Parallel Reachability in Almost Linear Work and Square Root Depth

Arun Jambulapati
ICME
Stanford University
Palo Alto, CA
jmblpati@stanford.edu

Yang P. Liu
Department of Mathematics
Stanford University
Palo Alto, CA
yangpatil@gmail.com

Aaron Sidford
MS&E
Stanford University
Palo Alto, CA
sidford@stanford.edu

Abstract—**In this paper we provide a parallel algorithm that given any n-node m-edge directed graph and source vertex s computes all vertices reachable from s with $\widetilde{O}(m)$ work and $n^{1/2+o(1)}$ depth with high probability in n. This algorithm also computes a set of $\widetilde{O}(n)$ edges which when added to the graph preserves reachability and ensures that the diameter of the resulting graph is at most $n^{1/2+o(1)}$. Our result improves upon the previous best known almost linear work reachability algorithm due to Fineman [1] which had depth $\widetilde{O}(n^{2/3})$.**

Further, we show how to leverage this algorithm to achieve improved distributed algorithms for single source reachability in the CONGEST model. In particular, we provide a distributed algorithm that given a n-node digraph of undirected hop-diameter D solves the single source reachability problem with $\widetilde{O}(n^{1/2} + n^{1/3+o(1)}D^{2/3})$ rounds of the communication in the CONGEST model with high probability in n. Our algorithm is nearly optimal whenever $D = O(n^{1/4-\epsilon})$ for any constant $\epsilon > 0$ and is the first nearly optimal algorithm for general graphs whose diameter is $\Omega(n^\delta)$ for any constant δ.

I. INTRODUCTION

Given a n-vertex m-edge *directed graph* or *digraph* $G = (V, E)$ and a vertex $s \in V$ the *single source reachability problem* asks for the set of vertices $T \subseteq V$ reachable from s, i.e. the vertices $t \in V$ for which there is a s to t path in G. This problem is perhaps one of the simplest graph optimization problems. It easily solvable in linear, $O(n + m)$, time by any of a number of classic graph exploration algorithms, e.g. breadth first search (BFS), depth first search (DFS), etc. and is often one of the first graph problems considered in an introductory algorithms course. The reachability problem is prevalent in theory and practice and algorithms for solving it are leveraged to solve more complex graph optimization problems, including computing strongly connected components, shortest paths, maximum flow, spanning arborescenses, etc.

Given the fundamental nature of the reachability problem and the utility of reachability algorithms, the reachability problem is often one of the first considered when investigating resource constrained computation. However, despite the simplicity of solving single source reachability with optimal time complexity, obtaining optimal algorithms for this problem under constraints of parallelism [1]–[3], distributed computation [4], [5], space utilization [6], [7], and

dynamic updates [8] are all notoriously difficult. In many cases, reachability lies at the heart of well-known long-standing open problems in complexity theory. For example, reachability is known to be complete for non-deterministic log space (NL) computation [6] and obtaining sufficiently efficient dynamic reachability algorithms would break the popular 3-SUM conjecture and refute the strong exponential time hypothesis (SETH) [8].

In the parallel and distributing models of computation, single-source reachability is itself a fundamental barrier towards achieving efficient graph optimization. Despite extensive study, until a recent breakthrough of Fineman [1] the best parallel reachability algorithm all required trading off depth versus work and all known algorithms that had linear work had the trivial $O(n)$ depth. In distributed computation, for example the popular CONGEST-model [9], though there have been algorithmic improvements over the trivial $O(n)$ round protocol [4], [5], the best known algorithms are polynomial factors larger then the known $\Omega(D + \sqrt{n})$-round lower bound (where here D denotes the undirected hop-diameter of the graph) [10].

Given these complexity theoretic barriers related to reachability and the prevalance of parallel and distributed models of computation, improved parallel and distributed reachability algorithms are highly coveted. In this paper we provide improved reachability algorithms under each computational model. The main results of this paper are as follows.

Theorem 1 (Parallel Reachability). *There is a parallel algorithm that given a n-node m-edge digraph solves the single source reachability problem with work $\widetilde{O}(m)$ and depth $n^{1/2+o(1)}$ with high probability in n.*

Theorem 2 (Distributed Reachability). *There is a distributed algorithm that given a n-node digraph of undirected hop-diameter D solves the single source reachability problem with $\widetilde{O}(n^{1/2} + n^{1/3+o(1)}D^{2/3})$ rounds of the communication in the CONGEST model with high probability in n.*

Theorem 1 improves the previous best $\widetilde{O}(n^{2/3})$ depth bound achieved by a parallel nearly linear work algorithm due to Fineman [1]. Theorem 2 is nearly optimal whenever $D = O(n^{1/4-\varepsilon})$ for some $\varepsilon > 0$ due to a known $\tilde{\Omega}(\sqrt{n} + D)$

lower bound [10] and is the first nearly optimal algorithm for general directed graphs where $D = \Omega(n^\delta)$ for for constant $\delta > 0$. (See Section I-B for a more detailed comparison to and discussion of previous work.)

Our results build upon a recent breakthrough result of Fineman [1] and a simple, yet-powerful decompositional tool regarding reachability known as *hopsets* or *shortcuts*. Shortcuts are edges which if added to the graph, do not change which pairs of vertices can reach each other. It is well known that in a graph of diameter D, i.e. largest shortest-path distance between a pair of vertices which can reach each other is D, we can compute reachability in work $O(m)$ and depth $O(D)$. Consequently, a natural approach towards improved reachability algorithms would simply be to find a small set of shortcuts in nearly linear time which decrease the diameter of a graph.

Computing shortcutters is a tantalizing approach to improved reachability algorithms. A simple folklore random sampling argument can be used to show that for every n-node digraph and every parameter t there exists a set of $O(t^2 \log^2 n)$ shortcuts such that adding them makes the diameter of the graph is at most $O(n/t)$. Consequently, there is a nearly linear, $O(n \log^2 n)$, number of edges which would reduce the diameter to $O(\sqrt{n})$. Finding such a set of shortcuts in nearly linear work and $O(\sqrt{n})$ depth would immediately yield linear work $O(\sqrt{n})$ depth algorithms for reachability. Unfortunately, even obtaining almost linear time algorithms for constructing a set of shortcutters which reduce the diameter to almost square root n was open prior to this work.

Fortunately, recent work of Fineman [1] provided some hope towards achieving this goal. This work provided the first nearly linear time algorithm for computing a nearly linear number of shortcutters which provide any polynomial diameter reduction from the trivial $O(n)$ bound. In particular, Fineman's work provided a nearly linear time algorithm which computed with high probability a nearly linear number of shortcutters that decrease the diameter to $\widetilde{O}(n^{2/3})$ and leveraged this result to obtain a parallel reachability algorithm with $\widetilde{O}(m)$ work and $\widetilde{O}(n^{2/3})$ depth that succeeds with high probability in n.

Though an impressive result and a considerable break-through, this work left open the question of how well parallel almost linear work algorithms could match the depth bound that would be optimistically predicted by hopsets, i.e. $O(n^{1/2})$. Further, this work left open the question of whether these improvements could be transferred to additional resource constrained computational problems. In this paper we make progress on both questions with Theorem 1 and Theorem 2. We provide an $\widetilde{O}(m)$ work and $n^{1/2+o(1)}$ depth algorithm that computes a set of $\widetilde{O}(n)$ shortcutters that reduces the diameter to $n^{1/2+o(1)}$ and use this to achieve improved distributed algorithms.

We achieve our results by strengthening and simplifying

parts of Fineman's algorithm (see Section III for an overview of the approach). Further, we provide a fairly general strategy to turn improved parallel reachability algorithms into improved distributed algorithms in the CONGEST-model, building off approaches of [4], [5]. Interestingly, we show that even Fineman's algorithm can be be modified to achieve improved distributed algorithms (albeit with weaker bounds).

Ultimately we hope this work sheds light on the structure of single source reachability, may lead to faster reachability in more resource constrained computational environments, and may ultimately lead to more practical massive scale graph processing.

Paper Outline: The rest of the paper is structured as follows. In the remainder of this introduction we formally state our results in Section I-A and compare to previous work in Section I-B. In Section II we cover technical preliminaries and leverage this notation to provide a more technical approach overview in Section III. In Section IV we then provide an $\widetilde{O}(m)$ time algorithm for computing $\widetilde{O}(n)$ shortcuts which decrease the diameter to $n^{1/2+o(1)}$ with high-probability. This serial algorithm demonstrates many of the key insights we ultimately build upon to achieve our parallel reachability algorithms in Section V and our distributed reachability algorithms in Section VI.

A. Our results

In this paper we provide several parallel and distributed algorithms for efficiently constructing diameter-reducing hopsets and computing reachability in digraphs. Here we provide a brief overview of these results. Throughout this section (and the rest of the paper) we use $\widetilde{O}(\cdot)$ to hide polylogarithmic factors in n and we use w.h.p. as shorthand for "with high probability in n" where in both cases n is used to denote the number of vertices in the original input graph.

First, in Section IV we provide a sequential algorithm for efficiently computing diameter-reducing hopsets. This algorithm improves upon the previous best diameter bound of $\widetilde{O}(n^{2/3})$ [1], known to be achievable by a nearly linear time algorithm. Our main result is as follows:

Theorem 3 (Sequential Diameter Reduction). *For any parameter k, there is an algorithm that given any n-node and m-edge digraph in $\widetilde{O}(mk)$ time computes $\widetilde{O}(nk)$ shortcuts such that adding these edges to the graph reduces to the diameter to $n^{1/2+O(1/\log k)}$ w.h.p.*

This result forms the basis for our improved parallel reachability algorithms. We first argue that careful modification and application of the algorithmic and analytic insights of Section IV suffice to obtain similar diameter improvements from a parallel algorithm. Our main result of Section V is the following parallel analog of Theorem 3.

Theorem 4 (Parallel Diameter Reduction). *For any parameter k, there is a parallel algorithm that given any n-node and m-edge digraph with $\widetilde{O}(mk + nk^2)$ work and $\mathrm{poly}(k) \cdot n^{1/2 + O(1/\log k)}$ depth computes a set of $\widetilde{O}(nk)$ shortcuts such that adding these edges to the graph reduces the diameter to $n^{1/2 + O(1/\log k)}$ w.h.p.*

Setting $k = O(\log n)$ immediately gives the following corollary.

Corollary I.1. *There is a parallel algorithm that given any n-node and m-edge digraph performs $\widetilde{O}(m)$ work in depth $n^{1/2 + o(1)}$ and computes a set of $\widetilde{O}(n)$ shortcuts such that adding these edges to the graph reduces the diameter to $n^{1/2 + o(1)}$ w.h.p.*

Further, since single source reachability can be solved by BFS in linear work and depth proportional to the diameter of the graph applying this corollary and then leveraging BFS immediately proves Theorem 1, our main result on a parallel solution to single source reachability.

We leverage this parallel reachability result to provide our improved distributed reachability algorithms in Section VI. Formally we consider the CONGEST-model where given a n-node digraph G there is a separate processor for each node and in every round, for every vertex u, its processor may send $O(\log n)$-bits of information to each of its neighbors (i.e. vertices v for which either (u, v) or (v, u) is an edge). Here the distributed reachability question we consider is how to design a messaging scheme so that each node learns if it is reachable from a single given source in as few rounds as possible.

Our main result regarding such distributed algorithms is given by Theorem 2. Formally, we show that if D is the diameter of the undirected graph associated with G (i.e. there is an edge in the undirected graph between u and v if and only if either (u, v) of (v, u) is an edge in G) then we can design a distributed algorithm that solves single source reachability in $\widetilde{O}(n^{1/2} + n^{1/3 + o(1)} D^{2/3})$ rounds of communication in the CONGEST-model with high probability in n. Due to a known lower bound of $\tilde{\Omega}(\sqrt{n} + D)$ [10] this result is nearly optimal whenever $D = O(n^{1/4 - \varepsilon})$ for any $\varepsilon > 0$. Further, to the best of our knowledge this is the first nearly optimal algorithm for general directed graphs when $D = \Omega(n^\delta)$ for any constant $\delta \in (0, 1/4)$.

Interestingly, we achieve this improved distributed algorithm by following a fairly general framework inspired by [4], [5]. We argue that there is a fairly general procedure for converting "nice enough" work-efficient parallel reachability algorithms into improved bounds on distributed reachability. This procedure first computes a set of shortcuts using known prior work on distributed algorithms [9]. Leveraging these shortcuts, the procedure then solves reachability by applying a work-efficient parallel reachability algorithm over the a graph with shortcuts added. We then argue that if the

	Work	Span
Parallel BFS	$O(m)$	$\widetilde{O}(n)$
Parallel Trans. Closure	$\widetilde{O}(n^\omega)$	$\widetilde{O}(1)$
Spencer's [2]	$\widetilde{O}(m + n\rho^2)$	$\widetilde{O}(n/\rho)$
UY [3]	$\widetilde{O}(m\rho + \rho^4/n)$	$\widetilde{O}(n/\rho)$
Fineman's [1]	$\widetilde{O}(m)$	$\widetilde{O}(n^{2/3})$
This paper	$\widetilde{O}(m)$	$n^{1/2 + o(1)}$

Table I
SUMMARY OF PREVIOUS RESULTS FOR PARALLEL SINGLE-SOURCE DIGRAPH REACHABILITY. HERE $\rho \in [1, n]$ CAN BE CHOSEN ARBITRARILY AND $\omega < 2.373$ DENOTES THE MATRIX MULTIPLICATION EXPONENT. THIS TABLE WAS MODIFIED FROM THE ONE IN [1].

reachability algorithm is "nice enough" we can bound the distributed round complexity of the resulting algorithm as a function of the work and depth of the parallel reachability algorithm. Ultimately, Section VI shows that in addition to proving Theorem 2 we could have used this framework and Fineman's recent work [1] to obtain improved distributed algorithms (even without using our work of Section V); albeit with worse bounds.

B. Related Work

The problem of computing single-source reachability from source s in n-node m-edge digraph G is one of the most fundamental questions in the theory of parallel algorithms. A more complete survey of previous results can be found in [1] and we describe them briefly here.

Two folklore algorithms exist for parallel reachability. First, the complete transitive closure of G can be computed in parallel by repeatedly squaring the adjacency matrix of G. This achieves a polynomial running time with polylogarithmic depth, but the $\widetilde{O}(n^\omega)$ work algorithm where $\omega < 2.373$ is the matrix multiplication constant [11] is currently not known to be nearly linear even for dense graphs. Second, a straightforward modification to standard breadth-first search, called "Parallel BFS," enables us to compute the single-source reachability from s in $O(m)$ time and $O(n)$ depth. This has near optimal work, but its depth is trivial, i.e. it is essentially a fully serial algorithm.

Procedures by Spencer [2] and Ullman and Yannakis [3] provide work-depth tradeoffs that interpolate between the extremes of these two naive procedures, yet neither improve upon the $\widetilde{O}(n)$ depth bound in the case of $\widetilde{O}(m)$ work. Fineman [1] provided a breakthrough by demonstrating the $\widetilde{O}(m)$ work algorithm for single-source reachability achieving sublinear depth $\widetilde{O}(n^{2/3})$. Our results build on those of Fineman by improving the depth to $n^{1/2 + o(1)}$. These results are summarized in Table I.

Our algorithm, like [1], solves a more general problem than single-source reachability: we show show that our algorithm gives work-efficient parallel construction of an $\widetilde{O}(n)$-edge hopset of diameter $n^{1/2 + o(1)}$. With this hopset

we can answer arbitrary single-source reachability queries in nearly-linear work and $n^{1/2+o(1)}$ depth. A natural question is whether our hopset construction can be improved. However even from the perspective of constructibility the true tradeoff between diameter and number of added edges is not known. As mentioned previously a straightforward random construction provides a $O(t^2 \log^2 n)$-edge hopset with diameter $O(n/t)$, but it is not known how to improve upon this result in any regime. Building off of [12], [13] demonstrates that $O(n)$-edge hopsets cannot ensure diameter less than $O(n^{1/6})$ and $O(m)$-edge hopsets cannot achieve $O(n^{1/11})$ diameter; no results are known in other parameter regimes. Nevertheless we conjecture that in the case of nearly-linear sized hopsets our construction is tight and therefore any polynomial improvement to our algorithm must somehow avoid the hopset paradigm.

Distributed algorithms in the CONGEST model [9] have been studied extensively over the past two decades. Though there have been multiple improvements to the round complexity of approximately solving single source shortest paths in this model (see for example [14]–[16] for the relevant literature), there has been comparatively little progress on the solving the same problem on directed graphs [4], [15], [16]. For the single-source reachability problem considered in this paper the previous state of the art for this problem is due to [5], which solved the problem in $\widetilde{O}(D + \sqrt{n}D^{1/4})$ rounds w.h.p. This algorithm in turn improved upon the $\widetilde{O}(D + \sqrt{n}D^{1/2})$ w.h.p. round bound of [4], which was (to the best of our knowledge) the first non-trivial distributed algorithm for this problem.

II. PRELIMINARIES

We denote vertex set of a graph G by $V(G)$, and the edge set by $E(G)$. We simply write these as V and E when the graph G is clear from context. For a $V' \subseteq V$, we let $G[V']$ denote the induced subgraph on V', i.e. the graph with vertices V' and edges of G that have both endpoints in V'.

Digraph relations: Let G be a *directed graph* or *digraph* for short. We say that $u \preceq v$ if there is a directed path from u to v in G. In this case we say that u *can reach* v, or that v *is reachable from* u. We say that $u \npreceq v$ if there is no directed path from u to v in G. In this case we say that u cannot reach v, or that v is not reachable from u. When $u \preceq v$ and $v \preceq u$ we say that u and v are in the same strongly connected component. We define the *descendants* of v to be $R_G^{\mathrm{Des}}(v) \overset{\text{def}}{=} \{u \in V(G) : v \preceq u\}$ and the *ancestors* of v to be $R_G^{\mathrm{Anc}}(v) \overset{\text{def}}{=} \{u \in V(G) : u \preceq v\}$. We say that u and v are *related* if $u \preceq v$ or $v \preceq u$. We define the *related vertices* of v as $R_G(v) \overset{\text{def}}{=} R_G^{\mathrm{Des}}(v) \cup R_G^{\mathrm{Anc}}(v)$. Throughout, the letter R we use in the notation should be read as "related" or "reachable". We say that u is *unrelated* to vertex v if $u \in V(G) \backslash R_G(v)$.

We extend this notation to subsets $V' \subseteq V$ in the natural way. We define the ancestors, descendants, and related vertices to V' as

$$R_G^{\mathrm{Des}}(V') \overset{\text{def}}{=} \bigcup_{v \in V'} R_G^{\mathrm{Des}}(v)\, , \ R_G^{\mathrm{Anc}}(V') \overset{\text{def}}{=} \bigcup_{v \in V'} R_G^{\mathrm{Anc}}(v),$$
$$\text{and } R_G(V') \overset{\text{def}}{=} R_G^{\mathrm{Des}}(V') \cup R_G^{\mathrm{Anc}}(V').$$

We say that a vertex v is related to a subset V' if $v \in R_G(V')$. When the graph G is clear from context, we will often drop the G subscript and simply write (for example) $R^{\mathrm{Des}}(v), R^{\mathrm{Anc}}(v)$, and $R(v)$ instead of $R_G^{\mathrm{Des}}(v), R_G^{\mathrm{Anc}}(v), R_G(v)$.

We further extend this notation to induced subgraphs of G. Let G' be a subgraph of G, possibly with a different set of vertices and edges than G. We say that $u \preceq_{G'} v$ if there is a directed path from u to v in the subgraph G'; we say that v is reachable from u *through* G' in this case. Define

$$R_{G'}^{\mathrm{Des}}(v) \overset{\text{def}}{=} \{u \in V(G') : v \preceq_{G'} u\},$$
$$R_{G'}^{\mathrm{Anc}}(v) \overset{\text{def}}{=} \{u \in V(G') : u \preceq_{G'} v\},$$
$$\text{and } R_{G'}(v) \overset{\text{def}}{=} R_{G'}^{\mathrm{Des}}(v) \cup R_{G'}^{\mathrm{Anc}}(v).$$

We similarly extend this definition to subsets of vertices $V' \subseteq V(G')$ as

$$R_{G'}^{\mathrm{Des}}(V') \overset{\text{def}}{=} \cup_{v \in V'} R_{G'}^{\mathrm{Des}}(v), R_{G'}^{\mathrm{Anc}}(V') \overset{\text{def}}{=} \cup_{v \in V'} R_{G'}^{\mathrm{Anc}}(v),$$
$$\text{and } R_{G'}(V') \overset{\text{def}}{=} R_{G'}^{\mathrm{Anc}}(V') \cup R_{G'}^{\mathrm{Des}}(V').$$

As our algorithm performs recursion on subgraphs of G, this notation enables us to reference specific subproblems as our algorithm progresses.

A *shortcut* refers to adding an edge (u, v) to a graph G where $u \preceq v$ in G. Adding the edge does not affect the reachability structure of G. A *shortcutter* v is a node we add shortcut edges to and from. A *hopset* refers to a collection of shortcuts.

Paths: Our analysis will consider paths in the graph as well as the relations between the vertices on the path and other vertices in the graph. Let G be a digraph. We denote a path $P = \langle v_0, v_1, \ldots, v_\ell \rangle$, where all the v_i are vertices of G and $(v_i, v_{i+1}) \in E(G)$. Here, the length of the path is ℓ, where we have that $v_0 \preceq v_1 \preceq \ldots \preceq v_\ell$. We say that the head of the path is $\mathrm{head}(P) \overset{\text{def}}{=} v_0$ and the tail is $\mathrm{tail}(P) \overset{\text{def}}{=} v_\ell$. We now make the following definitions.

Path-related vertices: We adopt a similar convention as [1] and for a path $P = \langle v_0, v_1, \ldots, v_\ell \rangle$ we say that v is *path-related* if $v \in R_G(P)$. Further, for any path P in digraph G, we define $s(P, G) \overset{\text{def}}{=} |R_G(P)|$ as the number of path-related vertices. All path-related vertices are one of the following three types:

- **Descendants:** We say that a vertex v is a descendant of the path P if $v \in R_G^{\mathrm{Des}}(P) \backslash R_G^{\mathrm{Anc}}(P)$. Note that this holds if and only if $v_0 \preceq v$ and $v \npreceq v_\ell$.

- **Ancestors:** We say that a vertex v is an ancestor of the path P if $v \in R_G^{\mathrm{Anc}}(P) \backslash R_G^{\mathrm{Des}}(P)$. Note that this holds if and only if $v_0 \not\preceq v$ and $v \preceq v_\ell$.
- **Bridges:** We say that a vertex v is a bridge of the path P if $v \in R_G^{\mathrm{Des}}(P) \cap R_G^{\mathrm{Anc}}(P)$. Note that this holds if and only if $v_0 \preceq v$ and $v \preceq v_\ell$.

A vertex which is not a descendant, ancestor, or bridge for a path P is called *unrelated* to P. Later in Section V we explain how to extend all these definitions to the distance-limited case.

Subproblems: During our algorithms' recursions, we will make reference to the induced recursive calls made. Consider a graph G and a path P in G. During a call to an algorithm on the graph G, we define a *subproblem* to be an induced subgraph $G[V']$ along with a subpath P' of P which lies inside $G[V']$ on which we perform a recursive execution.

Miscellaneous: We let $B(n,p)$ be the binomial random variables over n events of probability p. We have the following standard fact about binomial random variables:

Lemma II.1 (Chernoff Bound). *Let $X \sim B(n,p)$ be a binomial random variable. Then*

$$\Pr\left[X > (1+\delta)np\right] \le \exp\left(-\frac{\delta^2}{2+\delta}np\right).$$

The diameter of a directed graph G is defined as $\max\{d(u,v) : u,v \in V(G) \text{ and } d(u,v) < \infty\}$ i.e. the longest shortest path between two vertices u,v where u can reach v.

For functions $f(n)$ and $g(n)$ we say that $f(n) = \widetilde{O}(g(n))$ if $f(n) = O(g(n) \cdot \mathrm{poly} \log n)$. In particular, $\widetilde{O}(1) = O(\mathrm{poly} \log n)$.

III. OVERVIEW OF APPROACH

Here we provide an overview of our approach towards achieving our algorithmic improvements on reachability. First, we provide a blueprint for our sequential nearly linear time algorithm for computing diameter reducing hopsets. While further work is needed to make this algorithm implementable in low depth and more insights are needed to obtain our parallel and distributed results (and we discuss these briefly), we believe this simple sequential algorithm demonstrates the primary algorithmic insights of the paper.

For the remainder of this section, let $G = (V,E)$ be a digraph for which we wish to efficiently compute diameter-reducing shortcuts. For simplicity we consider the case where G is a directed acyclic graph (DAG); the analysis of the general case is essentially identical and for the purposes of reachability (ignoring parallel computation issues) we can contract every strongly connected component to a single vertex.

Our shortcutting algorithms follows the general blueprint leveraged by Fineman [1] for efficiently computing diameter reducing shortcuts. Briefly, Fineman's algorithm consists of

the following iteration: in every step, a "shortcutter" vertex v is selected from V uniformly at random. It then constructs three sets: v's ancestors $R_G^{\mathrm{Anc}}(v)$, v's descendants denoted $R_G^{\mathrm{Des}}(v)$, and the set of notes *unrelated to v* $U_G(v) \stackrel{\mathrm{def}}{=} V \backslash \{R_G^{\mathrm{Des}}(v) \cup R_G^{\mathrm{Anc}}(v)\}$. The algorithm then adds shortcut edges from v to every node in $R_G^{\mathrm{Des}}(v)$ and from every node in $R_G^{\mathrm{Anc}}(v)$ to v. The algorithm then computes the induced graphs $G_D = G[R_G^{\mathrm{Des}}(v)], G_A = G[R_G^{\mathrm{Anc}}(v)]$ and $G_U = G[U_G(v)]$ and recursively applies the procedure to each of these three graphs.

To analyze this procedure, consider any path P in G. The algorithm in [1] considers how the shortcuts the algorithm constructs affect the distance between the endpoints of P. When a shortcutter vertex v is picked, there are four possibilities with how it interacts with P: it is either

1) Unrelated to every node in P.
2) An ancestor to some nodes in P forming a subpath P_1 and unrelated to the remaining subpath P_2.
3) A descendant to some nodes in P forming a subpath P_2 and unrelated to the remaining subpath P_1.
4) An ancestor to the tail of P and a descendant to the head of P.

Consider shortcutting through any vertex v and following the recursion of the above algorithm. By the above, it is clear that after shortcutting through any vertex v one of three things can happen: either P remains intact in a subproblem (case 1), it gets split into exactly two pieces in two different subproblems (cases 2 or 3), or the connectivity between the endpoints of P is resolved through v, i.e. we can go from P's head to tail in two edges by going through v (case 4). Thus we either split P into at most two pieces or we ensure the endpoints P are distance 2 from each other. Let P_i be the pieces P is split into at some state of the algorithm's execution, and let V_i be the subproblem vertex set containing P_i.

The key insight of Fineman is to define the following function (which we defined in Section II) and to use it to reason about the effect of this random process:

$$s(P_i, G_i) \stackrel{\mathrm{def}}{=} \left|\{u \in V(G_i) | u \text{ is related to some node in } P_i\}\right|.$$

Observe that $s(P_i, G_i)$ is an overestimate of the length of P_i, and that $s(P,G) \le n$. Define

$$L(P) \stackrel{\mathrm{def}}{=} \sum_i s(P_i, G_i)$$

to be the sum over all $s(P_i, G_i)$ at this state of our algorithm. $L(P)$ is a random variable, but by reasoning about the above cases we can reason about how $L(P)$ changes in expectation. For any subpath P_i, consider the induced subproblems after shortcutting through a randomly selected node v. If v lands in case 1 nothing changes, and if v lands in case 4 we resolve the connectivity of P_i and set $s(P_i, G_i)$ to 0, as there is no remaining subproblem. In cases 2 and 3 however, we

split P_i into two pieces: call these P_{i1} and P_{i2}. In these cases, Fineman is able to argue that a randomly chosen node can ensure that the number of nodes which are related to either P_{i1} or P_{i2} decreases by some constant factor c in expectation. Thus if $f(x)$ is the expected shortcut length of a path P_i with $s(P_i, G_i) = x$ we can essentially guarantee by induction that $f(x) \leq \max_{a+b=cx} f(a) + f(b)$. Fineman achieves a constant $c = 3/4$: this gives a bound of $f(n) \leq O(n^{1/\log(8/3)})$. A sophisticated refinement of this argument allows him to obtain his claimed $\widetilde{O}(n^{2/3})$ bound.

Our algorithm is almost identical to that of Fineman with one crucial modification: we pick more than one shortcutter node before we recurse. Specifically, we shortcut from k random vertices in the graph instead of only a single vertex. After shortcutting, we partition the vertices of the graph into subsets, much like Fineman's algorithm partitioned the vertices in ancestors, descendants, and unrelated vertices. In our partitioning scheme, two vertices are in the same subset if and only if they have the same relationship to each of the k shortcutters. As an example, two vertices u_1 and u_2 are *not* in the same subset if say u_1 is an ancestor of shortcutter v and u_2 is a descendant of shortcutter v. If we pick k shortcutter nodes from cases 2, 3, or 4 at a time and partition in the way described, we are able to guarantee that the number of path-related nodes after we recurse decreases by a factor of $\frac{2}{k+1}$ in expectation after we recurse (Lemma IV.4). Although the path splits into $k + 1$ pieces after recursing, analyzing the resulting recursion in the same manner as Fineman reveals that $k = \omega(1)$ will ensure our algorithm will shortcut paths to length $n^{1/2+o(1)}$ as desired. Unfortunately, we are not able to guarantee this directly. The above analysis requires that in any recursive level we pick either 0 or k path-relevant shortcutters in any recursive level; however we do not know how to obtain such fine-grained control without knowing the path.

Intuitively, we would like to pick as many shortcutter vertices as possible while staying within our nearly-linear work bound– the more shortcutters we pick, the more likely we are to obtain the $\frac{2}{k+1}$ reduction in path-related nodes. However, we cannot simply pick the same number of shortcutters in every level of recursion: because the number of path-related nodes goes down rapidly, picking k shortcutters per level of recursion will eventually only enable us to pick a single path-relevant shortcutter per round. Instead, we show that after each level of recursion the structure of the subproblems is such that we can pick k *times more* shortcutters while still having nearly-linear work. This, combined with a new inductive analysis in Lemma IV.6 to get around the fact that we don't have as precise control over the change in $L(P)$ enables us to obtain our result.

Parallel Implementation: Our techniques as described give us a nearly-linear work algorithm which constructs a nearly-linear number of shortcuts that reduce the diameter to $n^{1/2+o(1)}$. We make our construction parallel in a similar

fashion to Fineman. The key insight to [1]'s parallelization is to consider D^{search}-*restricted* searches; instead of computing the ancestor, descendant, and unrelated sets with full graph traversals from a vertex v, Fineman computes collections of D^{search}-ancestors and D^{search}-descendants. These are the set of ancestors (resp. descendants) which are reachable from v at distance at most D^{search}. Now although these can be computed in low depth, we cannot use these as a direct replacement for the full ancestor and descendant sets as we can no longer guarantee an expected decrease in $L(P)$.

Fineman gets around this issue with a new idea. Let G be a digraph. Assume that we could efficiently find a set of edges F to add to G such that if s and t are nodes at distance D from each other their distance in $G \cup F$ is at most $D/5$ w.h.p. Then for any nodes u, v at distance more than D from each other we observe that their distance in $G \cup F$ is halved w.h.p., i.e. this breaks up the $u - v$ shortest path into chunks of length D and observe that each subpath's size falls by a constant factor with constant probability. Thus by repeating this procedure on $G \cup F$ and iterating $O(\log n)$ times we observe that every pair of reachable nodes u, v can be brought within distance D. Doing this reduction only costs logarithmic factors in total work and parallel depth.

Fineman therefore modifies his recursion in the following way. In every level set $D^{search} = (\kappa + 1)D$, where κ is a random variable. Fineman then constructs the D^{search}-ancestors and D^{search}-descendants, but he then defines the unrelated set to be set of all nodes which are *not κD-ancestors or κD-descendants*. This modification duplicates all nodes at distance between κD and $(\kappa + 1)D$ from the shortcutter v, but now any path of length D is partitioned into two contiguous subpaths, copies of which can be found in these three induced sets. Fineman argues that the expected increase in the number of nodes can be controlled and that an analogous bound on $L(P)$ in expectation can be obtained as in the serial setting by picking and choosing the specific copies of subpaths to split P into in the recursion. Combining these pieces allows him to obtain his claimed $\widetilde{O}(n^{2/3})$ depth algorithm. Our approach (Section V) will build off of these ideas with several modifications for our new algorithm to leverage our inductive analysis.

Application to Distributed Reachability: Let G be a n-vertex directed graph with undirected hop diameter D. We describe our approach for solving the single source reachability problem in the CONGEST model on G. Our approach involves combining our parallel diameter reduction algorithm with the approaches of Ghaffari and Udwani [5] and Nanongkai [4]. The approach (loosely) involves using $\widetilde{O}(\alpha + n/\alpha)$ rounds of communication in the CONGEST model to reduce the problem to computing reachability on a set of vertices S of size α, with the difference that the vertices must communicate via global broadcasting, as the vertices in S aren't actually connected in the original graph. We then simulate our parallel reachability algorithm

on S. By analyzing our parallel reachability algorithm, we can analogously get a bound on the number of rounds needed to simulate it in the CONGEST model.

IV. SEQUENTIAL ALGORITHM

The main goal of this section is to prove Theorem 3 showing that for all k there is an $\widetilde{O}(mk)$ time algorithm which adds $\widetilde{O}(nk)$ shortcuts which reduce the diameter to $n^{1/2+O(1/\log k)}$ w.h.p. In Section IV-A we present our algorithm for achieving this result. In Section IV-B we bound the work of the algorithm and the number of shortcuts it adds. In Section IV-C we provide our main technical lemma regarding diameter reduction and then in Section IV-D we apply this lemma repeatedly to prove that the algorithm reduces diameter, thereby proving Theorem 3.

A. Algorithm Description

Here we present our sequential short-cutting algorithm (see Algorithm 1). Before stating the algorithm, we give some definitions and intuition for the quantities defined in the algorithm. Let G be the graph that we input to our algorithm and consider the following.

- **Inputs** k, r: k is a parameter governing the speed that we recurse at. Intuitively, our algorithm picks shortcutters so that graphs at one level deeper in the recursion are "smaller" by a factor of k. This is made precise in Lemma IV.1. $r \leq \log_k n$ is the depth of recursion that the algorithm is currently at, where we start at $r = 0$.
- **Set** S: S is the set of vertices from which we search and build shortcuts from.
- **Set** F: F is the final set of shortcuts we construct.
- **Probability** p_r: At recursion depth r, for each vertex $v \in V(G)$, we put v in S with probability p_r.
- **Labels** $v^{\mathrm{Des}}, v^{\mathrm{Anc}}, \text{✗}$: We want to distinguish vertices by their relations to vertices in S. Therefore, when we search from a vertex v we add a label v^{Des} to add vertices in $R_G^{\mathrm{Des}}(v) \backslash R_G^{\mathrm{Anc}}(v)$, a label v^{Anc} to all vertices in $R_G^{\mathrm{Anc}}(v) \backslash R_G^{\mathrm{Des}}(v)$, and a label ✗ to all vertices in $R_G^{\mathrm{Des}}(v) \cap R_G^{\mathrm{Anc}}(v)$. The label ✗ should be understood as "eliminating" the vertex (since it is in the same strongly connected component as v and we have shortcut through v already).

Algorithm 1 $\mathrm{SEQ}(G, k, r)$. Takes a graph G, parameter k and recursion depth $r \leq \log_k n$ (starts at $r = 0$). Returns a set of shortcut edges to add to G. Sequential diamater reduction algorithm. n denotes the number of vertices at the top level of recursion.

1: $p_r \leftarrow \min\left\{1, \frac{20k^{r+1}\log n}{n}\right\}$;   ▷ Begin level r of recursion.
2: $S \leftarrow \emptyset$;
3: **for** $v \in V$ **do**
4:     With probability p_r do $S \leftarrow S \cup \{v\}$;
5: **end for**
6: $F \leftarrow \emptyset$;
7: **for** $v \in S$ **do**
8:     **for** $w \in R_G^{\mathrm{Des}}(v)$ **do** add edge (v, w) to F;
9:     **end for**
10:     **for** $w \in R_G^{\mathrm{Anc}}(v)$ **do** add edge (w, v) to F.
11:     **end for**
12:     **for** $w \in R_G^{\mathrm{Des}}(v) \backslash R_G^{\mathrm{Anc}}(v)$ **do** add label v^{Des} to vertex w.
13:     **end for**
14:     **for** $w \in R_G^{\mathrm{Anc}}(v) \backslash R_G^{\mathrm{Des}}(v)$ **do** add label v^{Anc} to vertex w.
15:     **end for**
16:     **for** $w \in R_G^{\mathrm{Des}}(v) \cap R_G^{\mathrm{Anc}}(v)$ **do** add label ✗ to vertex w.
17:     **end for**
18: **end for**
19: $W \leftarrow \{v \in V : v$ has no label of ✗$\}$.
20: $V_1, V_2, \ldots, V_\ell \leftarrow$ partition of W such that $x, y \in V_i$ if and only if x and y have the same exact labels.   ▷ Vertices in the V_i have no label of ✗.
21: **for** $1 \leq i \leq \ell$ **do**
22:     $F \leftarrow F \cup \mathrm{SEQ}(G[V_i], k, r+1)$
23: **end for**
24: **return** F

Our algorithm can be thought of as an extension of Fineman's shortcut construction procedure. In every iteration, we seek to add as many shortcutters as possible while staying within our claimed work bound. Thus, in the first iteration we add $\widetilde{O}(k)$ shortcutters w.h.p. and perform $\widetilde{O}(mk)$ work. We then partition the nodes into clusters such that any two nodes x and y which are in the same V_i have exactly the same labels assigned to them by the shortcutters, none of which are ✗. We will show how to implement this step later (Lemma IV.3). We then recursively apply the algorithm within each cluster with a sampling probability that is a factor of k larger. We will show two things. First, we show that the increase in sampling probability is offset by a decrease in the number of related pairs such that the work done in an iteration is the same w.h.p. (Lemma IV.1 and Lemma IV.2). Second, we show that if we pick q shortcutters that are path related to a path P we get an

expected decrease in the number of path-related nodes to all the induced subproblems of P (Lemma IV.4). This second fact enables us to replace the recursion in Fineman with one that decreases more quickly (Lemma IV.6): this gives our depth improvement.

B. Work and Shortcut Bound

In this section we bound the work and number of short-cuts added by Algorithm 1. In any recursion level of our algorithm there are two sources of work. The first source is from computing the requisite labels $v^{\text{Des}}, v^{\text{Anc}}$, and ✗ for every node v we shortcut from. The second source comes from grouping the nodes by these labels to generate the subproblems for the next level. We will bound both of these sources of work by using a useful fact on the number of ancestors and descendants a node has in the subproblem it belongs to in any level.

Lemma IV.1. *Consider an execution of* $\text{SEQ}(G, k, 0)$ *on n-node m-edge G. With probability $1 - n^{-10}$ in each recursive execution of $\text{SEQ}(G', k, r)$ in line 22 of Algorithm 1 the following holds*

$$R_{G'}^{\text{Des}}(v) \leq nk^{-r} \quad \text{and} \quad R_{G'}^{\text{Anc}}(v) \leq nk^{-r} \quad \text{for all } v \in G'.$$

Proof: We prove by induction on r. Clearly the claim is true for the one recursive call at $r = 0$. We will show that assuming the claim for all recursive calls with $r = j$ the result holds for all $r = j + 1$ problems with probability at least $1 - n^{-11}$. By applying union bound over all $\widetilde{O}(1)$ values of r encountered in the algorithm implies the result.

Assume the result holds for every recursive execution with $r = j$. Let $v \in V$ be any vertex, and let G' be the induced subgraph our algorithm is recursively called on with $r = j + 1$ which contains v. We prove the claim for $R_{G'}^{\text{Des}}(v)$ as the claim for $R_{G'}^{\text{Anc}}(v)$ follows by a symmetric argument.

Observe that the recursive call $\text{SEQ}(G', k, j + 1)$ is ultimately called through an execution of $\text{SEQ}(H, k, j)$ on some $H \subseteq G$. Let Q be the set of nodes in $V(G')$ which are descendants of v in H. Now if $|Q| = R_H^{\text{Des}}(v)$ is less than nk^{-r} we are done since the induced subgraphs we recurse on only decrease in size. Thus assume $|Q| \geq nk^{-r}$.

Let $Q_1, Q_2, \cdots$ be the strongly connected components of Q, and consider any topological order over these subsets of Q, where Q_i precedes Q_j whenever a path from Q_i to Q_j exists. Consider any $x, y \in Q$ where y precedes x in this order. We investigate the random choices in $\text{SEQ}(H, k, j)$ that lead to G''s formation. Observe that if we chose y as a shortcutter for H, G' would not contain x since v is in G' yet x and v receive different labels from y: v is y's ancestor but x is either a descendant of or unrelated to y. Further, we observe that if we shortcut from y any node z in y's strongly connected component would also fail to be in G': z would be given an ✗ label. Thus if we shortcut the graph with any of the nk^{-j} nodes which are earliest in the topological order

of Q (which are closest to v in H) we can guarantee that v in the G' level has at most nk^{-j} descendants. Since we choose each node with probability $\frac{20k^j \log n}{n}$ we fail to do this with probability at most

$$\left(1 - \frac{20k^j \log n}{n}\right)^{nk^{-j}} \leq e^{-20 \log n} = n^{-20}.$$

By union bounding over all vertices in G' and over all induced subgraphs encountered at level $r = j$ we see that our bound holds for all recursive calls with $r = j$ with probability at least $1 - n^{-11}$. The result follows. ∎

We now bound the number of labels any vertex v receives in any recursive execution which contains it. This will provide us with an elegant way to bound the total work of our procedure.

Lemma IV.2. *Consider an execution of $\text{SEQ}(G, k, 0)$ on n-node m-edge G with $k \geq 2$. With probability $1 - 2n^{-10}$, every recursive execution $\text{SEQ}(G[V_i], k, r)$ assigns at most $80k \log n$ labels to every node $w \in V_i$ in lines 12, 14, and 16, where ✗ labels assigned by different shortcutters are counted as distinct labels.*

Proof: Note that v receives a label from a shortcutter u only if u is related to v. By Lemma IV.1 we have that at most $2nk^{-r}$ nodes are related to v for all v in all executions of SEQ with probability $1 - n^{-10}$. Since we pick nodes in the rth level with probability $p_r = \frac{20k^{r+1} \log n}{n}$ we see that the probability that more than $80k \log n$ labels are given to v, assuming that at most $2nk^{-r}$ vertices are related to v, is at most

$$\Pr\left[B\left(2nk^{-r}, \frac{20k^{r+1} \log n}{n}\right) > 80k \log n\right]$$
$$\leq \exp\left(-\frac{40}{3}k \log n\right) \leq n^{-12}$$

by Lemma II.1. Thus v receives at most $80k \log n$ labels with probability at least $1 - n^{-12}$. Union bounding this over all nodes in all recursive executions of SEQ implies the result. ∎

Finally, we conclude this subsection by bounding the total work of SEQ, as well as the number of shortcut edges it adds.

Lemma IV.3. *Consider an execution of $\text{SEQ}(G, k, 0)$ on n-node m-edge G with $k \geq 2$. With probability $1 - 2n^{-10}$ $\text{SEQ}(G, k, 0)$ runs in $\widetilde{O}(mk)$ time and adds $\widetilde{O}(nk)$ short-cuts.*

Proof: By our given probability of failure, we may assume Lemma IV.1 and Lemma IV.2 hold deterministically.

We begin by considering a single recursive execution $\text{SEQ}(G[V_i], k, r)$ generated by SEQ. We will bound the number of shortcuts added by this call and amount of work it performs before it recurses in 22. We will then aggregate these bounds over all recursive executions and obtain our

final result. For convenience, let $G[V_i]$ have $\hat{n}$ nodes and $\hat{m}$ edges.

We first bound the number of shortcuts added by $\text{SEQ}(G[V_i], k, r)$. By Lemma IV.2 we observe that every recursive execution $\text{SEQ}(G[V_i], k, r)$ assigns $\widetilde{O}(k)$ labels to every $w \in V_i$. As each label corresponds to a shortcut we add in lines 8 and 10, we see that $\text{SEQ}(G[V_i], k, r)$ adds $\widetilde{O}(k)$ edges to every $w \in V_i$: this is $\widetilde{O}(\hat{n}k)$ edges in total.

We now bound the work performed by $\text{SEQ}(G[V_i], k, r)$. Within a call to SEQ, we perform work in two places: within the loop in line 7 and when generating the partition in line 19. We bound the contributions of these sources in order. First, observe that the loop in 7 can be implemented by computing breadth-first searches forwards and backwards from every w in the shortcutter set S. The amount of work needed to apply the labels and and the shortcuts themselves is clearly $\widetilde{O}(\hat{n}k)$ by the above argument, so we need only to bound the cost of running these traversals.

Observe that by IV.2 $\text{SEQ}(G[V_i], k, r)$ assigns $\widetilde{O}(k)$ labels to every $w \in V_i$. Now the number of labels w receives is within a factor of two the number of times it is visited in searches. Thus w is visited $\widetilde{O}(k)$ times in our traversals. Each time we encounter w in a traversal we perform a constant amount of work for each edge incident upon it. Thus if $\delta_i(w)$ is the undirected degree of w in $G[V_i]$, the total work performed by $\text{SEQ}(G[V_i], k, r)$ is

$$\widetilde{O}\left(\sum_{w \in V_i} k\delta_i(w)\right) = \widetilde{O}(\hat{m}k).$$

We finally bound the cost of generating the partition in line 19. We implement this in two parts. First, we check each vertex to see whether it has an ✗ label and discard any vertex which does. Next, we define an order over all possible combinations of labelings a node could receive. We then sort the remaining nodes by this order: we can then trivially read off the partition. To implement this order of labelings, pick an arbitrary ordering on the individual labels we distribute to nodes. To compare two labeling schemes a and b we internally sort a and b by our arbitrary ordering, and then determine the order amongst a and b lexicographically.

To implement this procedure, we first note that by Lemma IV.2 every node receives at most $\widetilde{O}(k)$ labels. Determining which of the $\hat{n}$ nodes have an ✗ label clearly takes $O(\hat{n}k)$ time. It is straightforward to verify that comparing two labelings each with at most $\widetilde{O}(k)$ labels with this scheme requires $\widetilde{O}(k)$ time: thus this partitioning can be found in $\widetilde{O}(\hat{n}k)$ time using a mergesort. Combining this with the previous bound we see that $\text{SEQ}(G[V_i], k, r)$ requires $\widetilde{O}(\hat{m}k)$ time before recursing.

We now obtain our final work and shortcut bounds by aggregating. If we consider the set of recursive calls $\text{SEQ}(H, k, r)$ for any fixed value of r, we see that the calls are applied to a disjoint collection of subgraphs of G. Thus,

the total number of nodes in all of these subproblems is n, and the total number of edges is at most m. Thus cost of performing all of these calls without recursing is $\widetilde{O}(mk)$, and these calls collectively add $\widetilde{O}(nk)$ shortcuts. As there are at most $\widetilde{O}(1)$ different values of r our claim follows. ∎

C. Path Related Nodes and Main Helper Lemma

We now prove a significant helper lemma that will enable us to prove our diameter bound. We begin with some context. Recall that the goal of our algorithm is, for any path $P \in G$ with endpoints s and t, to find a bridge for P. If in a recursive call to S we succeed in finding a bridge for P we add the corresponding shortcuts to connect s and t with a length 2 path: there is nothing more for us to do. If instead we do not find a bridge in S, we observe that P gets split amongst several different node-disjoint subproblems: we then seek to find bridges for each of these subproblems separately. Thus the collection of paths P_j represents the "residual" paths left for our algorithm to resolve: we either pick a bridge and entirely resolve the path or split it into pieces. While this splitting of the path may seem counterproductive, we show that the total number of path-related vertices in the next recursion level summed over all P_i decreases significantly in expectation when we recurse. We thus can ensure some form of progress whether we resolve the path or not.

In the below lemma, for a path P' in a subgraph $G' \subseteq G$, we define $s(P', G')$ to be the number of vertices in G' that are related to P', as was done in Section II.

Lemma IV.4. *Let G be a digraph and let P be a path in G. Let T be a uniformly random subset of $V[G]$, where any node $v \in V[G]$ is in T with some probability p. Define $S = T \cap R_G(P)$, and let $|S| = t$. Consider running lines 5-13 of $\text{SEQ}(G, k, r)$ (Algorithm 1) with this choice of S. Then there exists a partition of P into exactly $t + 1$ (possibly empty) subpaths $P_1, P_2, \cdots, P_{t+1}$ which satisfies the following conditions:*

1) *If S contains a bridge for P, then all the P_j are empty.*
2) *If S contains no bridges for P, then the vertex disjoint union of the P_j is exactly P.*
3) *Each P_j is inside some $G[V_{f(j)}]$ generated by SEQ for a recursive execution for some $f(j)$.*

Further,

$$\mathbb{E}_{|S|=t}\left[\sum_i s(P_i, G[V_{f(i)}])\right] \leq \frac{2}{t+1} \cdot s(P, G).$$

Here, the expectation is conditioned on the event that $|S| = t$; equivalently, we may take the expectation over a uniformly random subset of t elements from $R_G(P)$.

Before we prove this lemma, we describe our general proof strategy. Depending on what S is, we will construct a partition of P which satisfies our four constraints. If the

set S contains a bridge for P, our partition will be empty: P's endpoints are connected through the bridge. If S does not contain any bridges, we simply consider all nonempty subpaths of P inside the recursively generated subproblems induced by S: we will show that this partition does not form too many pieces. To prove the expected decrease in the number of path-related nodes, we will explicitly use the randomness of S. Consider any set C of ancestors and descendants of P. Let v be a vertex inside C, and imagine shortcutting P with $C - \{v\}$ and forming the subpaths by our partition. Now consider the event that v is path-relevant for one of these induced subpaths. We will show that there are at most two choices of v from C such that this happens. The result follows with some computation.

Lemma IV.5. *Let P be a path, and let $A = \{a_1, a_2, \cdots, a_{l+1}\}$ be ancestors (resp. descendants) of nodes in P. If we pick a_i at random from this collection and shortcut using the other l points, a_i is path-relevant for one of the subproblems with probability at most $1/(l+1)$.*

Proof: Index the path from head to tail [1], and let $\alpha(x)$ be the lowest-indexed node p such that $x \preceq p$. Assume node $a \in A$ remains path-relevant after shortcutting through all the other a_i. We will show that a must satisfy the following:

- a is a strictly minimal element amongst the A: no a_i can have a as an ancestor.
- Amongst all a_i which are unrelated to a, $\alpha(a) > \alpha(a_i)$.

For the first claim, assume for the sake of contradiction that some a_j had a as an ancestor. Note that since we shortcut from a_j it labels a with either a_j^{Anc} or ✗ depending on whether a an reach a_i. In the latter case we are done since we do not recurse on nodes which receive ✗. In the former case we observe that all the a_i can reach the path yet no node in the path can reach any a_i since the a_i are all ancestors. Thus every node $p \in P$ receives either no label from a_j or a a_j^{Des} label. As a receives a different label from every node in P we conclude that it cannot be path-relevant to any path subproblem in the next level of the recursion.

For the second claim, assume there existed a_j which was unrelated to a such that $\alpha(a_j) \leq \alpha(a)$. This implies that if a is an ancestor to a node $p \in P$ then a_j is also an ancestor for p. Thus every node in the path is either unrelated to a or a descendant of a_j. As all of a_j's descendants get a label a_j^{Des} yet a receives no label from a_j, we conclude that a can only be placed in a subproblem (if at all) with a piece of the path that it is unrelated to: thus a ceases to be path relevant.

We now show that there is at most one vertex amongst A which satisfies both of these conditions. Assume for the sake of contradiction that both a_i and a_j would remain path relevant if we shortcutted through $A \backslash a_i$ and $A \backslash a_j$ respectively. If these two vertices were related, then by the

---

[1]We assign the head an index of 0.

---

first of our conditions the one which was an ancestor will not remain path-related; contradiction. If the two vertices were unrelated, then $\alpha(a_i) \geq \alpha(a_j)$ or vice-versa: the vertex with the smaller α value cannot remain path related by our second condition. Thus at most one node satisfies our condition, and the claim follows.

An identical proof can be used in the case where the a_i are all path descendants. ∎

With this, we complete the proof of Lemma IV.4.

Proof of Lemma IV.4: We begin by defining the partition P_i. If S contains a bridge for P, we set all the $P_i = \emptyset$: this clearly satisfies the conditions. If S does not contain a bridge, we look at the induced subgraphs generated by Seq in a recursive call. Let Q_i denote P intersected with V_i– the part of P that lies in $G[V_i]$. We choose the P_j to be exactly those Q_i which are nonempty. We need only show that there are at most $|S| + 1$ nonempty P_j: conditions $1, 2, 3$ are trivial. Observe that as S does not contain any bridges, every path-relevant shortcutter picked is either an ancestor or a descendant of P. Thus every shortcutter $s \in S$ induces a "cut" of P into two contiguous pieces each assigned a different label from P. It is straightforward to verify that this implies that P can be split into at most $|S| + 1$ contiguous regions each internally with the same labels: this forms our partition P_i.

We now turn our attention to the second fact. Consider picking a random point v from $R_G(P)$. We will show that v is counted in some $s(P_i, G[V_{f(i)}])$ (that is, it remains path-relevant for some subpath) with probability at most $\frac{2}{t+1}$. Let the points of S be $s_1, s_2, \cdots, s_t$, and observe that the $t + 1$ points $v, s_1, \cdots, s_t$ form a random sample from $s(P, G)$. Amongst the $s_1, \ldots s_t$ we chose, some points are path ancestors, some are path descendants, and some are both (bridges). Assume that there are α ancestors, δ descendants, and β bridges amongst the $s(P, G)$ path related points. We relabel the s_i as $x_1, \cdots, x_a$, $y_1, \cdots, y_d$, and $z_1, \cdots, z_b$ where the x are path ancestors, the y are path descendants, and the z are path bridges. Here, among the s_i there are a ancestors, d descendants, and b bridges. There are two cases: either we picked at least one path bridge (in which case the endpoints of the path are linked and the path gets completely resolved) or we picked 0 bridge nodes. The probability that we picked 0 bridge vertices is at most

$$\left(1 - \frac{\beta}{s(P,G)}\right)^t.$$

Now as v is also a randomly chosen vertex: with probability $p = \frac{\alpha}{s(P,G)}$ it is an ancestor, with probability $q = \frac{\delta}{s(P,G)}$ it is a descendant, and with probability $g = \frac{\beta}{s(P,G)}$ it is a bridge. If we condition on the event that none of the s_i were bridges, we observe that $a \sim B(t, \frac{p}{p+q})$ and $d \sim B(t, \frac{p}{p+q})$. Thus the probability v is path relevant is at most

$$\mathbb{E}\left[\frac{p}{a+1} + \frac{q}{d+1}\right] + g.$$

by simply applying Lemma IV.5 to the cases where v is an ancestor and v is a descendant separately and union bounding the events. We now recall a useful fact: if $X \sim B(n, p)$, $\mathbb{E}[(X+1)^{-1}] = \frac{1-(1-p)^{n+1}}{p(n+1)} < \frac{1}{p(n+1)}$. By applying this fact we observe that the probability v is path relevant conditioned on us never picking a bridge shortcut is at most $\frac{2}{t+1} + g$. Thus by multiplying by the chance of us never picking a bridge, we see that the final probability v survives is at most

$$\left(1 - \frac{\beta}{s(P,G)}\right)^t \left(\frac{2}{t+1} + g\right) = (1-g)^t \left(\frac{2}{t+1} + g\right).$$

This can be verified to be at most $\frac{2}{t+1}$, and the result follows. ∎

D. Recursion and Inductive Diameter Bound

With this helper lemma in place, we now use it to prove our claimed diameter bound.

Lemma IV.6 (Inductive diameter bound). *Consider running* $\textsc{Seq}(G, k, 0)$, *and consider a recursive execution* $\textsc{Seq}(G[V'], k, r)$ *with path P' inside $G[V']$. Let $t \leq \log_k n$ be the largest value of r ever encountered in our recursive calls. If we complete our algorithm's recursion from* $\textsc{Seq}(G[V'], k, r)$, *the expected distance from* $\mathrm{head}(P')$ *to* $\mathrm{tail}(P')$ *after applying our computed shortcuts is at most* $(4\sqrt{2})^{\bar{r}} s(P', G')^{1/2}$, *where* $\bar{r} = \log_k n - r$. [2]

Proof: We proceed by induction on r with base case $r = \log_k n$. Note that at this stage we have no more recursion to do: P' must consist of a single node and thus the distance from $\mathrm{head}(P')$ to $\mathrm{tail}(P')$ is 0. Therefore, the result holds for $r = \log_k n$.

For our induction step, consider an inductive execution of algorithm $\textsc{Seq}(G[V'], k, r)$, and assume the result for depth $r + 1$. Consider the subexecutions directly induced by $\textsc{Seq}(G[V'], k, r)$. We shortcut P' in the following way. Say that our algorithm chose a set S of t vertices in $R_G(P')$ as shortcutters. If one of the vertices in S is a bridge of P', we would simply traverse the bridge and go from our path's head to tail in 2 edges. Otherwise, by Lemma IV.4 we would split P' into $t + 1$ subpaths $P'_1, P'_2, \cdots, P'_{t+1}$ where

- The disjoint vertex union of the P'_i is P'.
- Each P'_i is inside some $G[V_{f(i)}]$ on which $\textsc{Seq}$ executes $\textsc{Seq}(G[V_{f(i)}], k, r+1)$ on for some $f(i)$.

We can get from the head to tail of P' by inductively traversing each P'_i in the order we encounter them, while using t extra edges to go between these paths. Now in either case we use at most $t + 2$ edges to traverse between the subpaths formed by our recursion. We additionally use some number of edges to traverse from the tail to head of each of the subpaths we form. By our inductive hypothesis, we

[2]We note that the bound obtained here is weaker than the one obtained in the parallel setting. We give a less tight analysis for this lemma in pursuit of a shorter proof.

see that the length of each P'_i after applying our shortcuts satisfies

$$\mathbb{E}\left[\text{shortcut length of } P'_i\right] \leq (4\sqrt{2})^{\bar{r}-1} \mathbb{E}[s(P'_i, G[V'_i])^{1/2}].$$

Thus if we condition on the fact that $|S| = t$ the expected shortcut length of P' is at most

$$\mathbb{E}\left[\text{shortcut length of } P' \,\middle|\, |S| = t\right]$$

$$\leq t + 2 + \sum_{i=1}^{t+1} (4\sqrt{2})^{\bar{r}-1} \mathbb{E}\left[s(P'_i, G[V'_i])^{1/2} \,\middle|\, |S| = t\right]$$

$$\leq 2 + \sum_{i=1}^{t+1} (2\sqrt{2})^{-1} (4\sqrt{2})^{\bar{r}} \mathbb{E}\left[s(P'_i, G[V'_i])^{1/2} \,\middle|\, |S| = t\right]$$

as $(4\sqrt{2})^{\bar{r}-1} \mathbb{E}[s(P'_i, G[V'_i])^{1/2}] \geq 1$ for every i. We obtain

$$\mathbb{E}\left[\text{shortcut length of } P' \,\middle|\, |S| = t\right]$$

$$\leq 2 + \sum_{i=1}^{t+1} (2\sqrt{2})^{-1} (4\sqrt{2})^{\bar{r}} \mathbb{E}\left[s(P'_i, G[V'_i])^{1/2} \,\middle|\, |S| = t\right]$$

$$\leq 2 + \sum_{i=1}^{t+1} (2\sqrt{2})^{-1} (4\sqrt{2})^{\bar{r}} \mathbb{E}\left[s(P'_i, G[V'_i]) \,\middle|\, |S| = t\right]^{\frac{1}{2}}$$

$$\leq 2 + (2\sqrt{2})^{-1} (4\sqrt{2})^{\bar{r}} \sqrt{t+1} \mathbb{E}\left[\sum_{i=1}^{t+1} s(P'_i, G[V'_i]) \,\middle|\, |S| = t\right]^{\frac{1}{2}}$$

$$\leq 2 + (2\sqrt{2})^{-1} (4\sqrt{2})^{\bar{r}} \sqrt{t+1} \left[\frac{2}{t+1} s(P', G[V'])\right]^{\frac{1}{2}}$$

$$\leq 2 + 2^{-1} (4\sqrt{2})^{\bar{r}} s(P', G[V'])^{\frac{1}{2}}$$

$$\leq (4\sqrt{2})^{\bar{r}} s(P', G[V'])^{1/2},$$

where we used Jensen's inequality in the second inequality, Cauchy-Schwarz in the third, and Lemma IV.4 in the fourth. As our final bound is independent of the value of t, the result follows. ∎

With this result in place, we can now prove our final theorem statement by setting $r = 0$ in Lemma IV.6.

Theorem 5. *Let $G = (V, E)$ be a digraph with n nodes and m edges. Then with probability $1 - 2n^{-10}$ algorithm* $\textsc{Seq}(G, k, 0)$ *runs in $\widetilde{O}(mk)$ time and constructs a set F of $\widetilde{O}(nk)$ shortcuts such that an arbitrary path P is shortcut to length $n^{1/2 + O(1/\log k)}$ w.h.p.*

Proof: By Lemma IV.3 our claimed work and shortcut bound follow immediately. Let P be any path from s to t. By Lemma IV.6 we observe that the expected length of P after applying our constructed shortcuts is at most $(4\sqrt{2})^{\log_k n} (s(P,G))^{1/2}$. As $s(P,G) \leq n$, this equals $n^{1/2 + O(1/\log k)}$ as claimed. ∎

We observe that Theorem 3 follows from this by calling $\textsc{Seq}(G, k, 0)$ $O(\log n)$ times: the probability that any given pair s, t fails to have a path of length twice the bound in Theorem 5 between them after $c \log n$ calls is at most $1/n^c$

by Markov's inequality. By union bounding over all n^2 pairs of points the claim follows.

V. PARALLEL ALGORITHM

In this section, we show how to extend the nearly linear work sequential shortcutting algorithm of Section IV into a work-efficient, low depth parallel algorithm. In Section V-A we extend the definitions in Section II to the setting of distance limited searches and reachability, which we need throughout the section. In Section V-B we state our main algorithms PARALLELSC and PARALLELDIAM, and give intuition for how to reason about them. In Section V-C we bound the total work, depth, and number of shortcut edges added in the algorithm. Finally, in Section V-D we bound the diameter of the resulting graph after the execution of our algorithms.

A. Notation for Distance Limited Searches

Digraph distances and distance-limited relations: Let G be a digraph. For vertices $u, v \in V$ define $d(u, v)$ to be the length of the shortest path from u to v. We define $d(u, v) = +\infty$ if $v \notin R_G^{\text{Des}}(u)$, i.e. v is not reachable from u. We also define distance-limited reachability, which is a natural extension of the notion of reachability defined in Section II. For a parameter D, we define the D-descendants, D-ancestors, and D-related vertices to v as

$$R_G^{\text{Des}}(v, D) = \{u \in V : d(v, u) \le D\}$$
$$\text{and } R_G^{\text{Anc}}(v, D) = \{u \in V : d(u, v) \le D\}$$
$$\text{and } R_G(v, D) = R_G^{\text{Des}}(v, D) \cup R_G^{\text{Anc}}(v, D).$$

We extend all this notation to subgraphs G' of G in the natural way. Define $d_{G'}(u, v)$ to the length of the shortest path from $u \to v$ only using only vertices and edges in G'. Then we define $R_{G'}^{\text{Des}}(v, D)$, $R_{G'}^{\text{Anc}}(v, D)$, and $R_{G'}(v, D)$ as above. A vertex u is *unrelated within distance D* to a vertex v (with respect to a subgraph G') if $u \in V[G'] \backslash R_{G'}(v, D)$.

Distance limited path relations: In Algorithm PARALLELSC, our searches are limited to distance κD for some random parameter κ. Here we define distance limited path relations, analogous to the vertex and path relations defined in Section II. As before, we denote a path $P = \langle v_0, v_1, \ldots, v_\ell \rangle$, where all the v_i are vertices of G. We now make the following definitions. As the range the parameter κ is chosen from depends on r, the current recursion depth of the algorithm, our below definitions also depend on r. This dependence is made explicit in lines 3 and 4 of Algorithm PARALLELSC.

Throughout, we say that $u \preceq^s v$ if $d_G(u, v) \le s$, i.e. there is a path of length at most s from u to v. We say that $u \npreceq^s v$ if $d_G(u, v) > s$.

- **Fully path-related vertices.** We say that a vertex v is fully path-related if $v \in R_G(P, \kappa_{2r+1} D)$. In other words, $v \in R_G(P, \kappa D)$ for all $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$.

- **Partially path-related vertices.** We say that a vertex v is partially path-related if $v \in R_G(P, \kappa_{2r} D)$. In other words, $v \in R_G(P, \kappa D)$ for some $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$. We distinguish three types of fully or partially path-related vertices. The below definitions depend on the parameter κ chosen.

- **κD-Descendants.** We say that a vertex v is a κD-descendant of the path P if $v \in R_G^{\text{Des}}(P, \kappa D) \backslash R_G^{\text{Anc}}(P, \kappa D)$. Note that then we have that $v_0 \preceq^{(\kappa+1)D} v$ and $v \npreceq^{\kappa D} v_\ell$.

- **κD-Ancestors.** We say that a vertex v is an κD-ancestor of the path P if $v \in R_G^{\text{Anc}}(P, \kappa D) \backslash R_G^{\text{Des}}(P, \kappa D)$. Note that then we have that $v_0 \npreceq^{\kappa D} v$ and $v \preceq^{(\kappa+1)D} v_\ell$.

- **κD-Bridges.** We say that a vertex v is a κD-bridge of the path P if $v \in R_G^{\text{Des}}(P, \kappa D) \cap R_G^{\text{Anc}}(P, \kappa D)$. Note that then we have that $v_0 \preceq^{(\kappa+1)D} v$ and $v \preceq^{(\kappa+1)D} v_\ell$.

Now, we define the following quantities and briefly explain their importance in Algorithm 2. Further details are explained in the paragraph below (explanation of Algorithm 2 and Algorithm 3). Let G be the n-vertex m-edge digraph which we input, with vertex set V and edge set E.

- **Inputs k, r, r^{fringe}:** The input k denotes the speed that our algorithm recurses at, and intuitively digraphs one level lower in the recursion are "smaller" by a factor of k. This is made precise by Lemma V.1. The level $r \le \log_k n$ denotes the level of recursion the algorithm is at. Additionally, we have an inner recursion level $r^{\text{fringe}} \le \log n$ for the "fringe vertices" (defined below). For each level of recursion r, it has at most $\log n$ inner levels of recursion.

- **Set S:** S is the set of vertices from which we search and build shortcuts from.

- **Set F:** F is the final set of shortcuts we construct.

- **Search scale / distance D:** The parameter D denotes the scale on which the algorithm is running breadth first searches. Our main claim is that through one call of PARALLELSC$(G, k, 0, 0)$ an arbitrary path in G of length D will be shortcut to expected length $\frac{D}{10}$ (Lemma V.8).

- **Parameters κ_i, κ:** In the r-th recursion level, we randomly choose the parameter $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ to obtain our search radius κD for our breadth first searches.

- **Probability p_r:** At recursion depth r, for each vertex $v \in V(G)$, we put v in S with probability p_r.

- **Labels $v^{\text{Des}}, v^{\text{Anc}}, ✗$:** We want to distinguish vertices by their relations to vertices in S. Therefore, when we search from a vertex v we assign a label v^{Des} to add vertices in $R^{\text{Des}}(v, D) \backslash R^{\text{Anc}}(v, D)$, label v^{Anc} to all vertices in $R^{\text{Anc}}(v, D) \backslash R^{\text{Des}}(v, D)$, and label $✗$ to all vertices in $R^{\text{Des}}(v, D) \cap R^{\text{Anc}}(v, D)$. The label $✗$ should be understood as "eliminating" the vertex (since it is in the same strongly connected component as v and

we have shortcut through v already).

B. Main Algorithm Description

Algorithm 2 PARALLELSC$(G, k, r, r^{\text{fringe}})$. Takes a digraph G, parameter k, recursion depth $r \leq \log_k n$ (starts at $r = 0$), and inner fringe node recursion depth $r^{\text{fringe}} \leq \log n$. Returns a set of shortcut edges to add to G. n denotes the number of vertices at the top level of recursion, not the number of vertices in G.

1: $p_r \leftarrow \min\left(1, \frac{10k^{r+1}\log n}{n}\right)$.
2: $S \leftarrow \emptyset$.
3: $\kappa_{2r+1} \leftarrow 10^6 k^2 \log^5 n \left(1 + \frac{1}{4\log n}\right)^{-2r-1}$ and $\kappa_{2r} \leftarrow 10^6 k^2 \log^5 n \left(1 + \frac{1}{4\log n}\right)^{-2r}$.
4: Choose $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ uniformly at random. $\quad\triangleright$ Picking a random search distance
5: $D \leftarrow 100 \cdot \sqrt{2}^{\log_k n} \cdot n^{\frac{1}{2}} \cdot \log^2 n$.
6: **for** $v \in V$ **do**
7: $\quad$ With probability p_r do $S \leftarrow S \cup \{v\}$.
8: **end for**
9: $F \leftarrow \emptyset$.
10: **for** $v \in S$ **do**
11: $\quad$ **for** $w \in R^{\text{Des}}(v, (\kappa+1)D)$ **do** add edge (v, w) to F.
12: $\quad$ **end for**
13: $\quad$ **for** $w \in R^{\text{Anc}}(v, (\kappa+1)D)$ **do** add add (w, v) to F.
14: $\quad$ **end for**
15: $\quad$ **for** $w \in R^{\text{Des}}(v, \kappa D)\backslash R^{\text{Anc}}(v, \kappa D)$ **do** add label
16: v^{Anc} to vertex w.
17: $\quad$ **end for**
18: $\quad$ **for** $w \in R^{\text{Anc}}(v, \kappa D)\backslash R^{\text{Des}}(v, \kappa D)$ **do** add label
19: v^{Des} to vertex w.
20: $\quad$ **end for**
21: $\quad$ **for** $w \in R^{\text{Des}}(v, \kappa D) \cap R^{\text{Anc}}(v, \kappa D)$ **do** add
22: label ✗ to vertex w.
23: $\quad$ **end for**
24: $\quad$ $V_v^{\text{ring}} \leftarrow R(v, (\kappa+1)D)\backslash R(v, (\kappa-1)D)$.
25: $\quad$ $F \leftarrow F \cup \text{PARALLELSC}(G[V_v^{\text{ring}}], k, r, r^{\text{fringe}} + 1)$. $\quad\triangleright$ Recursion on fringe nodes
26: **end for**
27: $W \leftarrow \{v \in V : v \text{ has no label of } ✗\}$.
28: $V_1, V_2, \ldots, V_\ell \leftarrow$ partition of W such that $x, y \in V_i$ iff x and y have exactly the same labels. $\triangleright$ Vertices in the V_i have no label of ✗.
29: **for** $1 \leq i \leq \ell$ **do**
30: $\quad$ $F \leftarrow F \cup \text{PARALLELSC}(G[V_i], k, r+1, 0)$.
31: **end for**
32: **return** F

Algorithm 3 PARALLELDIAM(G, k). Takes a digraph $G = (V, E)$, parameter k. Modifies digraph G. Parallelizable diameter reduction algorithm.

1: **for** $i = 1$ to $10 \log n$ **do**
2: $\quad$ **for** $j = 1$ to $10 \log n$ **do**
3: $\quad\quad$ $S_j \leftarrow \text{PARALLELSC}(G, k, 0, 0)$, aborting if the work or shortcut edge count exceeds 10 times the bound in Lemma V.7.
4: $\quad$ **end for**
5: $\quad$ $E(G) \leftarrow E(G) \cup \left(\bigcup_j S_j\right)$
6: **end for**

Overview of Algorithm 2: Algorithm 2 is similar to Algorithm 1. All parts of Algorithm 1 can be implemented in low parallel depth except for the breadth first searches from the vertices in S (lines 10 to 24). To resolve this, a natural idea is to limit the distance of the breadth first searches to D, where D denotes the diameter bound given by Lemma IV.6. Running these incomplete searches introduces issues in the analysis though. To get around this, we follow an approach similar to [1] and perform some additional computation on the fringe of our breadth first searches. Specifically, we choose a random integer κ in the range $[\kappa_{2r+1}, \kappa_{2r}]$ (think of these as parameters which are $\text{poly}(\log n, k)$), and search from a vertex v to distance approximately κD. Then, we call the vertices in the set $R(v, (\kappa+1)D)\backslash R(v, (\kappa-1)D)$ the *fringe vertices*. We chose κ randomly to ensure that the expected number of fringe vertices is sufficiently small. We then recurse on the fringe vertices, which is done in line 25 of Algorithm 2.

In addition, we assign labels based on reachability *within distance κD*. As we show later in the section, the analysis as done in Section IV can be modified to tolerate these changes and obtain a similar result.

We would like to note some differences between the ways fringe vertices are handled in our algorithms compared to those in [1]. One difference is that we directly handle fringe vertices by recursing on only that set, while in Fineman's algorithm the fringe vertices are lumped into the recursion on ancestor sets. The reason for the difference is that our way of partitioning our vertex set before recursing is more involved. Additionally, in our algorithm, we explicitly track the depth of recursion on fringe vertices (parameter r^{fringe}) inside our algorithm. We must do this as our algorithm requires good control on the number of ancestors and descendants of a vertex (Lemma IV.1), and we do not obtain the required bound on the number of ancestors or descendants when we recurse on fringe vertices. Additionally, we must ensure that $r^{\text{fringe}} \leq \log n$ so that our algorithm still has low parallel depth.

Explanation of Algorithm 2 and Algorithm 3: Here, we give a detailed description of what each part of Algorithm 2 and Algorithm 3 is doing. Lines 1 and 2 of Algorithm 2

are simply initializing the set of vertices we search from S to the empty set and picking the probability p_r for which $v \in S$. In line 5, we choose our search scale D. This is chosen to be a constant factor larger than the bound given in Lemma IV.6 and the analogous Lemma V.8. In lines 3 and 4, we define the parameters $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$, which will define our search radius κD. Note that $\kappa_0 \geq \kappa_1 \geq \kappa_2 \geq \cdots$ and that $\kappa_i \geq \frac{1}{2}\kappa_0$ for all i, as $r \leq \log_k n$. This way, the search radius is decreasing with every recursion level. In lines 6 and 7, we are choosing the set of vertices to breadth first search from, and we are adding them to S. In line 9, we initialize the set of shortcut edges we are going to eventually add to the empty set.

In lines 11 and 13, we are adding all the shortcut edges through a vertex $v \in S$. In lines 16 to 22 we are applying the ancestor, descendant, and "eliminated" labels to other vertices. In line 25, we are running a recursion on the fringe vertices in our search from v. Note that we have increased r^{fringe} to $r^{\text{fringe}} + 1$ but have kept the parameter r the same.

In lines 27 and 28, we are processing the labels assigned to the vertices. We first remove from consideration all vertices which have a ✗ label, and then we partition the remaining ones into groups based on matching labels. In lines 29 and 30 we recurse on these groups we have created. Note that we have increased r to $r + 1$ and reset r^{fringe} to 0.

In Algorithm 3 we are simply running PARALLELSC for multiple iterations. Specifically, we can guarantee that the expected head to tail distance of any path of original length at most D is now at most $\frac{D}{10}$. Running this $O(\log n)$ times ensures that any fixed path of length D has been shortcut to length $\frac{D}{5}$ with high probability. Now, if we run this procedure $O(\log n)$ times, it is easy to see that any path's head to tail distance will get reduced to D with high probability: split this path into polynomially many paths of length at most D and note that with high probability each of these paths' lengths gets reduced by a constant factor. Thus the original path's length falls by a constant factor, and we continue to do this until the original path's length becomes at most D.

How to reason about the randomness: In order to reason about the randomness used in sampling κ and S, we should imagine that the during an execution of PARALLELSC$(G', k, r, r^{\text{fringe}})$ the algorithm first samples κ before doing anything else. After sampling κ, we then know precisely which vertices are κD-descendants, κD-ancestors, and κD-bridges with respect to the specific path P that we are analyzing. Given this, we can essentially proceed forwards with similar arguments to those in Section IV.

C. Shortcut and Work Bound

In this section, we prove many lemmas which help us bound the total amount of work and shortcut edges added. We attempt to make the bounds with high probability whenever possible, but some are in expectation.

We start by proving an analogue of Lemma IV.1.

Lemma V.1. *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on n-node m-edge digraph G. With probability $1 - n^{-10}$ in each recursive execution of the form* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *of Algorithm 2 the following holds:*

$$|R_{G'}^{\text{Des}}(v, \kappa_{2r}D)| \leq nk^{-r} \quad \text{and} \quad |R_{G'}^{\text{Anc}}(v, \kappa_{2r}D)| \leq nk^{-r}$$
$$\text{for all } v \in V(G') \ .$$

Proof: We proceed by induction. The base case $r = r^{\text{fringe}} = 0$ is clear. All recursive calls are one of the following forms: let G' be a digraph in which we made a call to PARALLELSC$(G', k, r, r^{\text{fringe}})$. We consider the cases of a recursive call to PARALLELSC$(G'[V_v^{\text{ring}}], k, r, r^{\text{fringe}} + 1)$ and a recursive call to PARALLELSC$(G'[V_i], k, r + 1, 0)$ separately (line 25 and 30). In the former case, the claim is clear, as in our recursive call to PARALLELSC$(G'[V_v^{\text{ring}}], k, r, r^{\text{fringe}} + 1)$ the parameter r stays fixed, and the digraph G' already satisfied

$$R_{G'}^{\text{Des}}(v, \kappa_{2r}D) \leq nk^{-r} \quad \text{and} \quad R_{G'}^{\text{Anc}}(v, \kappa_{2r}D) \leq nk^{-r}$$
$$\text{for all } v \in V(G') \ .$$

Now we consider the case referring to $G'[V_i]$. Consider a vertex $v \in V_i$. Order the vertices in $R_{G'}^{\text{Des}}(v, \kappa_{2r+1}D)$ as $w_1, w_2, \ldots, w_M$, where $M = |R_{G'}^{\text{Des}}(v, \kappa_{2r+1}D)|$. The ordering is such that if $i < j$ then either $w_i \not\succeq w_j$ or w_i and w_j are in the same strongly connected component. It is easy to see that such an ordering is possible.

Now, note for any t we have that

$$\left| R_{G'}^{\text{Anc}}(w_t, \kappa_{2r+1}D) \cap R_{G'}^{\text{Des}}(v, \kappa_{2r+1}D) \right| \leq t.$$

Therefore, if $w_t \in S$ for some $t \leq nk^{-r-1}$ then we would have that

$$|R_{G'[V_i]}^{\text{Des}}(v, \kappa_{2r+2}D)| \leq nk^{-r-1},$$

as $\kappa_{2r+2} \leq \kappa_{2r+1}$. The probability of this is at least

$$1 - (1 - p_r)^{\frac{n}{k^{r+1}}} \geq 1 - n^{-10}$$

from our choice of p_r, as desired. ∎

As we must recurse separately on fringe vertices (hence redoing the computation on them), we need to be able to control the number of fringe vertices. Therefore, it is natural to bound the expected number of times a fixed vertex is a fringe vertex for some breadth-first search.

Lemma V.2. *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on n-node m-edge digraph G and a recursive execution of the form* PARALLELSC$(G', k, r, r^{\text{fringe}})$. *Let $u \in V(G')$. The expected number of times u is in a recursive fringe subproblem, i.e.*

$$u \in R_{G'}(v, (\kappa + 1)D)) \backslash R_{G'}(v, (\kappa - 1)D))$$

for some $v \in S$ *(line 25), is at most* $\frac{1}{1000k \log^3 n}$.

Proof: For a vertex $v \in R_{G'}(u, \kappa_{2r}D)$, the probability that $v \in S$ and $u \in R_{G'}(v, (\kappa + 1)D)) \setminus R_{G'}(v, (\kappa - 1)D))$ over a random $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ is clearly at most $p_r \cdot \frac{2}{\kappa_{2r} - \kappa_{2r+1}}$. Also, we have that $|R_{G'}(u, \kappa_{2r}D)| \leq \frac{2n}{k^r}$ with high probability by Lemma V.1, so the expected number of times u is in a fringe subproblem is at most

$$p_r \cdot \frac{2}{\kappa_{2r} - \kappa_{2r+1}} \cdot \frac{2n}{k^r} \leq \frac{1}{1000k \log^3 n}$$

from our choice of $\kappa_{2r}, \kappa_{2r+1}, p_r$ and a direct computation. ∎

Now, we can show that the total size of all the digraphs we process during the algorithm doesn't increase much between levels of recursion. Additionally, at deep fringe recursion levels (for large r^{fringe}) exponentially few vertices are processed.

Lemma V.3 (Expected total size bounds). *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on n-node m-edge digraph G. For any recursion depth $r \leq \log_k n$ and fringe recursion depth r^{fringe} we have that:*

- *The expected value of the total number of vertices of the digraphs G' in all recursive executions* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *is at most $n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}}$.*
- *The expected value of the total number of edges of the digraphs G' in all recursive executions* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *is at most $m \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}}$.*

Proof: We focus on proving the first point, as the second is analogous. The first point follows directly by induction from a combination of Lemma V.1 and Lemma V.2. Specifically, we separate the cases $r^{\text{fringe}} > 0$ and $r^{\text{fringe}} = 0$. In the former case, note that this must result as a recursive call of the form PARALLELSC$(G''[V_v^{\text{ring}}], k, r, r^{\text{fringe}})$, where we also made a recursive call of the form PARALLELSC$(G'', k, r, r^{\text{fringe}} - 1)$. By induction and Lemma V.2, we know that the expected total number of vertices in all recursive calls to PARALLELSC$(G''[V_v^{\text{ring}}], k, r, r^{\text{fringe}})$ is at most

$$\frac{1}{1000k \log^3 n} \cdot n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}} - 1}}$$
$$\leq n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}}$$

as desired. We would like to note that the value $\frac{1}{1000k \log^3 n}$ is much smaller than we need to prove Lemma V.3; instead it is needed in Lemma V.9 below.

In the case $r^{\text{fringe}} = 0$, it must have resulted in a recursive call to PARALLELSC$(G''[V_i], k, r + 1, 0)$ for some digraph G'', where we also made a recursive call of the form

PARALLELSC$(G'', k, r, r^{\text{fringe}})$. As the V_i form a partition of $V(G'')$, we can see that the total number of vertices over all calls to PARALLELSC$(G''[V_i], k, r + 1, 0)$ for such G'' and V_i is at most

$$\sum_{r^{\text{fringe}} \geq 0} n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}} \leq n \cdot \left(1 + \frac{1}{\log n}\right)^{r+1}$$

as desired. ∎

Because exponentially few vertices are processed at large fringe recursion depths, we can guarantee that with high probability that $r^{\text{fringe}} \leq \log n$ always.

Corollary V.4 (Fringe depth is at most logarithmic). *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on an n-node m-edge digraph G. With probability at least $1 - n^{-10}$ we make no recursive calls of the form* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *and $r^{\text{fringe}} > \log n$.*

Proof: The expected value of the total number of vertices of the digraphs G' in all calls to PARALLELSC$(G', k, r, r^{\text{fringe}})$ where $r^{\text{fringe}} > \log n$ is at most

$$\sum_{r=0}^{\log_k n} \sum_{r^{\text{fringe}} > \log n} n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}}$$
$$\leq \frac{3n}{(\log n)^{\log n}} \leq \frac{1}{n^{10}}$$

by Lemma V.3. Thus, the claim follows. ∎

Additionally, we can bound the total sizes of all digraphs we process during the algorithm.

Corollary V.5 (Total size bound). *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on an n-node m-edge digraph G.*

- *The expected value of the total number of vertices in the digraphs G' in all recursive calls to* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *for some $r \leq \log_k n, r^{\text{fringe}}$ is at most $3n \log n$.*
- *The expected value of the total number of edges in the digraphs G' in all recursive calls to* PARALLELSC$(G', k, r, r^{\text{fringe}})$ *for some $r \leq \log_k n, r^{\text{fringe}}$ is at most $3m \log n$.*

Proof: Note that the total number of vertices over all the G' in recursive calls is at most

$$\sum_{r=0}^{\log_k n} \sum_{r^{\text{fringe}} \geq 0} n \cdot \left(1 + \frac{1}{\log n}\right)^r \cdot \frac{1}{(2 \log n)^{r^{\text{fringe}}}} \leq 3n \log n$$

by Lemma V.3. The edge bound follows similarly. ∎

We now proceed towards proving our ultimate bounds on expected total work and expected number of shortcut edges added.

Lemma V.6. *Consider an execution of* PARALLELSC$(G, k, 0, 0)$ *on an n-node, m-edge*

digraph G. *Consider a recursive execution of* PARALLELSC$(G', k, r, r^{\text{fringe}})$. *For all integers* $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$, *with probability* $1 - n^{-10}$ *over the choice of S we have that for all vertices $v \in V(G')$, the number of vertices $u \in S$ such that $v \in R_{G'}(u, \kappa D)$ is at most $50k \log n$.*

Proof: Fix $v \in V(G')$. By Lemma V.1, the number of u for which $v \in R_{G'}(u, \kappa D)$ is at most $2nk^{-r}$. Therefore, the expected number of $u \in S$ for which $v \in R_{G'}(u, \kappa D)$ is at most $p_r \cdot 2nk^{-r} \leq 20k \log n$. By a Chernoff bound (Lemma II.1), we have that the number of vertices $u \in S$ such that $v \in R_{G'}(u, \kappa D)$ is at most $50k \log n$ with probability at least

$$1 - \exp\left(-11k \log n\right) \geq 1 - n^{-11}.$$

The claim follows by union bound. ∎

Lemma V.7 (Work and depth bound). *An execution of* PARALLELDIAM(G, k) *on an n-node, m-edge digraph G can be implemented to do $\widetilde{O}(mk + nk^2)$ total work in expectation, add $\widetilde{O}(nk)$ shortcut edges in expectation, and have parallel depth $\widetilde{O}(\text{poly}(k)\sqrt{2}^{\log_k n} n^{\frac{1}{2}})$ with high probability.*

Proof: It suffices to show that an execution of PARALLELSC$(G, k, 0, 0)$ on an n-node, m-edge digraph G can be implemented to do $\widetilde{O}(mk)$ total work in expectation, add $\widetilde{O}(nk)$ shortcut edges in expectation, and have parallel depth $\widetilde{O}(\text{poly}(k)\sqrt{2}^{\log_k n} n^{\frac{1}{2}})$ with high probability. We show this in the two paragraphs below. Then the lemma follows as all digraphs G that we call PARALLELSC$(G, k, 0, 0)$ on during an execution of PARALLELDIAM(G, k) will have $\widetilde{O}(m + nk)$ edges with high probability.

Work bound: By Lemma V.6, it is clear that the number of shortcut edges added is within a multiplicative $\widetilde{O}(k)$ of the total number of vertices in all digraphs in all recursive subproblems with high probability, hence is $\widetilde{O}(nk)$ in expectation by Lemma V.3.

Similarly, the total work from the breadth first searches is within a multiplicative $\widetilde{O}(k)$ of the total number of edges in all digraphs in all recursive subproblems, hence is $\widetilde{O}(mk)$ in expectation by Lemma V.3. The remaining nontrivial work comes from line 28. We implement line 28 in the following manner. Consider a recursive execution of the algorithm on a digraph PARALLELSC$(G', k, r, r^{\text{fringe}})$. We put an arbitrarily total ordering on the labels v^{Anc} and v^{Des}, and for each vertex $u \in V(G')$ we assume that its set of labels, none of which are ✗, are sorted according to this total ordering. Now, we sort all vertices $u \in V(G')$ that have no label which is ✗ lexicographically by their set of labels using Cole's mergesort algorithm [17], and then group them into the sets V_i based on contiguous groups that have the same label. Each comparison takes work $\widetilde{O}(k)$ with high probability as every vertex u has $\widetilde{O}(k)$ labels with high

probability by Lemma V.6. The total work of the merge sort is thus $\widetilde{O}(k|V(G')|)$ as desired.

Depth bound: Clearly, the breadth first searches (lines 10 to 24) can be implemented in $\widetilde{O}(\kappa_0 D) = \widetilde{O}(\text{poly}(k)\sqrt{2}^{\log_k n} n^{\frac{1}{2}})$ parallel depth by our choice of κ_0. Line 28 can be implemented in parallel depth $\widetilde{O}(k)$ via Cole's merge sort [17] as described in the above paragraph. ∎

D. Diameter Bound after Shortcutting

In this section we show the analogue of Lemma IV.6.

Lemma V.8 (Inductive parallel diameter bound). *Perform an execution of* PARALLELSC$(G, k, 0, 0)$ *on an n-node, m-edge digraph G. Consider a recursive execution of* PARALLELSC$(G', k, r, r^{\text{fringe}})$, *with all shortcut edges added. For any path $P \subseteq G'$ with length at most D, the expected distance from $\text{head}(P)$ to $\text{tail}(P)$ using edges in G' and shortcut edges is at most*

$$5(\bar{r} + 1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} s(P, G')^{\frac{1}{2}},$$

where $\bar{r} = \log_k n - r$, and $s(P, G')$ is the number of partially path related vertices to P in G', i.e. $s(P, G') = |R_{G'}(P, \kappa_{2r} D)|$.

Note that we have that

$$5(\log_k n + 1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\log_k n} n^{\frac{1}{2}}$$
$$\leq 10 \log n \sqrt{2}^{\log_k n} n^{\frac{1}{2}} \leq \frac{D}{10},$$

so that any path of length D will be shortcut to expected length $\frac{D}{10}$ in one run of PARALLELSC$(G, k, 0, 0)$.

Setup for proof of Lemma V.8: An execution of PARALLELSC$(G', k, r, r^{\text{fringe}})$ will generate many recursive subproblems, each of the form PARALLELSC$(G'[V_v^{\text{ring}}], k, r, r^{\text{fringe}} + 1)$ or PARALLELSC$(G'[V_i], k, r + 1, 0)$. These recursive subproblems (other than the fringe vertices) all involve disjoint sets of vertices, hence the path P that we are trying to analyze gets "split" into subpaths when we look at the recursive subproblems. Our next claim gives structure on how we can split up the path P into subpaths, which are contained in our various subproblems.

Lemma V.9 (Splitting up a path). *Consider the setup described in the above paragraph and Lemma V.8. Let κ be such that $\kappa_{2r+1} \leq \kappa \leq \kappa_{2r}$ and $S \subseteq R_{G'}(P, \kappa D)$ be the set of shortcutters. There exists paths $P_i^{S,\kappa}$ (possibly empty) for $1 \leq i \leq |S| + 1$ and $P_i^{S,\kappa,\text{fringe}}$ (possibly empty) for $1 \leq i \leq |S|$ such that*

1) *If a vertex $u \in S$ is a κD-bridge, then all $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$ are empty.*

2) *If no vertex $u \in S$ is a κD-bridge, then the vertex disjoint union of all $P_i^{S,\kappa}$ for $1 \le i \le |S| + 1$ and $P_i^{S,\kappa,\mathrm{fringe}}$ for $1 \le i \le |S|$ is exactly P.*
3) *Each $P_i^{S,\kappa}$ is inside V_{a_i} for some index a_i.*
4) *Each $P_i^{S,\kappa,\mathrm{fringe}}$ is a subset of $V_{u_i}^{\mathrm{ring}}$ for some vertex $u_i \in S$ (line 25).*

Additionally, we have that

$$\mathbb{E}_{\kappa, |S|=t}\left[\sum_{i=1}^{t+1} s(P_i^{S,\kappa}, G'[V_{a_i}])\right] \le \frac{2}{t+1} s(P, G') \quad (1)$$

and

$$\mathbb{E}_{\kappa, S}\left[\sum_{i=1}^{t} s(P_i^{S,\kappa,\mathrm{fringe}}, G'[V_{u_i}^{\mathrm{ring}}])\right] \le \frac{1}{1000 k \log^3 n} s(P, G'), \quad (2)$$

where $\mathbb{E}_{\kappa, |S|=t}$ refers first selecting $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ uniformly at random and then picking the shortcutter set S as a random subset of $R_{G'}(P, \kappa D)$ of size exactly t (as opposed to the way S is selected in Algorithm 2). $\mathbb{E}_{\kappa, S}$ refers to first selecting $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ uniformly at random and then selecting S as in Algorithm 2, where every vertex $u \in V(G')$ is in S with probability p_r.

Here, we assert that $S \subseteq R_{G'}(P, \kappa D)$ as searching to distance κD from vertices $u \notin R_{G'}(P, \kappa D)$ does not affect the path P.

Proof: We describe a process for constructing the $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\mathrm{fringe}}$, and then verify that this construction satisfies all the necessary constraints.

If a vertex $u \in S$ is a κD-bridge, then we set all the $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\mathrm{fringe}}$ to be empty. Otherwise, first consider the case where $u \in S$ is a κD-descendent. Let $v_0 \preceq v_1 \preceq \cdots \preceq v_\ell$ be the vertices on the path P. Define

$$\mathrm{bot}(u) = \min\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Anc}}(u, \kappa D)\}$$

and $\mathrm{top}(u) = \min\Big(\mathrm{bot}(u) + 1,$

$$\max\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Anc}}(u, (\kappa-1)D)\}\Big)$$

and $\mathrm{top}(u) = \mathrm{bot}(u) + 1$ in the case that the set $\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Anc}}(u, (\kappa-1)D)\}$ is empty. Define

$$W^{\mathrm{top}}(u) = \{v_i : 0 \le i < \mathrm{top}(u)\},$$
$$W^{\mathrm{bot}}(u) = \{v_i : \mathrm{bot}(u) < i \le \ell\},$$
$$W^{\mathrm{fringe}}(u) = \{\mathrm{top}(u) \le i \le \mathrm{bot}(u)\}.$$

In particular, it is easy to verify that $W^{\mathrm{fringe}}(u) \subseteq V_u^{\mathrm{fringe}}$. For the case where $u \in S$ is a κD-ancestor, define

$$\mathrm{top}(u) = \max\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Des}}(u, \kappa D)\}$$

and $\mathrm{bot}(u) = \max\Big(\mathrm{top}(u) - 1,$

$$\min\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Des}}(u, (\kappa-1)D)\}\Big),$$

where $\mathrm{bot}(u) = \mathrm{top}(u) - 1$ in the case that the set $\{i : 0 \le i \le \ell, v_i \in R_{G'}^{\mathrm{Des}}(u, (\kappa-1)D)\}$ is empty. We can now define $W^{\mathrm{top}}(u), W^{\mathrm{bot}}(u)$, and $W^{\mathrm{fringe}}(u)$ analogously.

Now, our goal is to define a set of natural subpaths of P which are inside various V_i (line 28). Specifically, for a subset $T \subseteq S$, we define

$$W_T = \left(\bigcap_{u \in T} W^{\mathrm{top}}(u)\right) \cap \left(\bigcap_{u \in S \setminus T} W^{\mathrm{bot}}(u)\right).$$

W_T was defined so that it is inside some subset V_i (line 28) and is a subpath of P. To show that W_T is contained inside some V_i it suffices to show that all vertices in $W^{\mathrm{top}}(u)$ receive the same label from u (and by symmetry the same holds for $W^{\mathrm{bot}}(u)$). In the case where u is a κD-descendant of P, it is easy to verify that $W^{\mathrm{top}}(u) \subseteq R_{G'}^{\mathrm{Anc}}(u, \kappa D)$, and that all vertices in $W^{\mathrm{bot}}(u)$ are unrelated to u. In the case where u is a κD-ancestor, it is easy to verify that $W^{\mathrm{bot}}(u) \subseteq R_{G'}^{\mathrm{Des}}(u, \kappa D)$, and that all vertices in $W^{\mathrm{top}}(u)$ are unrelated to u. This shows that W_T always lies inside V_i for some i.

It is easy to check that the number of nonempty W_T over subsets $T \subseteq S$ is at most $|S| + 1$. We denote these sets $W_1, W_2, \ldots, W_{|S|+1}$. For simplicity, label the paths $W^{\mathrm{fringe}}(u)$ for $u \in S$ as $W_{|S|+2}, \ldots, W_{2|S|+1}$. By definition, we see that P itself is the (not necessarily disjoint) union of $W_1, W_2, \ldots, W_{2|S|+1}$. Now, one can easily check that for any paths $W_1, W_2, \ldots, W_{2|S|+1}$ whose union is P, then there are subpaths $Z_1, Z_2, \ldots, Z_{2|S|+1}$ (possibly empty) such that $Z_i \subseteq W_i$ and that the vertex disjoint union of the Z_i is P.

Now, we define the $P_i^{S,\kappa}$ as the subpaths Z_j for $1 \le j \le |S| + 1$, and define the $P_i^{S,\kappa,\mathrm{fringe}}$ as the subpaths Z_j for $|S| + 2 \le j \le 2|S| + 1$, which then by definition were subpaths of $W^{\mathrm{fringe}}(u)$ for some u. Now we verify that these choices satisfy the conditions of Lemma V.9.

Construction satisfies item 1: This is by definition (see the first sentence of the proof).

Construction satisfies item 2: We have defined the Z_j so that their vertex disjoint union is P, and each of the $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\mathrm{fringe}}$ is one of the Z_j or empty.

Construction satisfies item 3: The definition

$$W_T = \left(\bigcap_{u \in T} W^{\mathrm{top}}(u)\right) \cap \left(\bigcap_{u \in S \setminus T} W^{\mathrm{bot}}(u)\right)$$

corresponds to assigning the labels to vertices $V(G')$ and partitioning into sets V_i. As each $P_i^{S,\kappa}$ was a subpath of some W_T, the claim follows.

Construction satisfies item 4: Each $P_i^{S,\kappa,\mathrm{fringe}}$ is a subpath of one of $W_{|S|+2}, \ldots, W_{2|S|+1}$, which were each $W^{\mathrm{fringe}}(u)$ for some u. We have noted that $W^{\mathrm{fringe}}(u) \subseteq V_u^{\mathrm{ring}}$.

Construction satisfies Eq. (2): Note that

$$\mathbb{E}_{\kappa,S}\left[\sum_{i=1}^{t} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])\right]$$

$$\leq \mathbb{E}_{\kappa,S}\left[\sum_{v\in S}\left|V_v^{\text{ring}}\cap R_{G'}(P,\kappa_{2r}D)\right|\right] \qquad (3)$$

as no vertices outside of $R_{G'}(P,\kappa_{2r}D)$ can become path related. The right hand side of Eq. (3) is counting for every vertex u in $R_{G'}(P,\kappa_{2r}D)$ the expectation over the choices of κ, S of the number of times that u is in a fringe subproblem, i.e. $u\in V_v^{\text{ring}}$ for some $v\in S$. By applying Lemma V.2, we can see that this expression in Eq. (3) is with high probability at most

$$\mathbb{E}_{\kappa,S}\left[\sum_{v\in S}\left|V_v^{\text{ring}}\cap R_{G'}(P,\kappa_{2r}D)\right|\right]$$

$$\leq \frac{1}{1000k\log^3 n}|R_{G'}(P,\kappa_{2r}D)| = \frac{1}{1000k\log^3 n}s(P,G').$$

Construction satisfies Eq. (1): The proof essentially follows the same shape as Lemma IV.4 and Lemma IV.5. Our main claim is the following.

Claim V.10. *Consider an arbitrary subset of vertices $\{u_1, u_2, \cdots, u_{t+1}\} \subseteq R_{G'}(P,\kappa D)$. Then there are at most 2 indices $1 \leq h \leq t+1$ such that setting $S = \{u_j : j \neq h\}$ and running the procedure described so far in the proof of Lemma V.9 to produce the sets $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$ results in u_h still being path related in a non-fringe subproblem, i.e. $u_h \in R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$ for some $1 \leq i \leq t+1$.*

Now we explain why Claim V.10 implies Eq. (1). Intuitively, in Eq. (1), we can think of choosing the set S with $|S| = t$ instead as first uniformly randomly choosing a subset $S' \subseteq R_{G'}(P,\kappa D)$ with $|S'| = t+1$ and then choosing uniformly randomly choosing $S \subseteq S'$ with $|S| = t$. We then apply Claim V.10. Formally, we have the following, where $\mathbb{1}(S, u)$ is the function evaluating to 1 when $u \in R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$ for some $1 \leq i \leq t+1$ and 0 otherwise.

$$\mathbb{E}_{\kappa,|S|=t}\left[\sum_{i=1}^{t+1} s(P_i^{S,\kappa}, G'[V_{a_i}])\right]$$

$$= \mathbb{E}_{\kappa,|S|=t}\left[\sum_{u\in R_{G'}(P,\kappa D)} \mathbb{1}(S, u)\right]$$

$$\leq s(G', P) \cdot \mathbb{E}_{\kappa,|S'|=t+1}\mathbb{E}_{\substack{S\subseteq S'\\|S|=t}}\left[\mathbb{1}(S'\backslash S, S)\right]$$

$$\leq \frac{2}{t+1}s(G', P)$$

by Claim V.10 as desired. Between the second and third lines we have used linearity of expectation on the inner sum and symmetry. For the remainder of this proof, we focus on proving Claim V.10. We instead prove a slightly stronger

and more restrictive version of Claim V.10 in the case that all the u_i are path ancestors (or all path descendants).

Claim V.11. *Consider an arbitrary subset of vertices $\{u_1, u_2, \cdots, u_{t+1}\} \subseteq R_{G'}(P,\kappa D)$ such that the u_i are all path ancestors or all path descendants. Then there is at most one index $1 \leq h \leq t+1$ such that setting $S = \{u_j : j \neq h\}$ and running the procedure described so far in the proof of Lemma V.9 to produce the sets $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$ results in u_h still being path related in a non-fringe subproblem, i.e. $u_h \in R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$ for some $1 \leq i \leq t+1$.*

We now explain why Claim V.10 follows from Claim V.11 and then show Claim V.11. To show Claim V.10, note that the claim is trivial if the set $S' = \{u_1, u_2, \cdots, u_{t+1}\}$ contains a bridge. Indeed, as picking the set S to contain a bridge leaves no path related subproblems, the only index j satisfying the constraints of Claim V.10 must be the j where u_j is the bridge. Now, assume that all of the vertices in S' are either path ancestors or descendants (not bridges). Applying Claim V.11 on the subset of path ancestors among S' and the subset of path descendants among S' immediately implies Claim V.10: we get one valid index from the path ancestors and one from the descendants, for a total of two.

We now show Claim V.11. Consider the set $S' = \{u_1, u_2, \cdots, u_{t+1}\}$, where all the u_i are path ancestors. Therefore, for any subset $S \subseteq S'$, if we shortcut using vertices in S, then all vertices in P will only get labels of the form u_i^{Des}, and no u_i^{Anc} labels. For every vertex $u \in S'$, define

$$\alpha(u) = \min\{i : 0 \leq i \leq \ell \text{ and } v_j \in R_{G'}^{\text{Des}}(u, \kappa D) \text{ for all } j \geq i\},$$

where we recall that the path P consists of vertices $v_0 \preceq v_1 \preceq \cdots \preceq v_\ell$. We claim that the only index j that could satisfy the condition in Claim V.11 is such that

- We have that $u_j \npreceq^{\kappa D} u_{j'}$ for all $j' \neq j$.
- $\alpha(u_j)$ is unique and minimal out of all the values of $\alpha(u_{j'})$ for the vertices $u_{j'}$ satisfying the first point.

Indeed, for an index j, if there is an index j' such that $u_j \preceq^{\kappa D} u_{j'}$ then vertex u_j receives the label $u_{j'}^{\text{Anc}}$ when we shortcut from $u_{j'}$. On the other hand, no vertices on the path P receive the label $u_{j'}^{\text{Anc}}$ as $u_{j'}$ is a path ancestor. Now, let

$$I = \{j : u_j \npreceq^{\kappa D} u_{j'} \text{ for all } j' \neq j\},$$

the set of indices which satisfied the first condition. Note that by definition, for any distinct indices $j, j' \in I$, the vertices u_j and $u_{j'}$ are unrelated within distance κD. Now, assume that $\alpha(u_j) \geq \alpha(u_{j'})$. We aim to show that in this case that $u_j \notin R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$ for any $1 \leq i \leq t+1$, which would complete the proof. Indeed, if $u_j \in R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$, then by the definition of $\alpha(u_j)$ we would need for $P_i^{S,\kappa}$ to be a subset of the path P from $v_{\alpha(u_j)}$ to v_ℓ. Otherwise, the path $P_i^{S,\kappa}$ would be forced to be part of a fringe subproblem induced by u_j. Now, again by the definition of $\alpha(u_{j'})$, we

know that all vertices in the path P from $v_{\alpha(u_j)}$ to v_ℓ will receive a label of $u_{j'}^{\text{Des}}$ because $\alpha(u_j) \geq \alpha(u_{j'})$, while vertex u_j does not receive a label from $u_{j'}$, as $u_{j'} \not\preceq^{\kappa D} u_j$. Therefore, $u_j \notin R_{G'[V_{a_i}]}(P_i^{S,\kappa}, \kappa D)$ for any $1 \leq i \leq t+1$, which completes the proof. ∎

Now, we prove Lemma V.8.

Proof of Lemma V.8: The proof is via induction and Lemma V.9. Indeed, the claim is trivial for $\bar{r} = 0$, as Algorithm 2 chooses all vertices as shortcutters as $p_r = 1$.

We first claim that (using the same notation as Lemma V.9) that with probability $1 - n^{-10}$ we have that $|R_{G'}(P, \kappa D)| \leq 100k \log n$. Indeed, note that

$$|R_{G'}(P, \kappa D)|$$
$$\leq |R_{G'}(\text{head}(P), (\kappa+1)D)| + |R_{G'}(\text{tail}(P), (\kappa+1)D)|$$
$$\leq 100k \log n$$

with probability at least $1 - n^{-10}$ by Lemma V.6.

For a path P, parameter κ, and set S we have by Lemma V.9 that we can partition P into disjoint paths $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$. Then the final length of P after shortcutting through S and recursive shortcutting is clearly at most the total length of all the $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$ after shortcutting, plus at most $2|S| + 2$ (for edges between adjacent paths). Note that this even holds in the case where S contains a bridge: in this case all the $P_i^{S,\kappa}$ and $P_i^{S,\kappa,\text{fringe}}$ are empty by definition, and the shortcutted length of P is 2, which is at most $2|S| + 2$.

By induction, we have that the expected length of P after shortcutting through S and recursive shortcutting is at most

$$\mathbb{E}_{\kappa,S}\Big[2|S| + 2 \tag{4}$$
$$+ \sum_{i=1}^{|S|+1} 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} s(P_i^{S,\kappa}, G'[V_{a_i}])^{\frac{1}{2}} \tag{5}$$
$$+ \sum_{i=1}^{|S|} 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])^{\frac{1}{2}} \Big] \tag{6}$$
$$= 2 + 2\mathbb{E}_{\kappa,S}[|S|] \tag{7}$$
$$+ 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|+1} s(P_i^{S,\kappa}, G'[V_{a_i}])^{\frac{1}{2}} \Big] \tag{8}$$
$$+ 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} \mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])^{\frac{1}{2}} \Big] \tag{9}$$

by induction, where $\mathbb{E}_{\kappa,S}$ refers to first selecting $\kappa \in [\kappa_{2r+1}, \kappa_{2r}]$ uniformly at random and then selecting S analogous to Algorithm 2, where every vertex $u \in R_{G'}(P, \kappa D)$ is in S with probability p_r. We only care about those $u \in R_{G'}(P, \kappa D)$ as those are the only path related vertices,

and picking other vertices can only make less vertices path related in the future.

We bound the pieces of Eqs. (7) to (9) now. We split the analysis into cases depending on the size of $\mathbb{E}_{\kappa,S}[|S|]$.

Case 1: $\mathbb{E}_{\kappa,S}[|S|] \geq s(P, G')^{\frac{1}{2}}$: We assume that the length ℓ of the path P must satisfy

$$\ell \geq 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} s(P, G')^{\frac{1}{2}},$$

or else the claim is trivially true. If

$$p_r \cdot s(P, G') \geq s(P, G')^{\frac{1}{2}}, \text{ then } p_r \geq s(P, G')^{-\frac{1}{2}}$$

so we have that the probability that some vertex of P is in S is at least

$$1 - (1 - p_r)^\ell \geq 1 - \exp(-\ell p_r) \geq 1 - n^{-5}$$

by our assumed bounds on ℓ and p_r. Therefore, our path P is shortcut to length 2 with high probability. This completes the analysis for this case.

Case 2: $\mathbb{E}_{\kappa,S}[|S|] \leq s(P, G')^{\frac{1}{2}}$: We now bound each of the pieces of Eqs. (7) to (9).

Bound on $\mathbb{E}_{\kappa,S}[|S|]$: Note that

$$\mathbb{E}_{\kappa,S}[|S|] \leq p_r s(P, G') \leq s(P, G')^{\frac{1}{2}}$$

by our assumed bound on p_r.

Bound on rightmost term in Eq. (8): Using the Cauchy-Schwarz inequality and Lemma V.9 Eq. (1) we can see that

$$5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|+1} s(P_i^{S,\kappa}, G'[V_{a_i}])^{\frac{1}{2}} \Big]$$
$$\leq 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \sum_t \Pr[|S| = t]\cdot$$
$$\mathbb{E}_{\kappa,|S|=t}\Big[\sum_{i=1}^{t+1} s(P_i^{S,\kappa}, G'[V_{a_i}])^{\frac{1}{2}} \Big]$$
$$\leq 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \sum_t \Pr[|S| = t]\cdot$$
$$\mathbb{E}_{\kappa,|S|=t}\Big[\Big((t+1)\sum_{i=1}^{t+1} s(P_i^{S,\kappa}, G'[V_{a_i}]) \Big)^{\frac{1}{2}} \Big]$$
$$\leq 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \sum_t \Pr[|S| = t]\cdot$$
$$\mathbb{E}_{\kappa,|S|=t}\Big[\Big((t+1)\sum_{i=1}^{t+1} s(P_i^{S,\kappa}, G'[V_{a_i}]) \Big) \Big]^{\frac{1}{2}}$$
$$\leq 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \sum_t \Pr[|S| = t] \cdot \sqrt{2} \cdot s(P, G')^{\frac{1}{2}}$$
$$= 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot \sqrt{2} \cdot s(P, G')^{\frac{1}{2}}.$$

Bound on Eq. (9): We follow the same plan as in the above paragraph. Using the Cauchy-Schwarz inequality, Lemma V.9 Eq. (2), and our observation above that $|R_{G'}(P, \kappa_{2r}D)| \le 100k \log n$ with high probability we can see that

$$5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot$$

$$\mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])^{\frac{1}{2}}\Big]$$

$$\le 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot$$

$$\mathbb{E}_{\kappa,S}\left[\left(|S|\sum_{i=1}^{|S|} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])\right)^{\frac{1}{2}}\right]$$

$$\le 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot$$

$$\mathbb{E}_{\kappa,S}\left[\left(|S|\sum_{i=1}^{|S|} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])\right)\right]^{\frac{1}{2}}$$

$$\le 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} \sum_t \Pr[|S| = t]\cdot$$

$$\sqrt{\frac{|R_{G'}(P, \kappa_{2r}D)|}{1000k \log^3 n}} \cdot s(P, G')^{\frac{1}{2}}$$

$$\le 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} \sum_t \Pr[|S| = t]\cdot$$

$$\sqrt{\frac{100k \log n}{1000k \log^3 n}} \cdot s(P, G')^{\frac{1}{2}}$$

$$\le 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot \frac{1}{2\log n} \cdot s(P, G')^{\frac{1}{2}}.$$

Summing all our contributions gives that the expression in Eqs. (7) to (9) is at most

$$2 + 2\mathbb{E}_{\kappa,S}[|S|]$$

$$+ 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|+1} s(P_i^{S,\kappa}, G'[V_{a_i}])^{\frac{1}{2}}\Big]$$

$$+ 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} \mathbb{E}_{\kappa,S}\Big[\sum_{i=1}^{|S|} s(P_i^{S,\kappa,\text{fringe}}, G'[V_{u_i}^{\text{ring}}])^{\frac{1}{2}}\Big]$$

$$\le 2 + 2s(P, G')^{\frac{1}{2}} + 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot \sqrt{2} \cdot s(P, G')^{\frac{1}{2}}$$

$$+ 5\bar{r}\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}-1} \cdot \frac{1}{2\log n} \cdot s(P, G')^{\frac{1}{2}}$$

$$\le 5(\bar{r}+1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\bar{r}} s(P, G')^{\frac{1}{2}}$$

as desired. ∎

To conclude the proof of the correctness of Algorithm 3, it suffices to argue after a run Algorithm 3, for any vertices s, t such that $s \preceq t$, after shortcutting there is now a path of length at most D from s to t.

Lemma V.12. *Perform an execution of* $\text{PARALLELDIAM}(G, k)$ *on an n-node, m-edge digraph G. With high probability for any vertices s, t such that $s \preceq t$ there is a path of length at most D from s to t in G, where D is as in Algorithm 2.*

Proof: By Lemma V.8 and the observation that

$$5(\log_k n + 1)\left(\sqrt{2} + \frac{1}{2\log n}\right)^{\log_k n} n^{\frac{1}{2}}$$

$$\le 10 \log n \sqrt{2}^{\log_k n} n^{\frac{1}{2}} \le \frac{D}{10},$$

we can see that for a path P of length at most P the probability that it is shortcut to length at most $\frac{D}{2}$ in a run of $\text{PARALLELSC}(G, k, 0, 0)$ which does not exceed the work or shortcut edge bound by ten times is at least $1 - \frac{1}{5} - \frac{2}{10} > \frac{1}{2}$ by Markov's inequality. Therefore, running $\text{PARALLELSC}(G, k, 0, 0)$ $10 \log n$ times as in Algorithm 3 shortcuts a path of length D to length at most $\frac{D}{2}$ with high probability. Doing this $10 \log n$ times then guarantees that an arbitrary path is shortcut to length at most D with high probability. As we only need to shortcut $O(n^2)$ paths (a single path for every pair of vertices s, t) this is sufficient. ∎

Combining Lemma V.7 and Lemma V.12 readily gives a proof of Theorem 4. Combining these along with a breadth first search to depth D now gives a proof of Theorem 1.

Proof of Theorem 1: Take $k = O(\log n)$. For input graph G, consider first performing an execution of $\text{PARALLELDIAM}(G, k)$ and then running a breadth first search from vertex s. Clearly this algorithm solves the single source reachability problem from s. Lemma V.7 gives with high probability the desired $\widetilde{O}(mk + nk^2)$ total work bound and shows that the algorithm adds at most $\widetilde{O}(nk)$ shortcuts with high probability. Lemma V.12 shows that the breadth first search runs in depth $D \le \widetilde{O}\left(n^{\frac{1}{2}+O\left(\frac{1}{\log k}\right)}\right)$, where D is as in Algorithm 2. By Lemma V.7 we have that an execution of $\text{PARALLELDIAM}(G, k)$ can be implemented in parallel depth $\widetilde{O}\left(\text{poly}(k) \cdot n^{\frac{1}{2}+O\left(\frac{1}{\log k}\right)}\right)$. This finishes the proof. ∎

VI. DISTRIBUTED SINGLE SOURCE REACHABILITY IN THE CONGEST MODEL

In this section we explain how to combine our parallel algorithm in Section V with the techniques of [4] and [5] to obtain new algorithms for the single source reachability problem in the CONGEST model. In Section VI-A we formally define the CONGEST model and the single source reachability problem, as well as covering some standard

results in the CONGEST model. In Section VI-B we state our main algorithm DISTRREACH for solving the single source reachability problem. Finally, in Section VI-C we sketch a proof of how to apply the results of Section V to bound the number of rounds our algorithm takes.

A. Preliminaries

The CONGEST model.: In the CONGEST model, we model a communication network as an undirected unweighted graph G with n vertices and m edges, where the processors are modeled as vertices and edges as bounded bandwidth links between processors. We let $V(G)$ and $E(G)$ denote the vertex set of G and the (directed) edge set of G respectively. The processors/vertices are assumed to have distinct polynomially bounded IDs. Each vertex has infinite computational power, but only knows little about the graph G; it only knows the IDs of its neighbors and no other information about the graph G.

Now we define the *single source reachability problem* that we consider. Here, each vertex also knows which of its neighbors are in-neighbors and which are out-neighbors in the directed graph G. Here, communication on edges are bidirectional even though the edges themselves are directed edges. Additionally, there is a single source vertex s. As vertex s can communicate that it is the source to all other vertices in $O(D)$ rounds (Lemma VI.1), we assume that all vertices know s at the start. The goal of the single source reachability problem is for every vertex $t \in V(G)$ to know whether is an $s \to t$ path in G at the end of the algorithm. In particular, s doesn't have to know whether there is an $s \to t$ path in G, only t must know.

The performance of an algorithm or protocol is judged by the number of *rounds* of distributed communication in the worst case. At the start of each round, every vertex can send a message of length $O(\log n)$ to each of its neighbors, where the message need not be the same across different neighbors. The messages arrive at the end of the round. Running times are analyzed in terms of the number of vertices n and the undirected/hop diameter of G, which we denote as D. As every vertex can learn n within $O(D)$ rounds, we assume that all vertices know n.

We now note a few standard results about distributed computation in the CONGEST model.

Lemma VI.1 (Global message broadcast protocol, [9]). *Suppose each $v \in V$ holds $k_v \geq 0$ messages of $O(\log n)$ bits each, for a total of $K = \sum_{v \in V} k_v$. Then all nodes in the network can receive these K messages within $O(K+D)$ rounds.*

Theorem 6 ([18]). *Consider k distributed algorithms $A_1, \ldots, A_k$. Let* dilation *be such that each algorithm A_i finishes in* dilation *rounds if it runs individually. Let* congestion *be such that there are at most* congestion *messages, each of size $O(\log n)$, sent through each edge (counted*

over all rounds), when we run all algorithms together. There is a distributed algorithm that can execute $A_1, \ldots, A_k$ in $O($dilation $+$ congestion $\cdot \log n)$ rounds in the CONGEST model.

Corollary VI.2 (Low distance breadth first search). *Consider a set $T \subseteq V(G)$ and a distance h. There is a protocol such that for every vertex v, it learns the subsets of T that are distance h to and from itself in $\widetilde{O}(|T| + h)$ rounds.*

B. Algorithm description

Algorithm overview: Our algorithm consists of multiple steps. The first is a standard procedure reminiscent of the classic $O(m\sqrt{n})$ time shortcutting algorithm of Ullman and Yannakakis [3]. It has been used in the earlier works of Ghaffari and Udwani [5] and Nanongkai [4]. We randomly sample each vertex in $V(G)$ with probability $\frac{10\alpha \log n}{n}$ to be in a subset $T \subseteq V(G)$. We also assert that our source $s \in T$. With high probability $|T| = \widetilde{O}(\alpha)$. Using Corollary VI.2 with parameter $h = \widetilde{O}(n/\alpha)$, for each vertex v, it learns exactly the subset of T that can reach itself within distance h, and the subset of T that it can reach within distance h. This takes $\widetilde{O}(\alpha + h) = \widetilde{O}(\alpha + \frac{n}{\alpha})$ rounds. Now, we build the *skeleton graph G_T* on T using the reachability relations learned from the breadth first searches. Concretely, for vertices $u, v \in T$, there is an edge $u \to v$ in graph G_T if $d_G(u, v) \leq h$. It is direct to see that with high probability that the reachability from s to T on G_T is the same as that on G.

Now, every vertex in T tries to learn whether it is reachable from s in graph G_T. After this, the vertices in T can global broadcast whether they are reachable from s using Lemma VI.1 in $O(D+|T|)$ rounds. Then every vertex in G can locally determine whether s can reach it, as desired.

In order to let every vertex in T learn whether it is reachable from s, we run a small variation on Algorithm 2 and Algorithm 3 on G_T. Intuitively, during every unit of time in the parallel computation every vertex globally broadcasts much of its computational transcript, such as its current labels, which new shortcut edges are being added, whether it has just been visited in a breadth first search, etc. Through the guarantees of Theorem 4 with $k = \log |T|$ we can see that this process essentially takes $\widetilde{O}(D|T|^{1/2+o(1)} + |T|) = \widetilde{O}(D\alpha^{1/2+o(1)} + \alpha)$ rounds. The first term comes from a fixed cost of $O(D)$ to globally broadcast during each stage of a $|T|^{1/2+o(1)}$ depth algorithm. The second term is the total number of messages sent by all vertices during the process, which corresponds to the nearly linear runtime of the algorithm. Combining this with the first paragraph gives an algorithm taking

$$\widetilde{O}\left(\frac{n}{\alpha} + \alpha + D\alpha^{1/2+o(1)}\right)$$

rounds. Properly trading off parameters then gives Theorem 2.

We remark that using the reachability algorithm of [1] (instead of our improved algorithm) would also imply a new result; precisely the number of rounds would be $\widetilde{O}(\sqrt{n} + n^{2/5}D^{3/5})$, which is nearly optimal for $D \leq n^{1/6}$.

We now give algorithm DISTRREACH (Algorithm 4). It is fairly clear that in this way, Algorithm 4 can simulate PARALLELDIAM in the CONGEST model.

Algorithm 4 DISTRREACH(G, s), distributed algorithm for solving the single source reachability problem from s in G in the CONGEST model.

1: Choose $T \subseteq V(G)$ as follows: each vertex $v \in V(G)$ has a $\frac{10\alpha \log n}{n}$ probability of being in T. Set $h = \frac{10n \log n}{\alpha}$. Apply Corollary VI.2 so that all vertices $v \in V(G)$ learn what vertices in T are within distance h to / from itself. Build the skeleton graph G_T with vertex set T as follows: for $u, v \in T$ there is an edge $u \to v$ if and only if $d_G(u, v) \leq h$.

2: Run a variant of PARALLELDIAM(G, k) on G_T. After every unit of parallel depth, every vertex globally broadcasts all new computation of the following forms, as well as an ID for the current recursive subproblem it is broadcasting for. Note that a vertex can broadcast / do computation for multiple subproblems simultaneously.

  - Vertex v is in the shortcutter set S.
  - Vertex v is visited by a breadth first search from a vertex $u \in S$.
  - Vertex v gets a new label / becomes a fringe vertex.
  - A new shortcut edge is added to the graph.

In addition, each subproblem must choose a "leader" to pick the κ parameter for that subproblem. The skeleton graph G_T is updated after each run of PARALLELSC$(G, k, 0, 0)$ to include all the new shortcut edges.

3: Simulate a BFS on G_T (with all the new shortcut edges) of depth $100\sqrt{2}^{\log_k n} n^{\frac{1}{2}} \log^2 n$ so that all $v \in T$ learn whether they are reachable from s. All vertices $v \in T$ globally broadcast whether they are reachable from s. Each vertex $v \in V(G)$ now locally computes whether it is reachable from s.

C. Analysis

The bulk of the analysis follows in the same way as the analysis in Section V.

Theorem 7. *Algorithm* DISTRREACH *when executed on an n-vertex m-edge graph G with high probability solves the single source reachability problem in* $\widetilde{O}\left(\alpha + (n/\alpha) + D\alpha^{1/2+o(1)}\right)$.

Proof: By Corollary VI.2, we have that Line 1 in Algorithm DISTRREACH can be implemented with $O(|T| + h) = \widetilde{O}(\alpha + n/\alpha)$ rounds.

Our procedures simulates running PARALLELDIAM(G_T, k) over the graph. After every unit of parallel depth in our algorithm, we globally broadcast the complete reachability information of all performed searches over the graph. This consists of $\widetilde{O}(\alpha)$ messages in total. Further, we observe that during each graph traversal we perform during PARALLELDIAM nodes will locally check if one of their neighbors have been marked by a search and broadcast themselves appropriately if they have not yet been marked by the same search. Thus every node communicates one message for every search it is visited in: this is just the total number of shortcuts, and the total amount of communication needed to perform all the searches of PARALLELDIAM is $\widetilde{O}(\alpha)$. Now PARALLELDIAM itself takes $\widetilde{O}(\alpha^{1/2+o(1)})$ parallel depth in total, thus we can simulate it over G_T in $\widetilde{O}(D\alpha^{1/2+o(1)} + \alpha)$ rounds. Thus by Lemma VI.1, it is direct to check (by the results of Section V) that step 2 can be implemented with $\widetilde{O}(D\alpha^{1/2+o(1)} + \alpha)$ rounds.

As G_T now has diameter $\alpha^{1/2+o(1)}$ with high probability by the proof of correctness of algorithm PARALLELDIAM, simulating a BFS and globally broadcasting the results takes $\widetilde{O}(D\alpha^{1/2+o(1)} + \alpha)$ rounds. Summing these contributions gives the desired bound. ∎

Now, we prove Theorem 2.

Proof of Theorem 2: The proof simply involves substituting various values of α into Theorem 7. We handle the cases $D \leq n^{1/4}$ and $D \geq n^{1/4}$ separately.

Case 1: $D \leq n^{1/4}$: We set $\alpha = \sqrt{n}$. This gives us an algorithm that runs in $\widetilde{O}(\sqrt{n} + Dn^{1/4+o(1)})$ rounds. It is easy to check that $Dn^{1/4+o(1)} \leq D^{2/3}n^{1/3+o(1)}$ for $D \leq n^{1/4}$, which completes the argument for this case.

Case 2: $D \geq n^{1/4}$: In this case, we set $\alpha = \frac{n^{2/3}}{D^{2/3}}$. It is easy to check that $D \geq n^{1/4}$ gives that $\alpha \leq \sqrt{n}$. Therefore, the resulting algorithm runs in $\widetilde{O}\left(n/\alpha + D\alpha^{1/2+o(1)}\right) = \widetilde{O}(D^{2/3}n^{1/3+o(1)})$ rounds as desired. ∎

***Keywords**-Distributed Computing; Parallel Computing;Data Structures and Algorithims*

REFERENCES

[1] J. T. Fineman, "Nearly work-efficient parallel algorithm for digraph reachability," in *STOC'18—Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing.* ACM, New York, 2018, pp. 457–470.

[2] T. H. Spencer, "Time-work tradeoffs for parallel algorithms," *J. ACM*, vol. 44, no. 5, pp. 742–778, Sep. 1997. [Online]. Available: http://doi.acm.org/10.1145/265910.265923

[3] J. D. Ullman and M. Yannakakis, "High-probability parallel transitive-closure algorithms," *SIAM J. Comput.*, vol. 20, no. 1, pp. 100–125, 1991. [Online]. Available: https://doi.org/10.1137/0220006

[4] D. 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*, 2014, pp. 565–573. [Online]. Available: https://doi.org/10.1145/2591796.2591850

[5] M. Ghaffari and R. Udwani, "Brief announcement: Distributed single-source reachability," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, 2015, pp. 163–165. [Online]. Available: https://doi.org/10.1145/2767386.2767444

[6] A. Wigderson, "The complexity of graph connectivity," in *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, 1992, pp. 112–132. [Online]. Available: https://doi.org/10.1007/3-540-55808-X_10

[7] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber, "A sublinear space, polynomial time algorithm for directed *s-t* connectivity," *SIAM J. Comput.*, vol. 27, no. 5, pp. 1273–1282, 1998. [Online]. Available: https://doi.org/10.1137/S0097539793283151

[8] A. Abboud and V. V. Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014, pp. 434–443. [Online]. Available: https://doi.org/10.1109/FOCS.2014.53

[9] D. Peleg, *Distributed Computing: A Locality-sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[10] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer, "Distributed verification and hardness of distributed approximation," in *STOC'11—Proceedings of the 43rd ACM Symposium on Theory of Computing*. ACM, New York, 2011, pp. 363–372. [Online]. Available: https://doi.org/10.1145/1993636.1993686

[11] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, 2012, pp. 887–898.

[12] W. Hesse, "Directed graphs requiring large numbers of shortcuts," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '03. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, pp. 665–669. [Online]. Available: http://dl.acm.org/citation.cfm?id=644108.644216

[13] S. Huang and S. Pettie, "Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts," in *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, 2018, pp. 26:1–26:12. [Online]. Available: https://doi.org/10.4230/LIPIcs.SWAT.2018.26

[14] R. Becker, A. Karrenbauer, S. Krinninger, and C. 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*, 2017, pp. 7:1–7:16.

[15] M. Ghaffari and J. Li, "Improved distributed algorithms for exact shortest paths," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, 2018, pp. 431–444.

[16] S. Forster and D. Nanongkai, "A faster distributed single-source shortest paths algorithm," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, 2018, pp. 686–697.

[17] R. Cole, "Correction: "Parallel merge sort" [SIAM J. Comput. **1**7 (1988), no. 4, 770–785; MR0953293 (89m:68015)]," *SIAM J. Comput.*, vol. 22, no. 6, p. 1349, 1993. [Online]. Available: https://doi.org/10.1137/0222081

[18] M. Ghaffari, "Near-optimal scheduling of distributed algorithms," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, 2015, pp. 3–12. [Online]. Available: https://doi.org/10.1145/2767386.2767417