Answering distance queries in directed graphs using fast matrix multiplication

Raphael Yuster [*] *Uri Zwick* [†]

Abstract

Let $G = (V, E, w)$ be a weighted directed graph, where $w : E \to \{-M, \ldots, 0, \ldots, M\}$. We show that G can be preprocessed in $\tilde{O}(Mn^\omega)$ time, where $\omega < 2.376$ is the exponent of fast matrix multiplication, such that subsequently, each distance $\delta(u, v)$ in the graph, where $u, v \in V$, can be computed exactly in $O(n)$ time. We also present a tradeoff between the processing time and the query answering time.

As a very special case, we obtain an $\tilde{O}(Mn^\omega)$ time algorithm for the *Single Source Shortest Paths* (SSSP) problem for directed graphs with integer weights of absolute value at most M. For sufficiently dense graphs, with small enough edge weights, this improves upon the $O(m\sqrt{n}\log M)$ time algorithm of Goldberg. We note that even the case $M = 1$, in which all the edge weights are in $\{-1, 0, +1\}$, is an interesting case for which no improvement over Goldberg's $O(m\sqrt{n})$ algorithm was known. Our new $\tilde{O}(n^\omega)$ algorithm is faster whenever $m > n^{\omega-1/2} \simeq n^{1.876}$.

1 Introduction

Shortest paths problems are among the most fundamental algorithmic graph problems. Two of the most studied shortest paths problems are the *Single Source Shortest Paths* (SSSP) problem and the *All Pairs Shortest Paths* (APSP) problem. In the SSSP problem, we are given a graph $G = (V, E)$ and a source vertex $s \in V$ and are required to find distances and shortest paths from s to all the other vertices in the graph. In the APSP problem we are asked to find distances and shortest paths between any two vertices in the graph.

A version of the shortest paths problem that generalizes both the single-source and the all-pairs variants of the shortest paths problem, is the following: Given a graph $G = (V, E)$, preprocess it such that each distance query in the graph can subsequently be answered fairly quickly. We are interested, of course, in preprocessing times that are faster than the known APSP algorithms, and query times that are faster than the known SSSP algorithms. This variant of the shortest paths problem is, in several respects, more interesting that both the APSP and SSSP problems, as in practice, we are usually interested in distances from more than just one source vertex, but are rarely interested in all the distances in the graph. Furthermore, we may not always know in advance which distances in the graph will be of interest to us.

In this paper we show that any *weighted* and *directed* graph $G = (V, E)$ on n vertices, with integer edge weights of absolute value at most M, can be preprocessed in $\tilde{O}(Mn^\omega)$ time, where $\omega < 2.376$ is the exponent of matrix multiplication, yielding a data structure of size $O(n^2)$ that can answer any distance query in the graph, exactly, in $O(n)$ time. We, in fact, present a family of algorithms that presents a tradeoff between the required preprocessing and query answering times.

[*]Department of Mathematics, University of Haifa, Haifa 31905, Israel. E–mail: raphy@research.haifa.ac.il

[†]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: zwick@cs.tau.ac.il

The preprocessing/query answering variant of the shortest paths problem was previously considered by Thorup and Zwick [TZ05]. For every fixed integer $k \geq 1$, they obtain an algorithm that preprocesses a weighted *undirected* graph in $O(mn^{1/k})$ expected time, generating a data structure of size $O(n^{1+1/k})$, using which each distance query can be answered, approximately, in $O(1)$ time. The *stretch* of the approximate distance returned is at most $2k-1$. The *approximate distance oracles* of [TZ05] are very different, however, from the algorithms presented here. The oracles of [TZ05] use sub-quadratic space and can answer queries in constant time, but they work only for *undirected* graphs, and the answers that they provide are *approximate*. It is shown in [TZ05], using a fairly simple counting argument, that for directed graphs, any data structure that can produce constant stretch approximations of all distances must use $\Omega(n^2)$ space.

The new preprocessing and query answering algorithms are obtained by careful modifications the APSP algorithm of Zwick [Zwi02] for *directed* graphs. The running time of this algorithm, on graphs with integer edge weights of absolute value at most M, is $O(M^{0.68}n^{2.58})$. The $\tilde{O}(Mn^{\omega})$ time preprocessing algorithm is faster whenever $M < n^{3-\omega}$, which is the interesting range of the parameter M, as when $M > n^{3-\omega}$, both algorithms require $\Omega(n^3)$ time. The fastest algorithm for solving the APSP problem for *undirected* graphs with non-negative integer edge weights of value at most M, obtained by Shoshan and Zwick [SZ99], runs in $\tilde{O}(Mn^{\omega})$ time, exactly the same as the running time of the new preprocessing algorithm.

The new algorithm can be used to answer any set of $O(Mn^{\omega-1})$ distance queries in $\tilde{O}(Mn^{\omega} + Mn^{\omega-1} \cdot n) = \tilde{O}(Mn^{\omega})$ time. As a very special case, we get that it can be used to solve the SSSP problem in $\tilde{O}(Mn^{\omega})$ time! In fact, it can solve the SSSP problem from $O(Mn^{\omega-2})$ sources within the same time bound.

The SSSP problem for directed graphs with non-negative edge weights can be solved in $O(m + n \log n)$ time (Dijkstra [Dij59], Fredman and Tarjan [FT87]), where m is the number of edges in the graph. For undirected graphs with integer edge weights, an $O(m)$ time algorithm was obtained by Thorup [Tho99]. The SSSP problem becomes much harder, however, when negative edge weights are allowed. The classical Bellman-Ford algorithm (Belman [Bel58]) solves the problem in $O(mn)$ time. Goldberg [Gol95], slightly improving a result of Gabow and Tarjan [GT91], obtained an $O(m\sqrt{n} \log N)$ algorithm for the SSSP problem in directed graphs with integer edge weights of value at least $-N$. No improved SSSP algorithms for graphs with negative edge weights were obtained for over a decade.

For certain values of the parameters M and m, our new $\tilde{O}(Mn^{\omega})$ time algorithm for the SSSP problem for directed graphs with integer edge weights of absolute value at most M improves on the $O(m\sqrt{n} \log M)$ running time of Goldberg's algorithm [Gol95]. This happens when the input graph is dense enough (m is large), and the edge weights are small enough (M is small). We note that even the case $M = 1$, in which all the edge weights are in $\{-1, 0, +1\}$, is an interesting case for which no improvement over Goldberg's $O(m\sqrt{n})$ algorithm was known. Our new $\tilde{O}(n^{\omega})$ algorithm is faster whenever $m > n^{\omega-1/2} \simeq n^{1.876}$.

This is the first time in which fast matrix multiplication algorithms are used to obtain an improved algorithm for the *single-source* shortest paths problem, and not for the *all-pairs* shortest paths problem.

The rest of this extended abstract is organized as follows. In the next section we summarize the facts we need about fast matrix multiplication algorithms and their use in shortest paths algorithms. In Section 3 we shortly review the APSP algorithm of Zwick [Zwi02]. In Section 4 we present our new preprocessing and query answering algorithms. In Section 5 we present a family of algorithms that exhibits a tradeoff between the preprocessing and the query answering times. In Section 6 we briefly mention an idea that can be used to speed-up the answering of structured collections of queries. In Section 7 we describe some applications of our results, including the improved SSSP algorithm. Most of the algorithms in the paper are randomized. We can, however, derandomize them, with essentially no loss in efficiency. Some of the ideas used in the derandomization of the algorithms are described in Section 8. The full details will appear in the full version of the paper. We end, in Section 9, with some concluding remarks and open problems.

Assumptions and conventions

Throughout the paper we assume that the input graphs supplied to our algorithms do not contain *negative cycles*. It is straightforward to modify our algorithms so that they could detect such cycles. For simplicity, we concentrate on the computation of *distances*. It is possible to extend our algorithms so that they would also return a concise representation of the shortest paths, though the details are not always trivial. The full details will be given in the full version of the paper. Also, as mentioned above, most of the algorithms presented in this extended abstract are randomized. We can obtain deterministic algorithms with essentially the same running time, though they are considerably more complicated than the randomized algorithms given here. Some details regarding the derandomization of our algorithms are given in Section 8. The full details will again be given in the full version of the paper.

2 Matrix multiplication and shortest paths

2.1 Algebraic products

Definition 2.1 (Matrix multiplication) *Let $A = (a_{ij})$ be an $\ell \times m$ matrix, and let $B = (b_{ij})$ be a $m \times n$ matrix. Their $\ell \times n$ product $C = (c_{ij}) = AB$ is defined as follows: $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$, for $1 \leq i \leq \ell$ and $1 \leq j \leq n$. Let $M(\ell, m, n)$ be the minimal number of algebraic operations needed to compute the product of an $\ell \times m$ matrix by an $m \times n$ matrix.*

Definition 2.2 (Matrix multiplication exponents) *Let $\omega(r, s, t)$ be the infimum of all the exponents ω' for which $M(n^r, n^s, n^t) = O(n^{\omega'})$. We let $\omega = \omega(1, 1, 1)$ be the exponent of square matrix multiplication.*

Theorem 2.3 (Coppersmith and Winograd [CW90]) $\omega < 2.376$.

Theorem 2.4 (Coppersmith [Cop97]) *If $r \leq 0.294$, then $\omega(1, r, 1) = 2$.*

Definition 2.5 *Let α be the supremum over all constants r for which $\omega(1, r, 1) = 2$.*

Coppersmith's result [Cop97] implies that $\alpha > 0.294$. The following lemmas are obtained by decomposing a given matrix product into smaller products. (See, e.g., Huang and Pan [HP98].)

Lemma 2.6 $\omega(1, r, 1) \leq \begin{cases} 2 & \text{if } 0 \leq r \leq \alpha, \\ 2 + \frac{\omega - 2}{1 - \alpha}(r - \alpha) & \text{if } \alpha \leq r \leq 1. \end{cases}$

Lemma 2.7 $\omega(1, r, r) = \omega(r, r, 1) \leq 1 + (\omega - 1)r$, *for* $0 \leq r \leq 1$.

2.2 Distance products

The computation of distances is easily reduced to the computation of *distance*, or *min-plus*, products:

Definition 2.8 (Distance products) *Let $A = (a_{ij})$ be an $\ell \times m$ matrix, and let $B = (b_{ij})$ be a $m \times n$ matrix. Their $\ell \times n$* distance product *$C = (c_{ij}) = A \star B$ 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 short-path(W_{n×n}, M)
V ← {1, 2, ..., n}
D ← W
for ℓ ← 1 to ⌈log_{3/2} n⌉
    B ← sample(V, (9n ln n)/(3/2)^ℓ)
    D ⟸^min ⟨D[V,B]⟩_{sM} ⋆ ⟨D[B,V]⟩_{sM}
endfor
```

```
algorithm preprocess(W_{n×n}, M)
V ← {1, 2, ..., n}
B ← V ; D ← W
for ℓ ← 1 to ⌈log_{3/2} n⌉
    B ← sample(B, (9n ln n)/(3/2)^ℓ)
    D[V,B] ⟸^min ⟨D[V,B]⟩_{sM} ⋆ ⟨D[B,B]⟩_{sM}
    D[B,V] ⟸^min ⟨D[B,B]⟩_{sM} ⋆ ⟨D[B,V]⟩_{sM}
endfor
```

Figure 1: The old shortest paths algorithm and the new preprocessing algorithm.

It is easy to see that if W is an $n \times n$ matrix containing the edge weights of an n-vertex graph, then W^n, the n-th power of W with respect to distance products, is the distance matrix of the graph.

As the fast algebraic matrix multiplication algorithms rely heavily on the ability to perform *subtractions*, they cannot be used directly for the computation of distance products. Nevertheless, we can get the following result, first stated by Alon *et al.* [AGM97], following a related idea of Yuval [Yuv76].

Lemma 2.9 *Let A be an $n^r \times n^s$ matrix and let B be an $n^s \times n^t$ matrix, both with elements taken from $\{-M, \ldots, 0, \ldots, M\} \cup \{+\infty\}$. Then, the distance product $A \star B$ can be computed in $\tilde{O}(Mn^{\omega(r,s,t)})$ time.*

Reducing the dependency on M in the above running time is a major open problem. Any improvement here will immediately improve the results of [SZ99], [Zwi02] and the results of this paper.

3 The APSP algorithm of Zwick

The new preprocessing and query answering algorithms rely heavily on the APSP algorithm of Zwick [Zwi02]. A concise description of this algorithm is given in Figure 1. The input to the algorithm is an $n \times n$ matrix $W = (w_{ij})$ containing the edge weights of a directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$. More specifically, if $(i, j) \in E$, then w_{ij} is the weight of the edge (i, j), and if $(i, j) \notin E$, then $w_{ij} = +\infty$. The edge weights of G, and hence the finite elements in the matrix W are assumed to be integers taken from the interval $[-M, M]$.

The algorithm is extremely simple. It is composed of $\lceil \log_{3/2} n \rceil$ iterations. In the ℓ-th iteration, the algorithm chooses a random subset B_ℓ of V of size $\min\{n, (9n \ln n)/(3/2)^\ell\}$. Thus, in the first $\log_{3/2}(9 \ln n) = O(\log \log n)$ iterations, we have $B_\ell = V$. ¿From that point onwards, the size of B_ℓ shrinks at each iteration by a factor of 2/3. (Note, however, that B_ℓ is not necessarily a subset of $B_{\ell-1}$.)

The main operation carried out during the ℓ-th iteration of the algorithm is the command

$$D \stackrel{\min}{\longleftarrow} \langle D[V, B] \rangle_{sM} \star \langle D[B, V] \rangle_{sM} .$$

This involves the computation of the distance product $\langle D[V, B] \rangle_{sM} \star \langle D[B, V] \rangle_{sM}$, see the definitions below and the lefthand size of Figure 2, and then the computation of the element-wise minimum between D and

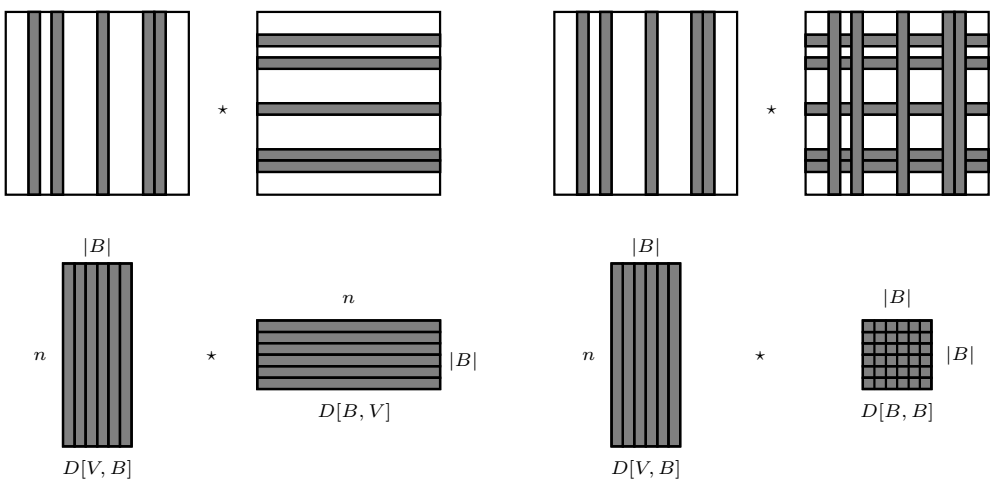

Figure 2: The distance products performed by **short-path** and **preprocess**.

the result of the product. We next define each one of these steps in more detail. Although the description is a bit technical, the operations are quite simple.

Definition 3.1 (Truncation) *If $D = (d_{ij})$ is a matrix and t is a threshold, we let $\langle D \rangle_t = (d'_{ij})$ be the matrix obtained from D by replacing all the entries that are larger than t by $+\infty$. In other words, $d'_{ij} = d_{ij}$, if $d_{ij} \leq t$, and $d'_{ij} = +\infty$, otherwise.*

Definition 3.2 (Selection) *Let D be an $n \times n$ matrix. Let $A, B \subseteq \{1, 2, \ldots, n\}$ be two subsets of indices. We let $D[A, B]$ be the $|A| \times |B|$ submatrix of D obtained by selecting the elements of D that belong to rows whose indices belong to A and to columns whose indices belong to B. More precisely, if $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and $B = \{b_1, b_2, \ldots, b_{|B|}\}$, where $1 \leq a_1 < a_2 < \cdots < a_{|A|} \leq n$ and $1 \leq b_1 < b_2 < \cdots < b_{|B|} \leq n$, and $D[A, B] = (d'_{ij})$, then $d'_{ij} = d_{a_i, b_j}$, for $1 \leq i \leq |A|$ and $1 \leq j \leq |B|$.*

Definition 3.3 (Min assignment) *Let $D = (d_{ij})$ and $D' = (d'_{ij})$ be two $n \times n$ matrices, and let $A, B \subseteq \{1, 2, \ldots, n\}$ be two subsets of indices. By $D[A, B] \xleftarrow{\min} D'[A, B]$ we denote the operation $d_{ij} \leftarrow \min\{d_{ij}, d'_{ij}\}$, for every $i \in A$ and $j \in B$.*

We claim that, with high probability, the matrix D computed by **short-path** is the distance matrix of the input graph. In other words, $d_{ij} = \delta(i, j)$, for every $i, j \in V$, where $\delta(i, j)$ denotes the distance from i to j in the graph. The claim follows from the following Lemma whose proof is taken from [Zwi02]:

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 ℓ-th iteration of the processing algorithm, with high probability, we have $d_{ij} = \delta(i, j)$.*

Proof: The proof is by induction on ℓ. For $\ell = 0$, the claim is obvious. We suppose, therefore, that the claim holds for $\ell - 1$ and show that it also holds for ℓ. Let $i, j \in V$ be two vertices and let p be a shortest path from i to j that uses at most $s = (3/2)^\ell$ edges. We may assume that p contains at least $2s/3 = (3/2)^{\ell-1}$ edges, as otherwise it follows from the induction hypothesis that the claim regarding i and j already holds after the $(\ell - 1)$-st iteration and thus continues to hold after the ℓ-th iteration. (As the matrix D is only updated using min assignments, the elements d_{ij} never increase.)

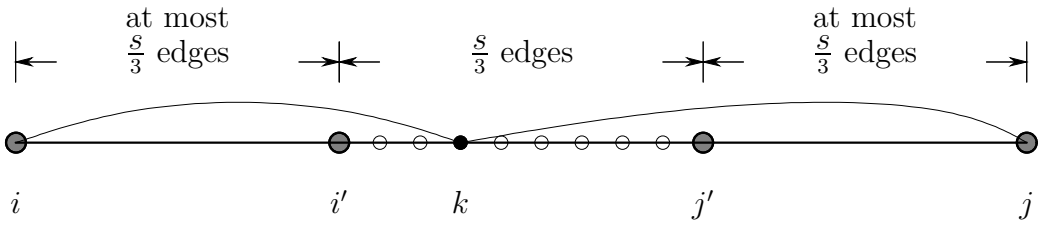

Figure 3: The correctness proof of algorithms **short-path** and **preprocess**.

If a and b are two vertices on p, we let $p(a,b)$ denote the portion of p that connects a and b. Let i' and j' be two vertices on p such that $p(i,i')$ and $p(j',j)$ each contain at most $s/3$ edges, and such that $p(i',j')$ contains exactly $s/3$ edges. (See Figure 3.) With high probability, at least one of the vertices on $p(i',j')$ belongs to $B = B_\ell$. Indeed, the probability that this does not hold 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 $O(\log n/n)$. (This probability can be further decreased by replacing 9 by a larger constant.)

Let $k \in B_\ell$ be a vertex on $p(i',j')$. As we just saw, such a vertex exists with high probability. Now, $p(i,k)$ and $p(k,j)$ are shortest paths that contain at most $2s/3$ edges. Thus, by the induction hypothesis, after the $(\ell-1)$-st iteration of the algorithm we have $d_{ik} = \delta(i,k)$ and $d_{kj} = \delta(k,j)$. Also, $\delta_{ik}, \delta_{kj} \leq (2s/3)M$. Thus, after the command $D \stackrel{\min}{\longleftarrow} \langle D[V,B] \rangle_{sM} \star \langle D[B,V] \rangle_{sM}$, we have $d_{ij} \leq d_{ik} + d_{kj} \leq \delta(i,k) + \delta(k,j) = \delta(i,j)$. It is easy to see that at all stages of the algorithm we have $d_{ij} \geq \delta(i,j)$. Thus, after the ℓ-th iteration we have $d_{ij} = \delta(i,j)$, with high probability, as required. □

We next analyze the complexity of the algorithm:

Lemma 3.5 *The running time of algorithm* **short-path**$(W_{n\times n}, M)$*, with* $M = n^t$*, is* $\tilde{O}(n^{2+\mu(t)})$*, where* $\mu(t) \geq 0$ *satisfies the equation* $\omega(1,\mu(t),1) = 1 + 2\mu(t) - t$*. Furthermore,* $\mu(t) \leq \frac{1-\alpha\beta+t}{2-\beta}$*, where* $\beta = \frac{\omega-2}{1-\alpha}$*, and thus the running time of the algorithm is* $\tilde{O}(M^{\frac{1}{2-\beta}} n^{2+\frac{1-\alpha\beta}{2-\beta}}) = O(M^{0.68} n^{2.58})$.

Proof: The most time consuming operation in the ℓ-th iteration is the computation of the distance product $\langle D[V,B] \rangle_{sM} \star \langle D[B,V] \rangle_{sM}$. As $|B_\ell| = \tilde{O}(n/s)$, where $s = (3/2)^\ell$, the cost of the product, by Lemma 2.9, is $\tilde{O}(sMn^{\omega(1,r,1)}) = \tilde{O}(n^{1-r}Mn^{\omega(1,r,1)})$, where $n^r = n/s$ and $0 \leq r \leq 1$. Alternatively, the product can also be computed using the naive algorithm in $\tilde{O}(n^3/s) = \tilde{O}(n^{2+r})$.

Let $M = n^t$ and let $\mu(t) \geq 0$ be the solution to the equation $\omega(1,\mu(t),1) = 1 + 2\mu(t) - t$. It is easy to see that when $r \geq \mu(t)$, the $\tilde{O}(n^{1-r}Mn^{\omega(1,r,1)})$ algorithm is faster, while when $r \leq \mu(t)$, the naive $\tilde{O}(n^{2+r})$ algorithm is faster. The cost of the product, for any $0 \leq r \leq 1$, is therefore, at most $\tilde{O}(n^{2+\mu(t)})$. As there are only $O(\log n)$ iterations, this is also the cost of the whole algorithm.

By Lemma 2.6, $\omega(1,r,1) \leq 2 + \beta(r-\alpha)$, for $r \geq \alpha$, where $\beta = \frac{\omega-2}{1-\alpha}$. It is easy to check that this implies that $\mu(t) \leq \frac{1-\alpha\beta+t}{2-\beta}$. The running time of the algorithm is, therefore, $\tilde{O}(M^{\frac{1}{2-\beta}} n^{2+\frac{1-\alpha\beta}{2-\beta}}) = O(M^{0.68} n^{2.58})$, as required. □

Combining these two lemmas, we get:

Theorem 3.6 *Let* $G = (V,E)$ *be a weighted directed graph on* n *vertices with integer edge weights of absolute value at most* $M = n^t$*. Let* W *be an* $n \times n$ *matrix containing the weights associated with the edges of* G*. Then, with high probability, the* $n \times n$ *matrix* D *computed by* **short-path**$(W_{n\times n}, M)$ *is the distance matrix of* G*. The running time of the algorithm is* $\tilde{O}(n^{2+\mu(t)}) = O(M^{0.68} n^{2.58})$.

```
algorithm dist(D_{n×n}, i, j)
return D[{i}, V] ⋆ D[V, {j}]
```

Figure 4: The query answering algorithm.

4 The new preprocessing and query answering algorithms

The new preprocessing algorithm is given on the righthand side of Figure 1. As can be seen it is fairly similar to the APSP algorithm of Zwick [Zwi02]. There are basically two differences between the two algorithms. The first difference is in the way the samples B_ℓ are chosen. In the APSP algorithm of Zwick [Zwi02], the ℓ-th sample B_ℓ is a random subset of V of size $\min\{n, (9n \ln n)/(3/2)^\ell\}$. In the new preprocessing algorithm, we start with $B_0 = V$ and then take B_ℓ to be a random subset of $B_{\ell-1}$, and not of V, of size $\min\{n, (9n \ln n)/(3/2)^\ell\}$. We thus have $V = B_0 \supseteq B_1 \supseteq \cdots \supseteq B_\ell \supseteq \cdots$. This simple modification is *crucial* for the operation of the preprocessing algorithm. (It also makes the derandomization of the algorithm a bit harder, as briefly discussed in Section 8.)

The second and more major difference between the two algorithms, which accounts for the faster preprocessing time, is that the distance product $\langle D[V,B]\rangle_{sM} \star \langle D[B,V]\rangle_{sM}$ is replaced by the two much smaller distance products $\langle D[V,B]\rangle_{sM} \star \langle D[B,B]\ \rangle_{sM}$ and $\langle D[B,B]\rangle_{sM} \star \langle D[B,V]\ \rangle_{sM}$. (See Figure 2 for a comparison of these products.)

The query answering algorithm, given in Figure 4 is extremely simple. It performs a distance product between the i-th row and the j-th column of the matrix D returned by the preprocessing algorithm. The query time is thus clearly $O(n)$. The correctness of the new preprocessing and query answering algorithms follows from the following Lemma:

Lemma 4.1 *If $i \in B_\ell$ or $j \in B_\ell$, and there is a shortest paths from i to j in G that uses at most $(3/2)^\ell$ edges, then after the ℓ-th iteration of the preprocessing algorithm, with high probability, we have $d_{ij} = \delta(i,j)$.*

Proof: The proof is by induction on ℓ and is a simple modification of the proof of Lemma 3.4. For $\ell = 0$, the claim is obvious. We suppose, therefore, that the claim holds for $\ell - 1$ and show that it also holds for ℓ. Let $i, j \in V$ be two vertices connected in G by a shortest path p that uses at most $s = (3/2)^\ell$ edges. Suppose that $i \in B_\ell$. The case $j \in B_\ell$ is analogous. As in the proof of Lemma 3.4, let i' and j' be two vertices on p such that $p(i,i')$ and $p(j',j)$ each contain at most $s/3$ edges, and such that $p(i',j')$ contains exactly $s/3$ edges. Again, with high probability, there exists a vertex $k \in B_\ell$ on $p(i',j')$. (Note that B_ℓ is still a uniformly random subset of V of size $\min\{n, (9n \ln n)/(3/2)^\ell\}$, hence the probability calculation carried out in Lemma 3.4 is still valid.)

Now, $p(i,k)$ and $p(k,j)$ are shortest paths that contain at most $2s/3$ edges. Thus, by the induction hypothesis, after the $(\ell-1)$-st iteration of the algorithm we have $d_{ik} = \delta(i,k)$ and $d_{kj} = \delta(k,j)$. (Note that $k \in B_\ell \subseteq B_{\ell-1}$.) Furthermore, $d_{ik}, d_{kj} \leq (2s/3)M$. As $i, k \in B_\ell$, after the command $D[B,V] \stackrel{\min}{\longleftarrow} \langle D[B,B]\rangle_{sM} \star \langle D[B,V]\ \rangle_{sM}$, we have $d_{ij} \leq d_{ik} + d_{kj} \leq \delta(i,k) + \delta(k,j) = \delta(i,j)$. It is easy to see that at all stages of the algorithm we have $d_{ij} \geq \delta(i,j)$. Thus, after the ℓ-th iteration we have $d_{ij} = \delta(i,j)$, with high probability, as required. □

The matrix D returned by the APSP algorithm is, with high probability, the distance matrix of the graph.

We cannot make the same claim regarding the matrix D returned by the new preprocessing algorithm. However, we can show that, with high probability, $D \star D$ is the distance matrix of the graph!

Lemma 4.2 *Let D be the matrix returned by the preprocessing algorithm. Then, with high probability, $D \star D$ is the distance matrix of the input graph.*

Proof: Let $D' = (d'_{ij}) = D \star D$. It is easy to see that $d'_{ij} \geq \delta(i,j)$, for every $i,j \in V$. We have to show that for every $i,j \in V$ there exists $k \in V$ such that $d'_{ij} \leq d_{ik} + d_{kj} = \delta(i,j)$.

Let $i,j \in V$. Let p be a shortest path from i to j in G and suppose that it uses at most $(3/2)^\ell$ and at least $(3/2)^{\ell-1}$ edges. Let $s = (3/2)^\ell$. As in the proof of Lemmas 3.4 and 4.1, let i' and j' be two vertices on p such that $p(i,i')$ and $p(j',j)$ each contain at most $s/3$ edges, and such that $p(i',j')$ contains exactly $s/3$ edges. Again, with high probability, there exists a vertex $k \in B_\ell$ on $p(i',j')$.

Now, $p(i,k)$ and $p(k,j)$ are shortest paths that use at most $2s/3$ edges, and $k \in B_\ell$. By Lemma 4.1, after the ℓ-th iteration of the algorithm, and hence also at the end of the algorithm, we have $d_{ik} = \delta(i,k)$ and $d_{kj} = \delta(k,j)$. As k lies on a shortest path from i to j, we have $d'_{ij} \leq d_{ik} + d_{kj} = \delta(i,j)$, as required. □

As $(D \star D)_{ij} = D[\{i\}, V] \star D[V, \{j\}]$, the correctness of the query answering algorithm follows immediately:

Corollary 4.3 *If D is the matrix returned by a call to* **preprocess**$(W_{n\times n}, M)$*, then with high probability, for every $i,j \in V$, we have* **dist**$(D,i,j) = \delta(i,j)$.

We next analyze the running time of the new preprocessing algorithm.

Lemma 4.4 *The running time of algorithm* **preprocess**$(W_{n\times n}, M)$ *is $\tilde{O}(Mn^\omega) = O(Mn^{2.38})$.*

Proof: The most time consuming operation in the ℓ-th iteration is the computation of the two distance products $\langle D[V,B]\rangle_{sM} \star \langle D[B,B]\ \rangle_{sM}$ and $\langle D[B,B]\rangle_{sM} \star \langle D[B,V]\ \rangle_{sM}$. As $|B_\ell| = \tilde{O}(n/s)$, where $s = (3/2)^\ell$, the cost of the first product, by Lemma 2.9, is $\tilde{O}(sMn^{\omega(r,r,1)}) = \tilde{O}(n^{1-r}Mn^{\omega(r,r,1)})$, where $n^r = n/s$ and $0 \leq r \leq 1$. By Lemma 2.7, the cost of the product is therefore $\tilde{O}(n^{1-r}Mn^{\omega(r,r,1)}) = \tilde{O}(Mn^{(1-r)+\omega(r,r,1)}) = \tilde{O}(Mn^{(1-r)+(1+(\omega-1)r)}) = \tilde{O}(Mn^{2+(\omega-2)r}) \leq \tilde{O}(Mn^\omega)$, where the last inequality follows from the fact that $r \leq 1$. The cost of the second product is identical. As there are only $O(\log n)$ iterations, the total complexity of the algorithm is also $\tilde{O}(Mn^\omega)$, as required. □

Putting everything together, we get:

Theorem 4.5 *Let $G = (V,E)$ be a directed graph on n vertices with integer edge weights of absolute value at most M. Let W be a matrix containing the edge weights of the graph. Algorithm* **preprocess**$(W_{n\times n}, M)$ *runs in $\tilde{O}(Mn^\omega)$ time and returns a matrix $D_{n\times n}$. For every $i,j \in V$, the query answering algorithm* **dist**$(D_{n\times n}, i, j)$ *runs in $O(n)$ time. With high probability, the matrix $D_{n\times n}$ returned by the preprocessing algorithm is such that* **dist**$(D_{n\times n}, i, j)$ *returns the distance from i to j in the graph, for every $i,j \in V$.*

As stated, the preprocessing algorithm is randomized and may sometimes return incorrect results, though this only happens with a negligible probability. In Section 8 we describe a slightly more complicated deterministic version of the algorithm, with almost the same running time, that never errs.

5 A tradeoff between the preprocessing and query answering times

The preprocessing algorithm of the previous section runs in $\tilde{O}(Mn^{\omega}) = O(Mn^{2.38})$ time and enables answering distance queries in $O(n)$ time. The APSP algorithm of Zwick [Zwi02] runs in $\tilde{O}(n^{2+\mu(t)}) = O(M^{0.68}n^{2.58})$ time, where $M = n^t$, and enables answering distance queries in $O(1)$ time. By combining the two algorithms we can obtain, for every $\mu(t) \leq r \leq 1$, a preprocessing algorithm that runs in $\tilde{O}(Mn^{1-r+\omega(1,r,1)})$ and which enables answering distance queries in $\tilde{O}(n^r)$ time. (Recall that $\mu(t)$ is the solution of the equation $\omega(1, \mu(t), 1) = 1 + 2\mu(t) - t$.) Note that by Theorem 2.3 and 2.4 and Lemma 2.6, $\tilde{O}(Mn^{1-r+\omega(1,r,1)}) = \tilde{O}(Mn^{(2-\alpha\beta)-(1-\beta)r}) = \tilde{O}(Mn^{2.85-0.46r})$, for every $r \geq \frac{1-\alpha\beta+t}{2-\beta}$.

Theorem 5.1 *For every $\mu(t) \leq r \leq 1$ there is an algorithm that preprocesses weighted directed graphs on n vertices with integer edge weights of absolute value at most $M = n^t$ in $\tilde{O}(Mn^{1-r+\omega(1,r,1)})$ time which enables answering distance queries in $\tilde{O}(n^r)$ time.*

Proof: The algorithm performs the first $(1-r)\log_{3/2} n$ iterations of algorithm **short-path**, and then the subsequent $r \log_{3/2} n$ iterations of algorithm **preprocess**. It is easy to see that the most expensive iteration is the last iteration of the first phase, and therefore, the total complexity of the algorithm is $\tilde{O}(Mn^{1-r+\omega(1,r,1)})$, as required.

Let B be the sample used in the last iteration of the first phase. Note that $|B| = \tilde{O}(n^r)$. To find the distance from a vertex i to a vertex j it is enough to compute the distance product $D[\{i\}, B] \star D[B, \{j\}]$ and this can be done in $O(n^r)$ time, as required. □

Corollary 5.2 *There is a preprocessing algorithm whose running time is $O(Mn^{2.5})$ which enables answering distance queries in $O(n^{3/4})$ time.*

6 Answering structured sets of queries

Let $G = (V, E)$ be a weighted directed graph with integer edge weights of absolute value of at most M. Let $U, W \subseteq V$ be two subsets of vertices. Suppose we are interested in all distances from i to j where $(i, j) \in U \times W$. We can do that in $\tilde{O}(Mn^{\omega} + |U||W|n)$ time, using the algorithms of Section 4, or in $\tilde{O}(Mn^{1-r+\omega(1,r,1)} + |U||W|n^r)$ time, for every $\mu(t) \leq r \leq 1$, as explained in Section 5.

For certain values of $|U|$, $|W|$ and M, we can speed-up the computation by computing a single distance product $D[U, B] \times D[B, W]$, using the algorithm of Lemma 2.9, instead of naively computing the products $D[\{i\}, B] \star D[B, \{j\}]$, for every $i \in U$ and $j \in W$. The analysis of this variant and the choice of the optimal parameter r turn out to be quite tedious. The details will appear in the full version of the paper.

7 Applications

Perhaps the most interesting application of the new algorithm is the improved algorithm obtained for the SSSP problem in weighted graphs with integer weights of small absolute value.

Theorem 7.1 *Let $G = (V, E)$ be a weighted directed graph on n vertices with integer edge weights of absolute value at most M. Let $s \in V$. Then, the distances from s to all the other vertices in the graph can be found in $\tilde{O}(Mn^{\omega})$ time, where $\omega < 2.376$ is the exponent of matrix multiplication.*

The new algorithm is much more general, however, as it can compute the distance between any pair of vertices $(i,j) \in S$, where $S \subseteq V \times V$ is an arbitrary set, in $\tilde{O}(Mn^{\omega} + |S|n)$ time, or in $\tilde{O}(Mn^{1-r+\omega(1,r,1)} + |S|n^r)$, for any $\mu(t) \leq r \leq 1$, as follows from Theorem 5.1. The set S is not even required to be known in advance. If the *size* of the set S is known in advance, we can choose the optimal value of the parameter r.

Theorem 7.2 *Let $G = (V,E)$ be a weighed directed graph on n vertices with integer edge weights of absolute value at most $M = n^t$. Let $S \subseteq V \times V$ where $|S| = n^{\gamma}$. Then, the distances from i to j, for every $(i,j) \in S$, can be found in $\tilde{O}(n^{\gamma + r(\gamma,t)})$ time, where $r = r(\gamma,t)$ satisfies the equation $\omega(1,r,1) = 2r + \gamma - t - 1$. Furthermore, for $\omega - 1 \leq \gamma - t$, we have $r(\gamma,t) \leq \frac{3-\alpha\beta-\gamma+t}{2-\beta}$, and hence the required running time is $\tilde{O}(M^{\frac{1}{2-\beta}} n^{\frac{3-\alpha\beta}{2-\beta}} |S|^{\frac{1-\beta}{2-\beta}}) = \tilde{O}(M^{0.68} n^{1.94} |S|^{0.32})$.*

Theorem 7.2 has two immediate corollaries:

Corollary 7.3 *Let $G = (V,E)$ be a weighed directed graph, where $|V| = n$ and $|E| = m$, with integer edge weights of absolute value at most M. If $n^{\omega-1} \leq m/M$, then the set of all edges in G which are the shortest paths between their endpoints can be found in $\tilde{O}(M^{0.68} n^{1.94} m^{0.32})$ time.*

Corollary 7.4 *Let $G = (V,E)$ be a weighed directed graph, where $|V| = n$ and $|E| = m$, with integer edge weights of absolute value at most M. If $n^{\omega-1} \leq m/M$, then the shortest cycle though each edge of G can be found in $\tilde{O}(M^{0.68} n^{1.94} m^{0.32})$ time.*

Using an observation of Johnson [Joh77] we also obtain the following result:

Theorem 7.5 *Let $G = (V,E)$ be a weighed directed graph, where $|V| = n$ and $|E| = m$, with integer edge weights of absolute value at most M. Let $U \subseteq V$ with $|U| \leq Mn^{\omega}/m$. Then, the distances of all pairs of vertices in $R \times V$ and $V \times R$ can be found in $\tilde{O}(Mn^{\omega})$ time.*

Proof: We introduce an auxiliary vertex s and add zero weight edges from it to all the other vertices in the graphs. We then use the $\tilde{O}(Mn^{\omega})$ time algorithm of Theorem 7.1 to compute the distances from s to all the other vertices. We next define a modified weight function of the edges $w'_{ij} = w_{ij} + \delta(s,i) - \delta(s,j)$, for every $(i,j) \in E$. By the triangle inequality, $w'_{ij} \geq 0$, for every $(i,j) \in E$. We can, therefore, use Dijkstra's algorithm (Dijkstra [Dij59], Fredman and Tarjan [FT87]) to compute shortest paths from each vertex $i \in U$ to all the other vertices of the graph. The total time required is $\tilde{O}(|U|m) = \tilde{O}(Mn^{\omega})$. Finally, it is easy to see that $\delta(i,j) = \delta'(i,j) - \delta(s,i) + \delta(s,j)$, for every $i,j \in V$, where $\delta'(i,j)$ are the distances with respect to the modified edge weights. □

8 Derandomization

The algorithms of Sections 4 and 5 can be derandomized using extensions of the techniques used in [Zwi02]. Some non-trivial new ideas are required, however. We sketch below some of the ideas used for the *unweighted* case, which is already a non-trivial case. Some additional complications arise in the weighted case.

The main idea is to replace the randomly selected sample B_{ℓ}, used in the ℓ-th iteration, by a deterministically constructed *s-bridging set*, for $s = (3/2)^{\ell-1}$. (See [Zwi02] for a definition.) A deterministic algorithm for constructing s-bridging sets with a running time of $O(n^2 s)$ is described in [Zwi02]. Unfortunately, this

algorithm is too slow for our purposes. We present, therefore, an improved deterministic algorithm with a running time of $O(n^2 \log n)$. Another problem that needs to be addressed is that the bridging set B_ℓ constructed during the ℓ-th iteration is usually not a subset of the bridging set constructed during the $(\ell-1)$-st iteration. To solve this problem we *rerun* iterations $1, 2, \ldots, \ell-1$ of the algorithm with the B_ℓ added to the bridging sets $B_1 \supseteq B_2 \supseteq \cdots \supseteq B_{\ell-1}$. This ensures the proper nesting of the bridging sets while slowing the algorithm down by only a logarithmic factor. The full details will appear in the full version of the paper.

9 Concluding remarks

We have shown that a weighted directed graph $G = (V, E)$ on n vertices, with integer edge weights of absolute value at most M, can be preprocessed in $\tilde{O}(Mn^\omega)$ time, where $\omega < 2.376$ is the exponent of matrix multiplication, such that any distance query can subsequently be answered, exactly, in $O(n)$ time. We also presented a tradeoff between the preprocessing and query answering times. As a very special, but interesting, case, this yields an improved SSSP algorithm for graphs that are dense enough, and whose edge weights are small enough. This is the first improved SSSP algorithm for graphs with negative edge weights for over a decade. It is also the first time that fast matrix multiplication algorithms are used to obtain an improved SSSP algorithm.

References

[AGM97] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.

[Bel58] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[Gol95] A.V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[GT91] H.N. Gablow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38(4):815–853, 1991.

[HP98] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.

[Joh77] D.B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.

[SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. of 40th FOCS*, pages 605–614, 1999.

[Tho99] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.

[TZ05] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[Yuv76] G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4:155–156, 1976.

[Zwi02] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.