Centralized, Parallel, and Distributed Multi-Source Shortest Paths via Hopsets and Rectangular Matrix Multiplication

Michael Elkin
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Ofer Neiman
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Consider an undirected weighted graph $G = (V, E, w)$. We study the problem of computing $(1+\epsilon)$-approximate shortest paths for $S \times V$, for a subset $S \subseteq V$ of $|S| = n^r$ sources, for some $0 < r \leq 1$. We devise a significantly improved algorithm for this problem in the entire range of parameter r, in both the classical centralized and the parallel (PRAM) models of computation, and in a wide range of r in the distributed (Congested Clique) model. Specifically, our centralized algorithm for this problem requires time $\tilde{O}(|E| \cdot n^{o(1)} + n^{\omega(r)})$, where $n^{\omega(r)}$ is the time required to multiply an $n^r \times n$ matrix by an $n \times n$ one. Our PRAM algorithm has polylogarithmic time $(\log n)^{O(1/\rho)}$, and its work complexity is $\tilde{O}(|E| \cdot n^{\rho} + n^{\omega(r)})$, for any arbitrarily small constant $\rho > 0$.

In particular, for $r \leq 0.313\ldots$, our centralized algorithm computes $S \times V$ $(1+\epsilon)$-approximate shortest paths in $n^{2+o(1)}$ time. Our PRAM polylogarithmic-time algorithm has work complexity $O(|E| \cdot n^{\rho} + n^{2+o(1)})$, for any arbitrarily small constant $\rho > 0$. Previously existing solutions either require centralized time/parallel work of $O(|E| \cdot |S|)$ or provide much weaker approximation guarantees.

In the Congested Clique model, our algorithm solves the problem in polylogarithmic time for $|S| = n^r$ sources, for $r \leq 0.655$, while previous state-of-the-art algorithms did so only for $r \leq 1/2$. Moreover, it improves previous bounds for all $r > 1/2$. For unweighted graphs, the running time is improved further to $\text{poly}(\log \log n)$ for $r \leq 0.655$. Previously this running time was known for $r \leq 1/2$.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Shortest paths, matrix multiplication, hopsets

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.27

Funding *Michael Elkin*: Funded by ISF grant 2344/19.
Ofer Neiman: Funded by ISF grant 1817/17.

Acknowledgements We are grateful to François Le Gall for explaining us certain aspects of the algorithm of [27], and to Shaked Matar for helpful discussions.

1 Introduction

We consider the problem of computing $(1+\epsilon)$-*approximate shortest paths* (henceforth, $(1+\epsilon)$-ASP) in undirected weighted graphs $G = (V, E, w)$, $|V| = n$, for an arbitrarily small $\epsilon > 0$. We study this problem in the centralized, parallel (PRAM) and distributed (Congested Clique) models of computation. Our focus is on computing $(1+\epsilon)$-ASP for $S \times V$, for a set $S \subseteq V$ of *sources*, $|S| = n^r$, for a constant parameter $0 < r \leq 1$.

This is one of the most central, fundamental and intensively studied problems in Graph Algorithms. Most of the previous research concentrated on one of the two following scenarios: the *single-source* ASP (henceforth, approximate SSSP), i.e., the case $|S| = 1$, and the *all-pairs* ASP (henceforth, APASP), i.e., the case $S = V$.

We next overview most relevant previous results and our contribution in the centralized model of computation, and then turn to the PRAM and distributed models.

1.1 Centralized Model

The classical algorithm of Dijkstra solves *exact* SSSP problem in time $O(|E| + n \log n)$ [22]. Thorup [36] refined this bound to $O(|E| + n \log\log n)$ when weights are integers. Employing these algorithms for the ASP problem for $S \times V$ results in running time of $O(|S|(|E| + n \log\log n))$. In the opposite end of the spectrum, Galil and Margalit [24], Alon et al. [2] and Zwick [40] showed that one can use fast matrix multiplication (henceforth, FMM) to solve $(1+\epsilon)$-APASP in time $\tilde{O}(n^\omega)$, where ω is the matrix multiplication exponent. (n^ω is the time required to multiply two $n \times n$ matrices. The currently best-known estimate on ω is $\omega < 2.372\ldots$ [38, 25, 14, 1].)

By allowing larger approximation factors, one can achieve a running time of $\tilde{O}(n^2)$ for APASP.[1] Specifically, Cohen and Zwick [12] devised an algorithm for 3-APASP with this running time, and Baswana and Kavitha [5] refined the approximation ratio to $(2, w)$. The notation $(2, w)$ means that for a vertex pair (u, v), their algorithm provides an estimate with a multiplicative error of 2, and an additive error bounded by the maximal weight of an edge on some shortest $u - v$ path in the graph.

Cohen [11], Elkin [17], and Gitlitz and the current authors [18] also showed that one can obtain a $(1+\epsilon, \beta \cdot w)$-approximation for the ASP problem for $S \times V$ in time $O(|E| \cdot n^\rho + |S| \cdot n^{1+1/\kappa})$, where $\beta = \beta(\epsilon, \kappa, \rho)$ is a quite large constant (as long as $\epsilon > 0, \rho > 0, 1/\kappa > 0$ are constant), and w is as in the result of Baswana and Kavitha [5].

However, if one insists on a purely multiplicative error of at most $1+\epsilon$, for an arbitrarily small constant $\epsilon > 0$, then for dense graphs ($|E| = \Theta(n^2)$), the best-known running time for ASP for $S \times V$ is $\tilde{O}(\min\{|S| \cdot n^2, n^\omega\})$ (the first term can be achieved by running Dijsktra from every source, the second term by using $(1+\epsilon)$-APASP). In the current paper we devise an algorithm that solves the problem in $\tilde{O}(n^{\omega(r)} + |E| \cdot n^{o(1)})$ time,[2] where $\omega(r)$ is the matrix multiplication exponent of *rectangular matrix multiplication*. That is, $n^{\omega(r)}$ is the time required to multiply an $n^r \times n$ matrix by an $n \times n$ matrix. Coppersmith [13] showed that for $r \leq 0.291$, $\omega(r) \leq 2 + o(1)$, and Le Gall and Urrutia [27] improved this bound further to $r \leq 0.313$. Denote $\alpha \geq 0.313$ as the maximal value such that $\omega(\alpha) \leq 2 + o(1)$. Therefore, our algorithm solves $(1+\epsilon)$-ASP problem for $S \times V$ in $n^{2+o(1)}$ time, as long as $|S| = O(n^\alpha)$. Moreover, the bound on our running time grows gracefully from $n^{2+o(1)}$ to n^ω, as the number of sources $|S|$ increases from n^α to n. When $S = V$, our bound matches the bound of Zwick [40]. See Table 1.

Furthermore, Dor et al. [15] showed that any $(2-\epsilon)$-ASP algorithm for $S \times V$ that runs in $T(n)$ time, for any positive constant $\epsilon > 0$ and any function $T(n)$, translates into an algorithm with running time $T(O(n))$ that multiplies two Boolean matrices with dimensions $|S| \times n$ and $n \times n$. Thus, the running time of our algorithm cannot be improved by more than a factor of $n^{o(1)}$ without improving the best-known algorithm for multiplying (rectangular) Boolean matrices.

In terms of edge weights, the situation with our algorithm is similar to that with the algorithm of Zwick [40]. Both algorithms apply directly to graphs with polynomially-bounded edge weights. Nevertheless, we argue that both of them can be used in conjunction with the Klein-Sairam's reduction of weights [32] to provide the same bounds for graphs with arbitrary weights.

---

[1] By $\tilde{O}(f(n))$ we mean $O(f(n) \cdot \log^{O(1)} f(n))$.

[2] In fact, our result holds for arbitrary $0 < \epsilon < 1$, see Theorem 5.

Table 1 Results on $(1+\epsilon)$-ASP for $S \times V$ in the centralized model for weighted graphs (previous running time is for dense graphs).

# of sources	Our running time	Previous running time
$n^{0.1}$	$n^{2+o(1)}$	$n^{2.1}$
$n^{0.2}$	$n^{2+o(1)}$	$n^{2.2}$
$n^{0.3}$	$n^{2+o(1)}$	$n^{2.3}$
$n^{0.4}$	$n^{2.011}$	$n^{2.373}$
$n^{0.5}$	$n^{2.045}$	$n^{2.373}$
$n^{0.6}$	$n^{2.094}$	$n^{2.373}$
$n^{0.7}$	$n^{2.154}$	$n^{2.373}$
$n^{0.8}$	$n^{2.222}$	$n^{2.373}$
$n^{0.9}$	$n^{2.296}$	$n^{2.373}$
n^{1}	$n^{2.373}$	$n^{2.373}$

1.2 Parallel Model

The situation in the parallel setting (PRAM) is similar to that in the centralized setting. The first parallel $(1+\epsilon)$-SSSP algorithm with polylogarithmic time (specifically, $(\log n)^{\tilde{O}((\log 1/\rho)/\rho)}$ and $O(|E| \cdot n^{\rho})$ work, for any arbitrarily small constant parameter $\rho > 0$, was devised by Cohen [11]. Her bounds were improved in the last five years by [20, 21, 33, 3, 19], culminating in polylogarithmic time and $\tilde{O}(|E|)$ work [33, 3]. All these aforementioned algorithms are randomized, except for the deterministic algorithm of Elkin and Matar [19] that requires polylogarithmic time $(\log n)^{O(\mathbf{1}/\rho)}$ and work $\tilde{O}(|E| \cdot n^{\rho})$.

On the opposite end of the spectrum, algorithms of Galil and Margalit [24], Alon et al. [2], and Zwick [40] (based on FMM) can be used in the PRAM setting. They give rise to deterministic polylogarithmic time $\tilde{O}(n^{\omega})$ work [40] for the $(1+\epsilon)$-APASP problem.

By using sparse spanners, the algorithm of Cohen [11] in conjunction with that of Baswana and Sen [4] provides polylogarithmic time and $O(|E| \cdot n^{1/\kappa} + |S| \cdot n^{1+1/\kappa})$ work for $(2+\epsilon)\kappa$-ASP for $S \times V$, where $\kappa = 1, 2, \ldots$ is a parameter. Recently, Gitlitz and the current authors [18] also showed that one can have $(1+\epsilon, \beta \cdot w)$-ASP for $S \times V$ in polylogarithmic time and $O(|E| \cdot n^{\rho} + |S| \cdot n^{1+1/\kappa})$ work, where $\beta = \beta(\epsilon, \kappa, \rho)$ is a large constant (as long as $\epsilon, \rho, 1/\kappa > 0$ are constant), and w is as above.

Nevertheless, if one insists on a purely multiplicative error of at most $1+\epsilon$, currently best-known solutions for the ASP problem for $S \times V$ that run in polylogarithmic time require work at least $\Omega(\min\{|S| \cdot |E|, n^{\omega}\})$. Our parallel algorithm for the problem with $|S| = n^r$ sources, $0 < r \leq 1$, has polylogarithmic time $(\log n)^{O(1/\rho)}$ and work $\tilde{O}(n^{\omega(r)} + |E| \cdot n^{\rho})$, for any arbitrarily small constant $\rho > 0$. Similarly to the centralized setting, this results in work $n^{2+o(1)} + \tilde{O}(|E| \cdot n^{\rho})$, for any arbitrarily small constant $\rho > 0$, as long as $|S| = O(n^{\alpha})$, $\alpha = 0.313$, and it improves Zwick's bound [40] of n^{ω} (which applies for $(1+\epsilon)$-APASP) for all values of $r < 1$. The aforementioned reduction of [15] implies that the work complexity of our algorithm cannot be improved by more than a factor of $n^{o(1)}$ without improving the best-known centralized algorithm for multiplying (rectangular) Boolean matrices.

Our algorithm uses FMM and hopsets. The ingredient that builds hopsets is randomized, but by using a new deterministic construction of hopsets from [19], one can make it deterministic, with essentially the same bounds. As a result our ultimate $(1+\epsilon)$-ASP algorithms (both centralized and parallel ones) become deterministic.

1.3 Distributed Model

In the Congested Clique model, every two vertices of a given n-vertex graph $G = (V, E)$, may communicate in each round by a message of $O(\log n)$ bits. The running time of an algorithm is measured by the number of rounds. Computing shortest paths in this model has been extensively studied in the last decade. An exact APSP algorithm was devised in [10] with running time $O(n^{1-2/\omega}) = O(n^{0.158\cdots})$ for unweighted undirected graphs (or with $1+\epsilon$ error in weighted directed graphs), and in $\tilde{O}(n^{1/3})$ time for weighted directed graphs. The latter result was improved in [26] to $n^{0.209}$ when the weights are constant.

The first algorithm with polylogarithmic time for weighted undirected graphs was devised by [6], who showed a $(1+\epsilon)$-approximate *single-source* shortest paths algorithm. In [9], among other results, a $(1+\epsilon)$-ASP algorithm with polylogarithmic time was shown for a set of $\tilde{O}(n^{1/2})$ sources. For unweighted graphs, the running time was recently improved by [16] to $\text{poly}(\log\log n)$, with a similar restriction of $O(n^{1/2})$ sources.

In the current paper we obtain an algorithm for the $(1+\epsilon)$-ASP in the Congested Clique model for weighted undirected graphs with polylogarithmic time, for a set of $|S| = O(n^{\frac{1+\alpha}{2}}) = O(n^{0.655\cdots})$ sources. For larger sets of sources, our running time gracefully increases until it reaches $\tilde{O}(n^{0.158})$ time when $S = V$ (see Table 2). Denoting $|S| = n^r$, our algorithm outperforms the state-of-the-art bound of [9] for all $0.5 < r < 1$. In the case of unweighted graphs, we provide a similar improvement over the result of [16]: our $(1+\epsilon)$-ASP algorithm has $\text{poly}(\log\log n)$ time, allowing up to $n^{0.655}$ sources.

Table 2 Results on $(1+\epsilon)$-ASP for $S \times V$ in the Congested Clique model (for any constant $\epsilon > 0$, and hiding constants and lower order terms).

# of sources	Our running time	Running time of [9]	Running Time of [10]
$n^{0.5}$	$\tilde{O}(1)$	$\tilde{O}(1)$	$n^{0.158}$
$n^{0.6}$	$\tilde{O}(1)$	$n^{0.06}$	$n^{0.158}$
$n^{0.7}$	$n^{0.006}$	$n^{0.13}$	$n^{0.158}$
$n^{0.8}$	$n^{0.04}$	$n^{0.2}$	$n^{0.158}$
$n^{0.9}$	$n^{0.1}$	$n^{0.26}$	$n^{0.158}$
n^{1}	$n^{0.158}$	$n^{1/3}$	$n^{0.158}$

1.4 Additional Results

We also devise an algorithm for the $(1+\epsilon)$-approximate *k-nearest neighbors* (henceforth, k-NN) problem in PRAM. Here k, $1 \le k \le n$, is a parameter. For a vertex v, let $z_1, z_2, \ldots$ be all other vertices ordered by their distance from v in non-decreasing order, with ties broken arbitrarily. A vertex u is *in the $(1+\epsilon)$-approximate k-NN of v* if it is no farther from v than $(1+\epsilon)d_G(v, z_k)$. The objective is to compute $(1+\epsilon)$-approximate shortest paths for some set $\mathcal{P}$ of pairs of vertices, that for every vertex $u \in V$ contains at least k pairs (u, v) with v being in the $(1+\epsilon)$-approximate k-NN of v. Our algorithm for this problem applies even in *directed* weighted graphs. It requires polylogarithmic time and $\tilde{O}(\min\{n^{\omega}, k^{0.702}n^{1.882} + n^{2+o(1)}\})$ work. For $k = O(n^{0.168})$, this work is $n^{2+o(1)}$, and for $k = o(n^{0.698})$, this bound is better than n^{ω}, i.e., it improves the bound for $(1+\epsilon)$-APASP problem.

From technical viewpoint, in this result we adapt a centralized algorithm of Yuster and Zwick [39] for sparse matrix multiplication to the PRAM setting. We then employ this algorithm in conjunction with the observation due to Censor-Hillel et al. [9] that the k-NN

problem boils down to computing k matrix products of sparse matrices. We generalize this observation and argue that this is the case not only for the exact k-NN problem, but also for its approximate variant.

1.5 Technical Overview

As was mentioned above, our algorithms employ hopsets. A graph $H = (V, E', w')$ is a $(1+\epsilon, \beta)$-*hopset* for a graph $G = (V, E, w)$, if for every vertex pair $u, v \in V$, we have

$$d_G(u,v) \leq d^{(\beta)}_{G \cup H}(u,v) \leq (1+\epsilon) d_G(u,v) \ . \tag{1}$$

Here $d^{(\beta)}_{G\cup H}(u,v)$ stands for β-*bounded distance* between u and v in $G \cup H$, i.e., the length of the shortest $u-v$ path between them with at most β edges (henceforth, β-*bounded path*).

Our algorithm is related to the algorithm of [9], designed for $(1+\epsilon)$-ASP for $S \times V$ in the distributed Congested Clique (henceforth, CC) model. Their algorithm starts with computing a $(1+\epsilon, \beta)$-hopset H for the input graph G. It then adds H to G, and creates an adjacency matrix A of $G \cup H$. It then creates a matrix B of dimensions $|S| \times n$, whose entries $B_{u,v}$, for $(u,v) \in S \times V$, are defined as $w(u,v)$ if $(u,v) \in E$, and ∞ otherwise. Then the algorithm computes distance products $B \star A, (B \star A) \star A, \ldots, (B \star A^{\beta-1}) \star A = B \star A^{\beta}$. By equation (1), $B \star A^{\beta}$ is a $(1+\epsilon)$-approximation of all distances in $S \times V$.

Censor-Hillel et al. [9] developed an algorithm for efficiently multiplying *sparse* matrices in the distributed CC model. They view the matrices $B, B \star A, \ldots, B \star A^{\beta-1}$, as sparse square $n \times n$ matrices, and as a result compute $B \star A^{\beta}$ efficiently via their (tailored to the CC model) algorithm. In particular, their algorithm does not use Strassen-like fast matrix multiplication (FMM) techniques, but rather focuses on carefully partitioning all the products that need to be computed in a naive matrix product of dimensions $|S| \times n$ by $n \times n$ among n available processors.

Our first observation is that this product can be computed much faster using best available fast *rectangular* matrix multiplication (FRMM) algorithms. This observation leads to our $(1+\epsilon)$-ASP algorithms for weighted graphs that significantly improve the state-of-the-art in all the three computational models that we consider (the centralized, PRAM, and distributed CC). We also need to convert matrix distance products into ordinary algebraic matrix products. This is, however, not difficult, and was accomplished, e.g., in [40]. We employ the same methodology (of [40]). Our algorithm then employs a fast *rectangular* MM in this model due to Le Gall [26]. This leads to our improved $(1+\epsilon)$-ASP algorithms in the distributed CC model (cf. Table 2).

Remarkably, while so far hopsets were used extensively in parallel/distributed/dynamic/streaming settings [11, 7, 34, 28, 29, 20, 21, 9], there were no known applications of hopsets in the classical centralized setting. Our results demonstrate that this powerful tool is extremely useful in the classical setting as well.

1.6 Organization

After reviewing some preliminary results in Section 2, we describe our algorithm for $(1+\epsilon)$-ASP for $S \times V$ in the standard centralized model in Section 3. In Section 5 we provide our algorithm for $(1+\epsilon)$-ASP for $S \times V$ in the Congested Clique model that substantially improves the number of allowed sources while maintaining polylogarithmic time (and $\mathrm{poly}(\log\log n)$ time, for unweighted graphs). In Section 6 we devise a PRAM algorithm for $(1+\epsilon)$-ASP for $S \times V$. In Section 7 we analyze the weight reduction of [31] in the context of our algorithm and the algorithm of [40]. Finally, in Appendix A we describe our PRAM algorithm for approximate distances to k-NN.

2 Preliminaries

Matrix Multiplication and Distance Product. Fix an integer n. For $0 \leq r \leq 1$, let $w(r)$ denote the exponent of n in the number of algebraic operations required to compute the product of an $n^r \times n$ matrix by an $n \times n$ matrix.

Let $1 \leq s, q \leq n$. Let A be an $s \times n$ matrix. We denote the entry in row i and column j of the matrix A by A_{ij}. The transpose of A is A^T. We use * to denote a wildcard, e.g., the notation A_{*j} refers to the vector which is the j-th column of A. For an $n \times q$ matrix B, define the distance product $C = A \star B$ by

$$C_{ij} = \min_{1 \leq k \leq n} \{A_{ik} + B_{kj}\} ,$$

for $1 \leq i \leq s$ and $1 \leq j \leq q$. For a parameter $\delta > 1$, we say that C' is a δ-approximation to C if for all i, j, $C_{ij} \leq C'_{ij} \leq \delta \cdot C_{ij}$.

The following theorem is an extension of a result from [40]. The latter applies to square matrices. We extend it to rectangular matrices, and argue that it is also applicable in a parallel setting.

▶ **Theorem 1** ([40]). *Let M, R be positive integers. Let A be an $n^r \times n$ matrix and B an $n \times n$ matrix, whose entries are all in $\{1, ..., M\} \cup \{\infty\}$. Then there is an algorithm that computes a $(1 + \frac{1}{R})$-approximation to $A \star B$ in deterministic time $\tilde{O}(R \cdot n^{w(r)} \cdot \log M)$.*

Proof. It was observed in [2] that the distance product $C = A \star B$ can be computed by defining $\hat{A}_{ij} = (n+1)^{M - A_{ij}}$ and, similarly, $\hat{B}_{ij} = (n+1)^{M - B_{ij}}$. Then C can be derived from $\hat{C} = \hat{A} \cdot \hat{B}$ by $C_{ij} = 2M - \lfloor \log_{n+1} \hat{C}_{ij} \rfloor$. Since the values of entries in the matrices $\hat{A}$ and $\hat{B}$ are of size $O(M \log n)$, and each algebraic operation (when computing the standard product $\hat{A} \cdot \hat{B}$) requires $\tilde{O}(M \log n)$ time, it follows that the running time is $\tilde{O}(M \cdot n^{w(r)})$.

Next, we show that the running time can be reduced to $\tilde{O}(R \cdot \log M \cdot n^{w(r)})$, at the expense of allowing $(1 + 1/R)$-approximation of the entries of $A \star B$.

If $R \geq M$ then our algorithm computes the exact distance product in $\tilde{O}(M \cdot n^{w(r)}) = \tilde{O}(R \cdot \log M \cdot n^{w(r)})$ time, and we are done.

Thus, we henceforth assume that $M > R$. We will also assume for simplicity that both R and M are integer powers of 2. If it is not the case, we can increase them by a factor at most 2, and guarantee this property. This increases the running time by at most a constant factor.

For each integer r, $\log_2 R \leq r \leq \log_2 M$, we define scaled matrices $A'(r), B'(r)$, by setting $A'_{ij}(r) = \lceil \frac{R}{2^r} \cdot A_{ij} \rceil$, if $A_{ij} \leq 2^r$, and setting it to ∞ otherwise. The entries $B'_{ij}(r)$ are defined analogously (with respect to B). Note that $A'_{ij}(r), B'_{ij}(r) \in \{0, 1, \ldots, R\} \cup \{\infty\}$.

We then compute the product matrices $C'(r) = A'(r) \star B'(r)$, for all $\log_2 R \leq r \leq \log_2 M$. Finally, the matrix C' is computed as entry-wise minimum of all the matrices $\frac{2^r}{R} \cdot C'(r)$. Note that we invoke $O(\log M)$ distance products of matrices with entries in the range $\{0, 1, \ldots, R\} \cup \{\infty\}$, and thus the overall running time is $\tilde{O}(\log M \cdot R \cdot n^{w(r)})$.

Observe that the matrix C' is entry-wise greater or equal than the matrix $C = A \star B$. In fact, this is the case for each of the matrices $\frac{2^r}{R} \cdot C'(r)$, as

$$\begin{aligned} C_{ij} &= \min_{1 \leq k \leq n} \{A_{ik} + B_{kj}\} \\ &\leq \frac{2^r}{R} \cdot \min_{1 \leq k \leq n} \{\lceil \frac{R}{2^r} \cdot A_{ik} \rceil + \lceil \frac{R}{2^r} \cdot B_{kj} \rceil\} \leq \frac{2^r}{R} \cdot C'_{ij}(r) . \end{aligned}$$

For the inequality in the opposite direction, consider some fixed pair of indices i, j, and let k be the witness for C_{ij}, i.e., $C_{ij} = A_{ik} + B_{kj}$. Assume without loss of generality that $A_{ik} \leq B_{kj}$. (Otherwise the index s below needs to be defined with respect to A_{ik}.) Let s be the positive integer that satisfies $2^{s-1} \leq B_{kj} < 2^s$. (If $B_{kj} = M$, we will however set $s = \log_2 M$. If $B_{kj} = 0$, then it is easy to verify that $C_{ij} = C'_{ij} = 0$.)

If $s \leq \log_2 R$ then for $r = \log_2 R$ we have

$$\begin{aligned} C'_{ij}(r) &= \frac{2^r}{R} \cdot C'_{ij}(r) = \frac{2^r}{R} \cdot \min_{1 \leq t \leq n} \{A'_{it}(r) + B'_{tj}(r)\} \\ &= \min_{1 \leq t \leq n} \{\lceil \frac{R}{2^r} \cdot A_{it} \rceil + \lceil \frac{R}{2^r} \cdot B_{tj} \rceil\} = A_{ik} + B_{kj} = C_{ij} \ . \end{aligned}$$

(Note that all terms in the minimum above are greater or equal than $A_{ik} + B_{kj}$.)

Hence we assume that $\log_2 R < s \leq \log_2 M$. Consider $r = s$. We have

$$\begin{aligned} \frac{2^r}{R} \cdot C'_{ij}(r) &= \frac{2^r}{R} \cdot \min_{1 \leq t \leq n} \{\lceil \frac{R}{2^r} \cdot A_{it} \rceil + \lceil \frac{R}{2^r} \cdot B_{tj} \rceil\} \\ &\leq \frac{2^r}{R} \cdot (\lceil \frac{R}{2^r} A_{ik} \rceil + \lceil \frac{R}{2^r} \rceil B_{kj}) \\ &\leq \frac{2^r}{R} \left(\frac{R}{2^r} A_{ik} + \frac{R}{2^r} B_{kj} + 2 \right) \\ &= (A_{ik} + B_{kj}) + 2 \cdot \frac{2^r}{R} \ . \end{aligned}$$

Recall that $B_{kj} \geq 2^{r-1}$, and thus $2^r \leq 2(A_{ik} + B_{kj})$. Hence

$$\frac{2^r}{R} \cdot C'_{ij}(r) \leq (A_{ik} + B_{kj}) + 4 \cdot \frac{A_{ik} + B_{kj}}{R} = C_{ij} \cdot (1 + 4/R) \ .$$

Hence $C'_{ij} \leq \frac{2^r}{R} \cdot C'_{ij}(r) \leq C_{ij} \cdot (1 + 4/R)$. ◀

▶ Remark 2. The algorithm of Theorem 1 boils down to $O(\log M)$ standard matrix multiplications, and choosing the minimum value for each entry. Thus, we can also apply it in the Congested Clique and PRAM models of computation. In the Congested Clique model, naively, the overhead is $O(R \cdot \log M)$. (See also Section 5 for a refined bound.) In the PRAM model, naively, the time grows by a factor of $O(R \cdot \log M)$. On the other hand, by the Chinese Remainders' theorem, one can also replace each matrix product with entries bounded by n^R by R matrix products with entries bounded by $n^{O(1)}$, and compute these products in parallel. Hence, in fact, the PRAM running time grows by a factor of $O(\log M)$, while the work complexity grows by a factor of $O(R \cdot \log M)$.

Witnesses. Given an $s \times n$ matrix A and an $n \times q$ matrix B, an $s \times q$ matrix W is called a *witness* for $C = A \star B$ if for all i, j, $C_{ij} = A_{iW_{ij}} + B_{W_{ij}j}$. It was shown in [23, 40] how to compute the matrix W in almost the same time required to compute C (up to logarithmic factors). This holds also for a witness for C' which is a c-approximation for C (see [40, Section 8]), for some $c \geq 1$. The witness can assist us in recovering the actual paths, rather than just reporting distance estimates. Since computing witnesses is done by an appropriate distance product, these witnesses can also be efficiently computed in the PRAM model.

Hopsets. Recall the definition of hopsets in the beginning of Section 1.5. A randomized construction of hopsets was gives in [11], see also [34, 29, 20]. The following version was shown in [21].

▶ **Theorem 3** ([21]). *For any weighted undirected graph $G = (V, E)$ on n vertices and parameter $\kappa > 1$, there is a randomized algorithm running in time $\tilde{O}(|E| \cdot n^{1/\kappa})$, that computes a $(1 + \epsilon, \beta)$-hopset H with $\beta = \left(\frac{\kappa}{\epsilon}\right)^{O(\kappa)}$ of size $O(n^{1+1/\kappa})$ (for every $0 < \epsilon < 1$ simultaneously) .*

We note that [19] provides a *deterministic* construction of hopsets with similar properties. There are two differences, which have essentially no effect on our result. First, the hopbound in [19] is $\beta = \left(\frac{\log n}{\epsilon}\right)^{O(\kappa)}$. Second, the construction there accepts $\epsilon > 0$ as a part of its input. Nevertheless, their hopsets can be used to make our results in PRAM deterministic, with essentially the same parameters. Our centralized algorithm can also be made deterministic using a hopset construction from [29].

Eliminating Dependence on Aspect Ratio. The aspect ratio of a graph G is the ratio between the largest to smallest edge weight. A well-known reduction by [32] asserts that to compute $(1+\epsilon)$-approximate shortest paths in $G = (V, E)$ with $|V| = n$, it suffices to compute $(1+\epsilon)$-approximate shortest paths in a collection of at most $\tilde{O}(|E|)$ graphs $\{G_t\}$. The total number of (non-isolated) vertices in all these graphs is $O(n \log n)$, the total number of edges is $\tilde{O}(|E|)$, and the aspect ratio of each graph is $O(n/\epsilon)$. This reduction can be performed in parallel (PRAM EREW) within $O(\log^2 n)$ rounds and work $O(|E|)$. Thus it can also be done in the standard centralized model in $\tilde{O}(|E|)$ time. See also Section 7 and [20, Section 4] for more details. Since in our algorithms the dependence on the aspect ratio will be logarithmic, in the sequel we assume that $M = \text{poly}(n)$.

3 Multi-Source Shortest Paths

Let $G = (V, E, w)$ be a weighted undirected graph and fix a set of s sources $S \subseteq V$. We compute a $(1+\epsilon)$-approximation for all distances in $S \times V$, by executing Algorithm 1.

Algorithm 1 $\texttt{ASP}(G, S, \epsilon)$.

1: Let H be an $(1+\epsilon, \beta)$-hopset for G;
2: Set $R = \beta/\epsilon$;
3: Let A be the adjacency matrix of $G \cup H$;
4: Let $B^{(1)} = A_{S*}$;
5: **for** t from 1 to $\beta - 1$ **do**
6: Let B' be a $(1 + 1/R)$-approximation to $B^{(t)} \star A$;
7: Let $B^{(t+1)}$ be entry-wise minimum between $B^{(t)}$ and B';
8: **end for**
9: **return** $B^{(\beta)}$;

The first step is to compute an $(1+\epsilon, \beta)$-hopset H, for a parameter $\kappa \geq 1$ with $\beta = \left(\frac{\kappa}{\epsilon}\right)^{O(\kappa)}$, as in Theorem 3. Let A be the adjacency matrix of $G \cup H$ and fix $R = \beta/\epsilon$. For every integer $1 \leq t \leq \beta$, let $B^{(t)}$ be an $s \times n$ matrix such that for all $i \in S$ and $j \in V$, $B^{(t)}_{ij}$ is a $(1 + \frac{1}{R})^{t-1}$-approximation to $d^{(t)}_{G \cup H}(i, j)$. Note that $B^{(1)} = A_{S*}$ is a submatrix of A containing only the rows corresponding to the sources S.

The following claim asserts that taking an approximate distance product of $B^{(t)}$ with the adjacency matrix yields $B^{(t+1)}$.

▷ Claim 4. Let $c, c' \geq 1$. Let A be the adjacency matrix of an n-vertex graph $G = (V, E)$, and let B be an $s \times n$ matrix (whose rows correspond to $S \subseteq V$) so that for all i, j, B_{ij} is a c-approximation to $d^{(t)}_G(i, j)$, for some positive integer t. Let $C = B \star A$ and C' be a c'-approximation to C. Then, for all i, j, C'_{ij} is a $c \cdot c'$-approximation to $d^{(t+1)}_G(i, j)$.

Proof. Consider a pair of vertices $i \in S$ and $j \in V$. By definition of the $\star$ operation, $C_{ij} = \min_{1 \le k \le n}\{B_{ik} + A_{kj}\}$. Let π be the shortest path in G from i to j that contains at most $t+1$ edges, and let $k \in V$ be the last vertex before j on π. Since B_{ik} is a c-approximation to $d_G^{(t)}(i,k)$ and A_{kj} is the edge weight of $\{k,j\}$, we have that $B_{ik} + A_{kj}$ is a c-approximation to $d_G^{(t+1)}(i,j)$. Hence $C_{ij} \le B_{ik} + A_{kj}$ is a c-approximation of $d_G^{(t+1)}(i,j)$ too. The assertion of the claim follows since $C_{ij} \le C'_{ij} \le c' \cdot C_{ij}$. ◁

Given $B^{(t)}$, we compute $B^{(t+1)}$ as a $(1+\frac{1}{R})$-approximation to $B^{(t)} \star A$. Using Theorem 1 this can be done within $\tilde{O}(R \cdot n^{w(r)})$ rounds. Thus, the total running time to compute $B^{(\beta)}$ is

$$\tilde{O}(\beta \cdot R \cdot n^{w(r)}) = \tilde{O}(n^{w(r)} \cdot (\kappa/\epsilon)^{O(\kappa)})$$

By Claim 4, $B^{(\beta)}$ is a $(1+\frac{1}{R})^{\beta-1} \le e^{\epsilon} = 1 + O(\epsilon)$ approximation to $d^{(\beta)}_{G \cup H}(u,v)$ for all $u \in S$ and $v \in V$. Since H is a $(1+\epsilon,\beta)$-hopset, the matrix $B^{(\beta)}$ is a $(1+O(\epsilon))$-approximation to $d_G(u,v)$, for all $u \in S$, and $v \in V$.

Reporting paths. For each approximate distance in $S \times V$ we can also report a path in G achieving this distance. To this end, we compute witnesses for each approximate distance product, and as in [40, Section 5] there is an algorithm that can report, for any $u, v \in V$, a path in $G \cup H$ of length at most $(1+\epsilon) \cdot d^{(\beta)}_{G \cup H}(u,v)$. In order to translate this to a path in G, we need to replace the hopset edges by corresponding paths in G. We use the fact that the hopsets of [21] have a path reporting property. That is, each hopset edge of weight W' has a corresponding path π of length W' in G, and every vertex on π stores its neighbors on the path. Thus, we can obtain a $u-v$ path in G in time proportional to its number of edges.

We conclude with the following theorem.

▶ **Theorem 5.** *Let $G = (V,E)$ be a weighted undirected graph, fix $S \subseteq V$ of size n^r for some $0 \le r \le 1$, and let $0 < \epsilon < 1$. Then for any $\kappa \ge 1$, there is a deterministic algorithm that computes a $(1+\epsilon)$-approximation to all distances in $S \times V$ that runs in time*

$$\tilde{O}(\max\{n^{w(r)} \cdot (\kappa/\epsilon)^{O(\kappa)}, |E| \cdot n^{1/\kappa}\}) .$$

Furthermore, for each pair in $S \times V$, a path achieving the approximate distance can be reported in time proportional to the number of edges in it.

One may choose κ as a slowly growing function of n, e.g. $\kappa = (\log\log n)/\log\log\log n$, so that $\kappa^{\kappa} \le \log n$ and $n^{1/\kappa} = n^{o(1)}$, and obtain running time $\tilde{O}(n^{\omega(r)} + |E| \cdot n^{o(1)})$ (for a constant $\epsilon > 0$). We stress that for all $r \le 0.313$, a result of [27] gives that $w(r) = 2 + o(1)$. So even for polynomially large set of sources S, with size up to $n^{0.313}$, our algorithm computes $(1+\epsilon)$-approximate distances $S \times V$ in time $n^{2+o(1)}$. In fact, for all $r < 1$, our bound improves the current bound for $(1+\epsilon)$-APASP [40].

Observe that if $r > 0.313$, then we can choose κ as a large enough constant, so that the running time to compute the hopset, which is $\tilde{O}(|E| \cdot n^{1/\kappa})$, is dominated by $n^{w(r)}$. Alternatively, if $|E| \le n^{2-\delta}$ we may choose $\kappa = 1/\delta$, so the running time to compute the hopset will be $\tilde{O}(n^2) = \tilde{O}(n^{w(r)})$ for all $0 \le r \le 1$. In both cases we obtain $\beta = (1/\epsilon)^{O(1)}$, and thus our algorithm for computing $(1+\epsilon)$-approximate shortest paths for $S \times V$ has running time $\tilde{O}(n^{w(r)}/\epsilon^{O(1)})$.

4 Approximate Distance Preservers

A direct application of our s-ASP algorithm is the problem of approximate D-preservers. Exact D-preservers were introduced in [8]. Given an unweighted n-vertex graph $G = (V, E)$ and a parameter D, a subgraph $G' = (V, H)$ of G ($H \subseteq E$) is called a *D-preserver* of G if for every pair $u, v \in V$ with $d_G(u, v) \geq D$, we have $d_{G'}(u, v) = d_G(u, v)$. It was shown in [8] that every unweighted graph (both undirected and directed) admits a D-preserver with $O(n^2/D)$ edges, and that this bound is tight. We will next describe an efficient construction of an *approximate* D-preserver that applies only for *undirected* graphs.

It is also well-known (see [37, 8]) that one can compute a D-preserver of size $O(\frac{n^2}{D} \log n)$ by sampling $O(\frac{n}{D} \log n)$ vertices S independently at random, computing a BFS tree rooted at each of them, and inserting all these trees into the ultimate D-preserver. The running time of this procedure is $\tilde{O}(\frac{m \cdot n}{D})$, where $m = |E|$.

For a pair of parameters D and $\epsilon > 0$, we say that a subgraph $G' = (V, H)$ is a $(1+\epsilon)$-*approximate D-preserver* of G if for every pair of vertices $u, v \in V$ with $d_G(u, v) \geq D$, we have $d_{G'}(u, v) \leq (1+\epsilon) \cdot d_G(u, v)$. Using our $(1+\epsilon)$-approximate s-ASP algorithm one can compute a $(1+\epsilon)$-approximate D-preserver within time $\tilde{O}(n^{w(r)})$, with $r = \frac{\log n - \log D}{\log n}$. This expression is strictly better than $\tilde{O}(n^3/D)$, for all values of D, i.e., at least for dense graphs ($m = \Theta(n^2)$), the new algorithm is always faster than the existing one.

The algorithm itself uses our $(1+\epsilon)$-ASP algorithm to compute $(1+\epsilon)$-approximate BFS trees rooted at all vertices of the sampled set S, and returns the union of them as a $(1+\epsilon)$-approximate D-preserver. For the stretch analysis, consider a pair $u, v \in V$ of vertices with $d_G(u, v) \geq D$. Let π be a shortest path between them. Since it contains at least $D+1$ vertices, with high probability at least one of the sampled vertices $s \in S$ belongs to the path. Thus the preserver G' satisfies $d_{G'}(u, s) \leq (1+\epsilon) \cdot d_G(u, s)$ and $d_{G'}(s, v) \leq (1+\epsilon) \cdot d_G(s, v)$. Thus

$$d_{G'}(u, v) \leq (1+\epsilon) \cdot (d_G(u, s) + d_G(s, v)) = (1+\epsilon) \cdot d_G(u, v) \ .$$

5 Improved ASP for $S \times V$ in the Congested Clique Model

In this section we show how to improve the $(1+\epsilon)$-ASP for $S \times V$ results of [9] and [16] in the Congested Clique model. Specifically, we show that given a weighted graph $G = (V, E)$ and a set of $S \subseteq V$ sources of size $|S| = n^r$, there is a $\text{poly}(\log n)$ time algorithm to compute $(1+\epsilon)$-ASP for $S \times V$ as long as $r < (1+\alpha)/2 \approx 0.655$. For unweighted graphs, we obtain an improved running time of $\text{poly}(\log \log n)$. More generally, for S of arbitrary size, $|S| = n^r$, the running time is given by $\tilde{O}(n^{f(r)})$, where the function $f(r)$ grows from 0 to $1 - \frac{2}{\omega} \approx 0.158$. (See Table 2 for more details.)

A polylogarithmic running time (respectively, $\text{poly}(\log \log n)$ time for unweighted graphs), was obtained only for $r \leq 1/2$ in [9] (resp., [16]). More generally, their running time for arbitrary S is $\tilde{O}(\frac{|S|^{2/3}}{n^{1/3}})$.

To achieve these improvements, we use the method of [10] combined with fast rectangular matrix multiplication in the Congested Clique model. The following theorem is from [26].

▶ **Theorem 6** ([26]). *Let $G = (V, E)$ be an n-vertex graph, and fix $0 < r \leq 1$. Let A and B be $n^r \times n$ and $n \times n$ matrices. Then there is a deterministic algorithm in the Congested Clique that computes $A \cdot B$ in $O(n^{1-2/\omega(r')})$ rounds, where r' is the solution to the equation:*

$$r' = 1 - (1 - r) \cdot \omega(r') \ . \tag{2}$$

(Recall that $\omega(r')$ is the exponent for $n^{r'} \times n$ MM.)

Using this theorem in conjunction with the reduction of Theorem 1, we obtain an approximate distance product in the Congested Clique model:

▶ **Corollary 7.** *Let $G = (V, E)$ be an n-vertex graph, and fix $0 < r \leq 1$. Let A and B be $n^r \times n$ and $n \times n$ matrices with entries in $\{1, 2, ..., M\} \cup \{\infty\}$, and fix any $R \geq 1$. Then there is a deterministic algorithm in the Congested Clique that computes a $(1+1/R)$-approximation to $A \star B$ in $O(R \cdot n^{1-2/\omega(r')} \cdot \log M)$ rounds, with r' as in (2).*

In fact, Le Gall [26] showed that k pairs of $n^r \times n$ and $n \times n$ matrices can be multiplied in $O(k^{2/\omega(r')} \cdot n^{1-2/\omega(r')})$ time. As a result, we improve the estimate in Corollary 7 to $O((R \cdot \log M)^{2/\omega(r')} \cdot n^{1-2/\omega(r')})$. Indeed, as we argued in the proof of Theorem 1, such an approximate distance product can be computed by calculating $O(\log M)$ distance products of matrices with entries in $\{0, 1, \ldots, R\} \cup \{\infty\}$. Each such distance product can, in turn, be computed via $O(R)$ distance products of matrices with small entries (via Chinese Remainders' Theorem; see the discussion that follows Lemma 2.2 in [40]). Hence overall, our algorithm needs to compute distance products of $O(R \cdot \log M)$ pairs of matrices, and this requires [26] $O((R \log M)^{2/w(r')} \cdot n^{1-2/w(r')})$ time.

5.1 ASP for $S \times V$ in Weighted Graphs

Here we apply the improved rectangular MM to ASP for $S \times V$, using the method of [9]. For completeness we sketch it below. The following theorem was shown in [9], based on a construction from [21]. It provides a fast construction of a hopset with logarithmic hopbound for the Congested Clique model.

▶ **Theorem 8** ([9])**.** *Let $0 < \epsilon < 1$. For any n-vertex weighted undirected graph $G = (V, E)$, there is a deterministic construction of an $(1 + \epsilon, \beta)$-hopset H with $\tilde{O}(n^{3/2})$ edges and $\beta = O(\log n/\epsilon)$, that requires $O(\log^2 n/\epsilon)$ rounds in the Congested Clique model.*

Now, we approximately compute β-bounded distances in the graph $G \cup H$, by letting B be the adjacency matrix of $G \cup H$, and $A^{(1)}$ the $|S| \times n$ matrix of sources. (Specifically, for every pair $(u, v) \in S \times V$, the entry $A^{(1)}_{u,v}$ contains $\omega((u, v))$ if $(u, v) \in E$, and ∞ otherwise.) Define $A^{(t+1)} = A^{(t)} \star B$, and by the definition of hopset, $A^{(\beta)}_{ij}$ is a $(1 + \epsilon)$-approximation to $d_G(i, j)$ for any $i \in S$ and $j \in V$. Each product is $(1 + 1/R)$-approximately computed by Corollary 7 within $\tilde{O}(R \cdot n^{1-2/\omega(r')} \cdot \log M)$ rounds. We obtain a $(1+\epsilon)(1+1/R)^{\beta}$-approximation. We set $R = O(\frac{\log n}{\epsilon^2})$. Recall also that $\beta = O(\log n/\epsilon)$. As a result we derive the following theorem:

▶ **Theorem 9.** *Given any n-vertex weighted undirected graph $G = (V, E)$ with polynomial weights, parameters $0 < r < 1$, $0 < \epsilon < 1$, and a set $S \subseteq V$ of n^r sources, let r' be the solution to equation (2). Then there is a deterministic algorithm in the Congested Clique that computes $(1 + \epsilon)$-ASP for $S \times V$ within $\tilde{O}(n^{1-2/\omega(r')}/\epsilon^{O(1)})$ rounds.*

In particular, for a constant $\epsilon > 0$, when $r < (1 + \alpha)/2 \approx 0.655$ the running time is $\tilde{O}(1)$. For $r = 0.7$, the solution is slightly smaller than $r' = 0.4$, for which $\omega(r') \approx 2.01$, and the number of rounds is $O(n^{0.006})$. When $r = 0.8$, the solution is roughly $r' = 0.59$, for which $\omega(r') \approx 2.085$, and the number of rounds is $O(n^{0.04})$. We show a few more values in the following Table 2. (Note that at $r = 1$ we converge to the result of [10] for APASP.)

5.2 ASP for $S \times V$ in Unweighted Graphs

In this section we show an improved algorithm for unweighted graphs, based on [16]. The first step of [16] was developing a fast algorithm for a sparse emulator: we say that $H = (V, F)$ is an (α, β)-*emulator* for a graph $G = (V, E)$ if for all $u, v \in V$, $d_G(u, v) \leq d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$.

▶ **Theorem 10** ([16], Theorem 24). *For any n-vertex unweighted graph $G = (V, E)$ and $0 < \epsilon < 1$, there is a randomized algorithm in the Congested Clique model that computes $(1+\epsilon, \beta)$-emulator H with $O(n \log\log n)$ edges within $O(\log^2 \beta/\epsilon)$ rounds w.h.p., where $\beta = O(\log\log n/\epsilon)^{\log\log n}$.*

Since the emulator is so sparse, all vertices can learn all of its edges within $O(\log\log n)$ rounds. Thus every pair of distance larger than β/ϵ already has an $1 + O(\epsilon)$ approximation, just by computing all distances in H locally. It remains to handle distances at most β/ϵ.

The next tool is a bounded-distance hopset that "takes care" of small distances. We say that $H' = (V, E')$ is a $(1+\epsilon, \beta', t)$-hopset if for every pair $u, v \in V$ with $d_G(u, v) \leq t$ we have the guarantee of inequality (1).

▶ **Theorem 11** ([16], Theorem 12). *There is a randomized construction of a $(1+\epsilon, \beta', t)$-hopset H' with $O(n^{3/2} \log n)$ edges and $\beta' = O(\log t/\epsilon)$ that requires $O(\log^2 t/\epsilon)$ rounds w.h.p. in the Congested Clique model.*

We use a $(1 + \epsilon, \beta', t)$-hopset H' for G with $t = \beta/\epsilon = O(\log\log n/\epsilon)^{\log\log n}$, so that $\beta' = \text{poly}(\log\log n/\epsilon)$. As before we let B be the adjacency matrix of $G \cup H'$, and $A^{(1)}$ be the $|S| \times n$ matrix of sources. Define $A^{(s+1)} = A^{(s)} \star B$. By the definition of bounded-distance hopset, $A^{(\beta')}_{ij}$ is a $(1+\epsilon)$-approximation to $d_G(i, j)$ for any $i \in S$ and $j \in V$ with $d_G(i, j) \leq t$. Each distance product is $(1+1/R)$-approximated using the algorithm from Corollary 7 within $\tilde{O}(R \cdot n^{1-2/\omega(r')} \cdot \log M)$ rounds. We note that since G is unweighted, the maximal entry in B and in any $A^{(s)}$ is $t \cdot (1 + \epsilon)$ (one can simply ignore entries of larger weight, i.e., replace them by ∞, since they will not be useful for approximating distances at most t). So we have $\log M = \text{poly}(\log\log n)$. In the current setting we assume $r \leq \frac{1+\alpha}{2} \approx 0.655$, and so $\omega(r') = 2$. The overall approximation factor is $(1+\epsilon)(1+1/R)^{\beta'}$. We set $R = \beta'/\epsilon = \text{poly}((\log\log n)/\epsilon)$, and get overall stretch $1 + O(\epsilon)$.

We conclude with the following theorem.

▶ **Theorem 12.** *Given any n-vertex unweighted undirected graph $G = (V, E)$, any $0 < \epsilon < 1$, and a set $S \subseteq V$ of at most $O(n^{0.655\ldots})$ sources, there is a randomized algorithm in the Congested Clique that w.h.p. computes $(1+\epsilon)$-ASP for $S \times V$ within $\text{poly}(\log\log n/\epsilon)$ rounds.*

Dory and Parter [16] provide also a deterministic counterparts of Theorems 10 and 11. Specifically, Theorem 5 of [16] provides a deterministic algorithm for building emulators with properties listed in Theorem 10 in time $O(\frac{\log^2 \beta}{\epsilon} + (\log\log n)^4)$. Theorem 12(2) in [16] provides a deterministic algorithm for building $(1 + \epsilon, \beta', t)$-hopsets with properties listed in Theorem 11, in time $O(\frac{\log^2 t}{\epsilon} + (\log\log n)^3)$. For our choice of parameters (given above), both these expressions are $\text{poly}(\log\log n, 1/\epsilon)$. As a result, we derive a deterministic counterpart of Theorem 12, i.e., $(1 + \epsilon)$-ASP for $S \times V$ in deterministic $\text{poly}(\log\log n, 1/\epsilon)$ time, for $|S| \leq n^{0.655\ldots}$.

6 PRAM Approximate Multi-Source Shortest Paths

The algorithm of Section 3 can be translated to the PRAM model. In this model, multiple processors are connected to a single memory block, and the operations are performed in parallel by these processors in synchronous rounds. The *running time* is measured by the number of rounds, and the *work* by the number of processors multiplied by the number of rounds.

To adapt our algorithm to this model, we need to show that approximate distance products can be computed efficiently in PRAM. The second ingredient is a parallel algorithm for hopsets. For the latter, the following theorem was shown in [21]. A deterministic analogue of it was recently shown in [19].

▶ **Theorem 13** ([21]). *For any weighted undirected graph $G = (V, E)$ on n vertices and parameters $\kappa \geq 1$ and $0 < \epsilon < 1$, there is a randomized algorithm that runs in parallel time $\left(\frac{\log n}{\epsilon}\right)^{O(\kappa)}$ and work $\tilde{O}(|E| \cdot n^{1/\kappa})$, that computes a $(1+\epsilon, \beta)$-hopset with $O(n^{1+1/\kappa} \cdot \log^* n)$ edges where $\beta = \left(\frac{\kappa}{\epsilon}\right)^{O(\kappa)}$.*

Matrix multiplication in PRAM. Essentially all the known fast matrix multiplication algorithms are based on Strassen's approach of divide and conquer, and thus are amenable to parallelization [30]. In particular, these algorithms which classically require time $T(n)$, can be executed in the PRAM (EREW) model within $O(\log^2 n)$ rounds and $\tilde{O}(T(n))$ work.

As was mentioned after Theorem 1, we can apply the reduction from MM to distance product in the PRAM model. Thus, we can compute a $(1+\frac{1}{R})$-approximate distance products of an $n^r \times n$ matrix by an $n \times n$ matrix in $O(R \cdot \text{poly}(\log n))$ rounds and $\tilde{O}(R \cdot n^{w(r)})$ work.

The path-reporting mechanism can be adapted to PRAM, by running the algorithm from [40] sequentially. Since we have only β iterations, the parallel time will be only $O(\beta)$ (which is a constant independent of n, as long as κ is constant). Once we got the path in $G \cup H$, we can expand all the hopset edges in parallel. We thus have the following result.

▶ **Theorem 14.** *Let $G = (V, E)$ be a weighted undirected graph, fix $S \subseteq V$ of size n^r for some $0 \leq r \leq 1$, and let $0 < \epsilon < 1$. Then for any $\kappa \geq 1$, there is a randomized parallel algorithm that computes a $(1+\epsilon)$-approximation to all distances in $S \times V$, that runs in $\left(\frac{\log n}{\epsilon}\right)^{O(\kappa)}$ parallel time, using work*

$$\tilde{O}(\min\{n^{w(r)} \cdot (\kappa/\epsilon)^{O(\kappa)}, |E| \cdot n^{1/\kappa}\}) \ .$$

Furthermore, for each pair in $S \times V$, a path achieving the approximate distance can be reported within parallel time $(\kappa/\epsilon)^{O(\kappa)}$, and work proportional to the number of edges in it.

Note that we can set κ to be an arbitrarily large constant, and obtain a polylogarithmic time and work $\tilde{O}(n^{\omega(r)} + |E|n^{1/\kappa})$.

7 Weight Reduction

In this section we argue that our s-ASP algorithm can be used in conjunction with Klein-Sairam weight reduction [31] (see also [11, 20, 19]) to replace the factor $\log M$ in the running time of its centralized version and in the work complexity of its parallel version by a factor of $O(\log^2 n/\epsilon)$ (independent of the aspect ratio of the graph).

The weight reduction produces $\lambda = \lceil \log M \rceil$ graphs $G^{(i)} = (V^{(i)}, E^{(i)})$, $i = 1, 2, \ldots, \lambda$, each with aspect ratio at most $\lceil n/\epsilon \rceil$. The vertex set $V^{(i)}$ of $G^{(i)}$ is the set of connected components of the subgraph of G in which all edges of weight at most $\epsilon \cdot 2^i/n$ are contracted. The edge set $E^{(i)}$ contains edges between nodes of $V^{(i)}$ with weight at most 2^i. Actually, we keep in the node set $V^{(i)}$ only "active" nodes, i.e., nodes that are not isolated in $G^{(i)}$.

The vertex sets $\{V^{(i)}\}_{i=1}^{\lambda}$ form a laminar family, which can be represented by a forest $\mathcal{F}$. There is an edge in $\mathcal{F}$ between a node $C^{(i+1)} \in V^{(i+1)}$ and a node $C^{(i)} \in V^{(i)}$ if and only if $C^{(i)}$ is merged into $C^{(i+1)}$ on scale $i+1$, i.e., $C^{(i+1)}$ is a union of one or more distinct sets from $V^{(i)}$, one of which is $C^{(i)}$. (It is possible that $C^{(i+1)} = C^{(i)}$.)

Denote the exponent of the running time of our centralized s-ASP algorithm by $2 \leq \zeta \leq \omega$, i.e., the running time is $\tilde{O}(n^{\zeta}) \cdot \text{poly}(1/\epsilon) \cdot \log M$. Then, once it is invoked on all the graphs $\{G_i\}_{i=1}^{\lambda}$ created by the weight reduction, the running time becomes $\log n/\epsilon \cdot \text{poly}(1/\epsilon) \cdot \sum_{i=1}^{\lambda} \tilde{O}(n_i^{\zeta})$. We next argue that

$$\sum_{i=1}^{\lambda} n_i^{\zeta} = \tilde{O}(n^{\zeta}) \ .$$

For the forest $\mathcal{F}$ as above, we denote by $f(\mathcal{F}) = \sum_{i=1}^{\lambda} n_i^{\zeta}$ the sum over all levels of $\mathcal{F}$, where each level i contributes its number n_i of nodes (that appear on level i) in the power ζ.

Observe that if a node C is active on level i (of $\mathcal{F}$), then by a level $i+\ell$, $\ell = \lceil \log n/\epsilon \rceil$, it necessarily merges into some supernode $\hat{C} \in V^{(i+\ell)}$, $C \subset \hat{C}$ ($C \neq \hat{C}$). (This is because any edge $e \in R^{(i)}$ incident on C will necessarily be contracted on or before level $i+\ell$.)

We say that a path π in $\mathcal{F}$ between some ancestor node $C^{(j)} \in V^{(j)}$ and some descendent node $C^{(i)} \in V^{(i)}$, $i \leq j$, is a *one-child* path if each of the nodes $C^{(i+1)}, C^{(i+2)}, \ldots, C^{(j)}$ has one single child in $\mathcal{F}$ (and $C^{(i)} = C^{(i+1)} = \ldots = C^{(j)}$). Such a path is said to be *maximal* if $C^{(j)}$ is a either a root or its parent has more than one child, and $C^{(i)}$ is either a leaf or has more than one child. Note that a maximal one-child path π may also be empty, if $C^{(i+1)} \in V^{(i+1)}$ is a parent of $C^{(i)} \in V^{(i)}$, $C^{(i+1)} \neq C^{(i)}$, and $C^{(i)}$ is either a leaf or has more than one child, and $C^{(i+1)}$ has more than one child. In this case we write $\pi = (C^{(i)})$.

Now consider a forest $\hat{\mathcal{F}}$ in which each maximal one-child path $\pi = (C^{(i)}, C^{(i+1)} = C^{(i)}, \ldots, C^{(j)})$ is replaced by a one-child path of length precisely ℓ. Let $\hat{\lambda} \leq \lambda \cdot \ell$ be the number of levels in $\hat{\mathcal{F}}$. Note that

$$f(\mathcal{F}) = \sum_{i=1}^{\lambda} n_i^{\zeta} \leq f(\hat{\mathcal{F}}) = \sum_{i=1}^{\hat{\lambda}} \hat{n}_i^{\zeta} \ ,$$

where $\hat{n}_i$ is the number of nodes on level i of the forest $\hat{\mathcal{F}}$. (This is because, by induction on i, we have $n_i \leq \hat{n}_i$, for every i.)

Let $\mathcal{F}'$ be the forest $\hat{\mathcal{F}}$ in which every maximal one-child path $\pi = (C^{(i)}, \ldots, C^{(j)} = C^{(i)})$ is replaced by an empty one-child path $(C^{(i)})$. Note that in $\mathcal{F}'$, every internal node has degree at least 2. Let n'_i denote the number of nodes on level i of $\mathcal{F}'$.

Observe that as the number of leaves is n, the overall number of distinct nodes in $\mathcal{F}'$ is at most $2n-1$. Each node C of $\mathcal{F}'$ contributes at most $n^{\zeta-1}$ to the sum $f(\mathcal{F}')$. (This is because if C belongs to a level on which the number of nodes is t, the total contribution of this level is t^{ζ}. Hence each node C on this level can be charged for at most $t^{\zeta-1} \leq n^{\zeta-1}$.) Hence $f(\mathcal{F}') \leq (2n-1) \cdot n^{\zeta-1} = O(n^{\zeta})$. (Alternatively, this can be seen by noting that the maximum of the sum $f(\mathcal{F}') = \sum_i n'^{\zeta}_i$ subject to $\sum_i n'_i = 2n-1$ is $O(n^{\zeta})$.)

Observe that every level of $\mathcal{F}'$ is duplicated ℓ times in $\hat{\mathcal{F}}$. Hence

$$f(\hat{\mathcal{F}}) = \ell \cdot f(\mathcal{F}') = O(\log(n/\epsilon) \cdot n^{\zeta}) \ .$$

Finally, as $f(\mathcal{F}) \leq f(\hat{\mathcal{F}})$, we conclude that $f(\mathcal{F}) = O(n^{\zeta} \cdot \log(n/\epsilon))$.

Hence the overall running time of the centralized version of our s-ASP algorithm (and the work complexity of its parallel version) is

$$\log(n/\epsilon) \cdot \mathrm{poly}(1/\epsilon) \cdot \tilde{O}(f(\mathcal{F})) = \tilde{O}(n^{\zeta}) \cdot \log^2(n/\epsilon) \cdot \mathrm{poly}(1/\epsilon) \ .$$

The time complexity of its parallel version is polylogarithmic in the aspect ratio (bounded by n/ϵ) of each of the graphs G_i.

In the distributed CC model one needs to compute the connected components along with their representatives in such a way that every vertex $v \in V$ represents $O(\log n)$ components (of all scales altogether). This is achieved by making sure that whenever a number of components

$C_1, C_2, \ldots, C_h$ with $|C_1| \geq |C_2| \geq \ldots \geq |C_h|$ merge into a higher-level component $\hat{C}$, a representative of one of the clusters $C_2, \ldots, C_h$ (but not C_1) becomes the representative of $\hat{C}$. See [20, 19] for more details.

Another issue arises when one needs to return output. One can compute an MST T of G, and to compute an exact distance labeling for this MST. (In the sequential setting this can be done in near-linear time.) These distance labels will provide an $(n-1)$-approximation of distances in G. Given a pair $u, v \in V$ with a distance estimate $\delta_{u,v}$, this provides us with $O(\log_{1+\epsilon} n) = O\left(\frac{\log n}{\epsilon}\right)$ scales on which one needs to look for a $(1+\epsilon)$-approximate distance estimate $\hat{d}_{u,v}$ for this pair. Indeed, note that in scales i for which $\epsilon/n \cdot 2^i > \delta_{u,v} \geq d_G(u,v)$, the whole path will be contracted. Also, in scales i for which $n \cdot 2^i < \delta_{u,v}/(n-1) \leq d_G(u,v)$, at least one edge in the $u-v$ path will have weight larger than 2^i. Hence u and v will be in different connected components of $G^{(i)}$.

We then identify components C_u, C_v, $u \in C_u$, $v \in C_v$, on each of the relevant scales i, and fetch the distance estimate between C_u and C_v in G_i. (One also needs to add to these estimates an upper bound on $Diam(C_u) + Diam(C_v)$, which is bounded by $O(\epsilon \cdot 2^i)$.) Finally, we then return the smallest among the resulting estimates. To summarize, this requires polylog(n) time per vertex pair, and $\tilde{O}(|S| \cdot n + |E|)$ time altogether.

The same approach can be used also in the PRAM and in the CC models (details are omitted). The time complexity will still be polylogarithmic in n, and the work complexity (in PRAM) is $\tilde{O}(|E|n^{\delta})$, for an arbitrarily small $\delta > 0$.

This completes the analysis of the weight reduction.

References

1. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
2. Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.
3. Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In *STOC*, 2020.
4. S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 384–396. Springer, 2003.
5. Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS*, pages 591–602, 2006. doi:10.1109/FOCS.2006.29.
6. Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 7:1–7:16, 2017. doi:10.4230/LIPIcs.DISC.2017.7.
7. Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702, 2009. doi:10.1109/FOCS.2009.16.
8. Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discret. Math.*, 19(4):1029–1055, 2005. doi:10.1137/S0895480103431046.
9. Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 74–83. ACM, 2019. doi:10.1145/3293611.3331633.

10 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 143–152, 2015.

11 Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. doi:10.1145/331605.331610.

12 Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001. doi:10.1006/jagm.2000.1117.

13 Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997. doi:10.1006/jcom.1997.0438.

14 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

15 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29:1740–1759, 2000.

16 Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, pages 59–68, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3405711.

17 M. Elkin. Computing almost shortest paths. In *Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 53–62, 2001.

18 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths and PRAM distance oracles in weighted graphs. *CoRR*, abs/1907.11422, 2019. arXiv:1907.11422.

19 Michael Elkin and Shaked Matar. Deterministic PRAM approximate shortest paths in polylogarithmic time and slightly super-linear work. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 198–207. ACM, 2021. doi:10.1145/3409964.3461809.

20 Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM J. Comput.*, 48(4):1436–1480, 2019. doi:10.1137/18M1166791.

21 Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019.*, pages 333–341, 2019. doi:10.1145/3323165.3323177.

22 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

23 Zvi Galil and Oded Margalit. Witnesses for boolean matrix multiplication and for transitive closure. *J. Complex.*, 9(2):201–221, 1993. doi:10.1006/jcom.1993.1014.

24 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.

25 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

26 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 2016. doi:10.1007/978-3-662-53426-7_5.

27 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67.

28 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 146–155, 2014. doi:10.1109/FOCS.2014.24.

29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 489–498, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897638.

30 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complex.*, 14(2):257–299, 1998. doi:10.1006/jcom.1998.0476.

31 Philip N. Klein and Sairam Subramanian. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 259–270, 1993. doi:10.1109/SFCS.1993.366861.

32 Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997. doi:10.1006/jagm.1997.0888.

33 Jason Li. Faster parallel algorithm for approximate shortest path. In *STOC*, 2020.

34 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573, 2014. doi:10.1145/2591796.2591850.

35 Yossi Shiloach and Uzi Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *J. Algorithms*, 2(1):88–102, 1981. doi:10.1016/0196-6774(81)90010-9.

36 Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69(3):330–353, 2004. doi:10.1016/j.jcss.2004.04.003.

37 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.

38 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

39 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

40 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Approximate Distances to k-Nearest Neighbors in PRAM

In this section, given a weighted *directed* graph $G = (V, E)$, we focus on the task of approximately computing the distances from each $v \in V$ to its k nearest neighbors. The main observation is that we work with rather sparse matrices, since for each vertex we do not need to store distances to vertices that are not among its k nearest neighbors.

In [39] fast algorithms for sparse matrix multiplication were presented. Recall that $\alpha \in [0, 1]$ is the maximal exponent so that the product of an $n \times n^\alpha$ by $n^\alpha \times n$ matrices can be computed in $n^{2+o(1)}$ time. Currently by [27], $\alpha \geq 0.313$. Let $\gamma = \frac{\omega-2}{1-\alpha}$.

▶ **Theorem 15** ([39]). *The product of two $n \times n$ matrices each with at most m nonzeros can be computed in time*

$$\min\{O(n^\omega), m^{\frac{2\gamma}{\gamma+1}} \cdot n^{\frac{2-\alpha\gamma}{\gamma+1}+o(1)} + n^{2+o(1)}\} .$$

We present the following adaptation to distance products in the PRAM model. In our setting, a matrix will be sparse if it contains few non-infinity values.

▶ **Lemma 16.** *For $R \geq 1$, the $(1+\frac{1}{R})$-approximate distance product of two $n \times n$ matrices each with at most m non-infinities can be computed in parallel time $O(R\log^{O(1)} n)$ and work*

$$\tilde{O}(R \cdot \min\{n^{\omega}, m^{0.702} \cdot n^{1.18} + n^{2+o(1)}\}) \ . \tag{3}$$

Proof. The $(1+\frac{1}{R})$-approximate distance product of Theorem 1 involves $O(\log M) = O(\log n)$ standard matrix multiplications. These multiplications can be done in parallel, and we need to compute entry-wise minimum of these matrices. This can also be done very efficiently in PRAM (See e.g., [35]). By the reduction described in the proof of Theorem 1, the resulting matrices will have $O(m)$ nonzeros (and entries of size $O(n^R)$). Thus the parallel time required to compute each such multiplication is $O(R\log^{O(1)} n)$. Using the currently known bounds on ω and α, we have $\gamma \approx 0.542$. Plugging this in Theorem 15, the work required is as in (3). ◀

For an $n \times n$ matrix A, denote by $\mathrm{trun}_k(A)$ the matrix A in which every column is truncated to contain only the smallest k entries, and ∞ everywhere else. Clearly this operation can be executed in $\mathrm{poly}(\log n)$ parallel time and $\tilde{O}(n^2)$ work. For a vertex $i \in V$, let $N_k(i)$ be the set of k nearest neighbors of i.

▷ Claim 17. Let G be a weighted directed graph. For some $t \geq 1$, and $c, c' \geq 1$, let A be an $n \times n$ matrix such that for every $1 \leq i \leq n$ and every $j \in N_k(i)$, A_{ij} is a c-approximation to $d_G^{(t)}(i,j)$, and ∞ for $j \notin N_k(i)$. Then, if B is a c'-approximation to $A^T \star A$, then for each i and $j \in N_k(i)$, we have that B_{ij} is a $(c \cdot c')$-approximation to $d_G^{(2t)}(i,j)$.

Proof. Let h be the middle vertex on the shortest path with at most $2t$ edges between i and j (so that there are at most t edges on the sub-paths from i to h and from h to j). Since $j \in N_k(i)$, the triangle inequality implies that $h \in N_k(i)$ and $j \in N_k(h)$. Thus, A_{ih} (resp., A_{hj}) is a c-approximation to $d_G^{(t)}(i,h)$ (resp., $d_G^{(t)}(h,j)$). By definition of distance product, $(A^T \star A)_{ij} \leq c \cdot d_G^{(t)}(i,h) + c \cdot d_G^{(t)}(h,j) \leq c \cdot d_G^{(2t)}(i,j)$. So B_{ij} is a $c \cdot c'$-approximation to $d_G^{(2t)}(i,j)$. (Note also that $(A^T \star A)_{ij} \geq d_G^{(t)}(i,h) + d_G^{(t)}(h,j) = d_G(i,j)$.) ◁

Our algorithm to compute approximate shortest paths to k nearest neighbors proceeds by computing $\lceil \log k \rceil$ times an approximate distance product, truncating each time to the smallest k entries in each column. See Algorithm 2. (This algorithm is based on an analogous algorithm from [9], devised there in the context of the Congested Clique model.) One difference between our algorithm and that of [9] is that on line 4 we apply a parallel version of Yuster-Zwick's sparse matrix multiplication [39], as opposed to an algorithm due to [9] for multiplying sparse matrices in the Congested Clique model. Another difference is that we are computing approximate distance products, as opposed to [9] that compute exact distance products. The latter (exact) computation applies to the Congested Clique model, and it is not clear if it can be performed in the centralized or PRAM models.

Since each matrix has $m = O(nk)$ non-infinities, and there are only $O(\log k)$ iterations, the parallel time is $R \cdot \log^{O(1)} n$ and the total work, using the bound of (3) with $m = O(nk)$, is

$$\tilde{O}(R \cdot \min\{n^{\omega}, k^{0.702} \cdot n^{1.882} + n^{2+o(1)}\}) \ .$$

The correctness of the algorithm follows from Claim 17, as the shortest path from a vertex v to a neighbor $u \in N_k(v)$ can have at most k edges. The approximation we obtain is $(1+\frac{1}{R})^{\lceil \log k \rceil} = 1 + O(\epsilon)$. We remark that the truncation might actually remove the distance from $v \in V$ to some $u \in N_k(v)$, because the computed distances are approximate, and so

Algorithm 2 `Approx` k-`NN`(G,ϵ).

Let A be the adjacency matrix of G;
Let $R = \lceil (\log k)/\epsilon \rceil$;
for i from 1 to $\lceil \log k \rceil$ **do**
  Let A' be a $(1+1/R)$-approximation to $(\mathrm{trun}_k(A))^T \star \mathrm{trun}_k(A)$;
  Let A be entry-wise minimum between A and A';
end for
return $\mathrm{trun}_k(A)$;

u can be replaced by a farther away vertex. Denote by $N'_k(v)$ the k vertices returned by Algorithm 2 for $v \in V$. This vertex set has the property that for every vertex $u \in N_k(v)$, there is a distinct vertex $u' \in N'_k(v)$, such that $d_G(v,u') \leq (1+\epsilon)d_G(v,u)$.

Next we provide a formal argument that shows that our algorithm computes an approximate k-NN. For a vertex $u \in V$, let $z_1(u), z_2(u), \ldots, z_{n-1}(u)$ denote the sequence of vertices in the monotonically non-decreasing order of distance from u. (Henceforth ties are broken consistently by the Ids.) Let n_u denote the number of vertices reachable from u in G. If $n_u \leq k$ then k-truncation has no effect on the computation for the vertex u, and thus the computed set $N'_k(u)$ will contain all the n_u vertices reachable from u, with distance estimates approximated up to $(1+1/R)^{\lceil \log k \rceil}$. We from now on therefore focus on the case $n_u > k$.

For an index $i = 1, 2, \ldots$, we say that a vertex v is a $(1+\epsilon)$-*replacement* of $z_i(u)$ if it satisfies $d_G(u,v) \leq (1+\epsilon)d_G(u,z_i(u))$. A $(1+\epsilon)$-*approximate k-NN* of u is a set S of size k that satisfies the following property: Let $i \in [k]$ be the minimum index so that $z_i(u) \notin S$, if exists. Then for each $j < i$, the computed distance estimates of $z_j(u)$ are at most $(1+\epsilon)$-approximations of the actual respective distance $d_G(u,z_j(u))$, and also S contains k distinct $(1+\epsilon)$-replacements of $z_i(u)$.

Consider the following algorithm, whose pseudocode is given by Algorithm 2. Let $B_0 = B'_0 = A_G$ be the adjacency matrix of the graph G. Let $A = \mathrm{trun}_k(A_G)$, $A_0 = A$ be the k-truncated matrix A_G. (The entries (u,v) that survive also contain distance estimates $\delta(u,v) = w(u,v)$. In other entries the estimates are set to ∞.)

Let B'_1 be a $(1+1/R)$-approximate $A_0^T \star A_0$. For every entry (x,y) we check if $B'_1(x,y) > A_0(x,y)$. If it is the case, we set $B_1(x,y) = A_0(x,y)$. Otherwise set $B_1(x,y) = B'_1(x,y)$. Set $A_1 = \mathrm{trun}_k(B_1)$, and iterate, i.e., repeat these operations $h = \lceil \log k \rceil$ times. The matrix A_h is the output matrix.

For every $i \in [0,h]$ and every vertex u, let $\hat{S}_u(i)$ denote the set of vertices v with $B_i(u,v) \neq \infty$, and $S_u(i)$ denote the set of vertices with $A_i(u,v) \neq \infty$. Also, let $Ball_u(i)$ denote the set of vertices v such that there exists a shortest $u-v$ path with at most 2^i hops. Let $p_u(i) = |Ball_u(i)|$. Observe that since $n_u > k$, for every $i \in [0,h-1]$ we have $p_u(i) \geq 2^i$, and $p_u(h) \geq k$. We also write $Ball'_u(i) = Ball_u(i) \cap \{z_1(u), \ldots, z_k(u)\}$, and $q_u(i) = |Ball'_u(i)|$. Note that $q_u(h) = k$.

▶ **Lemma 18.** *For every vertex $u \in V$ and index $i \in [h]$,* either

1. *The set $S_u(i)$ contains all the $q_u(i)$ vertices of $Ball'_u(i)$ themselves (with estimates that are $(1+1/R)^i$-approximations of the actual respective distances from u)*
2. Or: *Let $k_i < k$ be the smallest index such that $z_{k_i}(u) \notin S_u(i)$. Then $S_u(i)$ contains all the vertices of $\{z_1(u), \ldots, z_{k_i-1}(u)\} \cap Ball_u(i)$ (with estimates that are $(1+1/R)^i$-approximations of the actual respective distances from u), and also, $S_u(i)$ contains k distinct $(1+1/R)^i$-replacements of $z_{k_i}(u)$.*

▶ Remark. We will use the lemma with $i = h$, and deduce that for every vertex u, the set $S_u(i)$ is a $(1 + 1/R)^h$-approximate k-NN for u. (By $outdeg(u)$ we denote the out-degree of the vertex u, i.e., the number of its outgoing neighbors.)

Proof. The proof is by induction on i.

Base. For every $u \in V$, the set $S_u(0)$ contains the $\min\{k, outdeg(u)\}$ closest neighbors to u. In particular, it contains the closest vertex $z_1(u)$, and its estimate is $\delta(u, z_1(u)) = w(u, z_1(u)) = d_G(u, z_1(u))$. Note that $Ball'_u(0) = Ball_u(0) = \{z_1(u)\}$, i.e., $q_u(0) = 1$. Thus assertion 1 holds.

Step. First suppose that assertion 2 holds for u with respect to i. Let $k_i \leq k$ be the smallest index such that $z_{k_i}(u) \notin S_u(i)$. Then all vertices of $\{z_1(u), \ldots, z_{k_i-1}(u)\} \cap Ball'_u(i)$ belong to $S_u(i)$, and their distance estimates are $(1 + 1/R)^i$-approximate ones. Also, $S_u(i)$ contains k distinct $(1 + 1/R)^i$-replacements of $z_{k_i}(u)$.

There are two cases. If $k_{i+1} \geq k_i$ then, by definition, all the vertices $\{z_1(u), \ldots, z_{k_{i+1}-1}(u)\} \cap Ball'_u(i + 1)$ belong to $S_u(i + 1)$, and their distance estimates are $(1 + 1/R)^{i+1}$-approximate ones. Also, the remaining elements of $S_u(i + 1)$ have estimates that are smaller or equal than the estimates of the respective elements in $S_u(i)$, and thus they are $(1 + 1/R)^i$-replacements of $z_{k_i}(u)$. Hence they are also $(1 + 1/R)^{i+1}$-replacements of $z_{k_{i+1}}(u)$, and we are done.

Consider now the complementary case $k_{i+1} < k_i$. Then $S_u(i + 1)$ contains all vertices of $\{z_1(u), \ldots, z_{k_{i+1}-1}(u)\} \cap Ball'_u(i+1)$ with $(1+1/R)^{i+1}$-approximate estimates. (It is easy to verify that for every $j \in [0, h]$ and $v \in S_u(j)$, the estimate of v is a $(1 + 1/R)^j$-approximate one.) It follows that vertices from $\{z_{k_{i+1}}(u), z_{k_{i+1}+1}(u), \ldots, z_{k_i-1}(u)\}$ that belong to $S_u(i)$ were pushed out from $S_u(i + 1)$. For this to happen, the set $S_u(i + 1)$ must contain k distinct vertices with a better estimate than that of $z_{k_{i+1}}(u)$, i.e., with an estimate at most $(1 + 1/R)^i \cdot d_G(u, z_{k_{i+1}}(u))$. These k distinct vertices are $(1 + 1/R)^i$-replacements, and thus $(1 + 1/R)^{i+1}$-replacements too, of $z_{k_{i+1}}(u)$, proving that assertion 2 holds for u with respect to $i + 1$ in this case too.

Hence from now we assume that assertion 1 holds for u with respect to i. Thus, $S_u(i)$ contains all the $q_u(i)$ vertices of $Ball'_u(i)$, with estimates that may possibly be by a factor at most $(1 + 1/R)^i$ off their actual distance from u. The induction hypothesis with respect to i applies also to all these vertices $z_1(u), \ldots, z_{q_u(i)}(u)$ of $Ball'_u(i)$. [3]

For each $j \in [q_u(i)]$, let $q_j = q_{z_j(u)}(i)$.

Case 1. Suppose first that all these vertices also satisfy assertion 1 of the induction hypothesis for i, i.e., for every index $j \in [q_u(i)]$, the set $S_{z_j(u)}(i)$ contains the vertices $z_1(z_j(u)), \ldots, z_{q_j}(z_j(u))$ themselves with $(1 + 1/R)^i$-approximate estimates of their distance from $z_j(u)$.

Observe that for any vertex $z \in Ball'_u(i + 1)$, either $z \in Ball'_u(i)$, or $z \in Ball'_{z_j}(i)$ and z_j lies on a shortest $u - z$ path in G, for some index $j \in [q_u(i)]$. In both these cases, the $(1 + 1/R)$-approximate distance product computed on iteration $i + 1$ of the algorithm guarantees that the set $\hat{S}_u(i + 1)$ contains z, with a distance estimate which is at most $(1 + 1/R)(1 + 1/R)^i = (1 + 1/R)^{i+1}$ off the actual distance $d_G(u, z)$. Hence $Ball'_u(i + 1) \subseteq \hat{S}_u(i + 1)$.

[3] Actually, the indices of these vertices need not necessarily be consecutive with respect to the distance from u. But to keep the notation simple, we denote them as if they were consecutive.

Recall that $S_u(i+1)$ is the k-truncation of $\hat{S}_u(i+1)$, i.e., it contains k vertices of $\hat{S}_u(i+1)$ with the smallest estimates. If it contains all these vertices z with the aforementioned $(1+1/R)^{i+1}$-approximate estimates, then assertion 1 holds for u with respect to $i+1$. So (within Case 1) we are left with the subcase that at least one of these vertices $z \in Ball'_u(i+1)$ was pushed out of this k-truncation ($S_u(i+1)$). In the latter case, let $z' = z_r(u) = z_{k_{i+1}(u)}(u)$ be such a vertex with the smallest index r. It follows that $z_r \in S_u(i+1) \setminus Ball'_u(i+1)$, but all vertices of $Ball'_u(i+1)$ with smaller index (closer to u) belong to $S_u(i+1)$. By the above argument, these vertices have $(1+1/R)^{i+1}$-approximate distance estimates.

In addition, $S_u(i+1)$ must contain k vertices x whose estimate $\delta(u,x)$ satisfies

$$\delta(u,x) \leq \delta(u,z_r) \leq (1+1/R)^{i+1} \cdot d_G(u,z_r) \ .$$

As $d_G(u,x) \leq \delta(u,x)$ (this inequality holds for all estimates computed by our algorithm), it follows that each such vertex x is a $(1+1/R)^{i+1}$-replacement of $z_r = z_{k_{i+1}(u)}(u)$. This completes the proof for Case 1.

Case 2. In this case $S_u(i)$ contains all the $q_u(i)$ vertices of $Ball'_u(i)$ themselves (with $(1+1/R)^i$-approximate estimates), and at least one of these vertices $z_j \in Ball'_u(i)$ satisfies assertion 2 of the induction hypothesis with respect to i.

Recall that each $z_r \in Ball'_u(i+1)$ either belongs to $Ball'_u(i)$ (and then, by the assumption of this case, to $S_u(i)$), or to $Ball'_{z_j}(i)$, for some $z_j \in Ball'_u(i)$, and a shortest $u - z_r$ path traverses z_j.

If all $z_r \in Ball'_u(i+1)$ satisfy $z_r \in S_{z_j}(i)$ for some $z_j \in Ball'_u(i)$ (and a shortest $u - z_r$ path traverses z_j), then $Ball'_u(i+1) \subseteq \hat{S}_u(i+1)$. In this case the argument that we gave in Case 1 applies, and assertion of the lemma holds for u with respect to $i+1$ as well.

Otherwise, let r be the smallest index such that $z_r = z_r(u) \in Ball'_u(i+1) \cap Ball'_{z_j}(i)$, for some $z_j \in Ball'_u(i)$, and the shortest $u - z_r$ path contains z_j, and $z_r \notin S_{z_j}(i)$. (Moreover, $z_r \notin Ball'_u(i)$, and there exists no other shortest $u - z_r$ path that traverses some $z_t \in Ball'_u(i) \subseteq S_u(i)$, such that $z_r \in S_{z_t}(i)$. Indeed, in the latter case, z_r still reaches $\hat{S}_u(i+1)$, and the argument of Case 1 is applicable to it.)

For all vertices in $Ball'_u(i+1) \cap \{z_1, \ldots, z_{r-1}\}$, by the above argument, $S_u(i+1)$ contains them with $(1+1/R)^{i+1}$-approximate estimates. Also, by the induction hypothesis applied to to z_j, the set $S_{z_j}(i)$ contains k distinct $(1+1/R)^i$-replacements x of z_r that reach $\hat{S}_u(i+1)$, and they satisfy

$$\begin{aligned} \delta(u,x) &\leq (1+1/R) \cdot (\delta(u,z_j) + \delta(z_j,z_r)) \\ &\leq (1+1/R)((1+1/R)^i \cdot d_G(u,z_j) + (1+1/R)^i \cdot d_G(z_j,z_r)) \\ &= (1+1/R)^{i+1} \cdot d_G(u,z_r) \ . \end{aligned}$$

Hence all these vertices are $(1+1/R)^{i+1}$-replacements for z_r, and $S_u(i+1)$ contains either them, or k distinct vertices with yet smaller estimates. Thus assertion 2 holds for u with respect to $i+1$, proving the lemma. ◀

Our algorithm can also recover the paths with approximate distances for every $i \in V$ and $j \in N'_k(i)$. This is done by applying the algorithm from [40, Section 5], while executing the recursive calls in parallel.[4]

[4] Here is a brief sketch: Recall that we compute the witnesses for all the $O(\log k)$ distance products. Given a pair $i \in V$ and $j \in N'_k(i)$, if W is the witness matrix in the last iteration of the algorithm, then there are two cases: Either W_{ij} contains the middle vertex h (with at most $k/2$ hops to both i,j) on the approximate $i-j$ path. Then we can simply recurse in parallel on the pairs i,h and h,j, and then concatenate the paths. Otherwise, when $W_{ij} = 0$, we just return the edge (i,j).

▶ **Theorem 19.** *Let $G = (V, E)$ be a weighted directed n-vertex graph, and let $1 \leq k \leq n$ and $0 < \epsilon < 1$ be some parameters. Then there is a deterministic parallel algorithm that computes a $(1+\epsilon)$-approximation to all distances between any $u \in V$ and its k nearest neighbors, that runs in parallel time $O((\log^{O(1)} n)/\epsilon)$, using work*

$$\tilde{O}(\min\{n^{\omega}, k^{0.702} \cdot n^{1.882} + n^{2+o(1)}\}/\epsilon) \ .$$

Furthermore, for each $i \in V$ and $j \in N'_k(i)$, a path achieving the approximate distance can be reported in $O(\log k)$ parallel time and work proportional to the number of edges in it.

Note that for $k \leq n^{0.168}$ this work is $n^{2+o(1)}$, and while $k \leq n^{0.698}$ the work is smaller than n^{ω}.