Computing the diameter polynomially faster than APSP

Raphael Yuster [*]

Abstract

We present a new randomized algorithm for computing the diameter of a weighted directed graph. The algorithm runs in $\tilde{O}(M^{\omega/(\omega+1)}n^{(\omega^2+3)/(\omega+1)})$ time, where $\omega < 2.376$ is the exponent of fast matrix multiplication, n is the number of vertices of the graph, and the edge weights are integers in $\{-M,\ldots,0,\ldots,M\}$. For bounded integer weights the running time is $O(n^{2.561})$ and if $\omega = 2+o(1)$ it is $\tilde{O}(n^{7/3})$. This is the first algorithm that computes the diameter of an integer weighted directed graph polynomially faster than any known All-Pairs Shortest Paths (APSP) algorithm. For bounded integer weights, the fastest algorithm for APSP runs in $O(n^{2.575})$ time for the present value of ω and runs in $\tilde{O}(n^{2.5})$ time if $\omega = 2+o(1)$.

For directed graphs with *positive* integer weights in $\{1,\ldots,M\}$ we obtain a deterministic algorithm that computes the diameter in $\tilde{O}(Mn^\omega)$ time. This extends a simple $\tilde{O}(n^\omega)$ algorithm for computing the diameter of an *unweighted* directed graph to the positive integer weighted setting and is the first algorithm in this setting whose time complexity matches that of the fastest known Diameter algorithm for *undirected* graphs.

The diameter algorithms are consequences of a more general result. We construct algorithms that for any given integer d, report all ordered pairs of vertices having distance *at most* d. The diameter can therefore be computed using binary search for the smallest d for which all pairs are reported.

1 Introduction

Computing the diameter and, more generally, computing distances, are among the most fundamental algorithmic graph problems. In the *Diameter* problem we are given a graph and are required to find the largest distance between two vertices of the graph. The more general *All-Pairs Shortest Paths* (APSP) problem asks to find distances and shortest paths between all pairs of vertices of the graph. Clearly, any algorithm for APSP can be used as an algorithm for Diameter. The converse, however, is not necessarily true.

Unfortunately, at present, we do not know of any algorithm for *general* weighted graphs that solves Diameter asymptotically faster than APSP and the existence of such an algorithm is an open problem (see, e.g., [1, 3, 5]).

In the case of graphs with arbitrary real edge weights, no truly sub-cubic algorithm for APSP or for Diameter is known. The presently fastest algorithm for both is an algorithm of Chan [4] that runs in $O(n^3 \log^3 \log n / \log^2 n)$ time, where n in the number of vertices of the graph. For sufficiently sparse graphs, the APSP algorithm of Johnson [10] performs better as it runs in $O(mn + n^2 \log n)$ time where m is the number of edges.

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

When the edge weights are integers, fast matrix multiplication techniques are useful. For undirected graphs with integer edge weights in $\{1,\ldots,M\}$ an APSP (and thereby a Diameter) algorithm of Shoshan and Zwick [13] runs in $\tilde{O}(Mn^w)$[1] time, where $\omega < 2.376$ is the exponent of matrix multiplication. This algorithm generalizes earlier $\tilde{O}(n^\omega)$ algorithms of Seidel and of Galil and Margalit for the unweighted case [12, 8].

The situation becomes more involved in directed graphs, as negative edge weights may be allowed. In the presence of weights in $\{-M,\ldots,0,\ldots,M\}$, the fastest APSP algorithm (and presently fastest Diameter algorithm) is by Zwick [17]. It runs in $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ time, and an additional small speedup is obtained if fast rectangular matrix multiplication is used. For bounded M, this speedup results in a running time of $O(n^{2.575})$. Interestingly, Zwick's algorithm is also the fastest known algorithm for *unweighted* APSP in directed graphs. The directed unweighted setting is also the only known setting where Diameter has an algorithm that is presently faster than APSP. It was observed by Zwick (private communication) and also by Björklund (private communication) that the Diameter of a directed unweighted graph can be computed in $\tilde{O}(n^\omega)$ time using a repeated squaring argument of the adjacency matrix, combined with binary search. This approach, however, does not directly extend to the weighted setting even when the weights are positive integers in $\{1,\ldots,M\}$.

The main results of this paper present the first algorithm(s) for Diameter that are polynomially faster than APSP in integer weighted directed graphs. Our first algorithm computes the diameter of a directed graph with integer edge weights in $\{-M,\ldots,0,\ldots,M\}$ polynomially faster than the aforementioned APSP algorithm of Zwick. Our second algorithm computes the diameter of a directed graph with integer edge weights in $\{1,\ldots,M\}$ even faster. The time complexity we obtain in this case matches the time complexity of the aforementioned APSP algorithm of Shoshan and Zwick that applies to the *undirected* setting.

In fact, both algorithms are, respectively, consequences of two other algorithms that solve a related problem that can be viewed as more general than Diameter.

The problem we are considering is *Threshold APSP* where, in addition to the input graph we receive a threshold value d. The goal is to report all ordered pairs of vertices having distance *at most* d. Any algorithm for Threshold APSP can be converted into an algorithm for Diameter by applying Threshold APSP a logarithmic number of times. Indeed, since our edge weights are integers in $\{-M,\ldots,0,\ldots,M\}$ the diameter is either ∞ or is an integer in $\{0,\ldots,M(n-1)\}$[2]. Using binary search one can locate the diameter which is the smallest d such that Threshold APSP reports all ordered pairs of vertices. Observe that Threshold APSP is an interesting problem on its own. Especially interesting are the cases $d=0$ and $d=-1$ in the negative weight setting. The case $d=0$ can be used to find all pairs that have positive distance and the case $d=-1$ can be used to find all pairs that have negative distance. Using two consecutive values of d one can also locate all pairs having any specific given distance. Our main results are, therefore, algorithms for Threshold APSP.

Theorem 1.1 *Let $G=(V,E)$ be directed graph with n vertices and having integer edge weights in $\{-M,\ldots,0,\ldots,M\}$ and let d be an integer. Then, with high probability, the set of all ordered pairs of vertices having distance at most d can be computed in $\tilde{O}(M^{\omega/(\omega+1)}n^{(\omega^2+3)/(\omega+1)})$ time. In particular, the diameter of G can be computed in $\tilde{O}(M^{\omega/(\omega+1)}n^{(\omega^2+3)/(\omega+1)})$ time.*

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

[2] As usual, we assume that the input graph has non negative weight cycles. This can also be verified initially using known algorithms such as [16] in running times that are faster than the claimed running time of our algorithm.

For $\omega = 2.376$ and for bounded edge weights, the running time of our algorithm is $O(n^{2.561})$. It is, therefore, polynomially faster than the $O(n^{2.575})$ algorithm of Zwick. An even larger gap exists if $\omega = 2 + o(1)$, as may turn out to be the case. In this case, Zwick's algorithm, as well as an earlier algorithm of Alon, Galil, and Margalit [2], run in $\tilde{O}(n^{2.5})$ time. It is, thus, plausible that $\Omega(n^{2.5})$ is a barrier for APSP in directed graphs. Our algorithm, on the other hand, computes the diameter in $\tilde{O}(n^{7/3})$ time, assuming $\omega = 2 + o(1)$. Our result thus intensifies the plausibility of a true complexity gap between Diameter and APSP in weighted directed graphs.

For unbounded edge weights, our algorithm is also faster than Zwick's $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ algorithm (also if fast rectangular matrix multiplication is used in the latter) as long as $M < n^{3-\omega}$. But observe that for $M \geq n^{3-\omega}$ both algorithms become cubic and it is preferable to use Chan's general APSP algorithm in this case.

The main idea of our algorithm is as follows. We start by computing values that capture the actual distance between all pairs that are "far apart" (pairs whose distance is realized only by a path that has relatively many edges). This is done using the well-known "long shortest path" technique. The more difficult case is to capture distances connecting pairs that are not far apart. For this purpose we use an appropriately computed *redundant partial distance matrix* that is a generalization of partial distance matrices introduced in [16]. A scaled version of this matrix enables us to obtain a good *additive* approximation for the distance between such pairs. If the approximated value is sufficiently smaller than the threshold d then we already know that such pairs must be reported. For pairs whose approximation is close to d we can use a truncated and shifted version of the redundant partial distance matrix in order to compute their distance precisely.

We now state our result for the positive integer weighted case.

Theorem 1.2 *Let $G = (V, E)$ be directed graph with n vertices and having integer edge weights in $\{1, \ldots, M\}$ and let d be an integer. Then, the set of all ordered pairs of vertices having distance at most d can be computed in $\tilde{O}(Mn^{\omega})$ time. In particular, the diameter of G can be computed in $\tilde{O}(Mn^{\omega})$ time.*

Similar to other shortest paths algorithms, both of our algorithms for Threshold APSP can actually construct a *path data structure*. For two vertices u, v reported as having distance at most d, we can generate an actual path from u to v whose total weight is at most d in time that is proportional to the number of edges of the generated path. Observe, however, that for the purpose of computing the diameter, we do not even need to construct such a data structure. Once we have computed the diameter value d, we can use the results of Threshold APSP with the threshold $d - 1$ in order to locate all pairs of vertices that realize the diameter precisely. These are all pairs reported for the threshold d and not reported for the threshold $d - 1$. Now, if we pick any such pair u, v, we can perform a Single Source Shortest Paths computation from u in $\tilde{O}(Mn^{\omega})$ time (see, e.g. [16, 11]) and obtain an actual path that realizes the diameter.

The rest of this paper is organized as follows. The next section contains some preliminary definitions, notations, and known results that are required for the proof of the theorems. Section 3 describes redundant partial distance matrices that are an important ingredient in the proof of Theorem 1.1. Section 4 contains the proof of Theorem 1.1 and Section 5 contains the proof of Theorem 1.2. The final section contains some concluding remarks.

2 Preliminaries

Let $T(\ell, m, n)$ 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 2.1 (Matrix multiplication exponents) *Let $\omega(r, s, t)$ be the infimum of all the exponents ω' for which $T(n^r, n^s, n^t) = O(n^{\omega'})$. We let $\omega = \omega(1, 1, 1)$ be the exponent of square matrix multiplication.*

Theorem 2.2 (Coppersmith and Winograd [7]) $\omega < 2.376$.

In fact, Coppersmith [6] proved that if α is the supremum over all constants r for which $\omega(1, r, 1) = 2$ then $\alpha > 0.294$. The following lemmas are obtained by decomposing a given matrix product into smaller products. (See, e.g., Huang and Pan [9].)

Lemma 2.3 $\omega(1, r, 1) \le \begin{cases} 2 & \text{if } 0 \le r \le \alpha, \\ 2 + \frac{\omega-2}{1-\alpha}(r - \alpha) & \text{if } \alpha \le r \le 1. \end{cases}$

Lemma 2.4 $\omega(1, r, r) = \omega(r, r, 1) \le 1 + (\omega - 1)r$, *for* $0 \le r \le 1$.

Definition 2.5 (Distance products) *Let A be an $\ell \times m$ matrix, and let B be a $m \times n$ matrix. Their $\ell \times n$* distance product *$C = A \star B$ is defined as follows: $C[i, j] = \min_{k=1}^{m}\{A[i, k] + B[k, j]\}$, for $1 \le i \le \ell$ and $1 \le j \le n$.*

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 (recall that we assume no negative weight cycles). Namely, $W^n[i, j] = \delta(i, j)$ where $\delta(i, j)$ denotes the distance from i to j.

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. [2], following a related idea of Yuval [15].

Lemma 2.6 *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.*

Definition 2.7 (Truncation) *If D is a matrix and t is a positive integer, let $\langle D \rangle_t$ be the matrix obtained from D by replacing all the entries that are larger than t or smaller than $-t$ by $+\infty$. In other words, $\langle D \rangle_t[i, j] = D[i, j]$, if $|D[i, j]| \le t$, and $\langle D \rangle_t[i, j] = +\infty$, otherwise.*

Zwick [17] used several novel ideas combining truncated distance products, the notion of bridging sets, and fast matrix multiplication, to obtain the presently fastest APSP algorithm in dense directed graphs with integer edge weights.

Theorem 2.8 (Zwick [17]) *Let G be a directed graph with n vertices with integer edge weights in $\{-M, \ldots, 0, \ldots, M\}$. Let $M = n^t$. There is an algorithm that computes the distance matrix of G in $\tilde{O}(n^{2+\mu(t)})$ time, where $\mu = \mu(t)$ satisfies the equation $\omega(1, \mu, 1) = 1 + 2\mu - t$. In particular, the algorithm runs in $O(n^{2.575})$ time if M is bounded.*

Definition 2.9 (Partial distance matrix) *A* partial distance matrix *of a graph G is a matrix P such that $P \star P$ is the distance matrix of G.*

Generalizing the above result of Zwick, it was shown by the author and Zwick how a *partial distance matrix* can be computed much faster. Observe that once a partial distance matrix is computed, the distance between any given pair of vertices can be computed in $O(n)$ time.

Theorem 2.10 (Yuster and Zwick [16]) *Let G be a directed graph having n vertices and integer edge weights in $\{-M, \ldots, 0, \ldots, M\}$. There exists an algorithm that computes a partial distance matrix of G in $\tilde{O}(Mn^{\omega})$ time.*

3 Redundant partial distance matrices

For two vertices u, v of a graph, let $c(u, v)$ denote the smallest number of edges in a path that realizes $\delta(u, v)$.

Definition 3.1 ((β, γ)-redundant partial distance matrix) *Let $G = (V, E)$ be a directed graph. For nonnegative parameters β, γ with $0 \le \beta + \gamma \le 1$ we say that a matrix P whose rows and columns are indexed by V is a (β, γ)-*redundant partial distance matrix *((β, γ)-RPDM) if the following holds:*

1. *For each pair $u, v \in V$ with $c(u, v) \le n^{1-\beta}$ there exists some $x \in V$ such that $P[u, x] + P[x, v] = \delta(u, v)$. Furthermore:*
2. *There exists a path $p_{u,v}$ with $c(u, v)$ edges realizing $\delta(u, v)$ such that each segment of $p_{u,v}$ consisting of $\lceil n^{1-\beta-\gamma} \rceil$ edges contains a vertex x such that $P[u, x] + P[x, v] = \delta(u, v)$.*

The algorithm given in Figure 1 computes a (β, γ)-RPDM of a directed graph $G = (V, E)$ with integer edge weights in $\{-M, \ldots, 0, \ldots, M\}$ whose adjacency matrix is given as the input parameter W (non-edges represented by $+\infty$ in W). For subsets of vertices X and Y, the notation $P[X, Y]$ appearing in the algorithm denotes the sub-matrix of P consisting of the rows X and columns Y. For matrices R and S with the same dimensions, the notation $R \overset{\min}{\longleftarrow} S$ denotes the the assignment to R of the matrix whose entry $[u, v]$ is the minimum of $R[u, v]$ and $S[u, v]$. The algorithm in Figure 1 is a modification of the algorithm from [16] for computing partial distance matrices. In fact, the first `for` loop (consisting of $\lceil \log_{3/2}(n^{1-\beta-\gamma}) \rceil$ iterations) is identical to the algorithm in [16]. The second `for` loop differs from the first one in that B remains constant and is no longer decreased by sampling. We next prove the correctness and compute the running time of the algorithm **redundant-partial-distance-matrix**.

Lemma 3.2 *With high probability (at least $1 - O(\log n/n)$),* **redundant-partial-distance-matrix** *correctly computes a (β, γ)-RPDM.*

Proof. The first part of the proof is identical to the proof in [16]. Let B_ℓ denote the random subset B of the ℓ-th iteration (observe that when $\ell \le \lceil \log_{3/2}(n^{1-\beta-\gamma}) \rceil$ we are in the first `for` loop and otherwise we are in the second `for` loop). We first note that Lemma 4.1 of [16] remains intact: If $i \in B_\ell$ or $j \in B_\ell$, and 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, with high probability (at least $1 - O(\log n/n)$) we have $P[i, j] = \delta(i, j)$ (recall

```
algorithm redundant-partial-distance-matrix($W_{n\times n}, \beta, \gamma$)
$V \leftarrow \{1, 2, \ldots, n\}$
$B \leftarrow V$ ; $P \leftarrow W$
for $\ell \leftarrow 1$ to $\lceil \log_{3/2}(n^{1-\beta-\gamma}) \rceil$
    $s \leftarrow (3/2)^\ell$
    $B \leftarrow$ sample$(B, (9n \ln n)/s)$
    $P[V,B] \xleftarrow{\min} \langle P[V,B] \rangle_{sM} \star \langle P[B,B] \rangle_{sM}$
    $P[B,V] \xleftarrow{\min} \langle P[B,B] \rangle_{sM} \star \langle P[B,V] \rangle_{sM}$
endfor
for $\ell \leftarrow \lceil \log_{3/2}(n^{1-\beta-\gamma}) \rceil + 1$ to $\lceil \log_{3/2}(2n^{1-\beta}) \rceil$
    $s \leftarrow (3/2)^\ell$
    $P[V,B] \xleftarrow{\min} \langle P[V,B] \rangle_{sM} \star \langle P[B,B] \rangle_{sM}$
    $P[B,V] \xleftarrow{\min} \langle P[B,B] \rangle_{sM} \star \langle P[B,V] \rangle_{sM}$
endfor
return $P$
```

Figure 1: Computing a (β, γ)-redundant partial distance matrix.

that the only difference between our algorithm and the algorithm from [16] is that we stop sampling from B after the end of the first `for` loop; this is of course to our advantage since the probability of hitting a path with a larger sample is larger).

Let u and v be two vertices with $c(u,v) \le n^{1-\beta}$, and let $p_{u,v}$ be a path with $c(u,v)$ edges from u to v realizing $\delta(u,v)$. To establish the first part in the definition of a (β,γ)-RPDM, we only need to show that at the end of the algorithm there is, with high probability, some $x \in B_\ell$ on $p_{u,v}$. This is identical to the proof of Lemma 4.2 in [16] (as shown there, such an x already appears in B_ℓ where $(3/2)^\ell \ge c(u,v)$). But in order to satisfy the second part in the definition of a (β,γ)-RPDM we need to prove something stronger: we must show that, with high probability, any segment of $n^{1-\beta-\gamma}$ edges of $p_{u,v}$ contains some $x \in B_\ell$ at the end of the algorithm. As B_ℓ remains the same after the last iteration of the first `for` loop, we must prove that a random subset of size $9n \ln n / n^{1-\beta-\gamma}$ vertices hits every such segment of $p_{u,v}$. Indeed, the probability that no vertex of B_ℓ hits a specific segment is at most

$$(1 - 9 \ln n / n^{1-\beta-\gamma})^{n^{1-\beta-\gamma}} < n^{-9} .$$

As there are less than n segments to consider in $p_{u,v}$, and as there are less than n^2 pairs of vertices u, v to consider, we have that with high probability (larger than $1 - O(\log n/n)$), the set B_ℓ at the end of the first `for` loop (and hence also at the end of the algorithm; it is the same set) hits every segment of $n^{1-\beta-\gamma}$ edges of each $p_{u,v}$ with $c(u,v) \le n^{1-\beta}$. ■

Lemma 3.3 **redundant-partial-distance-matrix** *runs in* $\tilde{O}(Mn^\omega + Mn^{2+(\omega-2)(\beta+\gamma)+\gamma})$ *time.*

Proof. There are a logarithmic number of iterations, and the most time consuming operation in each iteration is the computation of a distance product. By Lemma 2.6, a distance product in the first `for` loop can be computed in $\tilde{O}(sM \times T(n, n/s, n/s))$ time. By Lemma 2.4, multiplying an $n \times n/s$ matrix with an $n/s \times n/s$ matrix requires $O(s(n/s)^{\omega})$ operations. Hence, the time to perform a single distance product is $\tilde{O}(Ms^2(n/s)^{\omega})$. Since $\omega \geq 2$, this is never larger than $\tilde{O}(Mn^{\omega})$ (this is the same argument that is used in [16] to show that their algorithm runs in $\tilde{O}(Mn^{\omega})$ time). By Lemma 2.6, a distance product in the second `for` loop can be computed in $\tilde{O}(sMn^{\omega(1,\beta+\gamma,\beta+\gamma)})$. This is maximized in the last iteration where $s = \Theta(n^{1-\beta})$. By Lemma 2.4 this amounts to $\tilde{O}(Mn^{2+(\omega-2)(\beta+\gamma)+\gamma})$. ■

4 Proof of Theorem 1.1

Let $G = (V, E)$ be a directed graph with n vertices and with integer edge weights taken from $\{-M, \dots, 0, \dots, M\}$. Recall that G is assumed to contain no negative weight cycles. Let d be any integer and let $D = \{(u, v) \ : \ \delta(u, v) \leq d\}$. Our goal is to construct the set D. Observe that if $d < -nM$ then $D = \emptyset$ (otherwise there are negative cycles). Also, if $d > nM$ then D is precisely the set of pairs (u, v) with $\delta(u, v) < \infty$ and in this case D can trivially be obtained from the transitive closure of the unweighted version of G. Hence we will assume that $|d| \leq nM$.

4.1 Retrieving distances that are realized by long paths

The first part of our algorithm computes the distances between all pairs of vertices for which $c(u, v)$ is large. More precisely, let β be a chosen such that

$$n^{\beta} = M^{\frac{w}{\omega+1}} n^{\frac{(\omega-1)^2}{\omega+1}} \tag{1}$$

and let $t = n^{1-\beta}$. We compute, for *each* ordered pair of vertices (u, v), and with very high probability, a value $\delta_t(u, v)$ which satisfies $\delta_t(u, v) \geq \delta(u, v)$ and if $c(u, v) \geq t$ then $\delta_t(u, v) = \delta(u, v)$.

Lemma 4.1 *There is an algorithm that computes, with probability at least* $1 - O(1/n)$*, values* $\delta_t(u, v)$ *for each ordered pair of vertices* (u, v) *such that* $\delta_t(u, v) \geq \delta(u, v)$ *and if* $c(u, v) \geq t$ *then* $\delta_t(u, v) = \delta(u, v)$*. The algorithm runs in* $\tilde{O}(n^{2+\beta} + Mn^{\omega})$ *time.*

Proof. Our algorithm uses the well-known "long shortest path" method to compute the distances between pairs whose distance is realized only by paths with at least t edges. The idea behind this simple method is that a random large subset $X \subset V$ hits all of these "long" shortest paths.

More formally, let

$$C = \{(u, v) : c(u, v) \geq t\}.$$

For each pair in C, fix a shortest path $p_{u,v}$ from u to v. Each such path contains at least $t+1$ vertices (including endpoints). Let X be a random subset of $8n \ln n/t$ vertices. For a pair $(u, v) \in C$, what is the probability that no element of X lies on $p_{u,v}$? Clearly, this probability is at most

$$\left(1 - \frac{c(u, v) + 1}{n}\right)^{|X|} < \left(1 - \frac{t}{n}\right)^{|X|} \leq \left(1 - \frac{t}{n}\right)^{8n \ln n/t} < \frac{1}{n^3} \, .$$

As $|C| < n^2$ is follows by the union bound that with high probability (at least $1 - O(1/n)$), X intersects each $p_{u,v}$ for $(u, v) \in C$.

So, assume that X is a set of $8n \ln n/t$ vertices intersecting all $p_{u,v}$ for $(u,v) \in C$. For each $x \in X$ our next goal is to compute, using a single-source (SSSP) algorithm, all the distances $\delta(x,v)$ and all the distances $\delta(v,x)$ for all $v \in V$.

Unfortunately, G has negative edge weights so performing $|X|$ applications of SSSP is too costly. As observed by Johnson [10], 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. An $\tilde{O}(Mn^\omega)$ time SSSP algorithm for directed graphs with integer weights in $\{-M,\ldots,0,\ldots,M\}$ was obtained in [11, 16]. It follows that the required reweighing of G can be obtained in $\tilde{O}(Mn^\omega)$ time. 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 $w_+(u,v)$ is just $w(u,v)+h(u)-h(v) \geq 0$. It now suffices to compute $SSSP$ from each vertex of X in G (and similarly in its edge-reversed version). This, in turn, can be performed in $O(n^2)$ time for each vertex of X, using Dijkstra's algorithm.

Altogether, the running time required to obtain all of the distances $\delta(x,v)$ and $\delta(v,x)$ for all $x \in X$ and $v \in V$ is

$$\tilde{O}(n^2|X| + Mn^\omega) = \tilde{O}(n^3/t + Mn^\omega) = \tilde{O}(n^{2+\beta} + Mn^\omega) \ .$$

Next, for each ordered pair of vertices (u,v), we compute

$$\delta_t(u,v) = \min_{x \in X} \delta(u,x) + \delta(x,v) \ .$$

Observe that the right hand side of the last equation is either infinity or a weight of *some* walk from u to v and thereby $\delta_t(u,v) \geq \delta(u,v)$. But for pairs $(u,v) \in C$ the property of X guarantees that, in fact, $\delta_t(u,v) = \delta(u,v)$, as required.

The running time to obtain all of the values $\delta_t(u,v)$ is $O(n^2|X|) \leq \tilde{O}(n^{2+\beta})$. The overall running time of the algorithm is therefore $\tilde{O}(n^{2+\beta} + Mn^\omega)$, as claimed. ■

4.2 Obtaining a good additive approximation

The second part of our algorithm computes *approximate* distances between all pairs of vertices for which $c(u,v)$ is relatively small. This process consists of a logarithmic number of steps, where each step guarantees to approximate distances $\delta(u,v)$ for a specific range of $c(u,v)$. More precisely, for $i = 0,\ldots,\lfloor (1-\beta)\log n \rfloor$, let t_i and β_i be defined by

$$t_i = n^{1-\beta_i} = \frac{n^{1-\beta}}{2^i}$$

and observe that $\beta_0 = \beta$ and $t_0 = t$. Step i computes, for *each* ordered pair of vertices, and with very high probability, a value $\delta^*_i(u,v)$ satisfying $\delta^*_i(u,v) \geq \delta(u,v)$ and if $t_i/2 \leq c(u,v) < t_i$ then $\delta^*_i(u,v) \leq \delta(u,v) + 2k_i$ for a suitably chosen approximation parameter k_i. Observe that for any pair (u,v) with $c(u,v) < t$, there exists *some* i for which $t_i/2 \leq c(u,v) < t_i$.

Let γ_i be defined by

$$n^{\gamma_i} = \left(n^{1-\beta_i}\right)^{\frac{\omega-1}{\omega}} \tag{2}$$

and observe that $n^{\beta_i+\gamma_i} \leq n$ so $\beta_i + \gamma_i \leq 1$. Since $\beta_0 = \beta$ we also define $\gamma = \gamma_0$.

```
algorithm additive-approximate($P_i, \gamma_i, k_i$)
$R_i \leftarrow P_i/k_i$
$X \leftarrow$ sample($V, 12n^{1-\gamma_i}\log n$)
$Q_i \leftarrow R_i[V,X] \star R_i[X,V]$
return $k_iQ_i$
```

Figure 2: Obtaining an additive approximation.

Let P_i be a (β_i,γ_i)-RPDM computed by the algorithm in Section 3. We use P_i to obtain the claimed additive approximation.

Algorithm **additive-approximate** in Figure 2 accepts P_i and γ_i as input, as well as an approximation parameter k_i. It returns a matrix such that with high probability, its entry $[u,v]$ is close to $\delta(u,v)$ whenever $t_i/2 \le c(u,v) < t_i$.

Lemma 4.2 *With probability $1-O(1/n)$, the matrix k_iQ_i returned by* **additive-approximate** *has the property that for each ordered pair of vertices (u,v) we have $k_iQ_i[u,v] \ge \delta(u,v)$ and if $t_i/2 \le c(u,v) < t_i$ then $k_iQ_i[u,v] \le \delta(u,v)+2k_i$. Using $k_i = Mn^{1-\beta_i-\gamma_i}$ the running time of* **additive-approximate** *is $\tilde{O}(n^{\gamma_i+\omega(1,1-\gamma_i,1)})$.*

Proof. Consider an ordered pair of vertices (u,v) for which $t_i/2 \le c(u,v) < t_i = n^{1-\beta_i}$. By the definition of P_i, we have that there exists a path $p_{u,v}$ with $c(u,v)$ edges realizing $\delta(u,v)$ where each segment of $\lceil n^{1-\beta_i-\gamma_i}\rceil$ edges of $p_{u,v}$ contains a vertex x such that $P[u,x]+P[x,v]=\delta(u,v)$. In particular, there are at least

$$\frac{c(u,v)}{\lceil n^{1-\beta_i-\gamma_i}\rceil} \ge \frac{t_i}{4n^{1-\beta_i-\gamma_i}} = \frac{n^{\gamma_i}}{4}$$

such vertices x. This, in turn, implies that a random subset X of $12n^{1-\gamma_i}\log n$ vertices contains, with high probability (at least $1-O(1/n)$) at least one such x for each pair (u,v) with $t_i/2 \le c(u,v) < t_i$. This means that, w.h.p., the entry $[u,v]$ of the product $P_i[V,X]\star P_i[X,V]$ contains the *precise* value of $\delta(u,v)$ for such a pair (u,v). Performing this distance product is, however, costly. Instead, we divide each finite element in P_i by the number k_i. More precisely, the line $R_i \leftarrow P_i/k_i$ denotes that $R_i[u,v] = \lceil P_i[u,v]/k_i\rceil$ for all elements of P. We have

$$P_i[u,x]+P_i[x,v] \le k_i(R_i[u,x]+R_i[x,v]) \le P_i[u,x]+P_i[x,v]+2k_i$$

and in particular, for all (u,v) with $t_i/2 \le c(u,v) < t_i$,

$$\delta(u,v) \le k_iQ_i[u,v] \le \delta(u,v)+2k_i\ .$$

Notice that for other pairs, the value $k_iQ_i[u,v]$ is *at least* $\delta(u,v)$ since the entries in R_i are rounded up.

When applying **additive-approximate** we will use $k_i = Mn^{1-\beta_i-\gamma_i}$. Observe that the finite entries in P_i have value $O(Mn^{1-\beta_i})$ and hence the finite entries in R_i have value $O(n^{\gamma_i})$. The running time of **additive-approximate** is dominated by the distance product of an $n \times |X|$ sub-matrix of R with an $|X| \times n$ sub-matrix of R. Hence, by Lemma 2.6 and since $|X| = \tilde{O}(n^{1-\gamma_i})$, we have that **additive-approximate** runs in $\tilde{O}(n^{\gamma_i + \omega(1,1-\gamma_i,1)})$ time. ■

4.3 Targeting pairs having distance at most d

The final part of our algorithm correctly reports all pairs (u, v) with $\delta(u, v) \le d$. In the previous parts of our algorithm we have computed, for each pair (u, v), values $\delta_t(u, v)$ and $\delta^*_i(u, v)$ for $i = 0, \ldots, \lfloor(1 - \beta)\log n\rfloor$. Let

$$\delta^*(u,v) = \min\{\delta_t(u,v)\ ,\ \min_{i=0}^{\lfloor(1-r)\log n\rfloor} \delta^*_i(u,v)\}\ .$$

The following lemma is a consequence of these computed values.

Lemma 4.3 *For any pair (u, v) we have $\delta(u, v) \le \delta^*(u, v) \le \delta(u, v) + 2Mn^{1-\beta-\gamma}$.*

Proof. Recall that all computed values are either infinite or upper bounds of weights of some walks from u to v and therefore $\delta(u, v) \le \delta^*(u, v)$. Now, if $c(u, v) \ge t$ then $\delta_t(u, v) = \delta(u, v)$. Otherwise, there is some i such that $t_i/2 \le c(u, v) < t_i$ in which case $\delta^*_i(u, v) \le \delta(u, v) + 2k_i$. Since $k_i = Mn^{1-\beta_i-\gamma_i}$ the claim follows once we observe that $\beta + \gamma \le \beta_i + \gamma_i$. Indeed this holds since by (2)

$$n^{\beta_i+\gamma_i} = n^{\frac{\omega-1+\beta_i}{\omega}} \ge n^{\frac{\omega-1+\beta}{\omega}} = n^{\beta+\gamma}\ .$$

■

By Lemma 4.3, any pair (u, v) with $\delta^*(u, v) \le d$ is reported as having $\delta(u, v) \le d$, as required. Similarly, any pair with $\delta^*(u, v) > d + 2Mn^{1-\beta-\gamma}$ is reported as having $\delta(u, v) > d$, as required. So we remain with the following set of pairs

$$C^* = \{(u,v) : d < \delta^*(u,v) \le d + 2Mn^{1-\beta-\gamma}\}$$

where we must determine, for each $(u, v) \in C^*$, whether or not $\delta(u, v) \le d$. In fact, we will determine $\delta(u, v)$ precisely for all $(u, v) \in C^*$.

Let C_i denote the set of pairs in C^* for which $t_i/2 \le c(u, v) < t_i$. Observe that we *do not* know C_i and, furthermore, there may be pairs $(u, v) \in C^*$ that are in no C_i as it may be that $\delta^*(u, v) = \delta_t(u, v)$. Consider again the (β_i, γ_i)-RPDM matrix P_i. By the definition of P_i, for each $(u, v) \in C_i$, there is a shortest path $p_{u,v}$ from u to v with $c(u, v)$ edges such that every segment of $\lceil n^{1-\beta_i-\gamma_i}\rceil$ edges of $p_{u,v}$ contains a vertex x such that $P_i[u, x] + P_i[x, v] = \delta(u, v)$. In particular, there is such an x where

$$P_i[u,x], P_i[x,v] \in [\frac{\delta(u,v)}{2} - Mn^{1-\beta_i-\gamma_i}\ ,\ \frac{\delta(u,v)}{2} + Mn^{1-\beta_i-\gamma_i}]\ .$$

But for pairs $(u, v) \in C^*$ we have, in particular, that

$$P_i[u,x], P_i[x,v] \in [\frac{d}{2} - 2Mn^{1-\beta-\gamma}\ ,\ \frac{d}{2} + 2Mn^{1-\beta-\gamma}]\ .$$

```
algorithm target-distances($P_i, \beta, \gamma, d$)
for $(u,v) \in V \times V$
    if $P_i[u,v] \in [\frac{d}{2} - 2Mn^{1-\beta-\gamma}\ ,\ \frac{d}{2} + 2Mn^{1-\beta-\gamma}]$
        $S_i[u,v] \leftarrow P_i[u,v] - \lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor$
    else
        $S_i[u,v] \leftarrow +\infty$
$R_i \leftarrow S_i \star S_i$
return $R_i + 2\lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor$
```

Figure 3: Computing distances for pairs in C_i.

Let S_i be the matrix obtained from P_i by replacing each entry not in $[\frac{d}{2} - 2Mn^{1-\beta-\gamma}\ ,\ \frac{d}{2} + 2Mn^{1-\beta-\gamma}]$ with $+\infty$, and by decreasing each remaining entry by $\lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor$. The distance product $R_i = S_i \star S_i$ has, therefore, the property that for each $(u,v) \in C_i$,

$$R_i[u,v] + 2\lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor = \delta(u,v)\ .$$

This procedure is summarized in Algorithm **target-distances** given in Figure 3. As each entry of S_i has value $O(Mn^{1-\beta-\gamma})$ the distance product $S_i \star S_i$ takes $\tilde{O}(Mn^{1-\beta-\gamma+\omega})$ time. The following lemma proves the correctness of our algorithm.

Lemma 4.4 *For each $(u,v) \in C^*$ we have*

$$\delta(u,v) = \min\{\delta_t(u,v)\ ,\ \min_{i=0}^{\lfloor (1-r)\log n \rfloor} R_i[u,v] + 2\lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor\}\ .$$

Furthermore, all the values $\delta(u,v)$ for $(u,v) \in C^*$ are computed in $\tilde{O}(Mn^{1-\beta-\gamma+\omega})$ time.

Proof. Let $(u,v) \in C^*$. If $c(u,v) \geq t$ then we already have $\delta_t(u,v) = \delta(u,v)$ as required. Otherwise, for some i, $t_i/2 \leq c(u,v) < t_i$ in which case we have shown that algorithm **target-distances** applied to P_i gives

$$R_i[u,v] + 2\lfloor \frac{d}{2} - 2Mn^{1-\beta-\gamma} \rfloor = \delta(u,v)\ .$$

Algorithm **target-distances** is applied $O(\log n)$ times, once for each P_i. As a single application of **target-distances** runs in $\tilde{O}(Mn^{1-\beta-\gamma+\omega})$ time, the claim regarding the running time follows. ■

4.4 Running time

It remains to show that all ingredients of our algorithm run in $\tilde{O}(M^{\omega/(\omega+1)} n^{(\omega^2+3)/(\omega+1)})$ time.

The first part of our algorithm, given as Lemma 4.1, runs in $\tilde{O}(n^{2+\beta} + Mn^{\omega})$ time. Recall that we can assume that $M \leq n^{3-\omega}$ (as otherwise the usual cubic algorithms are faster, as mentioned in the introduction) so by (1), $Mn^\omega = O(n^{2+\beta})$ and also

$$n^{2+\beta} = M^{\frac{\omega}{\omega+1}} n^{2+\frac{(\omega-1)^2}{\omega+1}} = M^{\frac{\omega}{\omega+1}} n^{\frac{\omega^2+3}{\omega+1}} .$$

Hence this part of the algorithm satisfies the claimed running time assertion.

Computing a (β_i, γ_i)-RPDM (and recall that we perform this $O(\log n)$ times) takes $\tilde{O}(Mn^\omega + Mn^{2+(\omega-2)(\beta_i+\gamma_i)+\gamma_i})$ time, as shown by Lemma 3.3. Using (2) it is straightforward to verify that

$$Mn^{2+(\omega-2)(\beta_i+\gamma_i)+\gamma_i} = Mn^\omega n^{(1-\beta_i)/\omega} .$$

As $\beta_i \geq \beta_0 = \beta$ the worst case occurs when $i = 0$. Plugging in the value of n^β from (1) we obtain that

$$Mn^{2+(\omega-2)(\beta+\gamma)+\gamma} = M^{\frac{\omega}{\omega+1}} n^{\frac{\omega^2+3}{\omega+1}} .$$

Hence this does not exceed the claimed running time assertion.

The next part is algorithm **additive-approximate** whose running time is $\tilde{O}(n^{\gamma_i+\omega(1,1-\gamma_i,1)})$ as stated in Lemma 4.2. We can use Lemma 2.3, but even if we use the naive bound $\omega(1, 1-\gamma_i, 1) \leq 2 + (\omega-2)(1-\gamma_i)$ we can see that this part is not a bottleneck of the algorithm. Indeed, since $\gamma_i \leq \gamma_0 = \gamma$ we have

$$n^{\gamma_i+\omega(1,1-\gamma_i,1)} \leq n^{\gamma_i+2+(\omega-2)(1-\gamma_i)} = n^{(3-\omega)\gamma_i+\omega} \leq n^{(3-\omega)\gamma+\omega} = n^{(3-\omega)\frac{\omega-1}{\omega}(1-\beta)+\omega} =$$

$$n^{(3-\omega)\frac{\omega-1}{\omega}+\omega} \left(M^{\frac{w}{\omega+1}} n^{\frac{(\omega-1)^2}{\omega+1}} \right)^{-(3-\omega)\frac{\omega-1}{\omega}} =$$

$$n^{\frac{\omega^3-6\omega^2+16\omega-9}{\omega+1}} M^{-\frac{(3-\omega)(\omega-1)}{\omega+1}} \leq n^{\frac{\omega^2+3}{\omega+1}}$$

where the last inequality is valid for all $\omega \leq 3$ (and is a strict inequality for $2 < \omega < 3$).

The final part is algorithm **target-distances** given as Lemma 4.4. It runs in $\tilde{O}(Mn^{1-\beta-\gamma+\omega})$ time. Plugging in the values from (1) and (2) we obtain

$$Mn^{1-\beta-\gamma+\omega} = M^{\omega/(\omega+1)} n^{(\omega^2+3)/(\omega+1)} .$$

We have established the claimed running time of our algorithm, thereby concluding the proof of Theorem 1.1. ■

5 Proof of Theorem 1.2

For a nonnegative integer k and for a positive integer M we define a set of nonnegative integers $F(k, M)$ recursively as follows.

$$F(k, M) = \begin{cases} \{0, \ldots, k\} & \text{if } k \leq M+1, \\ \{k\} \bigcup_{i=\lfloor (k-M)/2 \rfloor}^{\lceil (k+M)/2 \rceil} F(i, M) & \text{if } k > M+1. \end{cases}$$

For example, $F(100, 4) = \{100, 52, \ldots, 48, 28, \ldots, 22, 16, \ldots, 0\}$. Observe that $F(k, M)$ consists of $O(\log(k+M))$ intervals of consecutive integers.

Lemma 5.1 $|F(k,M)| = O(M \log k)$.

Proof. Assume that $k > 3M$ otherwise the claim clearly holds. The j'th level of the recursion defining $F(k,M)$ consists of unions of sets $F(i,M)$ where i is contained in the interval $[k/2^j - M - 1, k/2^j + M + 1]$. As there are $O(log(M+k))$ levels of recursion, the claim follows. ■

For a directed graph $G = (V,E)$ with edge weights in $\{1,\ldots,M\}$ let A_k denote the Boolean matrix whose rows and columns are indexed by V, and $A_k[u,v] = 1$ if and only if $\delta(u,v) \leq k$. Also define A_0 to be the identity matrix. For a given threshold value d our goal is to compute A_d.

Our algorithm will, in fact, compute all the matrices A_i for $i \in F(d,M)$. In particular, as $d \in F(d,M)$, we will eventually obtain the required A_d.

For $i \in F(d,M)$ we say that i is *primal* if $i \leq M+1$. We say that i belongs to *level* j if $F(i,M)$ appears in the j'th level of the recursion (observe that i may belong to more than one level). We denote by $L(d,M,j)$ the interval of consecutive integers forming the j'th level. As observed in the proof of Lemma 5.1, $L(d,M,j) \subset [d/2^j - M - 1, d/2^j + M + 1]$ so in particular $|L(d,M,j)| \leq 2M+3$. For example, consider again $F(100,4)$. Then 100 is at level 0. Level 1 consists of $\{48,\ldots,52\}$. Level 2 consists of $\{22,\ldots,28\}$. Level 3 consists of $\{9,\ldots,16\}$. Level 4 consists of $\{2,\ldots,10\}$. Level 5 consists of $\{1,\ldots,7\}$ (recursion continues only for $i > M+1$ so for $i \geq 6$ in this example). Level 6 consists of $\{1,\ldots,6\}$. Finally, level 7 consists of $\{1,\ldots,5\}$ and is the last level as it consists only of primal values.

The following lemma essentially proves Theorem 1.2.

Lemma 5.2 *Given all the matrices A_i for $i \in L(d,M,j+1) \cup \{0,\ldots,M+1\}$ the set of all matrices A_k for $k \in L(d,M,j)$ can be computed in $\tilde{O}(Mn^\omega)$ time.*

Proof. Consider some matrix A_k for $k \in L(d,M,j)$. If $k \leq M+1$ there is nothing to prove so assume that $k > M+1$. Suppose that u and v are vertices such that $\delta(u,v) \leq k$, and consider a shortest path from u to v. Let w be the first vertex on this path for which $\delta(u,w) \geq \lfloor (k-M)/2 \rfloor$. If there is no such vertex, then already $\delta(u,v) < \lfloor (k-M)/2 \rfloor$. Otherwise, since each weight is in $\{1,\ldots,M\}$, then $\delta(u,w) \leq \lfloor (k-M)/2 \rfloor + M - 1$. Furthermore, $\delta(w,v) \leq k - \lfloor (k-M)/2 \rfloor$. In any case A_k is just the Boolean **or** of the following Boolean Matrix products:

$$A_k = \vee_{i=\lfloor (k-M)/2 \rfloor}^{\lceil (k+M)/2 \rceil} A_i A_{k-i} \ . \tag{3}$$

By the definition of $F(d,M)$, if $k \in L(d,M,j)$ then all the indices i and $k-i$ in (3) belong to $L(d,M,j+1)$. Since Boolean matrix multiplication can be performed in $O(n^\omega)$ time, (3) shows that we can compute A_k in $O(Mn^\omega)$ time.

The only problem that remains is that we do not want to compute a single A_k. We want all A_k for all $k \in L(d,M,j)$ and as $|L(d,M,j)| = O(M)$ this takes $O(M^2 n^\omega)$ if we compute each A_k separately.

To overcome this problem we use the following matrix convolution idea. We construct a single matrix B that encodes all the matrices A_i for $i \in L(d,M,j+1)$ at once. Set $s = |L(d,M,j+1)|$ and let t be the smallest index in $|L(d,M,j+1)|$. Recalling that $L(d,M,j+1)$ is an interval of consecutive integers we have $L(d,M,j+1) = \{t, t+1, \ldots, t+s-1\}$. Let $B[u,v]$ be the following polynomial of degree at most $s-1$.

$$B[u,v] = \sum_{q=0}^{s-1} A_{q+t}[u,v] x^q \ .$$

Thus, the coefficient of x^q encodes the matrix A_{q+t}.

Now consider $C = B^2$ (product performed over the ring of polynomials in a single variable). Each element of C is therefore a polynomial of degree at most $2s-2$. Consider the coefficient of x^{k-2t} in $C[u,v]$. The only way it can be non-zero is if for some i, A_iA_{k-i} is nonzero in entry $[u,v]$. Hence, we obtain A_k just by examining the coefficients of x^{k-2t} in the entries of C.

The only thing that remains is to consider the complexity of computing C. Two matrices whose entries are polynomials of degree at most $s-1$ and whose coefficients are bounded integers (in our case the coefficients are either 0 and 1) can be multiplied in $\tilde{O}(sn^{\omega})$ time. The standard trick is to replace the variable x with a large number (say $n+1$ if the dimension of the matrices is n) so that no carry is introduced when reading the product entries as digits in base $n+1$, and thereby constructing the polynomials in the entries of the product. However, observe that replacing the variable x with $n+1$ causes the entries to become as large as $O(n^s)$ and hence consist of $O(s \log n)$ bits. Thus, each matrix operation incurs an $\tilde{O}(s)$-factor that cannot be ignored.

In our case we have $s = O(M)$ so we conclude that all the matrices A_k for $k \in L(d, M, j)$ can be computed in $\tilde{O}(Mn^{\omega})$ time. ■

The proof of Theorem 1.2 now follows from Lemma 5.2 by recalling that the number of levels is $O(\log(d+M))$. The only thing that remains is to compute the matrices $A_1, \ldots A_{M+1}$ that correspond to the primal indices. This, however, is relatively easy to do using distance products. In fact, the following lemma is a consequence of the result of Zwick from [17].

Lemma 5.3 *Let G be a graph with n vertices and integer edge weights in $\{1, \ldots, M\}$. There is an $\tilde{O}(Mn^{\omega})$ time algorithm that computes $\delta(u,v)$ for all pairs (u,v) which satisfy $\delta(u,v) \le M+1$.*

Proof. Any pair that has $\delta(u,v) \le M+1$ has at most $M+1$ edges on any shortest path from u to v. The result of [17], specifically, algorithm **rand-short-path**, its complexity analysis, and Lemma 4.2 therein, show that the exact distances for such pairs is computed in $\tilde{O}(Mn^{\omega})$ time. We note that during the computations of the distance products in each iteration of **rand-short-path** we never need to consider matrix entries with values exceeding $M+1$. As the bridging sets of each iteration are of size $\tilde{O}(n/s)$ (see Lemma 4.2 in [17]), the time to compute the distance product of the rectangular matrices, even without resorting to fast rectangular matrix multiplications, is $\tilde{O}(M(n/s)^{\omega}s^2)$. As in our case we always have $s = O(M)$, the result follows. ■

Observe that once we have $\delta(u,v)$ for all pairs (u,v) with $\delta(u,v) \le M+1$ then we immediately have the matrices $A_1, \ldots, A_{M+1}$, as required. This concludes the proof that all A_k for $k \in F(d,M)$ are computed in $\tilde{O}(Mn^{\omega})$ time, and hence the proof of Theorem 1.2.

6 Concluding remarks

We presented an algorithm that computes the diameter (and Threshold APSP) of an integer weighted directed graph polynomially faster than any presently known APSP algorithm. The algorithm is randomized and returns, with high probability, the *precise* diameter, as well as all pairs of vertices realizing it.

Obtaining a truly subcubic algorithm for computing the diameter (moreover Threshold APSP) of real-weighted graphs remains an open problem. The prospects in this case, however, seem gloomier. It is likely that Diameter and APSP are equivalent under sub-cubic reductions. A recent result of Vassilevska Williams and Williams [14] asserts that the existence of a truly subcubic algorithm for real-weighted APSP is equivalent under subcubic reductions to the existence of truly subcubic algorithms for a list of problems that seem "lighter" than APSP.

Acknowledgments

I thank Uri Zwick for some useful discussions. Special thanks to Andreas Björklund for fruitful discussions and insightful comments.

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*, 28:1167–1181, 1999.

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

[3] T.M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ACM/SIAM, 514–523, 2006.

[4] T.M. Chan. More algorithms for All-Pairs Shortest Paths in weighted graphs. *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, ACM Press, 590–598, 2007.

[5] F.R.K Chung. Diameters of graphs: Old problems and new results. Congressus Numerantium, 60:295–317, 1987.

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

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

[8] Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134:103–139, 1997.

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

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

[11] P. Sankowski. Shortest Paths in Matrix Multiplication Time. *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, LNCS, 770–778, 2005.

[12] R. Seidel. On the All-Pairs-Shortest-Path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[13] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 605-614, 1999.

[14] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix, and triangle problems. *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 645–654, 2010.

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

[16] R. Yuster and U. Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 389–396, 2005.

[17] U. Zwick. All Pairs Shortest Paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.