Nearly Work-Efficient Parallel Algorithm for Digraph Reachability

Jeremy T. Fineman
Georgetown University
Washington, District of Columbia, USA
jfineman@cs.georgetown.edu

ABSTRACT

One of the simplest problems on directed graphs is that of identifying the set of vertices reachable from a designated source vertex. This problem can be solved easily sequentially by performing a graph search, but efficient parallel algorithms have eluded researchers for decades. For sparse high-diameter graphs in particular, there is no known work-efficient parallel algorithm with nontrivial parallelism. This amounts to one of the most fundamental open questions in parallel graph algorithms: *Is there a parallel algorithm for digraph reachability with nearly linear work?* This paper shows that the answer is yes.

This paper presents a randomized parallel algorithm for digraph reachability and related problems with expected work $\tilde{O}(m)$ and span $\tilde{O}(n^{2/3})$, and hence parallelism $\tilde{\Omega}(m/n^{2/3}) = \tilde{\Omega}(n^{1/3})$, on any graph with n vertices and m arcs. This is the first parallel algorithm having both nearly linear work and strongly sublinear span, i.e., span $\tilde{O}(n^{1-\epsilon})$ for any constant $\epsilon > 0$. The algorithm can be extended to produce a directed spanning tree, determine whether the graph is acyclic, topologically sort the strongly connected components of the graph, or produce a directed ear decomposition, all with work $\tilde{O}(m)$ and span $\tilde{O}(n^{2/3})$.

The main technical contribution is an *efficient* Monte Carlo algorithm that, through the addition of $\tilde{O}(n)$ shortcuts, reduces the diameter of the graph to $\tilde{O}(n^{2/3})$ with high probability. While both sequential and parallel algorithms are known with those combinatorial properties, even the sequential algorithms are not efficient, having sequential runtime $\Omega(mn^{\Omega(1)})$. This paper presents a surprisingly simple sequential algorithm that achieves the stated diameter reduction and runs in $\tilde{O}(m)$ time. Parallelizing that algorithm yields the main result, but doing so involves overcoming several other challenges.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; **Shared memory algorithms**;

KEYWORDS

Parallel algorithm, randomized algorithm, graph search, reachability, shortcuts

ACM Reference Format:
Jeremy T. Fineman. 2018. Nearly Work-Efficient Parallel Algorithm for Digraph Reachability. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18)*. ACM, New York, NY, USA, 14 pages.
https://doi.org/10.1145/3188745.3188926

1 INTRODUCTION

There are essentially no good parallel algorithms known for the most basic problems on general directed graphs, especially when the graph is sparse. This paper yields several.

A good parallel algorithm should have polynomial parallelism and be (nearly) work efficient. The ***work*** $W(n)$ of a parallel algorithm on a size-n problem is the total number of primitive operations performed. Ideally, the work of the parallel algorithm should be similar to the best sequential running time $T^*(n)$ known for the problem. An algorithm is ***work efficient*** if $W(n) \in O(T^*(n))$ and ***nearly work efficient*** if $W(n) \in \tilde{O}(T^*(n)) = O(T^*(n)\cdot\text{poly}(\log n))$, where $\tilde{O}$ hides logarithmic factors.[1] (As a slight abuse of notation, $\tilde{O}(1)$ is used to mean $O(\text{poly}(\log n))$, where the n should be clear from context.)[2] The ***span*** $S(n)$, also called ***depth***, of a parallel algorithm is the length of the longest chain of sequential dependencies.[3] By Brent's scheduling principle [2], such an algorithm can generally be scheduled to run in $O(W(n)/p)$ time on $p \le W(n)/S(n)$ processors; adding more processors beyond that point does not yield asymptotic speedup. The limit $W(n)/S(n)$ is called the ***parallelism*** of the algorithm; an algorithm is ***moderately parallel*** if the parallelism is $\Omega(n^{\epsilon})$, for some constant $\epsilon > 0$, and ***highly parallel*** if the span is $\tilde{O}(1)$. The goal is to achieve speedup with respect to the best sequential algorithm, which is why work efficiency matters. A nearly work-efficient algorithm runs in $\tilde{O}(T^*(n)/p)$ time on $p \le W(n)/S(n)$ processors, but inefficient algorithms may require enormous numbers of processors to beat the sequential algorithm.

Remark. Aside from the context provided in this introduction and high-level ideas, most of the paper does not require any specific knowledge of parallel algorithms; the challenge lies in producing

[1]In addition to uncluttering the bounds, ignoring logarithmic factors is particularly convenient when comparing parallel algorithms — the precise bounds depend on the specifics of the parallel model, but the bounds typically only vary by logarithmic factors (see [11] for discussion) — allowing us to focus on the high-level discussion.
[2]The standard definition for soft-O is that $f(n) \in \tilde{O}(g(n))$ if $f(n) \in O(g(n)\,\text{poly}(\log g(n)))$. This paper uses $f(n) \in \tilde{O}(g(n))$ to mean $f(n) \in O(g(n)\,\text{poly}(\log n))$, with the only relevant difference being the meaning of $\tilde{O}(1)$.
[3]Older PRAM literature often characterizes algorithms by a number of processors and parallel running time. Span here is generally equivalent to parallel time, and work corresponds to the product of processors and time.

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188926

an algorithm with properties amenable to parallelization. Most implementation details are straightforward, so the parallel model and implementation details are deferred to Section 5.

Problem and history. Perhaps the most basic problem on directed graphs is the single-source reachability problem: given a directed graph $G = (V, E)$ and source vertex $s \in V$, identify the set of vertices reachable by a directed path originating at s. Throughout, let $n = |V|$ be the number of vertices and $m = |E|$ be the number of arcs, and for conciseness assume that $m \in \Omega(n)$. This problem has simple sequential solutions: both breadth-first search (BFS) and depth-first search (DFS) solve the problem in $O(m)$ time. There are two natural parallel algorithms for the reachability problem, which seem to be folklore. See Table 1 for a comparison. Parallel transitive closure [11], which amounts to repeated squaring of the adjacency matrix, is highly parallel but far from work efficient even for dense graphs. Parallel BFS is similar to sequential BFS, except that arcs from each layer (vertices with the same distance) are explored in parallel. Parallel BFS is work efficient (see, e.g., [16]), but the span is proportional to the diameter, which is $\Theta(n)$ in the worst case. Both algorithms fall short of our goals, but they are the state of the art.

The only other progress on general graphs are work/span tradeoffs. Ullman and Yannakakis [22] raised the question over 25 years ago of whether it is possible to solve digraph reachability with sublinear work without sacrificing work efficiency. Instead, their algorithm [22], henceforth termed UY, and Spencer's algorithm [20] exhibit tradeoffs between work and span. Though not originally described in the same terms, both algorithms can be parameterized by a value ρ, $1 \leq \rho \leq n$. Table 1 summarizes the performance bounds.[4] For $\rho = 1$, both algorithms are a parallel BFS. As ρ increases, the span decreases but the work increases. When $\rho = n$, both algorithms converge to transitive closure via regular $\Theta(n^3)$-work matrix multiplication. They differ for intermediate ρ. Spencer's algorithm is deterministic and, for sufficiently dense graphs, can be nearly work efficient with moderate parallelism. In contrast, UY is randomized and never simultaneously work efficient and moderately parallel, but it exhibits a better work/span tradeoff for sparse graphs.

Other work focuses on either restricted graph classes or sequential preprocessing. Kao and Klein [12] give an algorithm for reachability on planar digraphs with $\tilde{O}(n)$ work and $\tilde{O}(1)$ span. Klein [15] gives an algorithm that preprocesses the graph in $O(np)$ sequential time, where $p \geq 1$ is a parameter; after the preprocessing, reachability can be solved in $O(m/p)$ time on p processors.

1.1 Shortcutting Approach and Contributions

The high-level approach is intuitive: (1) reduce the diameter of the graph through the addition of ***shortcuts***, or arcs whose addition does not change the transitive closure of the graph; (2) run parallel BFS on the shortcutted graph. UY [22] fits this general strategy (and parallel BFS and transitive closure are extreme cases), but Spencer's algorithm [20] does not.

The number of shortcuts added is of utmost importance because it corresponds to the work performed during the BFS phase. Specifically if the BFS phase is to complete with $\tilde{O}(m)$ work, then the number of shortcuts must be limited to $\tilde{O}(m)$.

To understand the limits of what could be achieved through this approach, ignore for the now the cost of computing the shortcuts. It is known that $O(n)$ shortcuts are sufficient to achieve $\tilde{O}(\sqrt{n})$ diameter — UY [22] with $\rho = \sqrt{n}$, for example, accomplishes this task. Except for logarithmic factors, this is the best diameter reduction known for general graphs using a linear number of shortcuts. (Better bounds are known for, e.g., planar graphs [21].) Moreover, as Hesse [9] shows, there exists a family of graphs that cannot have their diameter reduced below $\Theta(n^{1/17})$ without adding $\Omega(mn^{1/17})$ shortcuts. In a recent breakthrough, Huang and Pettie [10] show a higher diameter lower bound of $\Omega(n^{1/11})$ when limited to $O(m)$ shortcuts.[5] The main lesson is: if a nearly work-efficient parallel algorithm for digraph reachability uses the shortcutting approach, then its span must be polynomial (specifically at least $\tilde{\Omega}(n^{1/11})$).

The main technical challenge is to produce the shortcuts efficiently, which is a challenge even ignoring parallelism. There is no $\tilde{O}(m)$-time *sequential* algorithm known to reduce every graph's diameter to $\tilde{O}(n^{1-\epsilon})$, for any constant $\epsilon > 0$. For contrast, consider the most natural approach (similar to UY [22]): sample $\sqrt{n}$ vertices, perform a graph search from each, and add shortcuts between all related pairs of samples. It is straightforward to prove that the resulting diameter is $O(\sqrt{n} \log n)$ with high probability, but the running time of the $\sqrt{n}$ independent searches is $O(m\sqrt{n})$.

This paper has the following main contributions:

- (Section 3.) An $\tilde{O}(m)$-time sequential Monte Carlo algorithm that shortens the diameter of any graph to $O(n^{2/3})$, with high probability, through the addition of $\tilde{O}(n)$ shortcuts.
- (Sections 4 and 5.) A Monte Carlo parallel algorithm having $\tilde{O}(m)$ work and $\tilde{O}(n^{2/3})$ span that shortens the diameter of any graph to $O(n^{2/3} \log n)$, with high probability, through the addition of $\tilde{O}(n)$ shortcuts.
- Applying the diameter reduction then parallel BFS yields a Las Vegas algorithm for single-source reachability with $\tilde{O}(m)$ work and $\tilde{O}(n^{2/3})$ span, with high probability.
- (Deferred to full version.) An extension that finds a *directed spanning tree*, i.e., a tree rooted at s containing all vertices reachable from s and using only arcs from G.

Applying existing reductions yields the following Las Vegas randomized parallel algorithms, both with $\tilde{O}(m)$ work and $\tilde{O}(n^{2/3})$ span with high probability:

- An algorithm that identifies and sorts the strongly connected components of the graph. (Use the new reachability algorithm in Schudy's algorithm [19].)
- An algorithm that finds a directed ear decomposition of any strongly connected graph. (Use the new directed spanning tree algorithm with Kao and Klein's algorithm [12].)

[4]The work bound stated by Ullman and Yannakakis [22] is worse, for small ρ, than the bound displayed in Table 1. The table shows the improved bound observed by Schudy [19].

[5]Closing the diameter gap between the $n^{1/11}$ lower bound and $\sqrt{n}$ upper bound is an interesting open question, but it is not addressed by this paper.

	Work	Span	Nearly work efficient?	Number of processors to achieve $\tilde{O}(n/k)$ runtime, for $m \in \Theta(n)$	
parallel BFS	$O(m)$	$\tilde{O}(n)$	Yes	Not Possible unless $k = \tilde{O}(1)$	
parallel Trans. Closure	$\tilde{O}(M(n))$	$\tilde{O}(1)$	No	$kM(n)/n \gg nk$	for $k \leq n$
Spencer's [20]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$	if $\rho = \tilde{O}(\sqrt{m/n})$	k^3	for $k \leq n$
UY [22]*	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$	if $\rho = \tilde{O}(1)$	k^2	for $k \leq n^{2/3}$ †
This paper*	$\tilde{O}(m)$	$\tilde{O}(n^{2/3})$	Yes	k	for $k \leq n^{1/3}$

Table 1: Comparison of parallel algorithms for single-source reachability. Two of the algorithms are parameterized by ρ, $1 \leq \rho \leq n$, which trades off work and span. $M(n)$ is the work of the best highly parallel $n \times n$ matrix multiplication, which is at least the current best sequential time of $O(n^{2.372869})$ [17].
***: the algorithm is randomized. Bounds are with high probability.**
†: for higher k, the dependence on k becomes worse and more complicated to state.

Algorithm 1: Sequential algorithm for shortcutting

```
SeqSC1($G = (V, E)$)
1  if $V = \emptyset$ then return $\emptyset$
2  select a pivot $x \in V$ uniformly at random
3  let $R^+$ denote the set of vertices reachable from $x$
4  let $R^-$ denote the set of vertices that can reach $x$
5  $S := \{(x, v) | v \in R^+\} \cup \{(u, x) | u \in R^-\}$     // add shortcuts
6  $V_F := R^+ \backslash R^-$ ;   $V_B := R^- \backslash R^+$ ;   $V_U := V \backslash (R^+ \cup R^-)$
7  return
   $S \cup$ SeqSC1($G[V_F]$) $\cup$ SeqSC1($G[V_B]$) $\cup$ SeqSC1($G[V_U]$)
```

1.2 Algorithm and Analysis Overview

The sequential algorithm is simple enough that the main subroutine is given immediately. (See also Algorithm 1.) The algorithm is recursive. First select a random vertex x, called the ***pivot***. Perform graph searches forwards and backwards from x to identify subsets R^+ and R^-, respectively. Add shortcuts from R^- to x and from x to R^+. The graph is next partitioned into four subsets of vertices: (i) the vertices reachable in both directions, (ii) the vertices V_F reachable in the forward direction but not the backward direction, (iii) the vertices V_B reachable in the backward direction but not the forward direction, and (iv) the vertices V_U that are unreachable in either direction. Recurse on the subgraphs induced by the three subsets V_F, V_B, and V_U; the vertices reached in both directions are ignored because the shortcuts have already reduced the diameter of that subgraph to 2 hops.

Ignoring the addition of shortcuts, Algorithm 1 is essentially the divide-and-conquer algorithm for topologically sorting the strongly connected components of a graph described by Coppersmith et al. [4]. Their proof thus carries over to prove that this algorithm runs in $O(m \log n)$ sequential time in expectation, but they do not address the diameter problem.

What should be surprising is that Algorithm 1 reduces the graph's diameter, captured by the following lemma. The proof is not obvious and leverages new insights and techniques.

Lemma 1.1. *Let $G = (V, E)$ be a directed graph, and consider any vertices $u, v \in V$ such that there exists a directed path from u to v in G. Let S be the shortcuts produced by an execution of Algorithm 1. Then with probability at least $1/2$ (over random choices in Algorithm 1), there exists a directed path from u to v in $G_S = (V, E \cup S)$ consisting of $O(n^{2/3})$ arcs.*

As a corollary (applying the union bound across at most n^2 related pairs), the union of shortcuts across $\Omega(\log n)$ independent executions of Algorithm 1 is sufficient to reduce the diameter of the graph to $O(n^{2/3})$ with high probability. More precisely, with $2 \lg n + k$ runs, the failure probability is at most $1/2^k$.

Unusual aspects and insight. The analysis focuses on shortcutting a particular path. But unlike most divide-and-conquer analyses, the division step here does not seem to effect progress. Partitioning a graph is good for reducing the problem size (which is what Coppersmith et al. [4] leverage), but it is not good for preserving paths — and once vertices fall in different subproblems, there can be no subsequent shortcuts between them. This feature is likely why previous algorithms, such as UY [22], perform independent searches on the original graph.

A key insight in the analysis is that the partitioning step also reduces by a constant factor the number of vertices that could cause the path to split again later. In doing so, the probability of splitting the path goes down, and hence the probability of shortcutting it goes up. The end effect is that the path is likely to be significantly shortcutted before it is divided into too many pieces.

The proof of this filtering insight (Lemma 3.4) leverages anti-symmetric relationships between certain vertices. Interestingly, the lack of symmetry in directed graphs is exactly the feature that makes good parallel algorithms for digraphs so elusive, but here asymmetry is crucial to the proof.

Building a parallel algorithm. The main obstacle to parallelizing Algorithm 1 is the graph searches employed to find R^+ and R^-. In fact, these searches are exactly the single-source reachability problem that we want to solve. The obvious solution to try is to instead limit the searches to a distance of $\tilde{O}(n^{2/3})$, but unfortunately doing so causes other problems. The parallel algorithm and the analysis are thus more involved. Section 4 provides a sequential algorithm with distance-limited searches. Given that, the parallel implementation (discussed briefly in Section 5) is straightforward.

2 PRELIMINARIES

This section provides definitions, notations, and the main probabilistic tools used throughout.

The subgraph of $G = (V, E)$ induced by vertices $V' \subseteq V$ is denoted by $G[V']$.

If there is a directed path (possibly empty) from u to v in digraph $G = (V, E)$, then u ***precedes*** v and v ***succeeds*** u, denoted $u \preceq v$. We say also that u ***can reach*** v and that v ***can be reached by*** u. If $u \preceq v$ and/or $v \preceq u$, then u and v are ***related***; otherwise they are ***unrelated***. The ***successors*** or ***forward reach of*** x is the set of nodes $R^+(G, x) = \{v | x \preceq v\}$. The ***predecessors*** or ***backwards reach of*** x is the set $R^-(G, x) = \{u | u \preceq x\}$.

A ***shortcut*** is any arc (u, v) such that $u \preceq v$ in G.

Paths and nonstandard notation. The analysis considers paths as well as the relationships between paths and vertices. A path $P = \langle v_0, v_1, \ldots, v_\ell \rangle$ is denoted by the sequence of its constituent vertices, with the arcs between consecutive pairs implied. For convenience, an path may be empty. The ***length*** of the path, denoted $length(P)$, is the number of arcs. For given path P, $length(P) = \ell$. An empty path and a path comprising a single vertex both have length 0. ***Splitting the path*** P ***into*** k ***pieces*** means partitioning it into k subpaths $\langle v_0, \ldots, v_{i_1} \rangle, \langle v_{i_1+1}, \ldots, v_{i_2} \rangle, \ldots, \left\langle v_{i_{k-1}+1}, \ldots, v_\ell \right\rangle$, where $0 \leq i_1 < \cdots < i_{k-1} < \ell$.

A vertex x and a path P can be compared in the following ways. The vertex x is a ***bridge of*** P if x can reach and can be reached by vertices on the path, i.e., if there exists $v_i, v_j \in P$ such that $v_i \preceq x$ and $x \preceq v_j$. Note that every vertex on the path is a bridge. A vertex x is an ***ancestor of*** P if x can reach some vertex on the path, but x cannot be reached by any vertex on the path. Similarly, x is a ***descendant of*** P if x can be reached by some vertex on the path, but x cannot reach any vertex on the path. The set of all bridges, ancestors, and descendants of P are denoted $Bridge(G, P)$, $Anc(G, P)$, and $Desc(G, P)$, respectively. Note that these sets are all disjoint by definition. If a vertex x is a bridge, ancestor, or descendant of the path P, then x and P are ***related***. Otherwise, they are ***unrelated***.

Tools. The analysis employs one relatively uncommon probabilistic tool — a special case of Karp's [13] probabilistic recurrence relations, restated next. Roughly speaking, this theorem relates two processes: (1) a random process where in each round the problem "size" (Φ in the theorem) reduces by a constant factor in expectation, and (2) a deterministic process where the problem size reduces by exactly that constant factor. The theorem says that if the random process uses a few extra rounds, it is very likely to experience at least the size reduction of the deterministic process.

Theorem 2.1 (Restatement of special case of Theorem 1.3[6] in [13]). *Consider a random process of the following form. Let $\mathcal{I}$ denote the set of all problem instances, and let $I_0 \in \mathcal{I}$ denote the initial problem instance. In the rth round, the process makes random choices and transforms the instance from I_{r-1} to I_r (a random variable). Let $\Phi : \mathcal{I} \to \mathbb{R}$ be any function satisfying $0 \leq \Phi(I_r) \leq \Phi(I_{r-1})$ for all relevant $r \geq 1$ and all feasible sequences $I_0, I_1, I_2, \ldots$ of instance outcomes.*

Suppose there exists some constant $p < 1$ such that for all instances $E[\Phi(I_r) | I_0, I_1, \ldots, I_{r-1}] \leq p \cdot \Phi(I_{r-1})$, and consider any integers $k \geq 0$ and $w \geq 0$. Then $\Pr\left\{\Phi(I_{k+w+2}) > p^k \cdot \Phi(I_0)\right\} \leq p^w$.

[6]Karp states the theorem very differently. The process described here corresponds to his recurrence $T(I) = a(\Phi(I)) + T(h(I))$, where $a(x) = 0$, $x < d$ and $a(x) = 1$, $x \geq d$, for $d = p^k \cdot \Phi(I_0)$. This recurrence counts the number of steps to reach the target size. (Note that d depends only on the initial instance and is constant in the recurrence.) The deterministic counterpart is $\tau(x) = a(x) + \tau(px)$, which has solution $u(\Phi(I_0)) = \lceil \log_{1/p}(\Phi(I_0)/d) \rceil \leq k + 1$.

3 SEQUENTIAL DIAMETER REDUCTION

This section focuses on proving the following theorem. The unmodified G is used to refer to subgraphs $G = (V, E)$. When the original input graph is intended, $\hat{G}$ is employed instead. Throughout, x denotes the pivot, and the vertex sets V_F (forward only), V_B (backward only), and V_U (unrelated) are used as setup in Algorithm 1.

Theorem 3.1. *There exists a randomized sequential algorithm that takes as input a directed graph $\hat{G} = (\hat{V}, \hat{E})$ and failure parameter $\gamma \geq 1$ with the following guarantees, where $n = \left|\hat{V}\right|$, $m = \left|\hat{E}\right|$, and without loss of generality $m \geq n/2$: (1) the running time is $O(\gamma m \log^2 n)$, (2) the algorithm produces a size-$O(\gamma n \log^2 n)$ set S^* of shortcuts, and (3) with probability at least $1 - 1/n^\gamma$, the diameter of $G_{S^*} = (V, E \cup S^*)$ is $O(n^{2/3})$.*

As mentioned in Section 1, the algorithm entails taking the union of shortcuts from $\Theta(\log n)$ runs of Algorithm 1. To make the running time worst case, there will be one minor modification introduced later: namely, an extra base case to truncate the recursion.

Sections 3.1 and 3.2 set up the main ideas for proof of Lemma 1.1 but instead prove a weaker distance bound of $O(n^{1/\lg(8/3)}) = O(n^{0.7067})$. Section 3.3 tightens the distance bound to $O(n^{2/3})$, thereby proving Lemma 1.1. It is worth emphasizing that Sections 3.2 and 3.3 use exactly the same algorithm — the only difference is the details of the analysis. Finally, Section 3.4 completes the proof of Theorem 3.1 by analyzing the running time and number of shortcuts.

3.1 Setup of the Analysis

Fix any simple path $\hat{P} = \langle v_0, \ldots, v_\ell \rangle$ in the graph up front. By partitioning the graph, each call to `SeqSC1` also splits the path into subpaths. The analysis tracks a collection of calls whose subgraphs contain subpaths of $\hat{P}$.

More precisely, a ***path-relevant subproblem***, denoted by pair (G, P), corresponds to a call `SeqSC1`(G) and an associated nonempty subpath P of $\hat{P}$ to shortcut. The starting subproblem is $(\hat{G}, \hat{P})$. The path-relevant subproblems are the subproblems for which $G \cap \hat{P} \neq \emptyset$, except that the base case occurs when a subpath P is shortcutted to two hops — all recursive subproblems arising beyond that point are *not* path relevant. The following lemma characterizes the path-relevant subproblems that arise when executing the call `SeqSC1`(G) with associated path P.

It is worth emphasizing that the algorithm has no knowledge of the path P; associating the subpath with the subproblem is an analysis tool only.

Lemma 3.2. *Let $P = \langle v_0, \ldots, v_\ell \rangle$ be a nonempty path in $G = (V, E)$, and consider the effect of a single call `SeqSC1`(G) in Algorithm 1. The following are the outcomes depending on pivot x:*

1. *(Base case.) If x is a bridge of P, then the shortcuts (v_0, x) and (x, v_ℓ) are created. There are no path-relevant subproblems.*

(2) *If x and P are unrelated, then P is entirely contained in $G[V_U]$; the one path-relevant subproblem is thus $(G[V_U], P)$.*
(3) *If x is an ancestor of P, then there exists some $v_k \in P$ such that $P_1 = \langle v_0, \dots, v_{k-1}\rangle$ is fully contained in $G[V_U]$ and $P_2 = \langle v_k, \dots, v_\ell\rangle$ is fully contained in $G[V_F]$. There are thus at most two path relevant subproblems: if P_1 is nonempty, $(G[V_U], P_1)$ is path relevant; if P_2 is nonempty, $(G[V_F], P_2)$ is path relevant.*
(4) *If x is a descendant of P, then there exists some $v_k \in P$ such that $\langle v_0, \dots, v_k\rangle$ is fully contained in $G[V_B]$ and $\langle v_{k+1}, \dots, v_\ell\rangle$ is fully contained in $G[V_U]$. This case gives rise to at most two path-relevant subproblems, as above.*

Proof. The proof follows from the definitions. Consider for example the last case, that x is a descendant of P. Here v_k is the latest vertex on the path such that $v_k \preceq x$. Then by transitivity, $v_i \preceq v_k \preceq x$ for all $i \le k$. Thus, P_1 is entirely contained in V_B. All v_j with $j > k$ are unrelated to x and hence in V_U. □

Cases 3 and 4 seem like bad cases because the number of path-relevant subproblems, and hence unshortcutted arcs in the final path, increases. Section 3.2 argues that these cases do make progress.

The path-relevant subproblems that arise during the execution of the algorithm induce a ***path-relevant subproblem tree***, where each node s corresponds to a call of SeqSC1 on some path-relevant subproblem $s = (G, P)$. For the analysis, it is convenient to consider the ***flattened path-relevant tree***, where each node corresponding to Case 2 in Lemma 3.2 is merged with its only child. Viewed algorithmically, a node in the flattened path-relevant tree corresponds to interpreting the algorithm as sampling multiple pivots x (and discarding some of the graph) until finally getting one that is related to the path P.

The analysis considers levels in the flattened path-relevant tree in aggregate. The point is to later fit the analysis to Theorem 2.1. Specifically, the analysis consists of a sequence of rounds, where the instance I_r in round r is the collection of subproblems defined by the nodes at depth r in the flattened path-relevant tree. We have the following lemma immediately. All that remains is bounding the remaining subpath lengths (Section 3.2).

Lemma 3.3. *Consider any graph $\hat{G} = (\hat{V}, \hat{E})$ and any path $\hat{P}$ from u to v. Consider an execution of Algorithm 1, let S be the shortcuts produced, and let $\{(G_1, P_1), \dots, (G_k, P_k)\}$ denote the set of path-relevant subproblems at level/depth r in the flattened path-relevant tree. Then there is a u-to-v path in $G_S = (\hat{V}, \hat{E} \cup S)$ of length at most $2^r + 2^{r-1} + \sum_{i=1}^k length(P_i)$.*

Proof. Let L_i denote the set of paths associated with leaves in the tree at depth i. Then a simple induction over levels proves that: the set of paths $\{P_1, \dots, P_k\} \cup \left(\bigcup_{i=1}^{r-1} L_i\right)$ constitute a splitting (partition) of path $\hat{P}$. (To perform the inductive step, apply Lemma 3.2 at each internal node.)

It remains to bound the path-length in G_S by positing a specific path: the concatenation of the shortcutted paths for the leaves and the full unshortcutted paths for the remaining subproblems. Each concatenation adds 1 arc, each leaf's path uses 2 shortcuts, and each remaining non-leaf path P_i has $length(P_i)$ arcs. Since the degree of each node is at most 2 (Lemma 3.2), the number of leaves above level r is at most 2^{r-1}, and the number of internal nodes (concatenations) above level r is also is at most 2^{r-1}. Adding everything together gives the bound. □

3.2 Asymmetry Leads to Progress

This section proves that with probability at least 1/2, the distance between u and v is at most $O(n^{1/\log(8/3)})$. The main tools are Theorem 2.1 and a proof that the number of path-related vertices decreases by a constant fraction, on average, with each level in the flattened path-relevant tree. More precisely, a vertex v is ***path active at level*** r if (1) v is part of some path-relevant subproblem at level r in the flattened tree, and (2) v is related to the path in that subproblem. The goal is to argue that the expected number of path-active vertices decreases with each level.

Recall that the each node in the flattened tree corresponds to sampling multiple pivots until finally drawing a pivot x that is path related. The analysis focuses on this path-related choice of x. Instead of reasoning about x as being drawn uniformly at random from path-related vertices, instead consider the following equivalent process for selecting x. First, toss a weighted coin to determine whether x is a bridge, ancestor, or descendant. Second, choose the specific pivot vertex from within the selected set uniformly at random.

The following lemma considers the effect of choosing x uniformly from all path ancestors. Choosing from path descendants is symmetric.

Lemma 3.4. *Consider any subproblem (G, P). Suppose that x is drawn uniformly at random from $Anc(G, P)$, let $\alpha = |Anc(G, P)|$, and let α' be denote the number of vertices in $Anc(G, P)$ that remain path active after recursing. Then $E[\alpha' | x \in Anc(G, P)] < \alpha/2$.*

Proof. Define the following binary relation over vertices in $Anc(G, P)$: u ***preserves*** u' means that if $x = u$, then u' remains path active. The relation is irreflexive by virtue of the fact that $x \in (R^-(G, x) \cap R^+(G, x))$ and hence not contained in any subproblems. The goal is to prove that it is also antisymmetric. Assuming the asymmetry, the total number of pairs satisfying the preserves relation is at most $\binom{\alpha}{2}$. The number of vertices preserved by x is denoted by α', and hence $E[\alpha'] \le \binom{\alpha}{2}/\alpha = (\alpha - 1)/2$. It remains only to prove that the preserves relation is antisymmetric.

Let $P = \langle v_0, v_1, \dots, v_\ell\rangle$. Consider any ancestor $u \in Anc(G, P)$ and let v_k be the earliest vertex in P such that $u \preceq v_k$. Similarly, consider any other ancestor vertex $u' \neq u$ and let $v_{k'}$ be its earliest related vertex in P.

The main claim is the following: u preserves u' only if one of the following holds:

- $u \prec u'$ and $u' \not\prec u$, or
- u and u' are unrelated and $k' < k$.

This claim alone directly implies the antisymmetry. Specifically, if u and u' are related, then they can only preserve each other in one direction. If u and u' are unrelated, the inequality is strict so preservation can also only be in at most one direction.

To prove the claim, consider the case that u is chosen as pivot, and let V_B, V_F, and V_U, denote the backward, forward, and unrelated vertex sets, respectively, as defined in Algorithm 1. For u to preserve u', u' must be active in its subproblem, i.e., u' must be in a recursive subproblem that contains a subpath, and u' must be related to that subpath. Since u is an ancestor of the path, none of the path falls in V_B. Thus, for u to preserve u', at least one of the following must be true: (1) $u' \in V_F$, or (2) $u' \in V_U$ and u' is related to P_1. The first condition directly implies $u \prec u'$. As for the second, Lemma 3.2 states that $P_1 = \langle v_0, \ldots, v_{k-1} \rangle$ falls in V_U, so u' related to P_1 implies $k' \leq k - 1 < k$. □

Lemma 3.4 states that if an ancestor is selected as pivot, the number of path-active ancestors decreases by half in expectation. The following lemma extends that reduction to the total number of path-active vertices. The worst case is that the number of ancestors equals the number of descendants, in which case the analysis is tight (to within additive constants).

Lemma 3.5. *Let η denote the total number of path-active vertices in some level-$(r-1)$ subproblem (G, P), and let η' be a random variable denoting the number of those vertices that are path active at level-r. Then $E[\eta'] < (3/4)\eta$.*

Proof. The analysis considers the following steps, which may afford the adversary more power. (1) To model any unrelated pivots, vertices and arcs may be removed adversarially. This step only reduces η further so is ignored. (2) A path-related pivot is selected uniformly at random, first by determining whether the pivot is an ancestor, bridge, or descendant, then by choosing uniformly at random from that set. If a bridge is selected, there are no path-related subproblems, so $\eta' = 0$. Otherwise, filtering occurs as per Lemma 3.4. This step is analyzed in more detail below.

At the start of step (2), let α, β, and δ denote the number of ancestors, bridges, and descendants, respectively, of path P in G, with $\alpha + \beta + \delta = \eta$. Scaling by the probabilities of selecting ancestors or descendants, we have:

$$\begin{aligned}
E[\eta'] &= \left(\frac{\alpha}{\eta}\right) E[\eta' | x \in Anc(G,P)] + \left(\frac{\delta}{\eta}\right) \cdot E[\eta' | x \in Desc(G,P)] \\
&< \left(\frac{\alpha}{\eta}\right)(\alpha/2 + \beta + \delta) + \left(\frac{\delta}{\eta}\right)(\alpha + \beta + \delta/2) \qquad \text{(Lemma 3.4)} \\
&= \frac{(\alpha+\delta)(\alpha/2 + \beta + \delta/2)}{\eta} + \frac{\alpha\delta}{\eta} \\
&= \frac{(\eta - \beta)(\eta + \beta)}{2\eta} + \frac{(\sqrt{\alpha\delta})^2}{\eta} \qquad (\eta = \alpha + \beta + \delta) \\
&\leq \frac{\eta^2}{2\eta} + \frac{((\alpha+\delta)/2)^2}{\eta} \qquad \text{(AM-GM inequality)} \\
&\leq (3/4)\eta\ .
\end{aligned}$$

□

For subproblem $s = (G, P)$, define $\phi(s)$ to be the number of path-active vertices in s. Define $\Phi(I_r) = \sum_{s \in I_r} \phi(s)$, where I_r is the collection of subproblems at level-r in the flattened tree. Then we have the following. Applying Theorem 2.1 then gives the main lemma.

Corollary 3.6. *For every possible collection I_{r-1} of subproblems, $E[\Phi(I_r) | I_{r-1}] \leq (3/4)\Phi(I_{r-1})$.*

Proof. Lemma 3.5 states that for each $s \in I_{r-1}$, we have $E[\phi(s_1) + \phi(s_2)] \leq (3/4)\phi(s)$, where s_1 and s_2 are random variables for the (at most) two path-relevant subproblems of s. The claim follows by linearity of expectation over all s. □

Lemma 3.7. *Let $\hat{G} = (\hat{V}, \hat{E})$ be a directed graph, and consider any vertices $u, v \in V$ such that there exists a directed path from u to v in $\hat{G}$. Let S be the shortcuts produced by an execution of Algorithm 1 and let $n = \left|\hat{V}\right|$. Then with probability at least $1/2$, there exists a directed path from u to v in $G_S = (\hat{V}, \hat{E} \cup S)$ consisting of $O(n^{1/\lg(8/3)})$ arcs.*

Proof. Choose an arbitrary simple path $\hat{P}$ from u to v in $\hat{G}$. At most every vertex is path active, so $\Phi(I_0) \leq n$. By Theorem 2.1 with Corollary 3.6, $\Pr\{\Phi(I_{r+5}) > (3/4)^r n\} < 1/2$. Observe that $\Phi(I_{r+5})$ is at least the number of bridge nodes that are still active in round $r+5$, and each node on an active subpath is a bridge node. Thus, by Lemma 3.3, running the algorithm to level $r+5$ is enough to yield a shortcutted path length of at most $O(2^r) + \Phi(I_{r+5}) \leq O(2^r) + (3/4)^r n$, with probability at least $1/2$. Setting both terms equal and solving for r gives $r = \log_{8/3} n$. Thus, with probability at least $1/2$, the shortcutted path has length $O(2^r) = O(2^{\log_{8/3} n}) = O(n^{1/\lg(8/3)})$. □

3.3 A Tighter Path-Length Bound (Lemma 1.1)

This section tightens the path-length bound to $O(n^{2/3})$, thereby proving Lemma 1.1.

The main difference versus Section 3.2 is a better potential function associated with subproblems. The 3/4 bound reduction in the number of path-active vertices, as stated in Lemma 3.5, is indeed tight in the worst case. But the worst case only occurs when the number of ancestors is equal to the number of descendants. When there is imbalance between the two, the reduction is better. Consider, for example, the extreme that there are no descendants — then the number of path active vertices reduces by 1/2 according to Lemma 3.4.

It turns out that leveraging the numbers of ancestors α, bridges β, and descendants δ is useful, but there is no requirement that potential merely take the sum of these three terms as in Lemma 3.5. In general, the potential function may be any function of the terms. In particular, this section defines a potential function ϕ on subproblems as follows:

$$\phi(s) = \begin{cases} 0 & \text{if } s \text{ is not path-relevant} \\ \psi(\alpha, \beta, \delta) & \text{otherwise} \end{cases}, \qquad (1)$$

where $\alpha = |Anc(G,P)|$, $\beta = |Bridge(G,P)|$, $\delta = |Desc(G,P)|$, and ψ is a function that obeys certain properties defined next.

Definition 3.8. Let $\psi : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be any function mapping three nonnegative real numbers to a nonnegative real number. The function ψ is ***well-behaved*** if the following apply:

(i) (Converting bridges to ancestors/descendants only helps): $\psi(\alpha + k_1, \beta, \delta + k_2) \leq \psi(\alpha, \beta + k_1 + k_2, \delta)$, for all $k_1, k_2 \geq 0$.
(ii) (Bridges are a lower bound): $\psi(\alpha, \beta, \delta) \geq \beta$.
(iii) (Monotonicity): $\psi(\alpha', \beta', \delta') \leq \psi(\alpha, \beta, \delta)$, for all $\alpha' \leq \alpha$, $\beta' \leq \beta$, and $\delta' \leq \delta$.
(iv) (Splittable): For $\beta_1 > 0$ and $\beta_2 > 0$ (both strictly positive), $\psi(\alpha_1, \beta_1, \delta_1) + \psi(\alpha_2, \beta_2, \delta_2) \leq \psi(\alpha_1 + \alpha_2, \beta_1 + \beta_2, \delta_1 + \delta_2)$.
(v) (Concavity): treating β and δ as constant, the resulting univariate function with respect to variable α is concave. Similarly, treating α and β as constant, the function on δ is concave.

A well-behaved function ψ is ***c*-reducing**, for constant $0 < c < 1$, if the following holds for all $\eta = \alpha + \beta + \delta > 0$:

$$(\alpha/\eta) \cdot \psi(\alpha/2, \beta, \delta) + (\delta/\eta) \cdot \psi(\alpha, \beta, \delta/2) \le c \cdot \psi(\alpha, \beta, \delta) \ .$$

Finally, a well-behaved function ψ has ***σ*-overhead**, for $\sigma \ge 1$, if $\psi(\alpha, \beta, \delta) \le \sigma \cdot \eta$.

The function $\psi(\alpha, \beta, \delta) = \alpha + \beta + \delta$ is well-behaved with overhead 1. As Lemma 3.5 shows, it is also (3/4)-reducing. The goal of this section is to first extend the analysis to any c-reducing well-behaved function, and then to show that there exists a function with $c < 3/4$.

Lemma 3.9. *Suppose that there exists a c-reducing well-behaved ψ, and define ϕ as in Equation 1. Consider any path-relevant subproblem $s = (G, P)$, and let s_1 and s_2 be random variables denoting any child path-relevant subproblems. Then $E[\phi(s_1) + \phi(s_2)] \le c \cdot \phi(s)$.*

Proof. The outline of the proof is the same as that of Lemma 3.5. (1) Allow the adversary to arbitrarily remove any vertices or arcs. By Definition 3.8-iii, removing path-related vertices entirely only reduces the potential. By Definition 3.8-i, removing relationships that thereby convert a bridge to an ancestor or descendant also can only reduce the potential. (2) A path-related pivot is selected uniformly at random, first by determining whether the pivot is an ancestor, bridge, or descendant, then by choosing uniformly from that set. If a bridge is selected, there are no path-related subproblems, so the resulting potential is 0. Otherwise, consider the expected reduction to the number of path-active ancestors or descendants and that impact on the potential. This step is analyzed in more detail below. (3) Adversarially divide the path-active vertices across up to two path-relevant subproblems. To be path-relevant, there must be at least one vertex on the path and hence at least one bridge. Thus Definition 3.8-iv can be applied, and any partitioning only reduces the potential further.

The remainder of the proof thus focuses on step (2). When the path-related pivot is selected, let $A = Anc(G, P)$ and $\alpha = |A|$, let $B = Bridge(G, P)$ and $\beta = |B|$, and let $D = Desc(G, P)$ and $\delta = |D|$. Let $\eta = \alpha + \beta + \delta$. Let α', β', and δ' be random variables denoting the number of vertices in A, B, and D, respectively, that are also path-active in any child subproblems.

$$\begin{aligned}
&E[\psi(\alpha', \beta', \delta')] \\
&\quad = \left(\frac{\alpha}{\eta}\right) E[\psi(\alpha', \beta', \delta') | x \in A] + \left(\frac{\delta}{\eta}\right) E[\psi(\alpha', \beta', \delta') | x \in D] \\
&\quad \le \left(\frac{\alpha}{\eta}\right) E\left[\psi(\alpha', \beta, \delta) | x \in A\right] + \left(\frac{\delta}{\eta}\right) E[\psi(\alpha, \beta, \delta') | x \in D] \\
&\qquad\qquad \text{(by Definitions 3.8-iii and 3.8-i)} \\
&\quad \le \left(\frac{\alpha}{\eta}\right) \psi(E[\alpha'], \beta, \delta) + \left(\frac{\delta}{\eta}\right) \psi(\alpha, \beta, E[\delta']) \\
&\qquad\qquad \text{(by concavity, i.e., Definition 3.8-v, and Jensen's inequality)} \\
&\quad \le \left(\frac{\alpha}{\eta}\right) \psi(\alpha/2, \beta, \delta) + \left(\frac{\delta}{\eta}\right) \psi(\alpha, \beta, \delta/2) \\
&\qquad\qquad \text{(by Lemma 3.4 and Definition 3.8-iii)} \\
&\quad \le c \cdot \psi(\alpha, \beta, \delta) \qquad \text{(by definition of c-reducing)}
\end{aligned}$$

To complete the proof, as already noted $\phi(s_1) + \phi(s_2) \le \psi(\alpha', \beta', \delta')$ by Definition 3.8-iv. □

As before, define $\Phi(I_r) = \sum_{s \in I_r} \phi(s)$, where I_r is the collection of subproblems at level-r in the flattened tree. Linearity of expectation yields the following:

Corollary 3.10. *Suppose that there exists a c-reducing well-behaved function ψ. Then given any collection I_{r-1} of subproblems, $E[\Phi(I_r) | I_{r-1}] \le c \cdot \Phi(I_{r-1})$.*

The following lemma, analogous to Lemma 3.7, completes the argument. Note that the c-reducing function ψ is used only in the analysis, so exhibiting a better function automatically strengthens the bound.

Lemma 3.11. *Suppose that there exists some c-reducing well-behaved function ψ, for constant c, with overhead σ.*

Let $\hat{G} = (\hat{V}, \hat{E})$ be a directed graph, and consider any vertices $u, v \in V$ such that there exists a directed path from u to v in $\hat{G}$. Let S be the shortcuts produced by an execution of Algorithm 1 and let $n = |\hat{V}|$. Then with probability at least 1/2, there exists a directed path from u to v in $G_S = (\hat{V}, \hat{E} \cup S)$ consisting of $O((\sigma n)^{1/\lg(2/c)})$ arcs.

Proof. The proof is similar to that of Lemma 3.7. Choose an arbitrary simple path $\hat{P}$ from u to v in $\hat{G}$. Let I_r be the collection of path-relevant subproblems at level-r in the flattened tree. At most every vertex is path active, so $\alpha + \beta + \delta \le n$. Since the function has overhead σ, it follows that $\Phi(I_0) = \psi(\alpha, \beta, \delta) \le \sigma n$.

Let $r' = r + \log_c(1/2) + 2 = r + \Theta(1)$. By Theorem 2.1 and Corollary 3.10, $\Pr\{\Phi(I_{r'}) > c^r \sigma n\} < 1/2$. That is, with probability $\ge 1/2$: $c^r \sigma n > \Phi(I_{r'}) = \sum_{s=(G,P) \in I_{r'}} \phi(s) \ge \sum_{s=(G,P) \in I_{r'}} |Bridge(G, P)|$, where the last inequality follows from Definition 3.8-ii.

Thus, by Lemma 3.3, running the algorithm to level r' is enough to yield a shortcutted path of length at most $O(2^{r'}) + \Phi(I_{r'}) \le O(2^r) + c^r \sigma n$ with probability at least 1/2. Setting both terms equal and solving for r gives $r = \log_{2/c}(\sigma n)$. Substituting back, with probability at least 1/2, the path length is $O((\sigma n)^{1/\lg(2/c)})$. □

3.3.1 A Better c-Reducing Function, and Proof of Lemma 1.1. The main idea of the potential is to capture any local imbalance between ancestors and descendants. A good choice of function is

$$\psi(\alpha, \beta, \delta) = \sqrt{(\alpha + \beta)(\delta + \beta)} \ ,$$

which captures imbalance through a geometric mean. The inclusion of β in both the α and β terms is primarily meant to capture both the constraint that converting a bridge to an ancestor/descendant does not increase the potential, and the constraint that $\psi(\alpha, \beta, \delta) \ge \beta$.

The remaining goal is to show that the function is well-behaved and $(1/\sqrt{2})$-reducing. As long as that is true, Lemma 3.11 directly implies Lemma 1.1, because $1/\lg(2/(1/\sqrt{2})) = 2/3$. Proof of the following lemma is deferred to the full version of the paper.

Lemma 3.12. *The function $\psi(\alpha, \beta, \delta) = \sqrt{(\alpha + \beta)(\delta + \beta)}$ is a $(1/\sqrt{2})$-reducing, well-behaved function with overhead 1.*

3.4 Runtime and Number of Shortcuts

This section completes the proof of Theorem 3.1 by analyzing the running time and number of shortcuts added. As stated, however,

Algorithm 2: Modified sequential algorithm for shortcutting

```
SeqSC2($G = (V, E)$)
1  if the recursion depth is $\lg n$ then return $\emptyset$
2  $S := \emptyset$
3  while $V \neq \emptyset$ do
4      select a vertex $x \in V$ uniformly at random
5      $R^+ := R^+(G, x)$
6      $R^- := R^-(G, x)$
7      $S := S \cup \{(x, v) | v \in R^+\} \cup \{(u, x) | u \in R^-\}$ // shortcuts
8      $V_F := R^+ \backslash R^-$ ;   $V_B := R^- \backslash R^+$ ;   $V_U := V \backslash (R^+ \cup R^-)$
9      $S := S \cup$ SeqSC2($G[V_F]$) $\cup$ SeqSC2($G[V_B]$)
10     $G := G[V_U]$
11 return $S$
```

the running time of Algorithm 1 is not worst case, so it does not meet the promise of a Monte Carlo algorithm.

This section instead analyzes Algorithm 2. Algorithm 2 is obtained from Algorithm 1 by replacing one of the recursive calls (specifically SeqSC1 ($G[V_U]$)) with a loop. There is also a new base case after $\lg n$ levels of recursion to make the bounds worst case, where (as always) n here refers to the number of vertices in the original graph $\hat{G}$. Aside from this one change, Algorithm 1 and Algorithm 2 are equivalent.

The following lemma indicates that the main path-length lemmas (e.g., Lemma 3.11) still hold even with the truncated execution. More precisely, proof of those lemmas only relies on the execution reaching a depth much less than $\lg n$ in the flattened path-relevant tree.

Lemma 3.13. *Consider an execution of Algorithm 1 and the corresponding flattened path-relevant tree. When mapped to an execution of Algorithm 2 with the same random choices, the first $\lg n - 1$ levels of the flattened tree all have recursion depth $< \lg n$ in Algorithm 2.*

Proof. The flattened tree only merges some of the calls corresponding to $G[V_U]$. Algorithm 2 merges all such nodes, which can only reduce the depth of nodes further. □

The next lemmas bound the number of shortcuts and running time.

Lemma 3.14. *Consider a graph $\hat{G} = (\hat{V}, \hat{E})$, and let $n = \left|\hat{V}\right|$. Each execution of Algorithm 2 creates $O(n \log n)$ shortcuts.*

Proof. Consider a call to SeqSC2(G) on $G = (V, E)$. Each shortcut added removes a vertex: if, e.g., (x, v) is created, then either $v \in V_B$ or $v \in V_F$, both of which sets are removed from G at the end of the iteration. Thus, there can be at most $|V|$ arcs added.

There are potentially many recursive subproblems, but by the same argument they are all disjoint subgraphs. Thus, the total number of arcs added at each level of recursion is $O(n)$. There are $O(\lg n)$ levels by construction, which completes the proof. □

Lemma 3.15. *Consider a graph $\hat{G} = (\hat{V}, \hat{E})$, and let $n = \left|\hat{V}\right|$ and $m = \left|\hat{E}\right|$. Algorithm 2 can be implemented to run in $O(m \log n)$ time.*

Proof. Proof is similar to Lemma 3.14, getting $O(m)$ total time at each level of recursion, assuming that the call SeqSC2(G) can be made to run in $O(|V| + |E|)$ time. Given a pivot $x \in V$, it is straightforward to implement each search, and build the induced subgraphs, to run in time $O(a)$ where a is the number of arcs explored. Each arc is only explored by one search in each direction, so the total number of arcs visited is $O(|E|)$. Finally, sampling vertices can be achieved by randomly permuting the vertices up front, iterating over that list, and checking whether the vertex has already been visited by a search. This takes a total of $O(|V|)$ time. □

Proof of Theorem 3.1. The full algorithm consists of $(2 + \gamma) \lg n$ independent runs of Algorithm 2. For each related pair $u \preceq v$, each run has probability $\geq 1/2$ of reducing the distance between those vertices to $O(n^{2/3})$ by Lemma 1.1. Thus, the probability that all runs fail is at most $1/2^{(2+\gamma)\lg n} = 1/(n^2 n^\gamma)$. Since there are at most n^2 related pairs, a union bound across runs gives a failure probability of $1/n^\gamma$ for the overall diameter. The running time and number of shortcuts are obtained by multiplying the bounds from Lemmas 3.15 and 3.14 by the $\Theta(\gamma \log n)$ runs. □

4 AN ALGORITHM WITH DISTANCE-LIMITED SEARCHES

This section presents a modified algorithm that is more amenable to being parallelized. For now, this algorithm can be viewed as a sequential algorithm — discussion of the parallel implementation is deferred to Section 5 and the full version of the paper. The main ideas are guided by certain sequential bottlenecks. As in Section 3, $\hat{G} = (\hat{V}, \hat{E})$ and $n = \left|\hat{V}\right|$ are used only to refer to the original graph.

There are two main obstacles to parallelizing Algorithm 2, but the first is more serious. Finding the set $R^-(G, x)$ or $R^+(G, x)$ entails a graph search, which can have linear span in a high-diameter graph. The solution for this problem is to modify the algorithm to use a ***D*-limited BFS**, returning only the vertices within D hops of the source x, but doing so introduces some other difficulties. This section thus focuses on modifying the algorithm to work with distance-limited searches for appropriate distance D.

The second obstacle is best exhibited by the loop in Algorithm 2. If there are no arcs in the graph, for example, the loop requires $\Omega(n)$ iterations. The solution is to perform multiple pivots in parallel, but in a controlled way that does not sacrifice much performance. This second obstacle is commonly addressed in parallel algorithms. Most related, Schudy [19] and Blelloch et al. [1] also use multiple pivots to parallelize the divide-and-conquer algorithm for strongly connected components [4], which is itself structurally identical to Algorithm 1. (Their algorithms, however, assume reachability as a black box; they do not address the first challenge.)

The full algorithm is given in pseudocode as Algorithm 3. Section 4.1 walks through the ideas incrementally, guided by rough intuitions behind the analysis. The key performance lemma, analogous to Lemma 1.1, is the following:

Lemma 4.1. *Let $\hat{G} = (\hat{V}, \hat{E})$ be a directed graph, let $n = \left|\hat{V}\right|$, let $m = \left|\hat{E}\right|$, and assume without loss of generality that $m \geq n/2$.*

Consider any directed path $\hat{P}$ from u to v with length($\hat{P}$) $\leq D$, for $D = \Theta(n^{2/3} \log n)$. Let S be the shortcuts produced by an execution *of Algorithm 3 on $\hat{G}$ starting with $h = \lg n$. Then with probability at least $1/2$: (1) there exists a path from u to v in $G_S = (\hat{V}, \hat{E} \cup S)$ with length at most $D/2$, (2) the number of shortcuts produced is $|S| = O(n \log^2 n)$, and (3) the total number of vertices and arcs visited by searches is $O(m \log^2 n)$. Moreover, the maximum distance used for any search is $O(n^{2/3} \log^{12} n)$;*

Using multiple runs of Algorithm 3 (see Section 4.2) yields the following:

Theorem 4.2. *There exists a randomized algorithm that takes as input a directed graph $\hat{G} = (\hat{V}, \hat{E})$ and uses distance-limited searches with the following guarantees. Let $n = \left|\hat{V}\right|$, $m = \left|\hat{E}\right|$, and without loss of generality $m \geq n/2$. Let $\gamma \geq 1$ be a parameter controlling failure probability. Then (1) the maximum distance used for any search is $O(n^{2/3} \log^{12} n)$; (2) the algorithm produces a size-$O(\gamma n \log^4 n)$ set S^* of shortcuts; (3) the total number of vertices and arcs visited by searches is $O(\gamma m \log^4 n + \gamma^2 n \log^8 n)$, and the searches dominate the overall number of primitive operations performed; and (4) with failure probability at most $1/n^\gamma$, the diameter of G_{S^*} is $O(n^{2/3} \log n)$,*

The remainder of this section is organized as follows. Section 4.1 describes the main subroutine, namely Algorithm 3. Section 4.2 extends the algorithm to perform multiple passes, thereby obtaining Theorem 4.2. Finally, Section 4.4 gives an overview of the interesting issues that arise in the analysis. Due to space constraints, details and all proofs are deferred to the full version of the paper.

Updated Notation. If there exists a path of length at most d from u to v, then $u \preceq_d v$. If $u \preceq_d v$ or $v \preceq_d u$, then u and v are ***d-related***. All other notations and definitions in Section 2 that depend on $\preceq$ (i.e., successors, predecesors, ancestors, descendents, bridges) are augmented with the term "d-limited" and a subscript d to indicate that the $\preceq$ in the definition should be replaced by $\preceq_d$. For example, $R^+_d(G, x) = \{v | x \preceq_d v\}$ denotes the ***d-limited successors*** of x.

4.1 The Algorithm

The main goal is to replace the searches $R^+(G, x)$ in Algorithms 1 and 2 with D-limited searches, for $D = \tilde{O}(n^{2/3})$. The good news is that some version of Lemma 3.4 still holds when restricted to pivots drawn from D-limited ancestors. The bad news is that Lemma 3.2 does not hold. For concreteness, consider a path $\langle v_0, v_1, \ldots, v_\ell \rangle$. It is possible, for example, that $x \preceq_D v_{2k}$ but $x \not\preceq_D v_{2k+1}$ for all $0 \leq k < \ell/2$. Thus all the even vertices would be in V_F and all the odd ones would be in V_U, splitting the path into $\Theta(\ell)$ pieces with no potential to shortcut them later. In contrast, when the search is *not* D-limited, $x \preceq v_k$ implies $x \preceq v_j$ for all $j \geq k$, so V_F contains a single contiguous subpath.

The solution is to extend the search a little further and duplicate vertices. That is, start with a distance of dD, for some $d = \tilde{O}(1)$. Any vertices reached this way are called ***core vertices***, and they are treated similarly to reached vertices in Algorithm 1. Then extend the search a little farther: to a distance of $(d+1)D$. Vertices discovered in the extended search are called ***fringe vertices***, denoted by F^+ and F^- in the code. Fringe vertices F^+ and incident arcs are duplicated (similarly for F^-), belonging to both to the unrelated subproblem $G[V_U]$ and the forwards subproblem $G[V_F \cup F^+]$.

The addition of fringe vertices fixes the path-splitting problem, giving an analog of Lemma 3.2, at least for paths of length

Algorithm 3: Shortcutting algorithm with distance-limited searches.

```
ParSC($G = (V, E)$, $h$)
  /* The value $h$ indicates how many more levels of
     recursion to perform. $\epsilon_\pi$, $N_k$, $N_L$, and $D$ are
     global parameters (independent of subproblem)
     set later.                                          */
1  if $h = 0$ then return $\emptyset$
2  $S := \emptyset$
3  randomly permute $V$, giving vertex sequence
   $X = x_1, x_2, \ldots, x_{|V|}$. Mark each $x_j$ live
4  split $X$ into subsequences $X_1, X_1, \ldots, X_{2k}$, with
   $|X_i| = |X_{k-i+1}| = \lfloor (1+\epsilon_\pi)^i \rfloor$ for $i < k$ and
   $|X_k| = |X_{k+1}| \leq \lfloor (1+\epsilon_\pi)^k \rfloor$
5  for $i := 1$ to $2k$ do
6      $d_{\min} = 1 + hN_kN_L - iN_L$                    // offset
       $d_{\max} = d_{\min} + N_L - 1$
7      choose random $d \in \{d_{\min}, d_{\min}+1, \ldots, d_{\max}-1\}$
8      foreach live $x_j \in X_i$ do
9          $R^-_j := R^-_{dD}(G, x_j)$
           $R^+_j := R^+_{dD}(G, x_j)$                    // core vertices
10         $F^-_j := R^-_{(d+1)D}(G, x_j) \backslash R^-_j$ ;
           $F^+_j := R^+_{(d+1)D}(G, x_j) \backslash R^+_j$      // fringe vertices
11         $S := S \cup \left\{(x_j, v) | v \in R^+_j \cup F^+_j\right\} \cup \left\{(u, x_j) | u \in R^-_j \cup F^-_j\right\}$
                                                          // add shortcuts
12         append a tag of $j$ to all vertices in $R^+_j \cup R^-_j$
13     foreach live $x_j \in X_i$ do
14         remove vertices with tag $< j$ from $R^+_j, R^-_j, F^+_j, F^-_j$
                                              // first core search wins
15         $V_{F,j} := R^+_j \backslash R^-_j$ ;     $V_{B,j} := R^-_j \backslash R^+_j$
16         $S := S \cup$ ParSC($G[V_{F,j} \cup F^+_j], h-1$) $\cup$
           ParSC($G[V_{B,j} \cup F^-_j], h-1$)      // include fringe
17     mark all vertices in $\bigcup_j (R^+_j \cup R^-_j)$ as dead in $X$
18     $V_U := V \backslash \bigcup_j (R^+_j \cup R^-_j)$
19     $G := G[V_U]$
20 return $S$
```

$\ell \leq D$. Consider again the bad example where $x \preceq_{dD} v_{2k}$ but $x \not\preceq_{dD} v_{2k+1}$. All of the even vertices are core vertices, but now all of the odd vertices are fringe vertices. Thus, the entire path is indeed contained in the subgraph $G[V_F \cup F^+]$. The analysis still treats the path as being partitioned across subproblems, but any fringe vertices on the path can be treated as belonging to whichever subproblem is better.

Unfortunately, duplicating fringe vertices introduces another problem — path-related fringe vertices can be active in multiple subproblems, thereby destroying the progress bound on Φ. In the worst case, almost all of the active vertices could be fringe vertices, and the total number of active vertices could thus increase drastically after partitioning around pivot x.

The solution is to select d (for search distance dD) randomly from the range $d \in \{1, 2, \ldots, N_L - 1\}$, for some $N_L = \tilde{O}(1)$ to be chosen later.[7] Any vertices in the fringe for distance dD are in the core for distances $d'D$, $d' > d$. Thus, on average, only an $O(1/N_L)$ fraction of vertices are on the fringe. For large enough N_L, the addition of these fringe vertices does not impact $\phi(s)$ much.

It is also important that the distances searched never increases. This is because any progress towards the number of active vertices is with respect to a particular search distance dD. The algorithm therefore selects the distance to search from the $N_L - 1$ options $\{d_{\min}, d_{\min} + 1, \ldots, d_{\max} - 1\}$, where $d_{\max} = d_{\min} + N_L - 1$, and $d_{\min}$ is offset by some value to allow for later decreases. With each choice of pivot(s), the offset decreases by at least N_L. More precisely, as in Algorithm 2, the main subroutine consists of a sequence of iterations, where some pivots are chosen in each iteration. The total number of iterations is bounded by some value N_k, meaning that the full range of distances effectively owned by a single call has size $N_k N_L$. Each time the recursion depth increases, the offset decreases accordingly by $N_k N_L$. We thus use an initial offset $d_{\min} = 1 + hN_k N_L$, and more generally $d_{\min} = 1 + hN_k N_L - iN_L$, where h is the number of levels of recursion to perform and i is the iteration number. As long as $h = \tilde{O}(1)$, $N_k = \tilde{O}(1)$, and $N_L = \tilde{O}(1)$, the maximum distance searched is $\tilde{O}(D) = \tilde{O}(n^{2/3})$ as desired.

Searches from Multiple Pivots. In addition to being more parallelizable, searching from multiple pivots is also necessary to keep the maximum number of iterations $N_k = \Theta(\log_{1+\epsilon_\pi} n) = \Theta(\log n/\epsilon_\pi)$ low, where $0 < \epsilon_\pi \le 1$ is chosen later.

A single recursive call ParSC consists of a sequence of iterations, like Algorithm 2 flattening the recursion of $G[V_U]$. Each iteration proceeds as follows. First, sample a set $\{x_j\}$ of pivots and perform independent searches from each of them, determining both the dD-limited core (R_j^+ and R_j^-) and the $(d+1)D$-limited fringe (F_j^+ and F_j^-) of each pivot. Add shortcuts to and from all reached vertices. To roughly simulate the effect of selecting one pivot at a time, if a vertex is part of x_j's core, then it is removed from the core and fringe sets for any x'_j with $j' > j$. Next, calculate the forward and backwards sets $V_{F,j}$ and $V_{B,j}$, respectively, as in Algorithms 1 and 2. Then launch the recursive subproblems $G[V_{B,j} \cup F_j^-]$ and $G[V_{F,j} \cup F_j^+]$, each including the fringe nodes found in that direction. Finally, remove all core vertices from the graph and start the next iteration.

Algorithm 3 uses the following process to control the pivot sampling. Randomly permute all of the vertices at the start of the call, creating a sequence $x_1, x_2, \ldots$ of pivots to consider. All pivots are initially *live*; the live pivots are those still in the graph. In each iteration, select the next group of pivots from the sequence, where the size of the group is discussed below. Perform searches from each live pivot, and ignore the dead ones. When a core vertex is removed from the graph, the vertex is also marked dead in the pivot sequence.

Number of pivots. The number of pivots (live or dead) selected in each iteration is controlled by the parameter $0 < \epsilon_\pi \le 1$. For the first $\Theta(1/\epsilon_\pi)$ iterations, only one pivot is used. In subsequent iterations, the number of pivots increases geometrically by roughly $(1+\epsilon_\pi)$. Were the only goal to keep the number of times a vertex is reached in a search to $O(\log n)$, setting $\epsilon_\pi = 1$ and following the geometric increase would be sufficient. To bound the number of times a path can split in a single iteration, however, it is important to achieve a tighter bound. There are $2k$ iterations total, where k is chosen to be large enough to include all vertices according to the following group sizes. The first k iterations follow a geometric increase, and the next k iterations follow a symmetric geometric decreases. More precisely, the number of pivots considered in both iteration i and $2k - i + 1$ is $\lfloor (1+\epsilon_\pi)^i \rfloor$; the middle iterations $2k$ and $2k + 1$ can be smaller.

[7] Read N_L as "number of layers".

Algorithm 4: Diameter reduction with distance-limited searches.

```
ParDiam($\hat{G} = (\hat{V}, \hat{E}, \gamma)$)
    /* The value $\gamma \ge 1$ controls failure probability */
1   $G' = (V', E') := \hat{G}$
2   for $i := 1$ to $\Theta(\log n)$ do
3       foreach $j \in \{1, 2, \ldots, \Theta(\gamma \log n)\}$ do
4           $S_j :=$ ParSC($G', \lg n$), aborting if number of shortcuts
            or work exceeds Lemma 4.1
5       $E' := E' \cup \left(\bigcup_j S_j\right)$          // add more arcs to $G'$
6   return $G'$
```

4.2 Full Diameter-Reduction Algorithm and Proof of Theorem 4.2

Like the algorithm in Section 3, to achieve diameter reduction with high probability requires multiple passes of Algorithm 3. But now more passes are necessary. The full algorithm, shown in Algorithm 4, is as follows. Perform $\Theta(\log n)$ iterations. In each iteration, perform $\Theta(\gamma \log n)$ independent executions of Algorithm 3 on the current graph. Add to the graph all of the shortcuts produced thus far, and continue to the next iteration on the updated graph.

The main reason for the extra passes of Algorithm 3 is that, due to the $\tilde{O}(D)$-limited searches, the analysis only considers paths of length D. The distance D is chosen to be large enough so that each iteration is enough to reduce the length of the path to $D/2$, with high probability, but a longer path needs to be subdivided.

Proof of Theorem 4.2, Assuming Lemma 4.1. Consider any two vertices $u \prec v \in V$. Let Δ_i denote the length of the shortest path from u to v in the graph after iteration i of the outer loop of Algorithm 4. The main claim is that with probability at least $1 - /n^{2+\gamma}$, for all i we have $\Delta_i \le D \cdot \max n/(D2^i), 1$. For $i = \Omega(\log n)$, i.e., when the main procedure returns, this reduces to $\Delta_i \le D$. Finally, taking a union bound across up to n^2 pairs u, v, the diameter bound is met with failure probability $1/n^\gamma$.

The proof is by induction on i. For $i = 0$, the length of the shortest path is at most n, so $\Delta_0 \le n = D \cdot n/(D2^0)$. For the inductive step (going from iteration i to $i+1$), consider the shortest path P from u to v in the current graph. If *length*$(P) \le D$, then the path is already short enough. Otherwise, subdivide the path into at most $(n/(D2^i))$ subpaths, each of length at most D. Consider each subpath. By Lemma 4.1, a single execution of Algorithm 3 shortens the subpath's length to $D/2$ with constant probability. Thus, for failure probability $1/n^{4+\gamma}$ can be achieved by repeating for $4 \lg n + \gamma \lg n$ runs. Taking a union bound over all $< n$ subpaths gives failure probability for this iteration of at most $1/n^{3+\gamma}$. If no failure occurs, concatenating the subpaths yields a path of length $(D/2) \cdot n/(D2^i) = n/(D2^{i+1})$ as desired.

Taking a union bound for failures across all $\Theta(\log n)$ iterations of the outer loop, the failure probability overall for this pair is at most $1/n^{2+\gamma}$.

The search distance follows directly from Lemma 4.1. The number of shortcuts follows from Lemma 4.1 by multiplying by the number of $\Theta(\gamma \log n)$ runs. As for the bound on total number of arcs visited, observe that the graph size is at most $\hat{E} + O(\gamma n \log^4 n)$ at the end. Thus, by Lemma 4.1, each run of Algorithm 3 visits $O((m + \gamma n \log^4 n)(\log^2 n)) = O(m \log^2 n + \gamma n \log^6 n)$ arcs. Multiplying by $\Theta(\gamma \log^2 n)$ runs completes the proof. □

4.3 Notation and Shorthand

It is often convenient to refer to iterations of the loop in Algorithm 3. During iteration i, quite a bit happens: some pivots are processed, some searches are performed, some induced subgraphs are built, etc., and the claims throughout refer to those objects. Defining every term concretely in every lemma statement or proof gets tedious and unwieldy. Instead, this paper adopts some notational conventions consistent with the pseudocode in Algorithm 3, using the variables to implicitly adopt the meaning of the code.

Concretely, for iteration i on graph $G = (V, E)$, the following notations are used with the same meaning as the pseudocode: h, X_i meaning the pivot sequence, and d meaning the random distance chosen. Moreover, for each $x_j \in X_i$, whenever notations R_j^+, R_j^-, F_j^+, F_j^-, $V_{F,j}$, or $V_{B,j}$ appear, they should also be interpreted to have the meaning laid out in the pseudocode.

Min and max distances. In each iteration i, the algorithm chooses a random distances in some size-$(N_L - 1)$ range, but at an offset that depends on the iteration. Specifically, the distance is drawn uniformly at random from $d \in \{d_{\min}, d_{\min} + 1, \ldots, d_{\max} - 1\}$. The minimum possible search distance for a core search is $d_{\min} D$. The maximum possible search distance for a core search is $(d_{\max} - 1)D$, and the maximum possible distance for a fringe search is $d_{\max} D$. Note that the offset used for $d_{\min}$ and $d_{\max}$ both rely on the current iteration i and recursion height h.

These min and max distances are useful for classifying vertex relationships as follows:

Definition 4.3. Consider any iteration i of Algorithm 3. To unclutter the notation, $\preceq_{\min}$ is used to denote $\preceq_{d_{\min} D}$, where $d_{\min} = 1 + hN_k N_L - iN_L$. Similarly, $\preceq_{\max}$ similarly denotes $\preceq_{d_{\max} D}$.

- Vertices u and v are ***never related*** if $u \not\preceq_{\max} v$ and $v \not\preceq_{\max} u$.
- Vertices u and v are ***partly related*** if $u \preceq_{\max} v$ or $v \preceq_{\max} u$.
- Vertices u and v are ***fully related*** if $u \preceq_{\min} v$ or $v \preceq_{\min} u$. If u and v are fully related, then they are also partly related.

When comparing a vertex v and a path P, the same terms apply in the natural way. For example, if v is fully related with any vertex in P, then v and P are fully related.

4.4 Overview of the Analysis

This section outlines the issues that arise in the analysis of Algorithm 3. The complete analysis appears in the full version of the paper.

For most of the main ideas, it is convenient to consider a version of the algorithm where in each iteration, only one pivot is selected. (This would be the case if, e.g., $\epsilon_\pi = \Theta(1/n)$ for appropriate choice of constant.) Using a larger number of pivots is indeed important for some aspects of the analysis, addressed in Section 4.4.5.

An analog of Lemma 3.4 is easy to show. The statement is complicated by the fact that this lemma does not count the impact of fringe vertices. Specifically:

Lemma 4.4. *Consider any iteration of the algorithm on remaining subproblem (G, P). Let dD be an arbitrary distance.*

Suppose that a pivot x is selected uniformly at random from $A = Anc_{dD}(G, P)$. Let $\alpha = |A|$ and let α' denote the total number of vertices in A that are either (i) reached by the forward core search and still path-active in the forward recursive subproblem, or (ii) not reached by the core search and still path-active in the next iteration. Then $E[\alpha'] < \alpha/2$.

When bounding the progress with respect to ϕ, however, one must incorporate the impact of fringe nodes. It is fairly easy to prove the following lemma bounding the number of fringe nodes.

Lemma 4.5. *Consider any iteration of the algorithm on remaining subproblem (G, P). Let V' be any subset of vertices, e.g., all path-related vertices. Let x be an arbitrary choice of pivot.*

Suppose that the search distance d is chosen uniformly at random from $\{d_{\min}, \ldots, d_{\max} - 1\}$. Then the expected number of vertices in V' that are fringe vertices is at most $O(|V'|/N_L)$.

These preliminary lemmas expose several issues, outlined in each of Sections 4.4.1–4.4.4. Section 4.4.5 revisits the multiple pivots.

4.4.1 The Flattened Path-Relevant Tree. Naturally, each node in the *unflattened* path-relevant tree should correspond to an iteration of the algorithm, and more specifically the graph G on which that algorithm is operating along with a path P to shortcut. The big question is how to setup the *flattened* tree, which was essential for the analysis in Section 3.

The natural choice would be to flatten any nodes selecting pivots that are dD-unrelated to the path P. Unfortunately, this choice effects some subtle changes to the distributions assumed in Lemmas 4.4 and 4.5. Consider, for example, the process where random pivots and random distances are sampled until finally selecting a pivot that happens to be dD-path-related. Then one cannot assume, as in Lemma 4.5, that d is chosen uniformly.

The solution is reinterpret the random process as follows:

(1) Toss a weighted coin to determine whether the pivot is never path related or partly path related. Repeat until getting a partly-path-related pivot, i.e., flatten the iterations corresponding to never-path-related pivots. Pessimistically assume that two path-relevant subproblems are created.
(2) Toss a weighted coin to decide if the pivot is fully path related or not. If not, Lemma 4.4 will not be applied — a different argument is necessary.

(3) Select the pivot uniformly from the appropriate set. If the pivot is fully path-related, apply Lemma 4.4, but with respect to distance $d_{\min} D$ — the actual distance is not yet known.
(4) Select the random distance and determine the number of fringe nodes. No information about the distance has yet been revealed, so Lemma 4.5 may be applied.

4.4.2 Balancing Fully and Partly Path-Related Pivots. The potential function incorporates all partly path-related vertices, e.g., using $\alpha = \left|Anc_{d_{\max}D}(G,P)\right|$. A nice consequence of reducing distances with each subsequent subproblem is as follows: any partly but not fully path-related vertex v is never path related in any child subproblems. Thus, regardless of random choices, v is not included in the potential in any subproblems.

The argument outlined in Section 4.4.1 considers the fully path-related vertices separately. In more detail, let α, β, and δ denote the numbers of partly path-related ancestors, bridges, and descendants, respectively. Let $\bar{\alpha}$, $\bar{\beta}$, and $\bar{\delta}$ denote the fully-path-related numbers, and let $p = (\bar{\alpha}+\bar{\beta}+\bar{\delta})/(\alpha+\beta+\delta)$. The idea is that if p is small, it is likely to select a fully-path-related pivot and the normal c-factor progress of the c-reducing function can be applied. If on the other hand p is large, then the c-reducing aspect is not applied, but significant progress is made automatically.

This logic imposes an additional requirement on the well-behaved c-reducing function ψ. Specifically

$$p \cdot c \cdot \psi(\bar{\alpha},\bar{\beta},\bar{\delta}) + (1-p)\psi(\bar{\alpha},\bar{\beta},\bar{\delta}) \le c \cdot \psi(\alpha,\beta,\delta) .$$

This condition is relatively easy to prove for $\psi(\alpha,\beta,\delta) = \alpha+\beta+\delta$, which is only $(3/4)$-reducing. It is harder to prove that this condition holds for $\psi(\alpha,\beta,\delta) = \sqrt{(\alpha+\beta)(\delta+\beta)}$, but it does.

4.4.3 Impact of Fringe Nodes on ψ. If using the linear function $\psi(\alpha,\beta,\delta) = \alpha+\beta+\delta$, Lemma 4.5 also implies a bound on the impact fringe nodes have to the potential. In particular, the expected number of path-active nodes increases by an additive $O((\alpha+\beta+\delta)/N_L)$. For $N_L = \Omega(\log n)$, this additive increase becomes a multiplicative $(1+O(1/\log n))$ factor, which is negligible given that the analysis only considers $O(\log n)$ levels.

Bounding the impact of fringe nodes for more complex ψ is much more difficult as the number of fringe nodes added does not easily relate to the change in ψ. The full analysis instead uses the following potential function:

$$\psi(\alpha,\beta,\delta) = \sqrt{(\alpha+\beta)(\delta+\beta)} \cdot C_\phi + (\alpha+\beta+\delta) , \qquad (2)$$

where $C_\phi = \Theta(\log^{3/2})$. The advantage of the linear term is that it makes it easier to bound the impact of fringe nodes. But the nonlinear term is weighted by more so that imbalance can still be exploited in roughly the same way.

Lemma 4.6. *The function ψ from Equation 2 is a well-behaved c-reducing function, for $c = (1/\sqrt{2})(1+O(1/\log n))$, with overhead $C_\phi = \Theta(\log^{3/2} n)$.*
Moreover, bounding the impact of adding $f = f_1 + f_2 + f_3$ fringe nodes, $\psi(\alpha+f_1,\beta+f_2,\delta+f_3) \le (1+O(1/\log n)) \cdot \psi(\alpha,\beta,\delta) + 3fC_\phi^2$.

To make the additive impact of fringe nodes small enough, the preceding lemma suggests choosing $N_L = \Omega(C_\phi^2 \log n) = \Omega(\log^4 n)$.

4.4.4 Analyzing Layers in the Flattened Tree. Most of the remainder of the analysis, at least for the single-pivot case, is similar to Section 3. However, there is one subtle challenge: Theorem 2.1 requires that the potential $\Phi(I_r) = \sum_{s \in I_r} \phi(s)$ satisfy $\phi(I_r) \le \phi(I_{r-1})$ for all feasible sequences $I_0, I_1, \ldots$ of instances. The inclusion of fringe nodes violates this requirement, since an unlucky outcome can cause the potential to increase, and it is not even clear whether such an increase is unlikely. The solution is to multiply the potential by $(1+\Theta(1/\log n))^{\lg n - r}$. This extra slack causes the potential to automatically decrease when moving to the next level in the tree, unless the number of fringe nodes far exceeds the expectation. An unlucky outcome can still cause the potential to increase, but with the proper setup Theorem 2.1 can be applied.

4.4.5 Multiple Pivots. The main problem with using a single pivot is that the number of iterations N_k is too large, and hence so is the maximum search distance. With multiple pivots, however, the graph may be searched from multiple places. Increasing the number of pivots geometrically ensures the following:

Lemma 4.7. *Restrict ϵ_π to be $0 < \epsilon_\pi \le 1$, and choose any $\gamma \ge 1$. Then for any iteration and vertex v: with probability at least $1 - 1/n^\gamma$, v is not visited more than $O(\gamma \log n)$ times.*

Thus, the total number of fringe nodes may increases by a $O(\log n)$ factor, which can be offset by increasing N_L further.

The other consequence of selecting multiple pivots is that there may be multiple partly-path-related pivots, and the path may be subdivided into many pieces. For technical reasons, the analysis only leverages reductions in potential arising from the first partly-path-related pivot selected. But the number of path-related subproblems generated cannot be ignored. The following lemma, used to bound the fanout, says that conditioning on the fact that some path-related pivot is chosen, the additional number of partly path-related pivots selected is extremely small.

Lemma 4.8. *Consider an iteration of the algorithm. Let x denote the number of partly-path-related pivots selected. Then $E[x] = O(\epsilon_\pi \log n)$. More importantly, $E[x | x \ge 1] \le 1 + O(\epsilon_\pi \log n)$.*

Moreover, if x path-related pivots are selected, then the number of path-relevant subproblems generated is at most $x+1$. Thus, as long as the total number of pivots selected in a level never exceeds $1+O(1/\log n)$ times the number of subproblems, the number of subproblems at level r is at most $(2(1+O(1/\log n)))^r = O(2^r)$ for $r = O(\log n)$. In short, for small-enough choice of ϵ_π, the increased fanout has no significant impact on the length of the shortcutted path.

5 PARALLEL VERSION

This section briefly discusses the parallel version of Algorithm 3 and Algorithm 4, with details deferred to the full version of the paper. This section assumes the reader is comfortable enough with parallel algorithms to infer the details, instead focusing only on the interesting issues.

The main results are as follows.

Theorem 5.1. *There exists a randomized parallel algorithm taking as input a directed graph $\hat{G} = (\hat{V}, \hat{E})$ with the following guarantees. Let $n = \left|\hat{V}\right|$, $m = \left|\hat{E}\right|$, and without loss of generality assume $m \ge n/2$.*

Then (1) the algorithm produces a size-$O(n \log^4 n)$ set S^* of shortcuts; (2) the algorithm has $O(m \log^6 n + n \log^{10} n)$ work; (3) the algorithm has $O(n^{2/3} \log^{19} n)$ span; and (4) with high probability, the diameter of $G_{S^*} = (\hat{V}, \hat{E} \cup S^*)$ is $O(n^{2/3} \log n)$.

Corollary 5.2. *There exists a randomized parallel CREW algorithm for digraph reachability that has work $O(m \log^6 n + n \log^{10} n)$ work and $O(n^{2/3} \log^{19} n)$ span, both with high probability.*

Proof. Perform the diameter reduction algorithm, then run a standard parallel BFS but limited to $O(n^{2/3} \log n)$ hops. The work and span of the diameter reduction dominates. If the BFS completes in the prescribed number of rounds, the algorithm terminates. Otherwise, keep repeating the diameter reduction and BFS until successful. □

Model. This paper adopts the now *de facto* standard ***work-span model*** [5], also called work-time [11] or work-depth model, which abstracts low-level details of the machine such as the number of processors or how parallel tasks are scheduled. The work-span model allows algorithms to be expressed through the inclusion of parallel loops, i.e., a parallel foreach. A parallel foreach indicates that each task corresponding to a loop iteration may execute in parallel, and that all parallel tasks must complete before continuing to the next step after the loop. It is generally straightforward to map algorithms from the work-span model to a PRAM model; see, e.g., [11, 14]. Like the *asynchronous PRAM model* [7], the work-span model requires that algorithmic correctness not be tied to any assumptions about how tasks are scheduled beyond the explicit ordering imposed by the loops. That is to say, it should not be assumed that the instructions across iterations execute in lock step.

The ***work*** of an algorithm is the same as the sequential running time in a RAM model (replacing all parallel loops by sequential loops). When multiple tasks are combined through a parallel loop, the combined ***span*** is the maximum of the span of the individual subproblems, plus the span of the loop itself. There are several variants to the work-span model. In a ***binary-forking model*** such as [5], the span of a k-way loop is $\Theta(\lg k)$. Much of the literature on parallel algorithms, however, adopts an ***unlimited-forking model***, where the span of launching k parallel tasks adds $O(1)$ to the span. Since many of the subroutines employed are analyzed in the latter model, this paper adopts the unlimited-forking model. PRAM algorithms, for example, correspond to an unlimited forking model. Both models only differ by logarithmic factors in the span.

The algorithm is a concurrent-read exclusive-write (***CREW***) algorithm. CREW means that multiple parallel tasks may read the same data, but they may not write to the same location.[8]

Performing Concurrent Searches. The key subroutine in Algorithm 3 are the dD-limited searches to find, e.g., R_j^+. One might simply replace the foreach loops by parallel loops, but the question is how the bookkeeping should be performed. Ordinarily, a BFS keeps track of already-visited vertices by either annotating vertices in the graph directly, or equivalently by keeping an extra array indexed by vertex. A natural way to perform multiple searches in parallel using a CREW algorithm would thus be to duplicate the bookkeeping efforts for each parallel search, but doing so would increase the work dramatically just to copy the graph or initialize the arrays.

The key property that allows an efficient implementation is Lemma 4.7 — with high probability, no vertex is visited by more than $O(\log n)$ parallel searches. The implementation may assume that this is the case, and just abort by returning immediately if a vertex gets visited too many times.

The main goal is to support the following for each call to `ParSC`.

Lemma 5.3. *Consider an iteration i in call to* `ParSC` *on graph $G = (V, E)$. Let n_e be the total number of arcs traversed by searches, counting an arc for each search that reaches it. There exists an algorithm implementing the iteration having $O(n_e \log^2 n + |X_i| \log n)$ work and $O(n^{2/3} \log^{13})$ span.*

The remainder of the section is devoted to exhibiting an algorithm that proves Lemma 5.3, focusing only on the core search. Extending to fringe searches is not much harder.

The set of searches from X_i (in one direction) are grouped together as a single modified BFS. Rather than marking a vertex with a single bit indicating whether it has been discovered, a vertex is tagged with a list of IDs of the pivots that have reached it. Every time this list of IDs changes, the vertex may be re-added to the frontier and all of its outgoing arcs explored again. Since a vertex is not visited too many times, the overhead is not too high.

In more detail, the algorithm is as follows. At the start of the call to `ParSC`, initialize $\Theta(\log n)$ space for each vertex to record the ID tags, initally all null. Use an array to store the frontier vertices along with the ID of the pivot from which this search originated; a vertex may appear in the frontier multiple times from different pivots. Save all frontiers so as to identify all vertices reached by the searches at the end and also to record all new shortcuts.

To start a set of searches from $|X_i|$, copy all live pivots x_j to the frontier array and associate with each pivot its own ID as the search originator. Also update each pivot's tag list to include itself.

Each round of the BFS operates as follows. Foreach vertex in the frontier in parallel, identify the number of outgoing arcs. Next, perform parallel prefix sums so that each arc has a distinct index in the next frontier array. Foreach arc (u, v) in parallel, let x_j be the associated pivot ID. Check whether v's ID set includes x_j; this check can be performed in $O(\log n)$ sequential time (both work and span) by scanning through v's tag list. If x_j is not present, record v and x_j in (u, v)'s slot in the next frontier; otherwise record null.

At this point, a vertex may appear many times in the frontier list, even from a single search. Sort the frontier list by vertex (high priority) and pivot ID (lower priority). Remove duplicate entries with a compaction pass. Now each vertex appears at most once for each search, so $O(\log n)$ times in total. For each slot j in the next frontier in parallel, let v be the vertex stored there. Check whether this is the first slot for vertex v, i.e., if $j - 1$ stores a different vertex. If so, scan through the $O(\log n)$ next slots (sequentially), and for each entry of v append the pivot tag to v's tag list.

Repeat this process for the number of rounds dictated by the distance dD for the core searches. When the searches complete, sort the arrays of all vertices reached by core searches. Foreach vertex v in core searches, in parallel, identify the lowest ID pivot

[8]CREW is usually a restriction applied to the PRAM [6, 8, 18] machine model, e.g., a CREW PRAM. In contrast, the work-span model is an algorithmic cost model, not a machine model. This paper proposes lifting the CREW qualifier to the work-span level rather than the PRAM level.

reaching v. Again use parallel prefix sums and then copy the lowest-ID occurrence of v to a new array for the recursive searches. Finally, sort the new array by pivot ID so that all vertices in the same induced subgraph are adjacent. Building the induced subgraphs for recursive calls can again be accomplished with arc counting, prefix sums, and sorting.

Updating $G[V_U]$. One could build $G[V_U]$ explicitly, but doing so would require processing the full graph. The goal expressed by Lemma 5.3 is to have work proportional to the number of arcs reached, but $G[V_U]$ could be much larger. Instead, simply mark vertices in V as dead when they have been reached by a core search. Augment the search to ignore dead vertices.

Completing the proof of Lemma 5.3. The basic subroutines used in each round such as prefix sums, compaction, etc, can all be performed in linear work and $O(\log n)$ span. (See e.g., [11].) Scanning the list of tags also requires $O(\log n)$ work per arc on the frontier and $O(\log n)$ span as it is performed sequentially. Using Cole's merge sort [3], the cost of a sort is $O(\log n)$ work per element sorted and $O(\log n)$ span. Multiplying the search distance by $O(\log n)$ thus gives the overall span bound. Since each arc may be reached by $O(\log n)$ searches, the bound is $O(\log^2 n)$ work per arc visited. □

Aborting Algorithm 3. To make the work (and shortcut) bound deterministic, Algorithm 4 needs the ability to abort any runs of Algorithm 3 that exceed the target work bound. (Exceeding the shortcut bound can be handled by simply discarding the result — a true abort is not necessary there.)

Unfortunately, the proof of Lemma 4.1 examines the work in aggregate across levels in the recursion tree. It is not clear how to make local abort decisions. One natural alternative is to augment the algorithm to check the elapsed time, and to return immediately if some threshold has been reached. Technically, however, this solution violates the work-span model as the target time bound would depend on both on how efficiently the program is scheduled and on the number of processors employed.

Nevertheless, it is possible to augment the algorithm to implement aborts as needed in the work-span model.

6 CONCLUSIONS

This work makes the first major progress toward work-efficient parallel algorithms for directed graphs, but it also exposes several new questions. First, can the performance be improved? Shaving logarithmic factors would be nice, but doing so seems premature — it is quite likely that $\tilde{O}(n^{2/3})$ is not the final answer. I would conjecture that an $n^{1/2+o(1)}$-diameter reduction is possible using a more sophisticated algorithm based on the one presented herein.

Is true work efficiency, i.e., $O(m)$ work, possible for the diameter-reduction problem? Achieving that would require first producing an $O(m)$-time sequential algorithm for the problem.

Hesse's lower bound provides a lower bound on work-efficient diameter reduction, but that is not a general lower bound on digraph reachability. Can digraph reachability be improved by relaxing the shortcutting requirements, perhaps by adopting some ideas from Spencer's algorithm? Are there good general lower bounds for work/span tradeoffs of these algorithms?

Finally, can the algorithm be extended to solve unweighted shortest paths? Solving the exact problem is likely difficult, but even an approximate solution would be progress.

ACKNOWLEDGEMENTS

Special thanks to Cal Newport and Justin Thaler for some useful discussions. This work is supported in part by NSF grants CCF-1718700, CCF-1617727, and CCF-1314633.

REFERENCES

[1] Guy E. Blelloch, Yan Gu, Julian Shun, and Yihan Sun. 2016. Parallelism in Randomized Incremental Algorithms. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. 467–478. https://doi.org/10.1145/2935764.2935766
[2] Richard P. Brent. 1974. The Parallel Evaluation of General Arithmetic Expressions. *J. ACM* 21, 2 (April 1974), 201–206.
[3] Richard Cole. 1988. Parallel Merge Sort. *SIAM J. Comput.* 17, 4 (Aug. 1988), 770–785. https://doi.org/10.1137/0217049
[4] Don Coppersmith, Lisa Fleischer, Bruce Hendrickson, and Ali Pinar. 2005. *A divide-and-conquer algorithm for identifying strongly connected components*. Technical Report RC23744. IBM Research.
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms* (2nd ed.). MIT Press, Cambridge, MA, USA.
[6] Steven Fortune and James Wyllie. 1978. Parallelism in Random Access Machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. 114–118. https://doi.org/10.1145/800133.804339
[7] P. B. Gibbons. 1989. A More Practical PRAM Model. In *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*. 158–168. https://doi.org/10.1145/72935.72953
[8] Leslie M. Goldschlager. 1978. A Unified Approach to Models of Synchronous Parallel Machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. 89–94. https://doi.org/10.1145/800133.804336
[9] William Hesse. 2003. Directed Graphs Requiring Large Numbers of Shortcuts. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 665–669. http://dl.acm.org/citation.cfm?id=644108.644216
[10] Shang-En Huang and Seth Pettie. 2018. Lower Bounds on Sparse Spanners, Emulators, and Diameter-reducing shortcuts. *ArXiv e-prints* (Feb. 2018). arXiv:cs.DS/1802.06271
[11] Joseph JáJá. 1992. *An Introduction to Parallel Algorithms*. Addison-Wesley.
[12] M.-Y. Kao and P. N. Klein. 1990. Towards Overcoming the Transitive-closure Bottleneck: Efficient Parallel Algorithms for Planar Digraphs. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*. 181–192. https://doi.org/10.1145/100216.100237
[13] Richard M. Karp. 1994. Probabilistic Recurrence Relations. *J. ACM* 41, 6 (Nov. 1994), 1136–1150. https://doi.org/10.1145/195613.195632
[14] Richard M. Karp and Vijaya Ramachandran. 1988. *A Survey of Parallel Algorithms for Shared-Memory Machines*. Technical Report UCB/CSD-88-408. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html
[15] Philip N. Klein. 1993. Parallelism, Preprocessing, and Reachability: A Hybrid Algorithm for Directed Graphs. *J. Algorithms* 14, 3 (May 1993), 331–343. https://doi.org/10.1006/jagm.1993.1017
[16] Philip N Klein and Sairam Subramanian. 1997. A Randomized Parallel Algorithm for Single-Source Shortest Paths. *J. Algorithms* 25, 2 (Nov. 1997), 205–220. https://doi.org/10.1006/jagm.1997.0888
[17] François Le Gall. 2014. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. 296–303. https://doi.org/10.1145/2608628.2608664
[18] Walter J. Savitch and Michael J. Stimson. 1979. Time Bounded Random Access Machines with Parallel Processing. *J. ACM* 26, 1 (Jan. 1979), 103–118. https://doi.org/10.1145/322108.322119
[19] Warren Schudy. 2008. Finding Strongly Connected Components in Parallel Using $O(\log^2 n)$ Reachability Queries. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*. 146–151. https://doi.org/10.1145/1378533.1378560
[20] Thomas H. Spencer. 1997. Time-work Tradeoffs for Parallel Algorithms. *J. ACM* 44, 5 (Sept. 1997), 742–778. https://doi.org/10.1145/265910.265923
[21] Mikkel Thorup. 1993. On shortcutting digraphs. In *Graph-Theoretic Concepts in Computer Science*, Ernst W. Mayr (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 205–211.
[22] Jeffrey D. Ullman and Mihalis Yannakakis. 1991. High Probability Parallel Transitive-closure Algorithms. *SIAM J. Comput.* 20, 1 (Feb. 1991), 100–125. https://doi.org/10.1137/0220006