. Several improvement then followed, culminating in a recent result of Chan [4] who gave an $O(n^3/\frac{\log^2 n}{\log\log n})$ algorithm. The question of whether the APSP problem can be solved in truly subcubic time, that is, in time $O(n^{3-c})$ for some $c > 0$, remains a major open problem.

Besides trying to obtain slight poly-logarithmic improvements over the naive $O(n^3)$ algorithm for general graphs, most of the research on the APSP problem focused on obtaining truely subcubic algorithms for graphs with *small integer edge weights*, where throughout the paper, when we say small integer edge weights we mean edge weights taken from the set $\{-M, \ldots, M\}$, with $M = n^{o(1)}$. This problem turns out to be closely related to the problem of fast (that is, subcubic) matrix multiplication algorithms. The first to realize this connection were Alon, Galil and Margalit [3] who showed how to solve the APSP problem in directed graphs with small integer edge weights in time[1] $O(n^{\frac{\omega+3}{2}})$, where $\omega < 2.376$ is the exponent of the fastest algorithm for multiplying two matrices, due to Coppersmith and Winograd [6]. Zwick [24] obtained an improved algorithm that can solve the APSP problem in directed graphs with small integer weights in time $O(n^{2.575})$. Even in the case of unweighted directed graphs, the fastest APSP algorithm is Zwick's $O(n^{2.575})$ time algorithm. The only lower bound on the APSP problem in directed graphs is $\Omega(n^\omega)$ that comes from the fact that APSP is at least as hard as boolean matrix multiplication. Therefore, even in unweighted directed graphs there is a gap between the $O(n^{2.575})$ upper bound and the $\Omega(n^\omega)$ lower bound. In fact, if $\omega = 2$ then both the algorithms of [3] and [24] run in $O(n^{2.5})$, so if $\omega = 2$ then the last progress on the APSP problem in directed graph with small weights (or even unweighted) was [3] from 1991. We finally note that the only (general) case where the APSP problem can be solved in time $O(n^\omega)$ is in the case of *undirected* graphs with small weights, see [15, 16, 21, 20].

Our focus in this paper is on the APSP problem in directed graphs with small positive integer weights. As we have discussed in the previous paragraph, even this spacial case of the problem is not well understood. From the results above, we know that this problem has an upper bound of $O(n^{2.575})$ and a lower bound of $\Omega(n^\omega)$. A natural question is therefore what can we do faster than we can exactly solve the APSP problem, that is, what can we do in time less than $O(n^{2.575})$? By faster we mean by a factor of $n^c$. The reason is (i) In most cases we don't even know the exact exponent, (ii) In most cases we disregard $n^{o(1)}$ factors as the fast matrix multiplication algorithms "hide" such factors anyway. To the best of our knowledge, the only result in this direction is an $O(n^\omega/\epsilon)$ algorithm of Zwick [24] that approximates the shortest path distances in a directed graph with positive edge weights [2] to within a *multiplicative* error $(1 + \epsilon)$. That is, if $\delta(u, v)$ denotes the length of the shortest path connecting $u$ and $v$, then Zwick's algorithm returns an estimate $\hat{\delta}(u, v)$ satisfying $\delta(u, v) \le \hat{\delta}(u, v) \le (1 + \epsilon)\delta(u, v)$.

Our motivation for studying better approximations for the APSP problem in directed graphs

[1]Throughout the paper, with a slight abuse of notation, we use $O(n^r)$ to denote a running time of $O(n^{r+o(1)})$.

[2]We note that as opposed to the previous algorithms we have discussed, Zwick's approximation algorithm has a good *logarithmic* dependence on the size of the edge weights.

stems from the fact that if one considers *undirected* graphs, then one can obtain much better approximations. Two notable example are an $O(n^{2.5})$ algorithm of Aingworth et. al. [2] that approximates the distances in *undirected unweighted* graphs to within an *additive* error 2, and an $O(n^2)$ time algorithm of Dor, Halperin and Zwick [9] that approximates the distances in *undirected unweighted* graphs to within an *additive* error $O(\log n)$. It is interesting to note that these two algorithms do *not* use fast matrix multiplication algorithms.

Our main result in this paper is that one can also obtain additive approximations in directed graphs. However, they are not as good as those that can be obtained in undirected graphs. We give an algorithm for computing additive approximations that are (arbitrarily) polynomially small in the actual distance between a pair of vertices. In fact our additive approximation gives a "smooth" transition between the fastest exact $O(n^{2.575})$ APSP algorithm and the $O(n^{\omega})$ $(1+\epsilon)$-multiplicative approximation algorithm. En-route we will also improve the running time of Zwick's $(1+\epsilon)$-multiplicative approximation algorithm whenever $\epsilon \ll 1$. We also show that our results are tight in the sense that the only way to obtain approximations, similar in quality to those achievable in undirected graphs, is to actually improve the running time of the fastest *exact* APSP algorithm.

As we have mentioned above, the study of the APSP problem in graphs with small weights is closely related to the problem of designing fast algorithms for matrix multiplication. Before turning to discuss our new results, which also apply fast matrix multiplication algorithms, let us briefly review the relevant results on fast matrix multiplication. Let $\omega(1,r,1)$ be the minimal exponent for which the product of an $n \times n^r$ matrix and an $n^r \times n$ matrix can be computed in $O(n^{\omega(1,r,1)})$ time[3]. The exponent $\omega = \omega(1,1,1)$ is usually called the exponent of fast matrix multiplication. Coppersmith and Winograd [6] proved that $\omega < 2.376$. Coppersmith [5] showed that $\omega(1,0.294,1) = 2$, that is, the product of an $n \times n^{0.294}$ matrix and an $n^{0.294} \times n$ matrix can be computed in $O(n^2)$. Let $\alpha = \sup\{0 \le r \le 1 : \omega(1,r,1) = 2\}$, so Coppersmith's [5] result mentioned above can be stated as $\alpha > 0.294$. Although one can trivially obtain bounds on $\omega(1,r,1)$ by breaking the two rectangular matrices into small square ones, Huang and Pan [18] obtained the following improved bound on $\omega(1,r,1)$.

**Theorem 1 ([18])** *Let* $\omega = \omega(1,1,1) < 2.376$ *and* $\alpha = \sup\{0 \le r \le 1 : \omega(1,r,1) = 2\} > 0.294$. *Then*

$$\omega(1,r,1) = \begin{cases} 2 & 0 \le r \le \alpha \\ 2 + \frac{\omega-2}{1-\alpha}(r-\alpha) & \alpha \le r \le 1 \end{cases} \qquad (1)$$

*In particular, for any* $0.294 \le r \le 1$ *we have* $\omega(1,r,1) \le 1.843 + 0.532 \cdot r$.

## 2 The New Results

Let us introduce the main result of this paper. We show that it is possible to compute arbitrary polynomially small additive approximations for the APSP in directed graphs with small weights, and still bit the running time of the fastest exact algorithm that computes APSP.

[3]More generally $\omega(r,s,t)$ denotes the exponent of the fastest algorithm for multiplying an $n^r \times n^s$ matrix with by an $n^s \times n^t$ matrix.

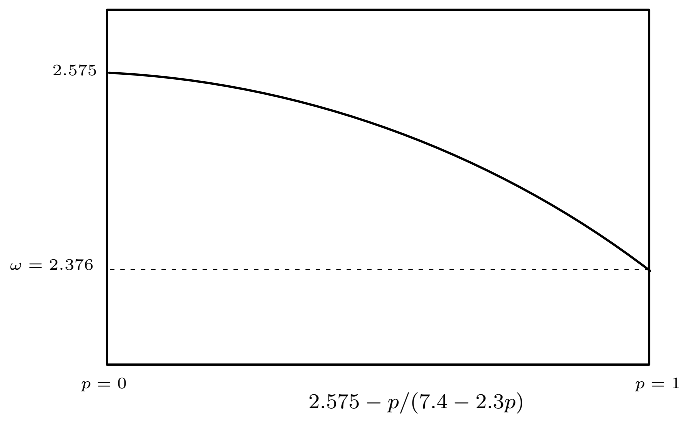

Figure 1: The exponent $2.575 - \frac{p}{7.4-2.3p}$ of the algorithm of Theorem 2 as a function of $p$.

**Theorem 2 (Main Result)** *For every $0 \leq p \leq 1$ there is a randomized $O(n^{2.575-\frac{p}{7.4-2.3p}})$ time algorithm that given a directed graph $G = (V, E)$, computes with high probability, for every pair $u, v \in V$ a value $\hat{\delta}(u, v)$ satisfying*

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + \delta^p(u, v) .$$

Observe that when $p = 1$ the running time of the above algorithm is $O(n^{2.376}) = O(n^{\omega})$. This corresponds to the case considered in [24] of computing estimates $\hat{\delta}(u, v)$ of the shortest paths satisfying $\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + O(\delta(u, v))$. Also, when $p = 0$, that is when we want to compute exact shortest paths, the running time of the above algorithm is $O(n^{2.575})$, which is the running time of the exact APSP algorithm of [24]. In particular, we get that for any $0 < p \leq 1$ the running time of the algorithm of Theorem 2 is faster than $O(n^{2.575})$ by a polynomial factor. So for any $0 \leq p \leq 1$ Theorem 2 gives a "smooth" transition from the fastest exact APSP algorithm and the fastest approximation with a linear error for directed graphs with small integer weights. See Figure 1. We stress again that in this paper we consider the APSP problem in directed graphs with small positive integer weights, while the exact $O(n^{2.575})$ APSP of [24] works also for small *negative* weights, and the approximation algorithm of [24] works also for *large* positive edge weights.

It is also interesting to consider the performance of our algorithm under the assumption that $\omega = 2$. As we briefly explain later, in that case the running time of the algorithm is $O(n^{2+\frac{1-p}{2-p}})$. Note that when $p \approx 1$ this running time becomes $O(n^2)$ and when $p \approx 0$ this running time becomes $O(n^{2.5})$. Recall again that assuming $\omega = 2$ the fastest APSP algorithm runs in $O(n^{2.5})$, so under the assumption that $\omega = 2$ the performance of the algorithm is also faster than that of the fastest exact APSP algorithm for any $0 < p \leq 1$. The resulting transition from $p = 0$ to $p = 1$ is similar to the one in Figure 1 only now it is between 2.5 and 2.

We note that the only way one can use previous results in order to obtain approximations with the quality of those given in Theorem 2 is to use the $(1 + \epsilon)$-multiplicative approximation algorithm of [24] with $1/n^{1-p}$. However, the running time of this algorithm is $O(n^{\omega+1-p})$ which is slower than the running time of the algorithm we obtain in Theorem 2 for any $0 \leq p < 1$ . As we show in the next subsection, when $\epsilon = 1/n^t$ we can improve the running time of the algorithm of [24] but even this improvement does not directly imply the result in Theorem 2 and more ideas are needed.

## 2.1 An improved multiplicative approximation algorithm

The main ingredient that we need in order to obtain our main result stated in Theorem 2 is an algorithm for computing $(1+\epsilon)$ multiplicative approximations of the shortest paths in direct graphs with small integer weights. As we will see later, we will need to apply this algorithm when $\epsilon = 1/n^t$ for some $t > 0$, that is when $\epsilon \ll 1$. Zwick's algorithm [24] for computing such approximations runs in time $O(n^{\omega+t})$. The following theorem shows that for such values of $\epsilon$ one can obtain a faster algorithm.

**Theorem 3** *For every $\epsilon = 1/n^t$ there is a randomized $O(n^{\omega(1,1-t,1)+t})$ time algorithm that given a directed graph $G = (V, E)$, computes with high probability, for every pair $u, v \in V$ a value $\hat{\delta}(u, v)$ satisfying*

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq (1 + \epsilon)\delta(u, v) \; .$$

Given the current bounds on $\omega(1, r, 1)$ that were given in Theorem 1, we see that when $\epsilon = 1/n^t$ the running time of the algorithm of Theorem 3 is $O(n^{\omega+0.468 \cdot t})$, which improves Zwick's algorithm [24] that runs in time $O(n^{\omega+t})$.

The algorithm given in Theorem 3 applies two of the main ideas from [24]. The first is the use of random sampling and rectangular matrix multiplication that was used in [24] in order to obtain an exact APSP algorithm. The second is the idea of *scaling* that was used in [24] in order to obtain an approximate APSP algorithm.

## 2.2 "Hardness" results

A natural qualitative question that arises given Theorem 2 is whether one can design approximation algorithms for APSP whose running time is faster than that of the fastest $O(n^{2.575})$ APSP algorithm by some polynomial factor[4] and yet supply better than a polynomially small additive error. For example, given the results of [9, 2] on additive approximations in undirected graphs that we have mentioned before, one can ask if for some $c > 0$, it is possible to design an $O(n^{2.575-c})$ time algorithm that will compute estimates $\hat{\delta}(u, v)$ satisfying $\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + O(\log \delta(u, v))$. As the following proposition shows, even if we consider a relaxed version of this problem, where the error is relative to $n$ rather than $\delta(u, v)$, such an algorithm would imply an improvement over the fastest APSP algorithm.

**Proposition 2.1** *Suppose that for some $r \geq \omega$ and $\epsilon \leq \frac{1}{2}$ there is an $O(n^r)$ time algorithm[5] that given a directed graph $G = (V, E)$ computes for any pair $u, v \in V$ a value $\hat{\delta}(u, v)$ satisfying*

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + n^{\epsilon} \; .$$

*Then there is an $O(n^{r+2r\epsilon})$ time algorithm for the exact APSP problem. In particular, if $\epsilon = o(1)$ and $r = 2.575 - c$ for some positive $c > 0$, then there is also an $O(n^{2.575-c})$ time algorithm for the exact APSP problem.*

[4]Remember that when we measure the running time in the context of fast matrix multiplication based algorithms, we disregard $n^{o(1)}$ factors so we are only interested in running time that is faster by a factor of $n^c$ for some $c > 0$.

[5]We assume (for convenience) that $r \geq \omega$ because this problem is at least as hard as computing transitive closure of a graph, which is equivalent to boolean matrix multiplication.

So the above proposition implies that if we are interested in an algorithm that runs faster than the best APSP algorithm by a polynomial factor, then unless we improve over the fastest APSP algorithms, all we can hope for is to obtain a polynomially small additive error.

The next proposition gives another such "hardness" result. One can ask if the $O(n^{\omega+0.468 \cdot t})$ running time of the algorithm of Theorem 3 can be improved. For simplicity we consider algorithms with running time $O(n^{\omega+\beta t})$ for some $0 < \beta < 1$, and show the following:

**Proposition 2.2** *Suppose that for some $\beta = 0.468 - c$ and for every $\epsilon = 1/n^t$ there is an $O(n^{\omega+\beta t})$ time algorithm that given a directed graph $G = (V, E)$ computes for any pair $u, v \in V$ a value $\hat{\delta}(u, v)$ satisfying $\delta(u, v) \leq \hat{\delta}(u, v) \leq (1 + \epsilon)\delta(u, v)$. Then there is an $O(n^{2.575-c/4})$ time APSP algorithm.*

We note that although the above proposition assumes that the running time of an alleged improved algorithm is of type $O(n^{\omega+\beta t})$, the argument gives similar hardness result for (essentially) any type of running time.

### 2.3 Organization and overview

The rest of the paper is organized as follows: in Section 3 we describe the algorithm whose running time was stated in Theorem 3. In Section 4 we apply Theorem 3 in order to obtain the main result of this paper stated in Theorem 2. In Section 4 we also prove Propositions 2.1 and 2.2. Section 5 contains some concluding remarks and open problems.

As we have already mentioned, we focus in this paper on directed graphs with small integer weights, that is, weights taken from the sets $\{1, \ldots, n^{o(1)}\}$. One can easily adapt the analysis to the case where the weights are taken from the set $\{1, \ldots, n^t\}$ but such a generalization does not require additional ideas and only complicates the analysis. Note also that the algorithms as stated above are randomized and only estimate the distances and do not produce actual paths with the estimated distances. The reason is that derandomizing these algorithms and turning them into algorithms that produce the paths, can be obtained by using known ideas that already appeared in previous papers.

## 3 The Improved Multiplicative Approximation Algorithm

In this section we consider directed graphs with positive integer weights from the set $\{0, \ldots, M\}$, where $M = n^{o(1)}$. We start with some definitions and a short overview of the approximation algorithm of [24] for computing $(1 + \epsilon)$ approximated distances in $O(n^\omega/\epsilon)$. We then proceed to present our improved $(1 + \epsilon)$ approximation algorithm whose running time is polynomially faster when $\epsilon \ll 1$.

### 3.1 Computing distance products

The computation of shortest paths lengths can be reduced to computation of *min-plus* products:

**Definition 3.1 (Min-plus products)** *The min-plus product $C = (c_{ij}) = A \star B$, where $A = (a_{ij})$ is an $\ell \times m$ matrix and $B = (b_{ij})$ is an $m \times n$ matrix is defined as follows: $c_{ij} = \min_{k=1}^{m}\{a_{ik} + b_{kj}\}$, for $1 \leq i \leq \ell$ and $1 \leq j \leq n$.*

```
algorithm dist-prod$(A, B, M)$

$a'_{ij} \leftarrow \begin{cases} (m+1)^{M-a_{ij}} & \text{if } a_{ij} \le M; \\ 0 & \text{otherwise.} \end{cases}$

$b'_{ij} \leftarrow \begin{cases} (m+1)^{M-b_{ij}} & \text{if } b_{ij} \le M; \\ 0 & \text{otherwise.} \end{cases}$

$C' \leftarrow$ fast-prod$(A', B')$

$c_{ij} \leftarrow \begin{cases} 2M - \lfloor \log_{m+1} c'_{ij} \rfloor & \text{if } c'_{ij} \le 0; \\ \infty & \text{otherwise.} \end{cases}$
return $C$
```

```
algorithm apx-dist-prod$(A, B, M, R)$
$\hat{C} \leftarrow \infty$
for $i \leftarrow \lfloor \log_2 R \rfloor$ to $\lceil \log_2 M \rceil$ do
    $A' \leftarrow$ scale$(A, 2^i, R)$
    $B' \leftarrow$ scale$(B, 2^i, R)$
    $C' \leftarrow$ dist-prod$(A', B', R)$
    $\hat{C} \leftarrow \min\{\hat{C}, \frac{2^i}{R} C'\}$
return $\hat{C}$
```

```
algorithm scale$(A, M, R)$
$a'_{ij} \leftarrow \begin{cases} \lceil \frac{R}{M} a_{ij} \rceil & \text{if } a_{ij} \le M; \\ \infty & \text{otherwise.} \end{cases}$
return $A'$
```

Figure 2: The algorithm **apx-dist-prod** and its subroutines

The min-plus product is known also as *distance product*. If $D$ is a matrix that contains the weights of the edges of a given graph then $D^n$, the $n^{th}$ power of $D$ with respect to distance product, is the distance matrix of that graph.

The next Lemma was first stated by [3], following a related idea of [23].

**Lemma 3.2** *Let $A$ be an $n \times n^r$ matrix and let $B$ be an $n^r \times n$ matrix, both with elements taken from $\{0, \dots, M\} \cup \{+\infty\}$. Then, the distance product $A \star B$ can be computed in $O(Mn^{\omega(1,r,1)})$ time.*

Based on this Lemma it is possible to use fast matrix multiplication in order to compute distance product. The algorithm **dist-prod** is given in Figure 3 (**fast-prod** is the fast matrix multiplication algorithm). It receives two matrices $A$ and $B$, where $A$ is an $n \times m$ matrix, $B$ is an $m \times n$ and $m = n^r$. The algorithm returns an $n \times n$ matrix $C$, where $C = A \star B$. The running time of the algorithm is $O(Mn^{\omega(1,r,1)})$.

## 3.2 The approximate distance product algorithm

In this section we present the $1 + \epsilon$ approximation algorithm of Zwick from [24] whose running time is $O(n^\omega/\epsilon)$. The algorithm is based on a clever scaling technique.

The main ingredient in the algorithm of [24] is the algorithm **apx-dist-prod**. The algorithm is given in Figure 3. It receives two matrices $A$ and $B$ and computes an approximation of their distance product. The algorithm **apx-dist-prod** uses the algorithm **scale** to scale the elements of $A$ and $B$ each time with a different scale factor. The algorithm **scale** gets a matrix $A$ whose elements are taken from $\{0, \dots, M\}$ and returns a matrix $A'$ whose elements are the elements of $A$ scaled and rounded up to elements taken from $\{0, \dots, R\}$.

Next, we restate a Lemma from [24] which shows that the matrix returned by **apx-dist-prod** is a good approximation of $A \star B$. Note that we adjust the Lemma to our future needs by stating it with respect to rectangular matrices and not just to square matrices as it is in [24]. For completeness the proof of this Lemma is included in the Appendix.

**Lemma 3.3** *Let $A$ be an $n \times n^r$ matrix let $B$ be an $n^r \times n$ matrix, and suppose that elements of $A$ and $B$ that are larger than $M$ are replaced with $\infty$. Set $C = A \star B$ and let $M$ and $R$ be powers of two. Let $\hat{C}$ be the matrix returned by* **apx-dist-prod**$(A, B, M, R)$*. Then, $c_{ij} \leq \hat{c}_{ij} \leq (1 + \frac{4}{R})c_{ij}$. The running time of* **apx-dist-prod**$(A, B, M, R)$ *is $O(\min\{R, M\} \cdot n^{\omega(1,r,1)} \cdot \log M)$.*

The algorithm of [24] that computes a $(1+\epsilon)$ approximation using the approximated computation of the distance product is given in the left side of Figure 3. Note that is uses **apx-dist-prod** when $A, B$ are *always* square matrices. It sets $R$ to be the first power of two that is greater or equal to $4\lceil\log_2 n\rceil / \ln(1+\epsilon)$. It follows from Lemma 3.3 that after $\lceil\log_2 n\rceil$ iterations the stretch of the elements of $F$ is at most:

$$\left(1 + \frac{4}{R}\right)^{\lceil\log_2 n\rceil} \leq \left(1 + \frac{\ln(1+\epsilon)}{\lceil\log_2 n\rceil}\right)^{\lceil\log_2 n\rceil} \leq 1 + \epsilon \ . \tag{2}$$

The total running time of the algorithm is $O(n^\omega/\epsilon)$ so if $\epsilon = 1/n^t$ the running time is $O(n^{\omega+t})$. As it may be clear by now, the main idea of this algorithm is that once we are ready to settle for approximated distances then we can reduce the order of the numbers that are involved in the computation of the distance products. However, the dependency in $\epsilon$ is still large. Notice also that the matrices $A'$ and $B'$ which **apx-dist-prod** passes to **dist-prod** are squared matrices. Thus, the main ingredient that [24] uses in order to obtain the speedup in his exact algorithm is not used here and the distance product is done between two square matrices. This is exactly the ingredient that we use in order to reduce the dependency in $\epsilon$. In particular, we use the bridging sets technique of [24] to obtain our improved algorithm. The improved algorithm is given in the right side of Figure 3.

The algorithm sets $R$ to be the first power of two that is greater or equal to $4\lceil\log_{3/2} n\rceil / \ln(1+\epsilon)$. As opposed to the approximation algorithm of [24] (and similarly to the exact algorithm of [24]) our algorithm is composed of $\lceil\log_{3/2} n\rceil$ iterations instead of $\lceil\log_2 n\rceil$. In the $\ell^{th}$ iteration, a random subset $B_\ell$ of $V$ of size $\min\{n, (9n \ln n)/(3/2)^\ell\}$ is chosen. Notice that $B_\ell = V$ in the first $\log_{3/2}(9 \ln n) = O(\log\log n)$ iterations. However, from that point onwards, the size of $B_\ell$ shrinks at each iteration by a factor of 2/3.

In the $\ell^{th}$ iteration the algorithm computes the approximate distance product between $F[V, B]$ and $F[B, V]$, that is, the rectangular matrices $F[V, B]$ and $F[B, V]$ obtained by taking only the columns of $F$ that corresponds to $B$ and the rows of $F$ that corresponds to $B$, respectively. As opposed to the exact algorithm of [24] (and similarly to the approximation algorithm of [24]) our algorithm computes the *approximation* of the distance product. Moreover, as the matrices that **apx-dist-prod** receives are rectangular then the distance product is done by exploiting this fact.

We claim that **apx-shortest-path**$_2$ computes, with high probability, $1+\epsilon$ approximations of the distances in the graph. This follows from the next Lemma, whose proof appears in the Appendix.

**Lemma 3.4** *Let $i, j \in V$. If there is a shortest path from $i$ to $j$ in $G$ that uses at most $(3/2)^\ell$ edges, then after the $\ell^{th}$ iteration, with high probability, $f_{ij} \leq (1 + \frac{4}{R})^\ell \delta(i, j)$.*

```
algorithm apx-shortest-path_1 $(D, \epsilon)$
$F \leftarrow D$
$M \leftarrow \max\{d_{ij} \mid d_{ij} \neq \infty\}$
$R \leftarrow 4\lceil \log_2 n \rceil / \ln(1+\epsilon)$
$R \leftarrow 2^{\lceil \log_2 R \rceil}$
for $\ell \leftarrow 1$ to $\lceil \log_2 n \rceil$ do
    $F' \leftarrow$ apx-dist-prod$(F, F, nM, R)$
    $F \leftarrow \min\{F, F'\}$
return $F$
```

```
algorithm apx-shortest-path_2 $(D, \epsilon)$
$F \leftarrow D$
$M \leftarrow \max\{d_{ij} \mid d_{ij} \neq \infty\}$
$R \leftarrow 4\lceil \log_{\frac{3}{2}} n \rceil / \ln(1+\epsilon)$
$R \leftarrow 2^{\lceil \log_2 R \rceil}$
for $\ell \leftarrow 1$ to $\lceil \log_{\frac{3}{2}} n \rceil$ do
    $s \leftarrow (\frac{3}{2})^{\ell}$
    $B \leftarrow$ sample$(V, 9n \ln n / s)$
    $F' \leftarrow$ apx-dist-prod$(F[*, B], F[B, *], sM, R)$
    $F \leftarrow \min\{F, F'\}$
return $F$
```

Figure 3: Zwick's approximation algorithm [24] on the left, and the improved algorithm on the right.

## 3.3 Proof of Theorem 3

We start by observing that if $\epsilon = 1/n^t$ then the algorithm **apx-shortest-path**$_2$ uses $R$ of order $O(\log n / \log(1 + 1/n^t)) = O(n^t)$ (we use the fact that $\log(1+\epsilon) \approx \epsilon$). Consider iteration $\ell$ of the algorithm and let $r$ be such that $s = (3/2)^{\ell} = n^{1-r}$, and note that this means that $|B_{\ell}| = O(n^r)$. The running time of the $\ell^{th}$ iteration of the algorithm is clearly dominated by the cost of **apx-dist-prod**. By Lemma 3.3, the cost of computing the distance product of $A$, whose size is $n \times n^r$, and $B$, whose size is $n^r \times n$, is $O(\min\{M, R\} \cdot n^{\omega(1,r,1)})$, where $M$ is the the largest element that appears in $A$ and $B$. When our improved algorithm **apx-shortest-path**$_2$ calls **apx-dist-prod** it passes two rectangular matrices $A$ and $B$, whose sizes are $n \times n^r$ and $n^r \times n$. From the assumption that the graph has small integer weights, it follows that at this stage $sM = O(n^{1-r})$. Hence, the running time of **apx-dist-prod** is $O(\min\{n^{1-r}, n^t\} \cdot n^{\omega(1,r,1)})$. As $\omega(1,r,1) + t$ is increasing with $r$ and $\omega(1,r,1) + 1 - r$ is decreasing with $r$, we infer that, ignoring poly-logarithmic factors, the worse running time of **dist-prod** is the iteration where $\omega(1,r,1) + t = \omega(1,r,1) + 1 - r$, that is, when $r = 1 - t$. Hence, the running time of the worst iteration is $O(n^{\omega(1,1-t,1)+t})$. The total running time of **apx-shortest-path**$_2$ is also $O(n^{\omega(1,1-t,1)+t})$ as it is dominated by the calls to **dist-prod** and there are only $O(\log n)$ such calls. Combining Lemma 3.4 with the same argument that is used in (2), we get that after $\lceil \log_{3/2} n \rceil$ iterations the algorithm **apx-shortest-path**$_2$ computes a $(1+\epsilon)$ approximation of the distances in the graph.

## 4 Proof of Main Result

In this section we prove Theorems 2 as well as Propositions 2.1 and 2.2. For the proof of Theorem 2 we will need the following well known fact, whose proof is included for completeness in the Appendix.

**Claim 4.1** *For every $0 \leq t \leq 1$, there is a randomized $O(n^{3-t})$ time algorithm, that given a directed graph $G$ with small positive integer weights, computes (with high probability) the shortest path from*

*$u$ to $v$ for every $u, v \in V$ that satisfy $\delta(u,v) \geq n^t$.*

**Proof of Theorem 2:** Let $0.294 \leq \ell \leq 1$ be a value to be chosen later. The key observation is that in order to compute approximations of $\delta(u,v)$ to within an additive error $\delta^p(u,v)$ it is enough to compute the exact shortest paths of $G$ of length at least $n^\ell$ (as in Claim 4.1) as well as compute a $(1+1/n^{(1-p)\ell})$-multiplicative approximation of the shortest paths of $G$ (in the sense of Theorem 3), and then take, for each pair $u, v$, the best of the two values as the estimate of the shortest paths in $G$. Indeed, if $\delta(i,j) \geq n^\ell$ then we can get the *exact* value of $\delta(u,v)$ via Claim 4.1. If $\delta(i,j) \leq n^\ell$ then the $(1+1/n^{(1-p)\ell})$-multiplicative approximation of Theorem 3 returns a value $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + \delta^p(u,v)$.

Thus, it remains to choose the value of $\ell$ that will minimize the total running time of the two algorithms. The running time of the algorithm of Claim 4.1 is $n^{3-\ell}$ and the running time of the $(1+1/n^{(1-p)\ell})$-multiplicative approximation algorithm of Theorem 3 is $n^{\omega(1,1-(1-p)\ell,1)+(1-p)\ell}$. Note that the running time of the first algorithm decreases with $\ell$ and the running time of the second algorithm increases with $\ell$, so we need to find the solution of the equation

$$\omega(1, 1-(1-p)\ell, 1) + (1-p)\ell = 3 - \ell \; . \tag{3}$$

By Theorem 1 we know that for any $0.294 \leq r \leq 1$ we have $\omega(1,r,1) = 1.843 + 0.532r$. So plugging this into the above equation we get:

$$1.843 + 0.532(1-(1-p)\ell) + (1-p)\ell = 3 - \ell \; ,$$

and this is equivalent to

$$2.376 + 0.468(1-p)\ell = 3 - \ell \; .$$

The solution of this equation is $\ell = \frac{0.624}{1.468 - 0.468p}$. Note that indeed for any $0 \leq p \leq 1$ we have $0.294 \leq \ell \leq 1$. We get that the running time of the algorithm is

$$O(n^{3-\frac{0.624}{1.468-0.468p}}) = O(n^{2.575 - \frac{p}{7.4-2.3p}})$$

■

We have commented after the statement of Theorem 2 that under the assumption $\omega = 2$ the running time of the algorithm of Theorem 2 is $O(n^{2+\frac{1-p}{2-p}})$. To see this, note that if $\omega = 2$ then equation (3) becomes $2 + (1-p)\ell = 3 - \ell$, whose solution is $\ell = \frac{1}{2-p}$, giving that the running time is indeed $O(n^{2+\frac{1-p}{2-p}})$. We end this section with the proofs of Propositions 2.1 and 2.2.

**Proof of Proposition 2.1:** Given an $n$-vertex graph $G = (V, E)$ let us construct a graph $G' = (V', E')$ on $m = n^{1+2\epsilon}$ vertices, where $G'$ is obtained from $G$ as follows: we replace every vertex $v$ with two vertices $v_{in}$ and $v_{out}$ that are connected by a path of length $n^{2\epsilon}-1$ on vertices $x_1^v, \ldots, x_{n^{2\epsilon}-2}^v$. All the above vertices are distinct, that is, the path connecting $v_{in}$ and $v_{out}$ and the one connecting $u_{in}$ and $u_{out}$ are disjoint. Hence, $G'$ indeed contains $m = n^{1+2\epsilon}$ vertices. Finally for every edge $(u, v)$ of $G$ we have an edge connecting $u_{out}$ to $v_{in}$ in $G'$. It is not difficult to see that for any $u \neq v$ we have $\delta_{G'}(u_{in}, v_{in}) = n^{2\epsilon} \cdot \delta_G(u,v)$.

Our assumption is that we can compute for all pairs $i, j \in V'$ an estimation $\hat{\delta}(i,j)$ satisfying

$$\delta_{G'}(i,j) \le \hat{\delta}_{G'}(i,j) \le \delta_{G'}(i,j) + m^{\epsilon} \le \delta_{G'}(i,j) + n^{2\epsilon} - 1 \tag{4}$$

in time $O(m^r) = O(n^{r+2r\epsilon})$, where in the rightmost inequality we have used the assumption that $\epsilon \le \frac{1}{2}$. We claim that this means that within the same time we can compute all the values $\delta_G(u,v)$. Indeed, combining the fact that $\delta_{G'}(u_{in}, v_{in}) = n^{2\epsilon} \cdot \delta_G(u,v)$ with (4) we infer that

$$n^{2\epsilon} \cdot \delta_G(u,v) \le \hat{\delta}_{G'}(u_{in}, v_{in}) \le n^{2\epsilon} \cdot \delta_G(u,v) + n^{2\epsilon} - 1 \ .$$

Observe that this means that

$$\delta_G(u,v) = \left\lfloor \frac{\hat{\delta}_{G'}(u_{in}, v_{in})}{n^{2\epsilon}} \right\rfloor ,$$

so all the exact distances can be computed with an additional cost of $O(n^2)$ time. ■

**Proof of Proposition 2.2:** Observe that if we compute estimates $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \le \hat{\delta}(u,v) \le (1+\epsilon)\delta(u,v)$, where $\epsilon = 1/n^t$, then in particular we have computed all the shortest paths of length at most $n^t$. Remember also that by Claim 4.1 we can compute all the shortest paths of length at least $n^t$ in time $n^{3-t}$. So if there is an $O(n^{\omega+(0.468-c)t})$ algorithm for computing the multiplicative estimates, then solving $3 - t = \omega + (0.468 - c)t$ we get that by choosing $t = \frac{3-\omega}{1.468-c}$ we obtain an algorithm for computing exact APSP in time $O(n^{3-\frac{0.624}{1.468-c}}) = O(n^{2.575-c/4})$. ■

## 5 Concluding Remarks and Open Problems

- A natural open problem is to improve the running time of the algorithm given in Theorem 1 for any $0 \le p \le 1$, that is, to obtain a faster transition between the exact and the multiplicative APSP algorithms.
- Our algorithm runs in time $O(n^{\omega+c})$ for any $p < 1$. Is there a $p < 1$ for which one can find additive approximation within an error $\delta^p(u,v)$ and still have running time $O(n^{\omega})$?
- The algorithm of Theorem 3 improves over the algorithm of [24] when $\epsilon \ll 1$ and the input graph has small integer weights. Is it possible to improve the result of [24] when $\epsilon \ll 1$ and still have running time that depends logarithmically on the size of the largest edge weight in the graph?

## References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, **The Design and Analysis of Computer Algorithms**, Addison-Wesley, Reading, 1974.

[2] D. Aingworth, C. Chekuri, P. Indyk and R. Motwani, Fast estimation of diameter and shortest paths (without matrix multiplication), SIAM Journal on Computing, 28 (1999), 1167-1181.

[3] N. Alon, Z. Galil and O. Margalit, On the exponent of the all pairs shortest path problem, Journal of Computer ans System Sciences, 54 (1997), 255-262. Also, Proc. of FOCS 1991.

[4] T. M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, Proc. of STOC 2007, to appear.

[5] D. Coppersmith, Rectangular matrix multiplication revisited, Journal of Complexity, 13 (1997), 42-49.

[6] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbol. Comput., 9 (1990) 251-280.

[7] T.H. Cormen, C.E. Leisserson, R.L. Rivest and C. Stein, **Introduction to Algorithms**, McGraw-Hill, 2nd ed., 2001.

[8] E.W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik, 1 (1959), 269-271.

[9] D. Dor, S. Halperin and U. Zwick, All pairs almost shortest paths, SIAM Journal on Computing, 29 (2000), 1740-1759.

[10] A. Czumaj, M. Kowaluk and A. Lingas, Faster algorithms for finding lowest common ancestors in directed acyclic graphs, manuscript, 2006.

[11] M.J. Fischer and A.R. Meyer, Boolean matrix multiplication and transitive closure, Proc. of the 12th Symposium on Switching and Automata Theory, East Lansing, Mich., (1971), 129–131.

[12] M.L. Fredman, New bounds on the complexity of the shortest path problem, SIAM Journal on Computing 5 (1976), 49-60.

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

[14] M.E. Furman, Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph, Dokl. Akad. Nauk SSSR, 11 (1970), no. 5, p. 1252.

[15] Z. Galil and O. Margalit, All pairs shortest distances for graphs with small integer length edges, Information and Computation 134 (1997), 103-139.

[16] Z. Galil and O. Margalit, All pairs shortest paths for graphs with small integer length edges, Journal of Computer and System Sciences, 54 (1997), 243-254.

[17] H.N. Gabow and R.E. Tarjan, Algorithms for two bottleneck optimization problems, Journal of Algorithms 9 (1988), 411-417.

[18] X. Huang and V.Y. Pan, Fast rectangular matrix multiplications and applications, Journal of Complexity, 14 (1998) 257-299.

[19] I. Munro, Efficient determination of the strongly connected components and the transitive closure of a graph, Unpublished manuscript, Univ. of Toronto, Toronto, Canada, 1971.

[20] R. Seidel, On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs, J. Comput. Syst. Sci. 51 (1995), 400-403.

[21] A. Shoshan and U. Zwick, All pairs shortest paths in undirected graphs with integer weights, Proc. of FOCS 1999, 605-614.

[22] M. Thorup and U. Zwick, Approximate distance oracles, Journal of the ACM, 52 (2005) 1-24.

[23] G. Yuval, An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications, Information Processing Letters 4 (1976), 155-156.

[24] U. Zwick, All-pairs shortest paths using bridging sets and rectangular matrix multiplication, Journal of the ACM 49 (2002) 289-317. Also, Proc. of FOCS 1998.

[25] U. Zwick, Exact and approximate distances in graphs - a survey, in Proc. of ESA 2001, 33-48.

## 6 Appendix

**Proof of Lemma 3.3:** If $M \leq R$ then the rounding performed by the **scale** procedure have no affect, and therefore the running time is given by the running time of **dist-prod** , which is $O(M \cdot n^{\omega(1,r,1)})$ by Lemma 3.2. Similarly, if $M \geq R$ then the calls to **scale** reduce the numbers to order $R$ and again the running time is $O(R \cdot n^{\omega(1,r,1)})$ by Lemma 3.2.

As the elements of $A$ and $B$ are rounded up in the scaling algorithm it follows that $c_{ij} \leq \hat{c}_{ij}$. In what follows we show that $\hat{c}_{ij} \leq (1 + \frac{4}{R})c_{ij}$. Let $c_{ij} = a_{ik} + b_{kj}$ and wlog let $b_{kj} \geq a_{ik}$. Assume that $2^{s-1} < b_{kj} \leq 2^s$, where $1 \leq s \leq \log_2 M$. The case that $s \leq \log_2 R$ is treated in the first iteration of **apx-dist-prod** when $i = \log_2 R$ and $\hat{c}_{ij} = c_{ij}$. Thus, we focus on the case that $\log_2 R < s \leq \log_2 M$. In the iteration where $i = s$ we have:

$$\frac{2^i \cdot a'_{ik}}{R} \leq a_{ik} + \frac{2^i}{R}, \quad \frac{2^i \cdot b'_{ik}}{R} \leq b_{ik} + \frac{2^i}{R} \ ,$$

and after the call to **dist-prod** we have:

$$\hat{c}_{ij} \leq \frac{2^i \cdot a'_{ik}}{R} + \frac{2^i \cdot b'_{ik}}{R} \leq a_{ik} + b_{kj} + \frac{2^{i+1}}{R} \leq (1 + \frac{4}{R})c_{ij} \ .$$

■

**Proof of Lemma 3.4:** The proof is by induction on $\ell$. For $\ell = 0$, the claim is obvious. Assuming the claim holds for $\ell - 1$, we prove that it also holds for $\ell$. Let $i, j \in V$ be two vertices and assume that the shortest path from $i$ to $j$ uses at most $s = (3/2)^\ell$ edges and at least $2s/3 = (3/2)^{\ell-1}$ edges, otherwise the claim regarding $i$ and $j$ already holds after the $(\ell - 1)$-st iteration from the induction hypothesis.

Let $i'$ and $j'$ be two vertices on the path from $i$ to $j$, where $i'$ is closer to $i$ and $j'$ is closer to $j$. Assume also that there are exactly $s/3$ edges between $i'$ and $j'$ and at most $s/3$ edges between $i$ and $i'$ and $j'$ and $j$.

At least one of the vertices on the path from $i'$ to $j'$ belongs to $B = B_\ell$, with high probability. The probability that this is not the case is at most $(1 - \frac{9 \ln n}{s})^{s/3} < n^{-3}$. As there are only $n^2$ pairs of vertices in the graph, and only $O(\log n)$ iterations, the probability that this condition will fail to hold even once throughout the running of the algorithm is at most $O(\log n/n)$.

Let $k \in B_\ell$ be a vertex on the path from $i'$ to $j'$. It is easy to see that both the path from $i$ to $k$ and the path from $k$ to $j$ have at most $2s/3$ edges, and thus, by the induction hypothesis we have $f_{ik} \le (1 + \frac{4}{R})^{\ell-1}\delta(i,k)$ and $f_{kj} \le (1 + \frac{4}{R})^{\ell-1}\delta(k,j)$. From Lemma 3.3 we know that after calling **apx-dist-prod** we have $f_{ij} \le (1 + \frac{4}{R})(f_{ik} + f_{kj})$. Combining it all together we get:

$$\begin{aligned}
f_{ij} &\le (1+\frac{4}{R})(f_{ik}+f_{kj}) \\
&\le (1+\frac{4}{R})((1+\frac{4}{R})^{\ell-1}\delta(i,k)+(1+\frac{4}{R})^{\ell-1}\delta(k,j)) \\
&= (1+\frac{4}{R})^{\ell}(\delta(i,k)+\delta(k,j)) \\
&= (1+\frac{4}{R})^{\ell}\delta(i,j)\ .
\end{aligned}$$

■

**Proof of Claim 4.1:** The algorithm samples a set of vertices $S$ of size $10n^{1-t}\log n$ and computes (using Dijkstra's algorithm) the single source shortest paths from all the vertices of $S$ as well as the shortest paths to all the vertices of $S$. As the running time of Dijkstra's [8, 13] algorithm is $O(m + n\log n) = O(n^2)$ we get that this step takes time $O(n^{3-t})$. Finally, for any pairs of vertices $u, v$, the estimate of the shortest paths between $u, v$ that the algorithm returns is calculated by $\min_{w\in S}(\delta(u,w) + \delta(w,v))$. Clearly this step can also be carried out in $O(n^{3-t})$ so the total running time is indeed $O(n^{3-t})$.

As for correctness, it is clearly enough to show that with high probability, if $\delta(u,v) \ge n^t$ then $S$ contains at least on vertex on (one of) the shortest paths from $i$ to $j$. Let us observe that as the weights of the graph are positive and small, if $\delta(u,v) \ge n^t$ then the shortest path from $u$ to $v$ contains $n^{t-o(1)}$ vertices. For simplicity, let us assume that there are actually $n^t$ such vertices. Fix any pair $u, v$ and a shortest path between them of length at least $n^t$. The probability that $S$ does not contain any of the vertices of this path is at most $(1 - 1/n^{1-t})^{|S|} \ll 1/n^3$. So by the union bound the probability that there is some pair $u, v$, satisfying $\delta(u,v) \ge n^t$, such that $S$ does not contain any vertex from a the shortest path connecting them, is $O(1/n)$. ■