Approximate shortest paths in weighted graphs

Raphael Yuster
Department of Mathematics
University of Haifa
Haifa 31905, Israel
E–mail: raphy@math.haifa.ac.il

Abstract

We present an approximation algorithm for the *all pairs shortest paths* (APSP) problem in weighed graphs. Our algorithm solves the APSP problem for weighted directed graphs, with real (positive or negative) weights, up to an *additive* error of ϵ. For any pair of vertices u, v, the algorithm finds a path whose length is at most $\delta(u,v) + \epsilon$. The algorithm is randomized and runs in $\tilde{O}(n^{(\omega+3)/2}) < O(n^{2.688})$ time, where n is the number of vertices and ω is the matrix multiplication exponent. The absolute weights are allowed to mildly depend upon n, being at most $n^{o(1)}$ (we note that even if the weights are constants, $\delta(u,v)$ can be linear in n, while the error requirement ϵ is a small constant independent of n). Clearly, ϵ-additive approximations generalize exact algorithms for integer weighted instances. Hence, if $\omega = 2 + o(1)$, this algorithm is as fast as any algorithm known for integer APSP in directed graphs, and is more general.

1 Introduction

Shortest paths problems are among the most fundamental algorithmic graph problems. In the shortest paths problem we are given a (possibly weighted, possibly directed) graph $G = (V, E)$ and a set $S \subset V \times V$ of pairs of vertices, and are required to find distances and shortest paths connecting the pairs in S. In the *Single Source Shortest Paths* (SSSP) problem we have $S = \{s\} \times V$ for some $s \in V$ and in the *All Pairs Shortest Paths* (APSP) problem we have $S = V \times V$.

Our main result is on the APSP problem. In its most general setting, the graph $G = (V, E)$ is a directed graph with real edge weights that can be positive or negative. Thus, the graph is associated with a weight function $w : E \to \Re$. Algorithms that manipulate real numbers as pure entities work in the *addition-comparison* model. In this model the only operations allowed on pairs of real numbers are additions and comparisons, and each of these operations has unit cost. The fastest algorithm for the general APSP problem was obtained by Chan [3]. Its runtime for n-vertex graphs is $O(n^3 \log^3 \log n / \log^2 n)$ which is almost cubic. This result improved earlier barely sub-cubic algorithms dating back to the first such algorithm by Fredman [6]. When edge weights are integers, we are no longer restricted to the addition-comparison model, and we can manipulate the bits of the integral weight. The fastest algorithm for APSP in directed graphs with integer (positive or negative) weights was obtained by Zwick [12] and runs in $\tilde{O}(n^{2+\mu})$ time[1], where $\mu < 0.575$ is a function of the exponent of rectangular matrix multiplication. It is assumed here that the weights are small (their absolute value is bounded by a constant, or, more liberally, at most $n^{o(1)}$). We note that if (as many researcher find plausible) two $n \times n$ matrices can be multiplied in $\tilde{O}(n^2)$ time, Zwick's algorithm, as well as an earlier algorithm of algorithm of Alon, Galil, and Margalit [2], runs in $\tilde{O}(n^{2.5})$ time.

The theoretical and practical importance of the APSP problem lead many researchers to look for faster algorithms that settle for *almost* shortest paths. The review article [13] contains a detailed survey of such algorithms. There are two natural ways to approximate shortest paths. The first is by *stretch-factor* algorithms. These types of algorithms guarantee an α-stretch factor. Namely, they compute a path whose length is at most $\alpha\delta(u,v)$, where $\delta(u,v)$ is the distance from u to v (assumed to be positive in this definition), and $\alpha > 1$. The second is *additive* approximations (also called *surplus* approximations). These types of algorithms guarantee an ϵ-additive error. Namely, they compute a path whose length is at most $\delta(u,v) + \epsilon$. The fastest stretch-factor algorithm for APSP was obtained by Zwick [12]. His algorithm achieves a stretch of $(1+\epsilon)$ in $\tilde{O}((n^{\omega}/\epsilon)\log(W/\epsilon))$ time. The algorithm assumes that the real weights are all positive and in the interval $[1, W]$. If $\epsilon = 1/n^t$, and $t \leq 0.706$, and $W = n^{o(1)}$, then a faster algorithm whose runtime is $\tilde{O}(n^{\omega+0.468t})$ was obtained by Roditty and Shapira [10]. Here and throughout this paper $\omega < 2.376$ denotes the matrix multiplication exponent [4].

Good additive approximations exist for *undirected* graphs. We mention two notable results. Aingworth, Chekuri, Indyk, and Motwani [1] obtained a 2-additive APSP algorithm in unweighted undirected graphs that runs in $\tilde{O}(n^{2.5})$ time. This was later improved by Dor, Halperin, and Zwick [5] who obtained a running time of $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$. They also obtain an $O(\log n)$-additive APSP algorithm that runs in almost optimal $\tilde{O}(n^2)$ time.

It is much more difficult to obtain good additive approximations for directed graphs, moreover for the general case where the weights are real, and possibly negative. One naive approach is to round (up) the real weights to the closest rational which is a multiple of some $O(1/n)$. This reduces the problem to computing exact solutions in integer weighted directed graphs where the absolute weights are $O(n)$. However, there is no truly sub-cubic algorithm known that handles such large integer weights. Likewise, one can obtain a constant additive approximation using the stretch factor approximation algorithm of Zwick mentioned above, with $\epsilon = O(1/n)$. However, this also results in a super-cubic runtime, and, in addition, does not directly apply to the case of negative edge weights.

Our main result obtains the first truly sub-cubic algorithm that achieves an ϵ-additive approximation. It applies to the most general setting, where positive and negative real weights are allowed. It is assumed that the absolute value of a weight is at most $n^{o(1)}$.

[1]Throughout this paper, $\tilde{O}(f(n))$ stands for $f(n)n^{o(1)}$.

Theorem 1.1 *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be a directed graph whose edges have real (positive or negative) weights, of absolute value at most $n^{o(1)}$. There is a randomized algorithm that computes APSP in G up to an ϵ-additive error in $\tilde{O}(n^{(\omega+3)/2})$ time.*

We assume that the input graph contains no negative cycles. In other words, for pairs u, v for which $\delta(u, v) = -\infty$, the result returned by the algorithm is meaningless. Clearly, any ϵ-additive approximation algorithm implies an exact algorithm for the case of integer weights. The converse, however, is not necessarily true. Hence, the algorithm of Theorem 1.1 is also an $\tilde{O}(n^{(\omega+3)/2})$ exact algorithm for the case of small (positive or negative) integer weights. This matches the runtime of the algorithm of Alon, Galil, and Margalit [2], but is strictly more general than it. In fact, if $\omega = 2 + o(1)$, then the runtime of the algorithm of Theorem 1.1 matches the runtime of the fastest known integer APSP algorithm of Zwick mentioned earlier. The algorithm of Theorem 1.1, though, is strictly more general as it can handle real weights, while still obtaining an ϵ-additive error.

The rest of this paper contains the proof of Theorem 1.1 in Section 2 and some concluding remarks and open problems in Section 3.

2 Proof of the main result

We construct a weight-approximated version of the input graph that is somewhat easier to work with. Let G_{int} be obtained from G be replacing each weight $w(e)$ with the weight $w_{int}(e) = \lceil \frac{2n}{\epsilon} w(e) \rceil$. Since $0 \le w_{int}(e) - \frac{2n}{\epsilon} w(e) < 1$ we know that whenever $\delta(u, v)$ is finite, then $\delta_{int}(u, v)$ is also finite and $0 \le \delta_{int}(u, v) - \frac{2n}{\epsilon}\delta(u, v) < n$. Hence, in order to obtain ϵ-additive approximations of distances in G it suffices to obtain n-additive approximations $\hat{\delta}_{int}$ of distances in G_{int}, since such an approximation implies

$$\delta(u, v) \le \frac{\epsilon}{2n} \hat{\delta}_{int}(u, v) \le \delta(u, v) + \epsilon \ .$$

It is our goal, therefore, to compute an approximation $\hat{\delta}_{int}(u, v)$ of lengths of actual paths in G_{int} that satisfies

$$\delta_{int}(u, v) \le \hat{\delta}_{int}(u, v) \le \delta_{int}(u, v) + n \ .$$

Denote by W the maximum absolute value of a weight of an edge of G, and recall that $W = n^{o(1)}$. For any two vertices u, v we have $|\delta(u, v)| < nW$ (from here onwards we will only consider pairs for which $\delta(u, v)$, and thus $\delta_{int}(u, v)$, is finite). Each weight of G_{int} is an integer in $\{-K, \ldots, K\}$ where $K = \lfloor (2W/\epsilon)n + 1 \rfloor$. Notice, though, that distances in G_{int} can be of absolute value as large as $\tilde{O}(n^2)$.

2.1 Computing distances for a given subset of pairs

An important part of our algorithm consists of computing *exact* distances in G_{int} for all pairs in a set of pairs $S \times V \cup V \times S$, where S is relatively large, but still a small part of V. In other words, we will solve SSSP in G_{int} from each vertex of S, and solve SSSP in the *edge-reversed* G_{int} from each vertex of S. It is costly, though, to compute SSSP in a graph with negative edge weights, such as G_{int}. As observed by Johnson [9], by an appropriate reweighing, we can settle for just *one* application of SSSP and then reduce the problem to SSSP in a graph with non-negative edge weights. Johnson's reweighing consists of running a single application of SSSP from a new vertex, denoted by r, connected with directed edges of weight 0 from r to each vertex of V. Goldberg [7], improving a result of Gabow and Tarjan [8], obtained an $O(m\sqrt{n}\log K)$ algorithm for the SSSP problem in directed graphs with integer edge weights of value at least $-K$ (here $m = O(n^2)$ denotes the number of edges of the graph). It follows that the reweighing of G_{int} can be obtained in $\tilde{O}(n^{2.5})$ time, in our case. The reweighing consists of assigning vertex weights $h(v)$ for each $v \in V$ (these are the distances from r to v after applying SSSP from r). The new weight, denoted by $w_{int+}(u,v)$ is just $w_{int}(u,v) + h(u) - h(v) \geq 0$. It now suffices to compute $SSSP$ from each vertex of S in G_{int+} (and similarly in its edge-reversed version). This, in turn, can be performed in $O(n^2)$ time for each vertex of S, using Dijkstra's algorithm. We therefore obtain:

Lemma 2.1 *Let $S \subset V$. There is an algorithm that computes $\delta_{int}(s,v)$ and $\delta_{int}(v,s)$ for each $s \in S$ and each $v \in V$ in $\tilde{O}(|S|n^2 + n^{2.5})$ time.*

Recall that Dijkstra's algorithm also computes shortest paths trees, hence a representation of shortest paths yielding the distances can be efficiently computed in the same time.

2.2 Approximating distances that are realized by few edges

Another important part of our algorithm consists of computing *approximate* distances in G_{int} connecting pairs of vertices for which there exists a shortest path (realizing the *exact* distance) that does not use too many edges. More precisely, for a pair of vertices u, v, let $c(u,v)$ be the smallest integer k such that $\delta_{int}(u,v)$ is realized by a path containing k edges. We describe a procedure that obtains an n-additive approximation of $\delta_{int}(u,v)$ for those pairs u, v having $c(u,v) \leq t$, where t will be chosen later. Notice that if $c(u,v) \leq t$, then $|\delta_{int}(u,v)| \leq Kt$.

Our procedure is a modified version of the stretch-factor approximation algorithm of Zwick from [12], although the latter only applies to graphs with positive edge weights. Indeed, let us first show how Zwick's algorithm can be applied, without change, in the special case where G (and hence G_{int}) only has positive edge weights. In this case, each edge of G_{int} is in $\{1, \ldots, K\}$. For a given $\gamma > 0$, the algorithm from [12] computes paths of stretch at most $1+\gamma$ in time $\tilde{O}((n^\omega/\gamma)\log K)$. If we apply it with $\gamma = \epsilon/(2Wt)$ the running time becomes, in our case, $\tilde{O}(tn^\omega)$. It computes paths of length $\hat{\delta}_{int}(u,v)$, of stretch at most $1+\gamma$, for all pairs of vertices of G_{int}. But let us examine what this means for those pairs having $c(u,v) \leq t$. For such pairs we have

$$\hat{\delta}_{int}(u,v) \leq (1+\gamma)\delta_{int}(u,v) = \delta_{int}(u,v) + \frac{\epsilon}{2Wt}\delta_{int}(u,v) \leq \delta_{int}(u,v) + \frac{\epsilon}{2Wt}Kt \leq \delta_{int}(u,v) + n \ .$$

```
algorithm scale$(A, M, R)$
The algorithm receives a matrix $A$. It returns a matrix $A'$ where elements outside the range
$\{-M, \ldots, M\}$ are changed to $\infty$ and other elements are scaled to the range $\{-R, \ldots, R\}$.
$a'_{ij} \leftarrow \begin{cases} \lceil Ra_{ij}/M \rceil & \text{if } -M \le a_{ij} \le M \\ +\infty & \text{otherwise} \end{cases}$
return $A'$
```

Figure 1: Scaling a matrix with positive and negative entries.

The stretch-factor approximation is meaningless when negative weight edges (and hence paths) exist. Still, we can modify the algorithm from [12] so that its consequence for additive approximations of those pairs for which $c(u, v) \le t$ remains almost intact. The modification will still run in $\tilde{O}(tn^{\omega})$ time, but its analysis will be somewhat more involved.

Recall that a weighted directed graph, such as G_{int}, is associated with its *distance matrix*. This matrix, denoted by D, has rows and columns indexed by V and has $D(u, v) = w_{int}(u, v)$. We assume that diagonal entries are 0 and if there is no edge from u to v then $D(u, v) = \infty$. If D_1 and D_2 are two distance matrices then the matrix $C = D_1 \star D_2$ is defined by $C(u, v) = \min_{w \in V} D_1(u, w) + D_2(w, v)$. We call C the *distance product* of D_1 and D_2. We also denote $D^2 = D \star D$ (not to be confused with the standard matrix product). Notice that $D^2(u, v)$ is the *exact* distance from u to v whenever $c(u, v) \le 2$. Similarly $D^i = D^{i-1} \star D$ (and which, by associativity, equals also $D^j \star D^{i-j}$ for any $j = 1, \ldots, i-1$) has the property that $D^i(u, v)$ is the exact distance from u to v whenever $c(u, v) \le i$.

It is therefore our goal to approximate the entries of $D^t(u, v)$. We will show how to compute a matrix B so that for any pair u, v we have $D^t(u, v) \le B(u, v) \le D^t(u, v) + n$, thereby obtaining an n-additive approximation for those pairs having $c(u, v) \le t$.

The procedure is based on exact distance products of scaled versions of the matrices to be (distance) multiplied. The simple algorithm **scale**(A, M, R), given in Figure 1, is a version of the **scale** algorithm from [12], where the only change is that it also scales matrices with negative entries. The elements of a matrix A in the range $\{-M, \ldots, M\}$ are scaled (and rounded up) to the range $\{-R, \ldots, R\}$. Other elements are replaced with infinity. Algorithm **scale** is used by algorithm **approx-dist-prod**(A, B, M, R), given in Figure 2, whose code is identical to the one from [12] (but we will apply it on matrices that also have negative entires). Algorithm **approx-dist-prod** computes an approximation of the distance product $A \star B$.

Suppose R is a power of 2 and that every finite element in A and B is in $\{-M, \ldots, M\}$. Let $C' = A \star B$ and let C be the matrix obtained by calling **approx-dist-prod**(A, B, M, R). The same

```
algorithm approx-dist-prod$(A, B, M, R)$
The algorithm receives two $n \times n$ matrices $A$ and $B$. Elements of $A$ and $B$ that are of
absolute value greater than $M$ are replaced by $\infty$. It returns an approximate distance
product $C'$ of $A$ and $B$ based on the resolution parameter $R$.

$C \leftarrow +\infty$
for $r \leftarrow \lfloor \log_2 R \rfloor$ to $\lceil \log_2 M \rceil$ do
begin
    $A' \leftarrow$ scale$(A, 2^r, R)$
    $B' \leftarrow$ scale$(B, 2^r, R)$
    $C' \leftarrow$ dist-prod$(A', B', R)$
    $C \leftarrow \min\{C\ ,\ (2^r/R)C'\}$
end
return $C$
```

Figure 2: Approximate distance products.

analysis as in [12] shows that for every i and j,

$$c'_{ij} \le c_{ij} \le c'_{ij} + 2M/R\ . \tag{1}$$

We show how to compute an approximation of D^t by repeated applications of **approx-dist-prod**. We will assume that t is a power of 2, and compute the approximation by repeated squaring, as shown in Figure 3.

Let us set R to be the smallest power of 2 greater than $4Wt(\log^2 t)/\epsilon$. Let M_s be the value of M that is set during the s'th iteration in algorithm **approx-power**. Likewise, let B_s be the matrix B during the s'th iteration in algorithm **approx-power**. The following lemma establishes bounds on M_s and on the entries of B_s.

Lemma 2.2 *In round s of algorithm* **approx-power** *we have:*

$$M_s \le 2^s K \left(\sum_{j=0}^{s} \frac{\binom{s}{j}}{R^j} \right)\ ,$$

$$D^{2^s}(i,j) \le B_s(i,j) \le D^{2^s}(i,j) + 2^s K \left(\sum_{j=1}^{s} \frac{\binom{s}{j}}{R^j} \right)\ .$$

Proof: The proof is by induction on s. For $s = 1$, note that in the first call to **approx-dist-prod**, each finite element of $B = D$ has a value in $\{-K, \ldots, K\}$, and $K = M$. Thus, by Equation (1),

```
algorithm approx-power$(D, t)$

$B \leftarrow D$
$M \leftarrow \max\{|b_{ij}| \ : \ |b_{ij}| \neq \infty\}$
$R \leftarrow 4Mt(\log^2 t)/\epsilon$
$R \leftarrow 2^{\lceil \log_2 R \rceil}$
for $s \leftarrow 1$ to $\lceil \log_2 t \rceil$ do
    $B \leftarrow$ approx-dist-prod$(B, B, M, R)$
    $M \leftarrow \max\{|b_{ij}| \ : \ |b_{ij}| \neq \infty\}$
return $B$
```

Figure 3: Computing an approximation of D^t.

in the resulting B_1 we have $D^2(i,j) \leq B_1(i,j) \leq D^2(i,j) + 2K/R$. The smallest value in B_1 is at least $-2K$ and the largest value is at most $2K + 2K/R$. Hence, $M_1 \leq 2K + 2K/R$. We now assume that the lemma holds for $s-1$ and prove it for s. In the call to **approx-dist-prod** during iteration s, each finite element of $B = B_{s-1}$ has a value in $\{-M_{s-1}, \ldots, M_{s-1}\}$, and $M = M_{s-1}$. Thus, in resulting B_s we have

$$B_{s-1}^2(i,j) \leq B_s(i,j) \leq B_{s-1}^2(i,j) + 2M_{s-1}/R \ .$$

On the other hand, by the induction hypothesis,

$$D^{2^{s-1}}(i,j) \leq B_{s-1}(i,j) \leq D^{2^{s-1}}(i,j) + 2^{s-1}K\left(\sum_{j=1}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right) \ ,$$

from which it follows that

$$D^{2^s}(i,j) \leq B_{s-1}^2(i,j) \leq D^{2^s}(i,j) + 2^sK\left(\sum_{j=1}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right) \ .$$

Therefore,

$$D^{2^s}(i,j) \leq B_s(i,j) \leq D^{2^s}(i,j) + 2^sK\left(\sum_{j=1}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right) + 2M_{s-1}/R \leq$$

$$D^{2^s}(i,j) + 2^sK\left(\sum_{j=1}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right) + \frac{2}{R}2^{s-1}K\left(\sum_{j=0}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right) = D^{2^s}(i,j) + 2^sK\left(\sum_{j=1}^{s} \frac{\binom{s}{j}}{R^j}\right) \ .$$

Similarly,

$$M_s \leq 2M_{s-1} + 2M_{s-1}/R = 2M_{s-1}(1+1/R) \leq 2^sK\left(\sum_{j=0}^{s-1} \frac{\binom{s-1}{j}}{R^j}\right)(1+1/R) = 2^sK\left(\sum_{j=0}^{s} \frac{\binom{s}{j}}{R^j}\right) \ .$$

■

Lemma 2.3 *The matrix B computed by* **approx-power** *satisfies $D^t(u,v) \le B(u,v) \le D^t(u,v) + n$ for all pairs u,v. The running time of* **approx-power** *is $\tilde{O}(n^\omega t)$.*

Proof: By our choice of R as the smallest power of 2 greater than $4Wt(\log^2 t)/\epsilon$ we have, from Lemma 2.2, that for $s = \log t$,

$$D^t(u,v) \le B(u,v) \le D^t(u,v) + tK\left(\sum_{j=1}^{s} \frac{\binom{s}{j}}{R^j}\right) <$$

$$D^t(u,v) + tK\log^2 t/R < D^t(u,v) + K\epsilon/(4W) < D^t(u,v) + n \ .$$

The running time follows from the fact that we apply a polylogarithmic number of calls to **dist-prod**, which, in turn, runs in $\tilde{O}(Rn^\omega) = \tilde{O}(n^\omega t)$, as shown by Yuval [11] (see also [12]).

2.3 Completing the proof of Theorem 1.1

We set t to be the smallest power of two larger than $n^{(3-\omega)/2}$.

We first apply the algorithm of Lemma 2.3 which computes a value $B(u,v)$ for each pair of vertices u,v. By Lemma 2.3, this requires $\tilde{O}(n^{(\omega+3)/2})$ time. Next, we choose a random subset $S \subset V$ consisting of $4n\ln n/t$ vertices. We apply the algorithm of Lemma 2.1 and obtain $\delta_{int}(s,v)$ and $\delta_{int}(v,s)$ for each $s \in S$ and each $v \in V$. By Lemma 2.1, this requires $\tilde{O}(n^{(\omega+3)/2})$ time. For each pair of vertices u,v let

$$\ell(u,v) = \min_{s \in S} \delta_{int}(u,s) + \delta_{int}(s,v) \ .$$

Notice that the runtime required for computing all the $\ell(u,v)$ is also $\tilde{O}(n^{(\omega+3)/2})$.

Our algorithm will return, for each pair u,v, the value

$$\hat{\delta}_{int}(u,v) = \min\{B(u,v)\ ,\ \ell(u,v)\} \ .$$

We claim that with very high probability, $\delta_{int}(u,v) \le \hat{\delta}_{int}(u,v) \le \delta_{int}(u,v) + n$. Consider a pair of vertices u,v for which $c(u,v) > t$. Let $p(u,v)$ be some path with $c(u,v)$ edges, realizing $\delta_{int}(u,v)$. As S was chosen randomly, the probability that none of the (at least t) internal vertex of $p(u,v)$ belongs to S is at most $(1-(4\ln n)/t)^t < 1/n^3$. Hence, with probability at least $1-1/n$, S has the property that for each pair u,v with $c(u,v) > t$, there is some path with $c(u,v)$ edges realizing $\delta_{int}(u,v)$, containing an internal vertex from S. Assuming S has this property, we have, using the fact that sub-paths of shortest paths are shortest paths, that $\ell(u,v) = \delta_{int}(u,v)$ whenever $c(u,v) > t$. On the other hand, for pairs u,v with $c(u,v) \le t$, we have, by Lemma 2.3, that $D^t(u,v) \le B(u,v) \le D^t(u,v) + n$. But since $c(u,v) \le t$, we have $D^t(u,v) = \delta_{int}(u,v)$, and hence $\delta_{int}(u,v) \le B(u,v) \le \delta_{int}(u,v) + n$.

We have proved that, with probability at least $1-1/n$, for all pairs u,v for which $\delta_{int}(u,v)$ is finite, we have $\delta_{int}(u,v) \le \hat{\delta}_{int}(u,v) \le \delta_{int}(u,v) + n$. Also notice that our approximation $\hat{\delta}_{int}(u,v)$ represents an actual path having this length. A data structure representing the actual paths is also easily obtained. For $\ell(u, v)$ this is obtained by the shortest path trees constructed by Dijkstra's algorithm (as noted in the paragraph following Lemma 2.1). For $B(u, v)$ this is obtained using the witnesses of **dist-prod**, as shown in [12]. ■

3 Concluding remarks

The algorithm of Theorem 1.1, running in $\tilde{O}(n^{(\omega+3)/2})$, is asymptotically as fast as any known algorithm for *integer* APSP, if $\omega = 2 + o(1)$. Still, for the current known upper bound of ω, Zwick's algorithm for integer APSP runs faster, in $\tilde{O}(n^{2+1/(4-\omega)})$ time (and even slightly faster using rectangular matrix multiplication). Although it is of some interest to improve the runtime of the algorithm of Theorem 1.1 to match the runtime of Zwick's algorithm, the major task is, clearly, to break the $\tilde{O}(n^{2.5})$ barrier assuming $\omega = 2 + o(1)$.

References

[1] 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), pp. 1167–1181.

[2] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, Journal of Computer and System Sciences 54 (1997), pp. 255–262.

[3] T. M. Chan, *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*, Proceedings of the 39^{th} ACM Symposium on Theory of Computing (STOC), ACM Press (2007), pp. 590–598.

[4] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation 9 (1990), pp. 251–280.

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

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

[7] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, SIAM Journal on Computing 24 (1995), pp. 494–504.

[8] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for general graph matching problems*, Journal of the ACM 38 (1991), pp. 815–853.

[9] D. B. Johnson, *Efficient algorithms for shortest paths in sparse graphs*, Journal of the ACM 24 (1977), 1–13.

[10] L. Roditty and A. Shapira, *All-Pairs Shortest Paths with a sublinear additive error*, Proceedings of the 35^{th} International Colloquium on Automata, Languages, and Programming (ICALP), LNCS (2008), pp. 622–633.

[11] G. Yuval, *An algorithm for finding all shortest paths using* $N^{2.81}$ *infinite-precision multiplications*, Information Processing Letters 4 (‘976), pp. 155–156.

[12] U. Zwick, *All-pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM 49 (2002), pp. 289–317.

[13] U. Zwick, *Exact and approximate distances in graphs - a survey*, Proceedings of the 9^{th} Annual European Symposium on Algorithms (ESA), LNCS (2001), pp. 33–48.