Centralized and Parallel Multi-Source Shortest Paths via Hopsets and Fast Matrix Multiplication

Michael Elkin\textsuperscript{1} and Ofer Neiman\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel.
Email: \{elkinm,neimano\}@cs.bgu.ac.il

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 in the parallel (PRAM) models of computation. 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 $\tilde{O}(|E| \cdot n^\rho + n^{2+o(1)})$, 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.

We also devise efficient algorithms for computing $(1 + \epsilon)$-approximate shortest paths from each vertex to its $k$ nearest neighbors in directed graphs. Here too the running time is only $O(n^{2+o(1)})$ even for polynomially large $k \leq n^{0.168}$.

Our algorithm combines fast matrix multiplication with hopsets. Related ideas were formerly used in the context of the Congested Clique model by Censor-Hillel et al. [CDKL19]. That model, however, suppresses heavy local computations. We show that these computations can be replaced by fast matrix multiplication.

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 and parallel (PRAM) models of computation. Our focus is on computing $(1 + \epsilon)$-ASP for $S \times V$, for a set $S \subseteq V$ of sources, for $|S| = n^r$, 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 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 model.
1.1 Centralized Model

The classical algorithm of Dijkstra solves exact SSSP problem in time $O(|E| + n \log n)$ [FT87]. Thorup [Tho04] refined this bound to $O(|E| + n \log \log n)$ when weights are integers. Employing these algorithms for 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 [GM97], Alon et al. [AGM97] and Zwick [Zwi02] showed that one can use fast matrix multiplication (henceforth, FMM) to solve $(1 + \epsilon)$-APASP in time $O(n^\omega)$, where $\omega$ is the matrix multiplication exponent. ($n^\omega$ is the time required to multiply two $n \times n$ matrices. The currently best-known estimate on $\omega$ is $\omega < 2.372\ldots$ [Wil12, Gal14, CW90].)

By allowing larger approximation factors, one can achieve a running time of $\tilde{O}(n^2)$ for APASP. Specifically, Cohen and Zwick [CZ01] devised an algorithm for 3-APASP with this running time, and Baswana and Kavitha [BK06] 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 [Coh00], Elkin [Elk01] and Gitlitz and the current authors [EGN19] also showed that one can obtain a $(1 + \epsilon, \beta \cdot w)$-approximation for the ASP problem for $S \times V$ in time $O(|E| \cdot n^\alpha + |S| \cdot n^{1+1/\kappa})$, 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 [BK06].

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 $\tilde{O}(\min\{|S| \cdot n^2, n^\omega\})$. In the current paper we devise an algorithm that solves the problem in $\tilde{O}(n^{\omega(r)} + |E| \cdot n^{o(1)})$ time, \footnote{In fact, our result holds for arbitrary $0 < \epsilon < 1$, see Theorem 3.} 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 [Cop97] showed that for $r \leq 0.291$, $\omega(r) \leq 2 + o(1)$, and Le Gall and Urrutia [GU18] improved this bound further to $r \leq 0.313$. Denote $\alpha = 0.313$ 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 $n^\omega$, as the number of sources $|S|$ increases from $n^\alpha$ to $n$. When $S = V$, our bound matches the bound of Zwick [Zwi02].

Furthermore, Dor et al. [DHZ00] 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 $T(\tilde{O}(n))$ 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 [Zwi02]. Both algorithms apply directly to graphs with polynomially-bounded edge weights. Nevertheless, both of them can be used in conjunction with the Klein-Sairam’s reduction of weights [KS97] to provide the same bounds for graphs with arbitrary weights.

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 [Coh00]. Her bounds were improved in the last five years by [EN19a, EN19b, Li20, ASZ20], culminating in polylogarithmic time and $\tilde{O}(|E|)$ work [Li20, ASZ20]. All these aforementioned algorithms are randomized.
On the opposite end of the spectrum, algorithms of Galil and Margalit [GM97], Alon et al. [AGM97], and Zwick [Zwi02] (based on FMM) can be used in the PRAM setting. They give rise to deterministic polylogarithmic time $\tilde{O}(n^\omega)$ work [Zwi02] for the $(1+\epsilon)$-APSP problem.

By using sparse spanners, the algorithm of Cohen [Coh00] in conjunction with that of Baswana and Sen [BS03] 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 [EGN19] 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 $\Omega\left(\min\{|S| \cdot |E|, n^\omega\}\right)$. 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 [Zwi02] of $n^\omega$ (which applies for $(1+\epsilon)$-APSP) for all values of $r < 1$. The aforementioned reduction of [DHZ00] 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 a simultaneous submission by an overlapping group of authors [EM20], 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 Additional Results

We also employ our approach to the $k$-nearest neighbors (henceforth, $k$-NN) problem. Here $k$, $1 \leq k \leq n$, is a parameter, and the objective is to compute $(1+\epsilon)$-approximate shortest paths between every pair $(u, v)$ of vertices, such that $u$ is among the $k$ closest vertices to $v$ (or vice versa). Our algorithm for this problem applies even in directed weighted graphs. It requires polylogarithmic time and $\tilde{O}\left(\min\{n^\omega, k^{0.702} n^{1.882} + n^{2+o(1)}\}\right)$ work. For $k = O\left(n^{0.168}\right)$, this work is $n^{2+o(1)}$, and for $k = o\left(n^{0.698}\right)$, this bound is better than $n^\omega$, i.e., it improves the bound for $(1+\epsilon)$-APSP problem.

In all the settings discussed above, we can also return a succinct representation of the approximate shortest paths, rather than just report distances. Each path can be reported in time proportional to the number of edges it contains (and within a constant number of rounds and work proportional to the number of edges in PRAM). To achieve this, we employ path-reporting hopsets from [EN19a, EN19b], in conjunction with witnesses of matrix multiplication from [GM93, Zwi02].

### 1.4 Technical Overview

Our algorithm for the $(1+\epsilon)$-ASP for $S \times V$ problem starts with constructing a hopset. A graph $H = (V, E', w')$ is an $(\epsilon, \beta)$-hopset for a graph $G = (V, E, w)$, if for every vertex pair $u, v \in V$, we have

$$d_G(u, v) \leq d_{G,H}^{(\beta)}(u, v) \leq (1 + \epsilon)d_G(u, v).$$  \hspace{1cm} (1)

Here $d_{G,H}^{(\beta)}(u, v)$ stands for $\beta$-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 $\beta$ edges (henceforth, $\beta$-bounded path).

The hopset is then added to the graph. We then create a rectangular matrix $B$ with dimensions $|S| \times n$. Its $(s, v)$th entry contains $w(s, v)$ if $(s, v) \in E$, $w'(s, v)$ if $(s, v) \in E'$, and infinity otherwise. This rectangular
matrix is then iteratively multiplied by the adjacency matrix of \( G \cup H \), i.e., of the graph union the hopset. This operation is repeated \( \beta \) times, and the resulting matrix contains \( \beta \)-bounded distances in \( G \cup H \) for all pairs \( S \times V \). By equation (1), these distances are \((1 + \epsilon)\)-approximate distances in \( G \).

Note that matrix products that we need to compute are distance products, i.e., products over the ring \( \{\text{min}, +\} \). Algorithms for FMM that we use apply for ordinary algebraic matrix product, i.e., over the ring \( \{+,*\} \). Like [Zwi02], we overcome this issue by rounding and rescaling weights, which loses a factor of \( 1 + \epsilon \) in the approximation. (However, this loss is in no way unavoidable, as we use hopsets.)

Our algorithm for the \( k \)-NN problem in directed graphs is based on sparse matrix multiplication algorithm by Yuster and Zwick [YZ05] (which in turn employs rectangular matrix multiplication). This algorithm does not use hopsets, and its approximation error is due to rounding and rescaling. We also present a version that computes exact shortest paths.

Remarkably, while so far hopsets were used extensively in parallel/distributed/dynamic/streaming settings [Coh00, Ber09, Nan14, HKN14, HKN16, EN19a, EN19b, CDKL19], 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.

### 1.5 Related Work

Censor-Hillel et al. [CDKL19] devised related algorithms in the context of the Congested Clique model, In that model, however, processors are allowed to conduct heavy local computations in zero time, and the focus is on minimizing communication between them. As a result, distance products between large matrices can be computed locally, once some vertex keeps the matrices in its memory. Our main insight is that the heavy local computations that are suppressed in [CDKL19] can be replaced by using fast centralized or parallel matrix multiplication algorithms.

### 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 \( A^T \). For an \( n \times q \) matrix \( B \), define the distance product \( C = A \ast B \) by

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

for \( 1 \leq i \leq s \) and \( 1 \leq j \leq q \). We say that \( C' \) is a \( \delta \)-approximation to \( C \) if for all \( i, j \), \( C_{ij} \leq C'_{ij} \leq \delta \cdot C_{ij} \).

The following theorem is implicit in [Zwi02]. We will provide a sketch of the proof since we would like to apply it in parallel setting (and also for completeness).

**Theorem 1** ([Zwi02]). 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 an \((1 + \frac{1}{R})\)-approximation to \( A \ast B \) in time \( O(R \cdot n^{w(r)} \cdot \log M) \).

**Sketch.** It was shown in [AGM97] that the distance product \( C = A \ast B \) can be computed by defining \( \hat{A}_{ij} = (n+1)^{M-A_{ij}} \) and similarly \( \hat{B}_{ij} \). Then \( C \) can be derived from \( \hat{C} = \hat{A} \cdot \hat{B} \) by \( C_{ij} = 2M - |\log_{n+1} \hat{C}_{ij}| \).
Since the values of entries in the matrices $\hat{A}$ and $\hat{B}$ are quite large, each algebraic operation (when computing the standard product $\hat{A} \cdot \hat{B}$) will take $O(M \log n)$ time. So the running time will be $\tilde{O}(M \cdot n^{w(r)})$.

In order to reduce the dependence on the maximal weight $M$ to logarithmic, one can apply scaling: for each $k = 0, 1, \ldots, \log M - \log R$, define $A'$ by setting

\[
A'_{ij} = \begin{cases} \left\lfloor \frac{A_{ij}}{2^k} \right\rfloor & A_{ij} \leq R \cdot 2^k \\ \infty & \text{otherwise} \end{cases}
\]

and similarly define $B'$. Then compute $C' = A' \ast B'$ by the above method. [Zwi02] showed that a $(1 + \frac{1}{\pi})$-approximation to $A \ast B$ can be obtained by taking the scaled up minimum (entry-wise) of all these $C'$. The point is that each $A'$ and $B'$ have entries in $\{1, \ldots, R\} \cup \{\infty\}$. Hence the running time is indeed $\tilde{O}(R \cdot n^{w(r)} \log M)$ (the factor $O(\log M)$ comes from the number of different choices of $k$).

\[\square\]

**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 $C = A \ast B$ if for all $i, j$, $C_{ij} = A_{iW_{ij}} + B_{W_{ij}j}$. It was shown in [GM93, Zwi02] 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 [Zwi02, 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 1.4. A randomized construction of hopsets was given in [Coh00], see also [Nan14, HKN16, EN19a]. The following version was shown in [EN19b].

**Theorem 2 ([EN19b]).** 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 $H$ of size $O(n^{1+1/\kappa})$, which is an $(\epsilon, \beta)$-hopset (for every $0 < \epsilon < 1$ simultaneously) with $\beta = \left(\frac{n}{\epsilon}\right)^{O(\kappa)}$.

We note that a forthcoming paper [EM20] 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 [EM20] 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 [HKN16].

**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 [KS97] 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 [EN19a, Section 4] for more details.

We conclude that if there exists an algorithm that computes $(1 + \epsilon)$-approximate shortest paths for a graph $G$ with $n$ vertices and aspect ratio $M$, whose running time is $O(f(n, M))$, then the running time can be made $\tilde{O}(f(n, n))$ (as long as $f$ is convex, which is always the case here). Since in our algorithms the dependence on the aspect ratio will be logarithmic, in all that follows we may assume $M = \text{poly}(n)$. 

5
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 Approximate MSP($G, S, \epsilon$)

1. Let $H$ be an $(\epsilon, \beta)$-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 $B^{t+1}$ be a $(1 + 1/R)$-approximation to $B^{(t)} \ast A$;
7. end for
8. return $B^{(\beta)}$;

The first step is to compute an $(\epsilon, \beta)$-hopset $H$, for a parameter $\kappa \geq 1$ with $\beta = \left(\frac{\kappa}{\epsilon}\right)^{O(\kappa)}$ as in Theorem 2. 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 $j \in V$, $B_{ij}^{(t)}$ is a $(1 + 1/R)^{t-1}$-approximation to $d_{G \cup H}^{(t)}(i, j)$. 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 1.** 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 $d_G^{(t)}(i, j)$, for some positive integer $t$. Let $C = B \ast A$ and $C'$ be a $c'$-approximation to $C$. Then, for all $i, j$, $C'_{ij}$ is a $c \cdot c'$-approximation to $d_{G}^{(t+1)}(i, j)$.

**Proof.** Consider a pair of vertices $i \in S$ and $j \in V$. By definition of the $\ast$ operation, $C_{ij} = \min_{1 \leq k \leq n} \{B_{ik} + A_{kj}\}$. Let $\pi$ 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 $\pi$. 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 $d_{G}^{(t+1)}(i, j)$. The assertion of the claim follows since $C_{ij} \leq C'_{ij} \leq c' \cdot C_{ij}$.

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

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

By Claim 1, $B^{(\beta)}$ is a $(1 + 1/R)^{\beta - 1} \leq e^\epsilon = 1 + O(\epsilon)$ approximation to $d_{G \cup H}^{(\beta)}(u, v)$ for all $u \in S$ and $v \in V$. Since $H$ is an $(\epsilon, \beta)$-hopset, $B^{(\beta)}$ 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 [Zwi02, 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 \((1 + \epsilon) \cdot d_{G \cup H}(u, v)\). 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 [EN19b] have a path reporting property. That is, each hopset edge of weight \( W' \) has a corresponding path \( \pi \) of length \( W' \) in \( G \), and every vertex on \( \pi \) stores its neighbors on the path. Thus, we can obtain a \( u-v \) path in \( G \) in time proportional to its number of edges.

With conclude with the following theorem.

**Theorem 3.** 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 deterministic algorithm that computes a \((1 + \epsilon)\)-approximation to all distances in \( S \times V \) that runs in time

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

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 \( \kappa \) as a slowly growing function of \( n \), e.g. \( \kappa = (\log \log n) / \log \log \log n \), so that \( \kappa^\kappa \leq \log n \) and \( n^{1/\kappa} = n^{o(1)} \), and obtain running time \( \tilde{O}(n^{w(r)} + |E| \cdot n^{o(1)}) \) (for a constant \( \epsilon > 0 \)). We stress that for all \( r \leq 0.313 \), a result of [GU18] 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 [Zwi02].

Observe that if \( r > 0.313 \), then we can choose \( \kappa \) 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 \( |E| \leq n^{2 - \delta} \) 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 \( 0 \leq r \leq 1 \). In both cases we obtain \( \beta = (1/\epsilon)^{O(1)} \), so our algorithm to compute \((1 + \epsilon)\)-approximate shortest paths for \( S \times V \) will have running time \( \tilde{O}(n^{w(r)}/\epsilon^{O(1)}) \).

## 4 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 [EN19b]. A deterministic analogue of it was recently shown in [EM20].

**Theorem 4 ([EN19b]).** 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 an \((\epsilon, \beta)\)-hopset with \( O(n^{1+1/\kappa} \cdot \log^* n) \) edges where \( \beta = \left(\frac{2}{\epsilon}\right)^{O(\kappa)} \).
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 [HP98]. 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 $O(T(n))$ work.

The algorithm of Theorem 1 boils down to $O(\log M) = O(\log n)$ standard matrix multiplications (albeit the matrices have large entries, with $O(R \log n)$ bits). Thus, we can compute an $(1 + \frac{1}{R})$-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 $O(R \cdot n^{w(r)})$ work.

The path-reporting mechanism can be adapted to PRAM, by running the [Zwi02] algorithm sequentially. Since we have only $\beta$ iterations, the parallel time will be only $O(\beta)$ (which is a constant independent of $n$). 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 5.** 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 $(\frac{\log n}{\epsilon})^{O(\kappa)}$ 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 $\kappa$ to be an arbitrarily large constant, and obtain a polylogarithmic time and work $\tilde{O}(n^{w(r)} + |E| n^{1/\kappa}).$

## 5 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 [YZ05] 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 [GU18], $\alpha \geq 0.313$. Let $\gamma = \frac{4}{\sqrt{77}}$.

**Theorem 6 ([YZ05]).** The product of two $n \times n$ matrices each with at most $m$ nonzeros can be computed in time

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

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 2.** 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

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

(2)
Proof. The \((1 + \frac{1}{\log k})\)-approximate distance product of Theorem 1 involves \(O(\log n)\) 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., [SV81]). By the reduction described in the proof of Theorem 1, the resulting matrices will have \(O(m)\) nonzeros (and entries of size \(O(n^{R})\)), so the parallel time to compute each such multiplication is \(O(R \log^{O(1)} n)\). Using the currently known bounds on \(\omega\) and \(\alpha\), by Theorem 6 the work required is as in (2).

For an \(n \times n\) matrix \(A\), denote by \(\text{trun}_{k}(A)\) the matrix \(A\) in which every column is truncated to contain only the smallest \(k\) entries, and \(\infty\) everywhere else. Clearly this operation can be executed in \(\text{poly}(\log n)\) 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 3.** Let \(G\) be a weighted directed graph. For some \(t \geq 1\), 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 \(d^{(t)}_{G}(i, j)\), and \(\infty\) for \(j \notin N_{k}(i)\). Then, if \(B\) is a \(c'\)-approximation to \(A^{T} \ast A\), for each \(i\) and \(j \in N_{k}(i)\), then we have that \(B_{ij}\) is a \(c \cdot c'\)-approximation to \(d^{(2t)}_{G}(i, j)\).

**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. \(A_{kj}\)) is a \(c\)-approximation to \(d^{(t)}_{G}(i, h)\) (resp. \(d^{(t)}_{G}(h, j)\)). By definition of distance product, \((A^{T} \ast A)_{ij} \leq c \cdot d^{(t)}_{G}(i, h) + c \cdot d^{(t)}_{G}(h, j) \leq c \cdot d^{(2t)}_{G}(i, j)\). So \(B_{ij}\) is a \(c \cdot c'\)-approximation to \(d^{(2t)}_{G}(i, j)\).

Our algorithm to compute approximate shortest paths to \(k\) nearest neighbors is done by simply computing \(\log k\) times an approximate distance product, truncating each time to the smallest \(k\) entries in each column. See Algorithm 2.

---

**Algorithm 2** \textsc{Approx}-\textsc{k-NN}(\(G, \epsilon\))

1: Let \(A\) be the adjacency matrix of \(G\);
2: Let \(R = \lceil \log k / \epsilon \rceil\);
3: for \(i\) from 1 to \(\lceil \log k \rceil\) do
4: Let \(A\) be a \((1 + 1/R)\)-approximation to \((\text{trun}_{k}(A))^{T} \ast \text{trun}_{k}(A)\);
5: end for
6: return \(\text{trun}_{k}(A)\);

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 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 3, 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 \((1 + \frac{1}{\log k})^{\lceil \log k \rceil} = 1 + O(\epsilon)\).

Our algorithm can also recover the paths for the approximate distance of every \(i \in V\) and \(j \in N_{k}(i)\) by applying the algorithm from [Zwi02, Section 5], while executing the recursive calls in parallel.\(^2\)

\(^2\)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
Theorem 7. 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
\[
\tilde{O}(\min\{n^c, k^{0.702} \cdot n^{1.882} + n^{2+o(1)}\} / \epsilon).
\]
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 \( n^c \).

5.1 Exact Distances

Here we show an efficient parallel algorithm, that given a weighted directed graph, computes the exact distances from each vertex to its \( k \) nearest neighbors, which runs in parallel time \( O(\min\{n^c, k^{0.702} \cdot n^{1.882} + n^{2+o(1)}\} / \epsilon) \).

Algorithm 3

```plaintext
1: Let \( A \) be the adjacency matrix of \( G \);
2: for \( i \) from 1 to \( \lceil \log k \rceil \) do
3:   Let \( A = (\text{trun}_k(A))^T \ast \text{trun}_k(A) \);
4: end for
5: return \( \text{trun}_k(A) \);
```

As there are \( O(\log k) \) iterations, we get \( \log^{O(1)} n \) parallel time and \( \tilde{O}(k \cdot n^2) \) work. The correctness of this algorithm follows from Claim 3 with \( c = c' = 1 \), and the fact that the shortest path between \( k \) nearest neighbors has at most \( k \) edges. The following theorem summarizes this result.

Theorem 8. Let \( G = (V, E) \) be a weighted directed \( n \)-vertex graph, and let \( 1 \leq k \leq n \). Then there is a deterministic parallel algorithm that computes all distances between any \( u \in V \) and its \( k \) nearest neighbors, that runs in parallel time \( \log^{O(1)} n \), using work \( \tilde{O}(k \cdot n^2) \).

Furthermore, for each \( i \in V \) and \( j \in N_k(i) \), a shortest path can be reported in \( O(\log k) \) parallel time and work proportional to the number of edges in it.

We remark that for general dense graphs there is no known algorithm to compute APSP with \( n^{3-\epsilon} \) work and \( \text{poly}(\log n) \) time. Hence this result is meaningful for essentially all values of \( k = o(n) \).

References

[AGM97] 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.
[ASZ20] Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. 2020. to appear in STOC.

[Ber09] 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.

[BK06] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In FOCS, pages 591–602, 2006.

[BS03] 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.

[CDKL19] 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.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. J. ACM, 47(1):132–166, 2000.

[Cop97] Don Coppersmith. Rectangular matrix multiplication revisited. J. Complex., 13(1):42–49, 1997.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. J. Symb. Comput., 9(3):251–280, 1990.

[CZ01] Edith Cohen and Uri Zwick. All-pairs small-stretch paths. J. Algorithms, 38(2):335–353, 2001.

[DHZ00] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. SIAM J. Comput., 29:1740–1759, 2000.

[EGN19] Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths and PRAM distance oracles in weighted graphs. CoRR, abs/1907.11422, 2019.

[Elk01] M. Elkin. Computing almost shortest paths. In Proc. 20th ACM Symp. on Principles of Distributed Computing, pages 53–62, 2001.

[EM20] Michael Elkin and Shaked Matar. Deterministic pram approximate shortest paths in polylogarithmic time and near-linear work. 2020. manuscript.

[EN19a] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. SIAM J. Comput., 48(4):1436–1480, 2019.

[EN19b] 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.
Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

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.

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

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

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.

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.

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.

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

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

Jason Li. Faster parallel algorithm for approximate shortest path. 2020. to appear in STOC.

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.

Yossi Shiloach and Uzi Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *J. Algorithms*, 2(1):88–102, 1981.

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.

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.
[YZ05] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.