SIAM J. COMPUT. © 2020 Society for Industrial and Applied Mathematics
Vol. 49, No. 5, pp. STOC18-500–STOC18-539

NEARLY WORK-EFFICIENT PARALLEL ALGORITHM FOR DIGRAPH REACHABILITY[*]

JEREMY T. FINEMAN[†]

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 article shows that the answer is yes, presenting 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 article 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.

Key words. parallel algorithms, randomized algorithms, graph search, reachability, shortcuts

AMS subject classifications. 68W10, 68W20, 05C85

DOI. 10.1137/18M1197850

1. Introduction.

There are essentially no parallel algorithms known to have provably good asymptotic guarantees 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] (In a slight abuse of notation, $\tilde{O}(1)$ is used to mean $O(\text{poly}(\log n))$, where

[*]Received by the editors July 2, 2018; accepted for publication (in revised form) September 29, 2020; published electronically October 22, 2020. An extended abstract appeared in the Proceedings of the 50th ACM Symposium on Theory of Computing, 2018.
https://doi.org/10.1137/18M1197850
Funding: This work was supported in part by NSF grants CCF-1718700, CCF-1617727, and CCF-1314633.

[†]Department of Computer Science, Georgetown University, Washington, D.C. 20057-1232 USA (jfineman@cs.georgetown.edu).

[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.

TABLE 1

Comparison of parallel algorithms for single-source reachability. Two of the algorithms are parameterized by ρ, $1 \le \rho \le n$, 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].

Algorithm	Work	Span	Nearly work efficient?	# of processors yielding $\tilde{O}(n/k)$ runtime for $m \in \Theta(n)$
Parallel BFS	$O(m)$	$\tilde{O}(n)$	Yes	N/A unless $k = \tilde{O}(1)$
Parallel TC	$\tilde{O}(M(n))$	$\tilde{O}(1)$	No	$\frac{kM(n)}{n}$ for $k \le n$
Spencer's [20]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$	if $\rho = \tilde{O}(\sqrt{\frac{m}{n}})$	k^3 for $k \le n$
UY [24]*	$\tilde{O}(m\rho + \frac{\rho^4}{n})$	$\tilde{O}(n/\rho)$	if $\rho = \tilde{O}(1)$	k^2 for $k \le n^{2/3}$ †
This paper*	$\tilde{O}(m)$	$\tilde{O}(n^{2/3})$	Yes	k for $k \le n^{1/3}$

*The algorithm is randomized. Bounds are with high probability.

†For higher k, the dependence on k becomes worse and more complicated to state.

the n should be clear from the 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 an algorithm with properties amenable to parallelization. Most details of the parallelization are straightforward, so the parallel model and realization 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 $m = |E|$ 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 (TC) [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

[2]The standard definition for soft-O is that $f(n) \in \tilde{O}(g(n))$ if $f(n) \in O(g(n)\,\mathrm{poly}(\log g(n)))$. This paper uses $f(n) \in \tilde{O}(g(n))$ to mean $f(n) \in O(g(n)\,\mathrm{poly}(\log n))$, with the only relevant difference being the meaning of $\tilde{O}(1)$.

[3]Older PRAM (parallel random access machine) 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.

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 is work/span tradeoffs. Ullman and Yannakakis [24] 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 [24], 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. Shortcut 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 shortcut graph. UY [24] 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 now the cost of computing the shortcuts. It is known that $O(n)$ shortcuts are sufficient to achieve $\tilde{O}(\sqrt{n})$ diameter—UY [24] 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 [23].) Moreover, as Hesse [9] shows that 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 that if a nearly work-efficient parallel algorithm for digraph reachability uses the shortcut 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 [24]): sample $\sqrt{n}$ vertices, perform

[4]The work bound stated by Ullman and Yannakakis [24] 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.

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($G[V_U]$)
```

a graph search from each, and add shortcuts between all pairs of sampled vertices that are related to each other (constituting $< n$ shortcuts). 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.
- (Section 6.) 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].)

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 affect 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 [24], 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 shortcut before it is divided into too many pieces.

The proof of this filtering insight (Lemma 3.4) leverages antisymmetric 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 realization (discussed briefly in section 5) is straightforward.

2. Preliminaries.

This section provides definitions, notation, 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 a path P into k pieces* means partitioning the path into k subpaths $\langle v_0, \ldots, v_{i_1}\rangle, \langle v_{i_1+1}, \ldots, v_{i_2}\rangle, \ldots, \langle v_{i_{k-1}+1}, \ldots, v_\ell\rangle$, where $0 \le i_1 < \cdots < i_{k-1} < \ell$.

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. (Although there is a generalization [22] of Karp's theorem designed specifically for analyzing parallel algorithms, the simplest form seems to be the most suitable because constant factors impact an exponent.) 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 $0 \le \Phi(I_r) \le \Phi(I_{r-1})$ for all relevant $r \ge 1$ and all feasible sequences $I_0, I_1, I_2, \ldots$ of instance outcomes.*

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

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 $\gamma \ge 1$ with the following guarantees, where $n = |\hat{V}|$, $m = |\hat{E}|$, and without loss of generality $m \ge n/2$:* (1) *the running time*

[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 $a(x) = 1$, $x \ge d$, 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 $u(\Phi(I_0)) = \lceil \log_{1/p}(\Phi(I_0)/d)\rceil \le k + 1$.

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 $G_{S^*} = (\hat{V}, \hat{E} \cup S^*)$ 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.

Subsections 3.1 and 3.2 set up the main ideas for proof of Lemma 1.1 but instead prove a weaker distance bound of $O(n^{1/\lg(8/3)}) = O(n^{0.7067})$. Subsection 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 shortcut 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 $P_1 = \langle v_0, \ldots, v_{k-1} \rangle$ is fully contained in $G[V_U]$ and $P_2 = \langle v_k, \ldots, v_\ell \rangle$ 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 $\langle v_0, \ldots, v_k \rangle$ is fully contained in $G[V_B]$ and $\langle v_{k+1}, \ldots, v_\ell \rangle$ 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, where 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 $i \leq k$. 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 unshortcut arcs in the final path, increases. Subsection 3.2 argues that these cases do make progress.

Algorithm 2: Algorithm corresponding to the flattened path-relevant tree.

```
PRAlg($G = (V, E), P = \langle v_0, v_1, \dots, v_\ell \rangle$)
1  select a pivot $x \in V$ uniformly at random
2  while $x$ is not related to $P$ do
3      remove from $G$ the vertices $R^-(G, x) \cup R^+(G, x)$ and their incident arcs
4      select a pivot $x \in V$ uniformly at random
5  $R^+ := R^+(G, x)$
6  $R^- := R^-(G, x)$
7  $V_F := R^+ \backslash R^-$ ;    $V_B := R^- \backslash R^+$ ;    $V_U := V \backslash (R^+ \cup R^-)$
8  if $x$ is a bridge then return the shortcuts $\{(v_0, x), (x, v_\ell)\}$
9  else if $x$ is an ancestor then
10     let $v_k$ be earliest vertex on $P$ in $V_F$
11     if $k = 0$ then return PRAlg($G[V_F], P$)
       else return PRAlg($G[V_U], \langle v_0, \dots, v_{k-1} \rangle$) ∪ PRAlg($G[V_F], \langle v_k, \dots, v_\ell \rangle$)
12 else // $x$ is a descendant
13     let $v_{k-1}$ be the latest vertex on $P$ in $V_B$
14     if $k - 1 = \ell$ then return PRAlg($G[V_B], P$)
       else return PRAlg($G[V_B], \langle v_0, \dots, v_{k-1} \rangle$) ∪ PRAlg($G[V_U], \langle v_k, \dots, v_\ell \rangle$)
```

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 $s = (G, P)$.

It is convenient to instead consider the *flattened path-relevant tree*, where each node corresponding to case 2 in Lemma 3.2 is merged with its only child. The reason is that the analysis proceeds level by level in the tree being analyzed, and all progress arguments rely on a random path-related vertex being selected. Algorithm 2 presents an algorithmic view of the flattened path-relevant tree being analyzed. Here multiple pivots are sampled until finally getting one that is related to the path, and at most two recursive calls are made. In light of Lemma 3.2 it is not hard to see that Algorithm 2 is equivalent to a subexecution of Algorithm 1, with some shortcuts and subproblems omitted. The analysis on the path length only leverages the shortcuts added in the base case of bridges.

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 simple 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 $O(2^r) + \sum_{i=1}^{k} \text{length}(P_i)$.*

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 $\{P_1, \dots, P_k\} \cup \left(\bigcup_{i=1}^{r-1} L_i\right)$ constitutes 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 shortcut paths for the leaves and the full unshortcut paths for the remaining subproblems. Each concatenation adds 1 arc, each leaf's path uses 2 shortcuts, and each remaining nonleaf 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 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 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 the 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' 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 the pivot were chosen to be $x = u$, then u' would remain 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 $P = \langle v_0, v_1, \ldots, v_\ell \rangle$. 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 paths 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 (the while loop in Algorithm 2), vertices and arcs may be removed adversarially. This step only reduces η further, so the worst case is that it does not occur at all. (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

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

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 any feasible collection I_{r-1} of subproblems at level $r-1$, we have $E[\Phi(I_r) | I_{r-1}] \leq (3/4)\Phi(I_{r-1})$.*

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, we have $\Pr\{\Phi(I_{r+5}) > (3/4)^r n\} < 1/2$. 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 shortcut 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 shortcut path has length $O(2^r) = O(2^{\log_{8/3} n}) = O(n^{1/\lg(8/3)})$. $\square$

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 compared to 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 the 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:

$$\phi(s) = \begin{cases} 0 & \text{if } s \text{ is not path-relevant,} \\ \psi(\alpha, \beta, \delta) & \text{otherwise,} \end{cases} \tag{3.1}$$

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) *(Partitionable.) 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 said to be c-reducing*, for constant $0 < c < 1$, if the* *following holds for all $\eta = \alpha + \beta + \delta > 0$:*

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

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

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* (3.1). *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 $E[\phi(s_1) + \phi(s_2)] \leq c \cdot \phi(s)$.*

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 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 the impact this reduction has on the potential. This step is analyzed in more detail below. (3) Adversarially partition 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. Note that as defined, α' (and similarly δ') does not include any former bridges that become ancestors (or descendants)—those are all counted in β'.

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

To complete the proof, as already noted $\phi(s_1) + \phi(s_2) \leq \psi(\alpha', \beta', \delta')$ by Definitions 3.8(iv) and 3.8(i). ◻

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, $E[\Phi(I_r)|I_{r-1}] \leq c \cdot \Phi(I_{r-1})$.*

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 $\alpha + \beta + \delta \leq n$. Since the function has overhead σ, it follows that $\Phi(I_0) = \psi(\alpha, \beta, \delta) \leq \sigma n$.

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 $\geq 1/2$, $c^r \sigma n > \Phi(I_{r'}) = \sum_{s=(G,P) \in I_{r'}} \phi(s) \geq \sum_{s=(G,P) \in I_{r'}} |Bridge(G,P)|$, 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 shortcut path of length at most $O(2^{r'}) + \Phi(I_{r'}) \leq O(2^r) + c^r \sigma n$ 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 $\psi(\alpha, \beta, \delta) \geq \beta$.

The remaining goal is to show that the function is a well-behaved, $(1/\sqrt{2})$-reducing function with overhead 1. As long as that is true, Lemma 3.11 directly implies Lemma 1.1, because $1/\lg(2/(1/\sqrt{2})) = 2/3$.

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

Proof. Most of the requirements of being well-behaved are trivial. The only difficult component is proving Definition 3.8(iv), i.e., that for $\beta_1 > 0$ and $\beta_2 > 0$,

$$\sqrt{(\alpha_1+\beta_1)(\delta_1+\beta_1)} + \sqrt{(\alpha_2+\beta_2)(\delta_2+\beta_2)} \\ \le \sqrt{(\alpha_1+\alpha_2+\beta_1+\beta_2)(\delta_1+\delta_2+\beta_1+\beta_2)}\ .$$

Let $y_i = \alpha_i + \beta_i$ for $i \in \{1,2\}$, and let $y = y_1 + y_2$. Similarly let $z_i = \delta_i + \beta_i$ and $z = z_1 + z_2$. The assumption that $\beta > 0$ implies that both $y > 0$ and $z > 0$. The proof focuses on the more general statement that for all $y = y_1 + y_2$ and $z = z_1 + z_2$, $\sqrt{y_1 z_1} + \sqrt{y_2 z_2} \le \sqrt{yz}$.

Let $\epsilon_y = y_1/y$ and $\epsilon_z = z_1/z$. It suffices to show for all $0 \le \epsilon_y, \epsilon_z \le 1$ that $\sqrt{\epsilon_y y \cdot \epsilon_z z} + \sqrt{(1-\epsilon_y)y \cdot (1-\epsilon_z)z} \le \sqrt{yz}$, or equivalently (dividing both sides by $\sqrt{yz}$) that $\sqrt{\epsilon_y \epsilon_z} + \sqrt{(1-\epsilon_y)(1-\epsilon_z)} \le 1$. Fix any ϵ_y (adversarially), treating $\sqrt{\epsilon_y}$ and $\sqrt{1-\epsilon_y}$ as constants. The expression is then maximized by setting $\epsilon_z = \epsilon_y$, so $\sqrt{\epsilon_y \epsilon_z} + \sqrt{(1-\epsilon_y)(1-\epsilon_z)} \le \epsilon_y + (1-\epsilon_y) = 1$, which completes the proof. ∎

LEMMA 3.13. *The function $\psi(\alpha,\beta,\delta) = \sqrt{(\alpha+\beta)(\delta+\beta)}$ is a $(1/\sqrt{2})$-reducing, well-behaved function with overhead* 1.

Proof. Let $\eta = \alpha + \beta + \delta$. That ψ is well-behaved is given by Lemma 3.12. Moreover, it has overhead 1 because $\sqrt{(\alpha+\beta)(\delta+\beta)} \le \eta$. The remainder focuses on showing that for all $\eta > 0$, $\frac{\alpha}{\eta}\sqrt{(\alpha/2+\beta)(\delta+\beta)} + \frac{\delta}{\eta}\sqrt{(\alpha+\beta)(\delta/2+\beta)} \le \frac{1}{\sqrt{2}}\sqrt{(\alpha+\beta)(\delta+\beta)}$.

Focus on one term at a time. The other term is symmetric. We have

$$\begin{aligned}
\left(\frac{\alpha}{\eta}\right)\sqrt{(\alpha/2+\beta)(\delta+\beta)} &= \left(\frac{\alpha}{\eta}\right)\sqrt{(1/2)\left(1+\frac{\beta}{\alpha+\beta}\right)(\alpha+\beta)(\delta+\beta)} \\
&= \left(\frac{\sqrt{(\alpha+\beta)(\delta+\beta)}}{\sqrt{2}}\right)\frac{\alpha\sqrt{1+\frac{\beta}{\alpha+\beta}}}{\eta} \\
&\le \left(\frac{\sqrt{(\alpha+\beta)(\delta+\beta)}}{\sqrt{2}}\right)\frac{\alpha\left(1+\frac{\beta}{2(\alpha+\beta)}\right)}{\eta} \\
&\qquad\qquad \text{(because } \sqrt{1+x} \le 1 + x/2 \text{ for } x \ge 0) \\
&\le \left(\frac{\sqrt{(\alpha+\beta)(\delta+\beta)}}{\sqrt{2}}\right)\frac{\alpha+\beta/2}{\eta}.
\end{aligned}$$

Adding both terms together, $(\alpha+\beta/2)/\eta + (\delta+\beta/2)/\eta = \eta/\eta = 1$, so we have that their sum is at most $1/\sqrt{2} \cdot \sqrt{(\alpha+\beta)(\delta+\beta)}$, as desired.

Though not necessary for the lemma, we close the proof by observing that the inequality $\frac{\alpha}{\eta}\sqrt{(\alpha/2+\beta)(\delta+\beta)} + \frac{\delta}{\eta}\sqrt{(\alpha+\beta)(\delta/2+\beta)} \le \frac{1}{\sqrt{2}}\sqrt{(\alpha+\beta)(\delta+\beta)}$ is tight. Consider $\beta = 1$ and $\alpha = \delta \approx \eta/2$ in the limit that η approaches infinity. As η increases, β's impact on the expression decreases. We are thus left with $\frac{1}{2}\sqrt{(\alpha/2)\delta} + \frac{1}{2}\sqrt{\alpha(\delta/2)} \approx \frac{1}{\sqrt{2}}\sqrt{\alpha\delta}$ ∎

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

Algorithm 3: 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 $S$
```

This section instead analyzes Algorithm 3. Algorithm 3 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, Algorithms 1 and 3 are equivalent.

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

LEMMA 3.14. *Consider an execution of Algorithm 1 and the corresponding flattened path-relevant tree of Algorithm 2. When mapped to an execution of Algorithm 3 with the same random choices, the first $\lg n - 1$ levels of the flattened tree all have recursion depth less than $\lg n$ in Algorithm 3.*

Proof. The flattened tree only merges some of the calls corresponding to $G[V_U]$. Algorithm 3 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 3.15. *Consider a graph $\hat{G} = (\hat{V}, \hat{E})$, and let $n = |\hat{V}|$. Each execution of Algorithm 3 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 whose sets are removed from G at the end of the iteration. Thus, there can be at most $2|V|$ shortcut 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 3.16. *Consider a graph $\hat{G} = (\hat{V}, \hat{E})$, and let $n = |\hat{V}|$ and $m = |\hat{E}|$. Algorithm 3 can be implemented to run in $O(m \log n)$ time.*

Proof. The proof is similar to that of Lemma 3.15, 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 to 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 3. 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 3.16 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 realization is deferred to section 5. 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 3, 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 3. 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 5. Subsection 4.1 walks through the ideas incrementally, guided by rough intuitions behind the analysis. The key performance lemmas, analogous to Lemma 1.1, are the following. The first lemma states that the resulting path is short enough; the second lemma bounds performance in number of shortcuts, total work performed, and maximum distance searched.

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 $m \geq n/2$. There exist settings of $D = \Theta(n^{2/3}\log n)$ and other constant parameters in Algorithm* 5 *such that the following holds.*

Consider a directed u-to-v path $\hat{P}$ with length$(\hat{P}) \leq D$. Let S be the shortcuts produced by an execution of Algorithm 5 *on $\hat{G}$. Then with probability at least* 7/10, *there exists a path from u to v in $G_S = (\hat{V}, \hat{E} \cup S)$ having length at most $D/2$.*

LEMMA 4.2. *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 $m \geq n/2$. There exist settings of constant* *parameters in Algorithm* 5*, consistent with the settings in Lemma* 4.1*, such that the following holds.*

Consider an execution of Algorithm 5 *on* $\hat{G}$ *and let* S *be the shortcuts produced. Then with probability at least* 9/10*,* (1) *the number of shortcuts produced is* $|S| = O(n \log^2 n)$*,* (2) *the total number of vertices and arcs visited by searches is* $O(m \log^2 n)$*, and* (3) *the maximum distance used for any search is* $O(n^{2/3} \log^{12.5} n)$*.*

By a union bound, with at least constant probability, Lemmas 4.1 and 4.2 both enter the success cases. Thus, using multiple runs of Algorithm 5 (see section 4.2), aborting each run immediately if the work or shortcut bound of Lemma 4.2 is violated, yields the following.

THEOREM 4.3. *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* $O(n^{2/3} \log^{12.5} n)$*;* (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. Subsection 4.1 describes the main subroutine, namely, Algorithm 5. Subsection 4.2 extends the algorithm to perform multiple passes, thereby obtaining Theorem 4.3. Lemma 4.12 and subsection 4.9 capture the bulk of the analysis, proving Lemma 4.1. Subsection 4.10 finishes the proof of Lemma 4.2 by analyzing the number of shortcuts and work performed.

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 notation and definitions in section 2 that depend on $\preceq$ (i.e., successors, predecessors, ancestors, descendants, 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.

Algorithm 4 gives an overview of the algorithm, with details omitted. Full pseudocode with more details is provided in Algorithm 5.

Algorithm 4 is presented with a recursive structure analogous to Algorithm 1, except that multiple pivots are considered in a single call. To capture the details (Algorithm 5) about pivot sets, however, it is more convenient to express the algorithm by a loop as in Algorithm 3.

The algorithm includes several key features, including distance-limited searches, duplicated fringe vertices, random search distances, and multiple pivots. Each of the features is explained incrementally and in more detail next, thinking in terms of the sequential algorithm as the starting point. The multiple pivots are the last feature added, so the reader should ignore the j's in the subscripts until that point.

Distance-limited searches. Recall that the goal is to replace the searches $R^+(G, x)$ in Algorithm 3 with D-limited searches for some $D = \tilde{O}(n^{2/3})$. The good news is that it is still possible to show progress on the number of ancestors, analogous to Lemma 3.4. Informally, if the pivot is chosen randomly from a D-limited ancestor, then the number of D-limited ancestors drops by 1/2 in expectation.

The bad news is that when partitioning the graph with D-limited searches, a

Algorithm 4: Outline of the distance-limited algorithm.

```
ParOverview($G = (V, E)$)
1 Select a group $X$ of random pivots.
2 Choose a random distance to search.
3 Core vertices. Perform distance-limited forward and backward "core" searches
  in $G$ from each pivot $x_j \in X$.
  Add shortcuts between each pivot and the vertices found in its searches.
  The searches partition the vertex set into the following $2|X| + 1$ subsets:
      • For each $j$, $V_{B,j}$ denotes the vertices (i) not reached by any core searches
        from earlier pivots, and (ii) reached by only the backward search from $x_j$.
      • For each $j$, $V_{F,j}$ denotes the vertices (i) not reached by any core searches
        from earlier pivots, and (ii) reached by only the forward search from $x_j$.
      • $V_U$ denotes vertices not reached by any core searches.
4 Fringe vertices. Extend each search to a slightly larger distance, again adding
  shortcuts from/to any vertices reached.
  For each pivot $x_j$, $F_j^-$ denotes the "fringe vertices" reached by $x_j$'s extended back-
  ward search but not the core backward search. Similarly, $F_j^+$ denotes the vertices
  newly discovered by the extended forwarded search.
5 foreach $x_j \in X$ do
6     recurse on ParOverview($G[V_{B,j} \cup F_j^-]$) and ParOverview($G[V_{F,j} \cup F_j^+]$)
7 recurse on ParOverview($G[V_U]$)
```

particular path may be shattered into far more than two pieces (contrary to Lemma 3.2 with unlimited searches), and as such the $O(2^r)$ term of Lemma 3.3 does not readily apply. For example, consider a path $P = \langle v_0, v_1, \ldots, v_\ell \rangle$ and single ancestor pivot x. It is possible to construct a graph such that $x \preceq_D v_k$ for even k, but $x \not\preceq_D v_k$ for odd k. Thus, all the even vertices on P would be in V_F, and all the odd vertices would be in V_U, splitting the path into $\Theta(\ell)$ pieces. This splitting violates the general assumption that each path-relevant subproblem contains a single subpath to shortcut. In contrast, when the search is *not* D-limited, $x \preceq v_j$ implies $x \preceq v_j$ for all $k \geq j$.

Fringe vertices and core vertices. 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 3. Then extend the search a little further, 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. The graph is partitioned according to the core searches as before, but the fringe vertices and incident arcs are duplicated and included in two subproblems. Specifically, the three subproblems are now $G[V_U]$, $G[V_B \cup F^-]$, and $G[V_F \cup F^+]$, where V_U, V_B, and V_F are defined as before with respect to core searches.

The addition of fringe vertices fixes the path-splitting problem, giving an analogue of Lemma 3.2, at least for paths of length $\ell \leq D$. Consider again the bad example where $x \preceq_{dD} v_k$ if and only if k is even. 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 $G[V_F \cup F^+]$.

Random distances. Unfortunately, duplicating fringe vertices introduces another problem—path-related fringe vertices can be active in multiple subproblems.

Specifically, it is not hard to construct examples such that for a given pivot (i) almost all of the path-related vertices are fringe vertices, and (ii) most of those vertices are active in two path-relevant subproblems. As such, the potential ϕ of a subproblem could increase dramatically when recursing, destroying the progress arguments of section 3.

The solution is to select d (for search distance dD) randomly from a range of $N_L - 1$ consecutive values for some $N_L = \tilde{O}(1)$ to be chosen later. (Read N_L as "number of layers.") 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.

Decreasing distances. The progress arguments on the potential of active vertices shall now be with respect to a particular distance. As such, it is important that distances searched never increase. The distance d is always selected from an interval of size $N_L - 1$ possibilities, but at some offset depending on recursion depth and, in the more precise Algorithm 5, the iteration number. (The larger the depths, the smaller the offset.)

Specifically, for any particular iteration of the for loop in Algorithm 5, the distance multiplier d is selected from some range $\{d_{\min}, d_{\min}+1, \ldots, d_{\max}\}$, with $d_{\max} = d_{\min} + N_L - 1$. The offset $d_{\min}$ decreases by N_L with each iteration, so there is no overlap between consecutive intervals. The offset decreases to the next multiple of $N_k N_L$ when recursing, where N_k is an upper bound on the number of iterations needed.[7] More precisely, let h be the number of levels of recursion yet to perform, and i the iteration number. Then $d_{\min} = 1 + hN_kN_L - iN_L$.

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 small. These details are expressed only with respect to Algorithm 5, not the overview.

For each recursive call to `ParSC`, pivots are chosen as follows. Before performing any iterations, randomly permute the vertices and mark all vertices as *live*; being *live* indicates that the vertex is still in the graph that remains in the current iteration. Next, partition the sequence into groups $X_1, X_2, \ldots, X_{2k}$, for some value k discussed below. For the ith iteration, the pivots selected are those vertices in group X_i that are still live. At the end of each iteration, mark any vertices no longer in the remaining graph (i.e., those reached by any core searches) as dead.

There are three important features in the sizing of the subsequences X_i, each important to some aspect of the analysis. (1) There should not be too many groups, as the maximum number $N_k \geq 2k$ of groups impacts the maximum search distance. (2) The group sizes should be no more than geometrically increasing, i.e., $|X_{i+1}| / |X_i|$ should be upper bounded by a constant. This feature is the standard one, and it ensures that each vertex is not reached by too many searches in a single iteration. (3) The size of X_i should be small relative to the total number of remaining pivots. Specifically, whenever $|X_i| > 1$, $|X_i| / \sum_{j=i+1}^{2k} X_j$ should be upper bounded by some small value.

[7]The larger decrease is not strictly necessary; decreasing the offset by N_L when recursing would also suffice. The larger decrease was chosen just so $d_{\min}$ could be expressed in closed form by level of recursion and iteration within that recursive call.

Algorithm 5: Shortcutting algorithm with distance-limited searches.

```
ParSC($G = (V, E)$)
  /* Here $n = |\hat{V}|$ denotes the size of the original graph, which is
     constant with respect to this algorithm. Similarly the values
     $\epsilon_\pi$, $N_k$, $N_L$, and $D$ are also constants, to be set later.        */
1  Let $h = \lg n - depth$, where $depth$ is the current recursion depth
2  if $h = 0$ then return $\emptyset$
3  $S := \emptyset$
4  randomly permute $V$, giving a sequence $X$ of all vertices. Mark each $x_j \in X$ live
5  split $X$ into subsequences $X_1, X_1, \ldots, X_{2k}$, with $|X_i| = |X_{2k-i+1}| = \lfloor (1+\epsilon_\pi)^i \rfloor$ for
     $i < k$ and $|X_k|, |X_{k+1}| \leq \lfloor (1+\epsilon_\pi)^k \rfloor$
6  for $i := 1$ to $2k$ do
7      $d_{\min} = 1 + hN_kN_L - iN_L$                                   // offset
       $d_{\max} = d_{\min} + N_L - 1$
8      choose random $d \in \{d_{\min}, d_{\min}+1, \ldots, d_{\max}-1\}$
9      foreach live $x_j \in X_i$ do
10         $R_j^- := R_{dD}^-(G, x_j)$
           $R_j^+ := R_{dD}^+(G, x_j)$                                 // core vertices
11         $F_j^- := R_{(d+1)D}^-(G, x_j) \backslash R_j^-$ ;
           $F_j^+ := R_{(d+1)D}^+(G, x_j) \backslash R_j^+$                       // fringe vertices
12         $S := S \cup \{(x_j, v) | v \in R_j^+ \cup F_j^+\} \cup \{(u, x_j) | u \in R_j^- \cup F_j^-\}$
                                                                      // add shortcuts
13         append a tag of $j$ to all vertices in $R_j^+ \cup R_j^-$
14     foreach live $x_j \in X_i$ do
15         remove vertices with tag $< j$ from $R_j^+$, $R_j^-$  // first core search wins
16         $V_{F,j} := R_j^+ \backslash R_j^-$ ;        $V_{B,j} := R_j^- \backslash R_j^+$
17         $S := S \cup$ ParSC($G[V_{F,j} \cup F_j^+], h-1$) $\cup$ ParSC($G[V_{B,j} \cup F_j^-], h-1$)
            // include fringe
18     mark all vertices in $\bigcup_j (R_j^+ \cup R_j^-)$ as dead in $X$
19     $V_U := V \backslash \bigcup_j (R_j^+ \cup R_j^-)$
20     $G := G[V_U]$
21 return $S$
```

To satisfy all three of these features, the group sizes are specified with respect to parameter ϵ_π (π for permutation), to be chosen later. Roughly speaking, the first k groups have sizes geometrically increasing by $(1 + \epsilon_\pi)$, and the remaining k groups have sizes geometrically decreasing by $(1 + \epsilon_\pi)$. More precisely, for $i < k$, $|X_i| = |X_{2k-i+1}| = \lfloor (1+\epsilon_\pi)^i \rfloor$. For the middle two groups, the size is upper bounded by $\lfloor (1+\epsilon_\pi)^k \rfloor$. The value k is the smallest value such that $2\sum_{i=1}^{k} \lfloor (1+\epsilon_\pi)^i \rfloor \geq |V|$.

4.2. Full diameter-reduction algorithm and proof of Theorem 4.3. Like the algorithm in section 3, achieving diameter reduction with high probability requires multiple passes of Algorithm 5. But now more passes are necessary. The full algorithm, shown in Algorithm 6, is as follows. Perform $\Theta(\log n)$ iterations. In each iteration, perform $\Theta(\gamma \log n)$ independent executions of Algorithm 5 on the current

Algorithm 6: Diameter reduction with distance-limited searches.

```
ParDiam($\hat{G} = (\hat{V}, \hat{E}, \gamma)$)
  /* The value $\gamma \geq 1$ 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 4.2
5     $E' := E' \cup \left(\bigcup_j S_j\right)$                     // add more arcs to $G'$
6 return $G'$
```

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 5 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 4.3, assuming Lemmas 4.1 and 4.2. 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 6. The main claim is that with probability at least $1 - 1/n^{2+\gamma}$, for all i we have $\Delta_i \leq D \cdot \max\left\{n/(D2^i), 1\right\}$. For $i = \Omega(\log n)$, i.e., when the main procedure returns, this reduces to $\Delta_i \leq D$. 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 $\Delta_0 \leq n = D \cdot n/(D2^0)$. 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 $\mathit{length}(P) \leq D$, 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 5 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 $(D/2) \cdot n/(D2^i) = n/(2^{i+1})$, 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 4.2. The number of shortcuts follows from Lemma 4.2 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 4.2, each run of Algorithm 5 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 5. 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 5, using the variables to implicitly adopt the meaning of the code.

Concretely, for iteration i on graph $G = (V, E)$, the following notation is used with the same meaning as the pseudocode: h, X_i meaning the group of pivots; $d_{\min}$ and $d_{\max}$ meaning the appropriate minimum and maximum distance for that iteration; and d indicating the specific random distance chosen. Moreover, for each $x_j \in X_i$, whenever the notation R_j^+, R_j^-, F_j^+, F_j^-, $V_{F,j}$, or $V_{B,j}$ appears, they should also be interpreted to have the meaning laid out in the pseudocode.

The minimum and maximum distances $d_{\min}$ and $d_{\max}$, respectively, of the iteration are also useful for characterizing vertex relationships as follows.

DEFINITION 4.4. *Consider any iteration i of Algorithm 5. To unclutter the notation, $\preceq_{\min}$ is used as a shorthand for $\preceq_{d_{\min}D}$, where $d_{\min} = 1 + hN_kN_L - iN_L$. Similarly, $\preceq_{\max}$ is a shorthand for $\preceq_{d_{\max}D}$.*

- *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. Specifically, if v is fully related to at least one vertex in P, then v and P are fully related. Similarly, if v is partly related to at least one vertex in P, then v and P are partly related. They are never related if all vertices in P are never related to v.

4.4. Setup and the path-relevant tree.

The definition of path-relevant subproblems and the path-relevant tree differ slightly from section 3.1 to account for the key differences in the algorithm. But the initial choices here are analogous.

A path-relevant subproblem, denoted by the pair (G, P), corresponds to an iteration of the for loop in Algorithm 5 (matching the recursion in Algorithm 4); G denotes the graph at the start of the iteration, and P is a nonempty subpath to shorten.

The following lemma, analogous to Lemma 3.2, considers the effect of an iteration on the path-relevant subproblems. Unlike Lemma 3.2, there may be more than two path-relevant subproblems.

LEMMA 4.5. *Let $P = \langle v_0, \ldots, v_\ell \rangle$ be a nonempty path in $G = (V, E)$ with $\ell \le D$ and consider the effect of a single iteration of the for loop in Algorithm 5. The following are the outcomes depending on group of pivots X_i and random distance d:*

1. *(Base case.) If X_i contains a live dD-limited bridge of P, then the shortcuts (v_0, x) and (x, v_ℓ) are created. There are no path-relevant subproblems.*
2. *If none of the live pivots in X_i is dD-related to P, then P is entirely contained in $G[V_U]$; the one path-relevant subproblem is thus $(G[V_U], P)$, the next iteration.*
3. *Suppose that k of the live pivots in X_i are dD-related to P but that none of them is a bridge. Let $x_1, \ldots, x_k$ denote these pivots. Then P can be partitioned into k subpaths $P_0, P_1, \ldots, P_k$ (listed in no particular order) such that* (i) *P_0 is fully contained in $G[V_U]$, and* (ii) *each P_j is fully contained in either $G[V_{F,j} \cup F_j^+]$ or $G[V_{B,j} \cup F_j^-]$. It follows that (G, P) gives rise to at most $k + 1$ path-relevant subproblems.*

Proof. (Case 1.) Suppose that some dD-related bridge x_j is selected. Then by definition there exist vertices $v_a, v_b \in P$ such that $v_a \preceq_{dD} x_j \preceq_{dD} v_b$. Since the path has length at most D, it follows that $v_0 \preceq_{(d+1)D} x_j \preceq_{(d+1)D} v_\ell$. Since v_0 and v_ℓ are within the fringe search distance, and shortcuts are added to all served vertices, the claimed shortcuts are added.

(Case 2.) None of the vertices is dD-related to P. Then none of the vertices in P is removed from V_U.

(Case 3.) Consider the dD-related pivots in rank order and the effect of processing each subsequent pivot, where P_0 is updated as each pivot is incorporated. Initially, $P_0 = P$. For the inductive step, suppose that the claim is true for the first $j-1$ related pivots.

Consider the jth pivot x_j. First, observe that because the searches are prioritized in pivot order, earlier core and fringe neighborhoods are unaffected by x_j. Thus, the only change due to this pivot is to P_0.

Let $P_0 = \langle v_a, \ldots, v_c \rangle$ be the subpath just before processing pivot x_j. Suppose that x_j is a dD-limited ancestor. (The case for a descendant is symmetric.) Then there exists an earliest vertex v_b on P such that $x_j \preceq_{dD} v_b$. Since v_b is the earliest such vertex, and x_j is not a bridge, no vertex on P before v_b is in a core search from x_j. It follows that the prefix $\langle v_a, \ldots, v_{b'-1} \rangle$ of P_0, where $b' = \min\{b, c\}$, remains in V_U after processing x_j. Since P has length $\ell \leq D$, x_j can reach all vertices on $\langle v_b, v_{b+1}, \ldots, v_\ell \rangle$ by a $(d+1)D$-limited search. It follows that the suffix $\langle v_{b''}, \ldots, v_c \rangle$ of P_0, where $b'' = \max\{a, b\}$, is fully contained in $G[V_{F,j} \cup F_j^+]$. ∎

As in section 3, the bulk of the analysis is with respect to the *flattened* path-relevant tree. The big question is when nodes should be flattened.

The most natural choice—flatten when case 2 applies—turns out not to work. The issue is the interaction between the two types of random choices. For some components of the analysis, it is convenient to leverage the fact that pivots are chosen uniformly at random. For other components, it is convenient to leverage the uniformly random distances. Conditioning on the assumption that a search from a pivot actually reaches the path at the chosen distance, however, biases those distributions in ways that are difficult to quantify.

Instead, interpret the random processes surrounding an iteration as follows:

1. First, reveal only how many live pivots $x_j \in X_i$ are partly related to the path. Flatten any iterations in which all pivots are either dead or never path related. An equivalent algorithm view is to repeatedly process groups until finally reaching an X_i that contains at least one partly path-related pivot. Pessimistically charge for the path being split into $k+1$ path-relevant subproblems, where k is the number of partly path-related pivots.
2. Next, reveal the pivots. This step is enough to bound the progress on potential with respect to core nodes.
3. Finally, reveal the random distance and bound the impact of fringe nodes.

The following lemma, analogous to Lemma 3.3, follows immediately.

LEMMA 4.6. *Consider any graph $\hat{G} = (\hat{V}, \hat{E})$ and any path $\hat{P} : u \rightsquigarrow v$ with length$(P) \leq D$. Let $n = |\hat{V}|$. Next consider an execution of Algorithm 5, and let S denote the set of shortcuts produced by the execution.*

Choose any level $r \leq \lg n$ to analyze. Let $I_0, I_1, \ldots, I_r$ denote the collection of path-relevant subproblems at the first r levels in the flattened tree. Let $\phi(s)$ be any function on subproblems $s = (G, P)$ satisfying $\phi(s) \geq$ length(P), and let $\Phi(I_r) = \sum_{s \in I_r} \phi(s)$. Then there is a u-to-v path in $G_S = (\hat{V}, \hat{E} \cup S)$ of length at most $O(\sum_{j=0}^{r} |I_j| + \Phi(I_r))$.

Proof. A simple induction over levels shows that all vertices on path $\hat{P}$ belong to either a subpath in I_r, or to a subpath at level $< r$ that has been shortcut to 2 hops. There thus exists a path that consists of the concatenation of any 2-hop bypasses with any subpaths that survive to level r in the tree. ∎

Unlike Lemma 3.3, where the number of subproblems at level r is $O(2^r)$ in the worst case, here the total number of subproblems is also a random variable. As such, we must prove not only that the potential Φ reduces sufficiently fast (section 4.8), but also that the number of subproblems is small (section 4.6).

4.5. Bounds on number of vertices searched.

Because the algorithm now searches from multiple pivots, vertices can be reached by multiple searches in a single path-relevant subproblem. These overlapping searches impact the size of the flattened path-relevant tree (Lemma 4.6), the number of fringe vertices, and the overall work performed. This section proves bounds that help to limit the negative impact of these effects.

The main observation is that with high probability each surviving vertex does not have too large a $d_{\max}D$-related neighborhood. As such, no vertex is reached by very many searches. To avoid any dependencies introduced by the fact that the analysis only focuses on iterations that include at least one partly path-related pivot, the failure case is lifted to the following definition. The full performance theorem must also include the impact of a failure occurring anywhere.

DEFINITION 4.7. *Consider a call to* `ParSC`$(G = (V, E))$ *and iteration i within the call. Let $y = \sum_{i'=1}^{i-1} |X_{i'}|$ denote the number of candidate pivots processed before iteration i begins. Let $G_i = (V_i, E_i)$ be the subgraph remaining at the start of the ith iteration. Let $\tau = |V| \ln n/y$, where $n = |\hat{V}|$ is the size the full graph.*

A **γ-neighborhood failure** *for iteration* ***i*** *is said to occur if there exists a $v \in V_i$ such that $|R^-_{d_{\max}D}(G_i, v)| > \gamma\tau$ or $|R^+_{d_{\max}D}(G_i, v)| > \gamma\tau$.*

The notion of neighborhood failures is used purely to drive the key lemmas of this section, each of which is conditioned on no neighborhood failures. The probability of such a failure occurring is bounded at the end of the section.

The next lemma says that, with high probability, no vertex is reached by more than $O(\log n)$ searches. This lemma is used to bound the total work performed as well as the number of fringe vertices.

LEMMA 4.8. *Consider any iteration i, suppose that there is no γ-neighborhood failure for i, and let X_i be the random set of pivots selected in iteration i. Suppose also that $\epsilon_\pi \leq 1$ and $\gamma \geq 1$.*

Then with failure probability $1/n^{\Theta(\gamma)}$, no vertex is $d_{\max}D$-related to more than $O(\gamma \log n)$ live pivots in X_i.

Proof. Use $G = (V, E)$ to refer to the graph at the beginning of the call, before the first iteration of the loop. Let y be the number of pivots considered before the iteration in question, and let v be a vertex to analyze. By choice of ϵ_π, $|X_i| \leq 3y$. (It can only be this large due to roundoff.) By assumption, v has $O(\gamma |V| \log n/y)$ $d_{\max}D$-limited predecessors and successors at the start of the iteration.

There are two cases. If $y > |V|/8$, then $O(\gamma |V| \log n/y) = O(\gamma \log n)$. Searching from *every* remaining vertex would thus only result in $O(\gamma \log n)$ searches reaching v.

If $y \leq |V|/8$, then $y + |X_i| \leq |V|/2$. For every pivot position $x_j \in X_i$, there are thus at least $|V|/2$ options to draw from. So we have $\Pr\{x_j \text{ live and } x_j \preceq_{dD} v\} = O(\gamma |V| \log n/(y |V|)) = O(\gamma \log n/y) = O(\gamma \log n/|X_i|)$, where the last step follows from $|X_i| \leq 3y$. Applying a Chernoff bound across all $|X_i|$ pivot positions, the number of searches that reach v is $O(\gamma \log n)$ with failure probability $1/n^{\Omega(\gamma)}$. To complete the proof, take a union bound across all v. ∎

Fringe searches. The next lemma bounds the number of fringe vertices and arcs explored in a single iteration i. Note that a particular vertex may be a fringe vertex for multiple searches. The lemma counts the total number of times that each vertex appears on the fringe. Similarly, arcs may be explored by multiple fringe searches, but just once per search. (An arc (u, v) is explored by x_j's fringe search if either u or v is a fringe vertex.)

The impact of fringe searches is important in two places. To bound progress with respect to potential, we care only about fringe vertices that are also path related. To bound total work, we care about all fringe vertices. The sets V' and E' model in the lemma captures both of these options.

LEMMA 4.9. *Consider an iteration i and $N_L \geq 2$. Assume that no γ-neighborhood failure has occurred for iteration i for large enough choice of constant γ. Let V' and E' denote any subset of vertices and arcs, respectively, that remain in the graph at the start of iteration i.*

Suppose that distance multiplier d is chosen uniformly at random from $N_L - 1$ options. Then the total number of fringe vertices from V' is $O(|V'| \log n / N_L)$ in expectation. Similarly, the total number of arcs explored by fringe searches is $O(|E'| \log n / N_L)$ in expectation.

Proof. By Lemma 4.8, with high probability each vertex in V' is $d_{\max}D$-related to at most $O(\log n)$ pivots and hence reached by at most $O(\log n)$ searches. A failure event can only increase the expectation by an additive $\Pr\{\text{failure}\} \cdot \left|\hat{V}\right|^2 \leq (1/n^{\gamma})n^2 \ll \log n$ for appropriate choice of constant γ in the high-probability bound. For the remainder, fix any (adversarial) set of pivots subject to the constraint that at most $O(\log n)$ of them are $d_{\max}D$-related to each vertex in V'.

Consider any $v \in V'$ and pivot x_j. Let Y_j^v be an indicator random variable for the event that v is on x_j's fringe, where $Y_j^v = 0$ if x_j and v are never related. For a partly related x_j, v is only on the fringe at one distance, so $\Pr\left\{Y_j^v\right\} \leq 1/(N_L - 1) \leq 2/N_L$. The total number of times v is on the fringe is thus $E[\sum_{x_j \in X_i} Y_j^v] = O(\log n) \cdot (2/N_L) = O(\log n / N_L)$. Summing across all v gives $E[\sum_{v \in V'} \sum_{x_j \in X_i} Y_j^v] = O(|V'| \log n / N_L)$.

The same argument applies to arcs, observing that the arc is explored whenever its endpoints are at the right distance. ∎

Number of subproblems. The following is used to bound the expected fanout of nodes in the flattened path-relevant tree.

LEMMA 4.10. *Consider any path-relevant subproblem (iteration i), and suppose that no γ-neighborhood failure occurs for i.*

Let z denote the number of partly path-related pivots selected in this iteration. Then $E[z | z \geq 1] \leq 1 + O(\epsilon_\pi \log n)$.

Proof. Let $G = (V, E)$ denote the graph before the first iteration of the loop in the containing call. By assumption, v has $O(\gamma |V| \log n / y)$ dD-limited predecessors and successors, where y is the number of pivots processed before this iteration.

Since $z \leq |X_i|$, the claim is trivial for $|X_i| = 1$. Suppose for the remainder that $|X_i| > 1$. Consider the suffix X_i' of pivots following the first partly path-related pivot. All pivots before that one may be chosen adversarially.

Let $y' = |V| - y - |X_i|$ denote the number of pivots to process strictly after this iteration. A key observation is that, except for the first and last $\Theta(1/\epsilon_\pi)$ iterations (for which $|X_i| = 1$), $|X_i| = O(y'\epsilon_\pi)$ and symmetrically $|X_i| = O(y\epsilon_\pi)$. The same bounds also apply to $|X'_i|$.

If $y > |V|/8$, then $O(\gamma |V| \log n/y) = O(\log n)$, meaning that v has only $O(\log n)$ dD-limited predecessors or ancestors remaining in total. In expectation, X'_i includes at most a $X'_i/(X'_i + y') \leq X'_i/y' = O(\epsilon_\pi)$ fraction of them. Thus, the expected number of partly path related pivots in X'_i is $O(\epsilon_\pi \log n)$.

If $y \leq |V|/8$, let Y_j be an indicator random variable for the event that $x_j \preceq_{dD} v$. Since there are $\Omega(|V|)$ pivots to choose from, $\Pr\{Y_j\} = O(|V| \log n/(y|V|)) = O(\log n/y)$. It follows that $E[\sum_{x_j \in X'_i} Y_j] = |X'_i| \cdot O(\log n/y) = O(\epsilon_\pi \log n)$. ∎

Finally, the following lemma bounds the probability of a γ-neighborhood failure occurring.

Lemma 4.11. *Consider a call* `ParSC(`$G = (V, E)$`)`. *The probability of any γ-neighborhood failure occurring during this call is at most $n^{\gamma-3}$*

Proof. Fix an iteration i and vertex $v \in V$ to analyze up front. Let y, G_i, and τ be as defined in Definition 4.7. The proof focuses on bounding the probability that v survives to the ith iteration and $\left|R^-_{d_{\max}D}(G_i, v)\right| > \gamma\tau$. Specifically, we shall show that the failure probability for v at iteration i is at most $1/n^\gamma$. Taking a union bound across both direction, n vertices, and $\leq n$ iterations completes the proof.

All searches before iteration i have distance larger than $d_{\max}D$. Moreover, arcs are not added to the graph of the next iteration, so the number of $d_{\max}D$-limited predecessors of v can only decrease or stay the same with each iteration. Thus, if v is to end with more than $\gamma\tau$ live predecessors, then more than $\gamma\tau$ of the vertices from the initial set $R^-_{d_{\max}D}(G, v)$ of predecessors must remain live throughout. The goal is thus to bound the probability that this event can occur without knocking v out of V_i.

For $y < \gamma \ln n$, the claim is vacuous ($\gamma\tau \geq \gamma(|V| \ln n)/(\gamma \ln n) = |V|$ in this case). Suppose instead that $y \geq \gamma \ln n$. Let $x_1, x_2, \ldots, x_y$ denote the sequence of pivots chosen before iteration i begins. Consider selecting each of them at random in turn. For each x_j, there are at most $|V|$ remaining choices. While the number of $d_{\max}D$-limited predecessors is above threshold, we thus have $\Pr\{x_j \text{ live and } x_j \preceq_{d_{\max}D} v\} \geq \gamma\tau/|V| = \gamma \ln n/y$. If any such pivot is selected, v would be included in a core search and thus not in V_i. The probability that no such pivot is selected is at most $\prod_{j=1}^{y} \Pr\{x_j \text{ dead or } x_j \not\preceq_{d_{\max}D} v\} \leq (1 - \gamma \ln n/y)^y \leq 1/e^{\gamma \ln n} = 1/n^\gamma$. ∎

4.6. Number of nodes in the flattened tree. We are now ready to bound the number of nodes in the flattened tree.

Lemma 4.12. *Consider any graph $\hat{G} = (\hat{V}, \hat{E})$ and any path $\hat{P} : u \rightsquigarrow v$ with length$(P) \leq D$. Let $n = |\hat{V}|$. Consider an execution of Algorithm 5 with parameter $\epsilon_\pi \leq 1/\lg^3 n$. Let $r \leq \lg n$ be any level to analyze and assume that no γ-neighborhood failure occurs in these first r levels.*

Let $I_0, I_1, \ldots, I_r$ denote the collection of path-relevant subproblems in the first r levels of the flattened tree. With failure probability at most $1/10$, $\sum_{j=0}^{r} |I_j| = O(2^r)$.

Proof. Consider a particular level j. The goal is to show that with failure probability at most $1/(10 \lg n)$, we have $|I_{j+1}| \leq 2(1 + O(1/\log n))|I_j|$. By a union bound, the probability that a failure occurs at any level is thus at most $1/10$, and as long as no failure occurs, the total number of subproblems is $(2(1+O(1/\log n)))^r = O(2^r)$ for $r = O(\log n)$. The remainder focuses on bounding the relative sizes between two levels.

Consider I_j and number the subproblems $1, 2, \ldots, q$. Let z_t be random variables denoting the number of partly path-related pivots selected in the tth subproblem, and let $z'_t = z_t - 1$. By Lemma 4.5, each pivot incurs one additional subproblem. Thus, the number of subproblems at level $j+1$ is at most $\sum_{t=1}^{q}(1+z_t) = \sum_{t=1}^{q}(2+z'_t) = 2q + \sum_{t=1}^{q} z'_t$. Let $Z = \sum_{t=1}^{q} z'_t$, meaning that the number of subproblems is $2q + Z$. Note that q is not a random variable, but Z is. By Lemma 4.10, $E[Z] = O(q\epsilon_\pi \log n) = O(q/\log^2 n)$. By Markov's inequality, $\Pr\{Z \geq 10 \lg n \cdot E[Z]\} \leq 1/(10 \lg n)$. Thus, with failure probability at most $1/(10 \lg n)$, the number of nodes in the next row is at most $2q + 10 \lg n \cdot O(q/\log^2 n) = 2q(1 + O(1/\log n))$. ∎

4.7. The new potential. Incorporating distance-limited searches into the algorithm impacts the potential function on active vertices in several ways. This section defines the new potential, outlines the main issues, and proves some properties about the potential function. The next section analyzes the progress with respect to the potential.

First, due to distance-limited searches, the definition of a vertex being path active is updated with respect to being *partly* related. That is, a vertex v is *path active* at some level r in the flattened tree if (1) v is part of some path-relevant subproblem at level r, and (2) v is partly related to the path in that subproblem.

As before, the goal is to show that for each problem, the child subproblems have potential that is a constant factor smaller in expectation. As long as the potential is well-behaved (in particular, satisfying Definition 3.8(iv)), the fact that there may be more than two subproblems does not change much. In fact, the analysis only leverages progress made by the first partly path-related pivot x.

Balancing fully and partly path-related pivots. The biggest challenge is the aforementioned interaction between the two types of random choices, distance and pivot. To bound the impact of fringe nodes, we shall apply Lemma 4.9, which requires that the distance be selected uniformly at random.

To disentangle the issues, instead of arguing progress with respect to partly path-related vertices, the goal is to argue progress with respect to *fully* path-related vertices. Because distances reduce with each successive subproblem, only the fully path-related vertices are active in the next subproblem.

At a high level, the argument is as follows. Start from the assumption that x is partly path related, with the distance still unknown. Next, reveal whether x is fully path related or merely partly path related. If x is fully path related, apply the c-reducing potential function with respect to fully path related vertices. If x is merely partly path related, apply a different argument.

In slightly more detail. Let α, β, and δ denote the number of partly path-related ancestors, bridges, and descendants, respectively. Let $\bar{\alpha}$, $\bar{\beta}$, and $\bar{\delta}$ denote the fully path-related numbers. Let $p = (\bar{\alpha} + \bar{\beta} + \bar{\delta})/(\alpha + \beta + \delta)$. The main idea is that if p is small (most are fully related), then it is likely to select a fully path-related pivot. In this case, one can leverage the c-reducing potential function with respect to the number of fully path-related vertices. If on the other hand p is large (most are not fully related), then significant progress should be made automatically as the search distance reduces.

This logic gives rise to an additional requirement on the function ψ.

DEFINITION 4.13. *$\langle \alpha, \beta, \delta \rangle$ is said to be* reducible to $\langle \bar{\alpha}, \bar{\beta}, \bar{\delta} \rangle$ *if there exist $k_1 \geq 0$ and $k_2 \geq 0$ such that $\bar{\beta} + k_1 + k_2 \leq \beta$, $\bar{\alpha} \leq \alpha + k_1$, and $\bar{\delta} \leq \delta + k_2$.*

Interpreted as the number of ancestors, bridges, and descendants in a subproblem, *being reducible means that the latter counts can be formed by removing vertices or relationships between vertices.*

DEFINITION 4.14. *Let $\psi : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be any c-reducing well-behaved function. Then ψ is* balanced *if for all $\langle \alpha, \beta, \delta \rangle$ reducible to $\langle \bar{\alpha}, \bar{\beta}, \bar{\delta} \rangle$,*

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

where $p = (\bar{\alpha} + \bar{\beta} + \bar{\delta})/(\alpha + \beta + \delta)$.

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 for the more involved functions.

Impact of fringe nodes. If using the linear function $\psi(\alpha, \beta, \delta) = \alpha + \beta + \delta$, then Lemma 4.9 also directly implies a bound on the impact of fringe nodes on 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 $(1+O(1/\log n))$, which is negligible when considered for only $\lg n$ levels.

Bounding the impact that fringe nodes have on more complex ψ is much more difficult as the number of fringe nodes added does not easily relate to the change in ψ.

Instead, the remainder of the analysis adopts the following function:

$$\psi(\alpha, \beta, \delta) = \sqrt{(\alpha + \beta)(\delta + \beta)} \cdot C_\phi + (\alpha + \beta + \delta) \ , \tag{4.1}$$

where $C_\phi = \Theta(\log^{3/2} n)$. The advantage of adding the linear term is that it makes it easier to bound the impact of fringe nodes. But as we have seen, a linear function is only (3/4)-reducing. As such, the nonlinear term is weighted significantly higher so that local imbalance can still be leveraged.

4.7.1. Proving properties for the new potential.

The remainder of the section focuses on proving the key properties about the updated potential function. These proofs are all purely algebraic.

LEMMA 4.15. *The function ψ from* (4.1) *is a well-behaved function with overhead $O(C_\phi)$.*

Proof. The function consists of two terms added together. Each is well-behaved, so their sum is as well.

As for the overhead, $\psi(\alpha, \beta, \delta) = C_\phi \sqrt{(\alpha + \beta)(\delta + \beta)} + \alpha + \beta + \delta \leq 2C_\phi(\alpha + \beta + \delta)$. □

LEMMA 4.16. *For $C_\phi = \Theta(\log^{3/2} n)$, the function ψ from* (4.1) *is c-reducing for $c = (1/\sqrt{2})(1 + O(1/\log n))$.*

Proof. Let $\psi_1(\alpha, \beta, \delta) = \sqrt{(\alpha + \beta)(\delta + \beta)}$ and $\psi_2(\alpha, \beta, \delta) = \eta = \alpha + \beta + \delta$. Thus, $\psi(\alpha, \beta, \delta) = C_\phi \cdot \psi_1(\alpha, \beta, \delta) + \psi_2(\alpha, \beta, \delta)$.

As shown in Lemma 3.5, ψ_2 is (3/4)-reducing. As shown in Lemma 3.13, ψ_1 is $(1/\sqrt{2})$-reducing.

The remainder of the proof focuses on balancing the two terms. Without loss of generality by symmetry, assume $\alpha \geq \delta$. There are two cases.

Case 1: $\delta \leq \alpha/C_\phi^{2/3}$. In this case, ψ_2 decreases significantly due to imbalance. In particular, we have

$$\begin{aligned}
(\alpha/\eta)\psi_2(\alpha/2,\beta,\delta) + (\delta/\eta)\psi_2(\alpha,\beta,\delta/2) \\
&= (\alpha/\eta)(\alpha/2+\beta+\delta) + (\delta/\eta)(\alpha+\beta+\delta) \\
&= (\alpha/\eta)(\alpha/2+\beta+\delta) + (\delta/\eta)(\eta) \\
&\leq \frac{\alpha(\alpha+\beta)+\alpha\beta}{2\eta} + 2\delta \\
&\leq \frac{\eta^2}{2\eta} + \frac{\alpha\beta}{2\eta} + \frac{2\alpha}{C_\phi^{2/3}} \\
&\leq \eta/2 + \frac{((\alpha+\beta)/2)^2}{2\eta} + \frac{2\eta}{C_\phi^{2/3}} && \text{(by AM-GM inequality)} \\
&\leq \eta/2 + \eta/8 + 2\eta/C_\phi^{2/3} \\
&< \eta/\sqrt{2} + O(\eta/\log n) \\
&= \psi(\alpha,\beta,\delta)(1/\sqrt{2})(1+O(1/\log n))\ .
\end{aligned}$$

The first term decreases faster (down to $1/\sqrt{2}$). Since each term is at worst $(1/\sqrt{2})(1+O(1/\log n))$-reducing, the sum is as well.

Case 2: $\delta > \alpha/C_\phi^{2/3}$. In this case, ψ_1 dominates by so much that it does not matter that ψ_2 decreases at all. Specifically, we have

$$\begin{aligned}
\psi_1(\alpha,\beta,\delta) &= \sqrt{(\alpha+\beta)(\delta+\beta)} \\
&> \sqrt{(\alpha+\beta)(\alpha/C_\phi^{2/3}+\beta)} \\
&\geq \sqrt{(\alpha+\beta)(\alpha+\beta)/C_\phi^{2/3}} \\
&= (\alpha+\beta)/C_\phi^{1/3} \\
&\geq \eta/(2C_\phi^{1/3}) && \text{because } \delta \leq \alpha \implies \delta \leq \eta/2\ .
\end{aligned}$$

It follows that $C_\phi\psi_1(\alpha,\beta,\delta) \geq C_\phi^{2/3}\eta/2 = \Omega(\eta\log n)$, or equivalently that $\eta = O(C_\phi\psi_1(\alpha,\beta,\delta)/\log n)$. The ψ_1 term decreases by $1/\sqrt{2}$ by Lemma 3.13. The overall final value adding the two terms together is thus at most $C_\phi \cdot \psi_1(\alpha,\beta,\delta)(1/\sqrt{2} + O(1/\log n)) = C_\phi \cdot \psi_1(\alpha,\beta,\delta)(1/\sqrt{2})(1+O(1/\log n))$. ∎

Lemma 4.17. *The function ψ from* (4.1) *is balanced.*

Proof. Adopt the notation from Definition 4.14.

First, let us assume that ψ satisfies

$$\psi(\bar\alpha,\bar\beta,\bar\delta) \leq \sqrt{p}\cdot\psi(\alpha,\beta,\delta) \text{ for all } \langle\alpha,\beta,\delta\rangle \text{ reducible to } \langle\bar\alpha,\bar\beta,\bar\delta\rangle. \tag{4.2}$$

This fact will be proved at the end.

Substituting in the above assumption, we have $p\cdot c\cdot\psi(\bar\alpha,\bar\beta,\bar\delta)+(1-p)\psi(\bar\alpha,\bar\beta,\bar\delta) \leq pc\sqrt{p}\cdot\psi(\alpha,\beta,\delta)+(1-p)\sqrt{p}\cdot\psi(\alpha,\beta,\delta) = \sqrt{p}(pc-p+1)\psi(\alpha,\beta,\delta)$. The claim follows as long as $\sqrt{p}(pc-p+1) \leq c$. To see that it is, fix c and observe how p changes. For $c = 2/3$, the expression on the left is maximized at $p = 1$, solving to exactly c. As c increases, the maximum of the function shifts even further to the right, meaning that the expression is still maximized (for $p \in [0,1]$) when at $p = 1$. Finally, $1/\sqrt{2}(1 + O(1/\log n)) \geq 2/3$, meaning that this logic applies.

The remainder of the proof focuses on (4.2). Define $\eta = \alpha+\beta+\delta$ and $\bar{\eta} = \bar{\alpha}+\bar{\beta}+\bar{\delta}$. Consider $\psi(\alpha,\beta,\delta) = C_\phi \cdot \psi_1(\alpha,\beta,\delta) + \psi_2(\alpha,\beta,\delta)$, and bound each term separately.

Showing that $\bar{\eta} = \psi_2(\bar{\alpha},\bar{\beta},\bar{\delta}) \leq \sqrt{p} \cdot \psi_2(\alpha,\beta,\delta) = \sqrt{p}\eta$ is trivial—by definition, $\bar{\eta} = p\eta \leq \sqrt{p}\eta$ for $p \leq 1$

To bound ψ_1, consider the following. The fraction p dictates how much α, β, and/or δ must be reduced by in total. The worst case for the desired inequality is to perform the required reduction in a way that keeps $\psi_1(\bar{\alpha},\bar{\beta},\bar{\delta})$ maximized. Without loss of generality, suppose $\alpha \leq \delta$. Then $\psi_1(\bar{\alpha},\bar{\beta},\bar{\delta})$ is maximized when $\bar{\alpha}$ and $\bar{\delta}$ are kept as large and as balanced as possible, so δ should be reduced first. If $p(\delta+\beta) > (\alpha+\beta)$, then only δ should be reduced, giving $\bar{\delta} + \bar{\beta} \leq p(\delta + \beta)$. The potential thus becomes $\psi_1(\bar{\alpha},\bar{\beta},\bar{\delta}) = \sqrt{(\bar{\alpha} + \bar{\beta})(\bar{\delta} + \bar{\beta})} \leq \sqrt{(\alpha + \beta)(p(\delta + \beta))} = \sqrt{p} \cdot \psi_1(\alpha,\beta,\delta)$. If p is smaller, then consider two phases p_1 and p_2 with $p = p_1p_2$ and $p_1 = (\alpha + \beta)/(\delta + \beta)$. During the first, the potential is maximized as above, leaving $\psi_1(\bar{\alpha},\bar{\beta},\bar{\delta}) = \sqrt{p} \cdot \phi_1$ as above. For the second phase, $\alpha = \delta$, so the best choice to keep the expression maximized is to balance the reductions from both counts simultaneously. During this regime, $\psi_1(\bar{\alpha},\bar{\beta},\bar{\delta}) = \sqrt{(\bar{\alpha} + \bar{\beta})(\bar{\delta} + \bar{\beta})} \leq \sqrt{p_2(\alpha + \beta)p_2(\delta + \beta)} = p_2\psi_1(\alpha,\beta,\delta)$. Multiplying the two together gives a maximum value of $p_2\sqrt{p_1} \cdot \phi_1 \leq \sqrt{p_1p_2} \cdot \phi_1(\alpha,\beta,\delta)$. ☐

Finally, the next lemma considers the impact of adding arbitrary vertices (specifically fringe vertices) to a subproblem. This lemma is the main reason for including the linear term in the updated potential. To transform the additive overhead to a multiplicative one, the bound suggests choosing a value of N_L of at least $N_L = \Omega(C_\phi^{7/3} \log n) = \Omega(\log^{9/2} n)$.

LEMMA 4.18. *For $f_1, f_2, f_3 \geq 0$ and $f = f_1 + f_2 + f_3$, the function ψ from* (4.1) *satisfies*

$$\psi(\alpha + f_1, \beta + f_2, \delta + f_3) \leq (1 + O(1/\log n)) \cdot \psi(\alpha,\beta,\delta) + 3fC_\phi^{7/3} \ .$$

Proof. The expression is maximized for $f_2 = f$. Let $\eta = \alpha + \beta + \delta$. Let $\psi_1 = \sqrt{(\alpha + \beta)(\delta + \beta)}$, so $\psi(\alpha,\beta,\delta) = C_\phi \cdot \psi_1 + \eta$. As a similar shorthand, let $\psi_1^f = \sqrt{(\alpha + \beta + f_1 + f_2)(\delta + \beta + f_2 + f_3)} \leq \sqrt{(\alpha + \beta + f)(\delta + \beta + f)}$.

Let $y = \alpha + \beta$ and $z = \delta + \beta$, and without loss of generality suppose that $y \leq z$. Consider the ψ_1 term first. We have $\psi_1^f \leq \sqrt{(y + f)(z + f)} \leq \sqrt{yz} + \sqrt{yf + zf} + f \leq \psi_1 + \sqrt{2fz} + f$. There are three cases.

Case 1: $2f \leq z/C_\phi^{10/3}$. Then $\sqrt{2zf} \leq \sqrt{z^2/C_\phi^{10/3}} = z/C_\phi^{5/3} \leq \eta/C_\phi^{5/3}$. Putting everything together gives

$$\begin{aligned}\psi^f &= \psi_1^f C_\phi + \eta + f \leq (\psi_1 + \eta/C_\phi^{5/3} + f)C_\phi + \eta + f \\ &= \psi(\alpha,\beta,\delta) + \eta/C_\phi^{2/3} + f(C_\phi + 1) \\ &\leq \psi(\alpha,\beta,\delta) \cdot (1 + 1/C_\phi^{2/3}) + 2fC_\phi = \psi(\alpha,\beta,\delta)(1 + O(1/\log n)) + 2fC_\phi.\end{aligned}$$

Case 2: $2f > z/C_\phi^{10/3}$. Then $\sqrt{2fz} \leq \sqrt{(2f)(2fC_\phi^{10/3})} = 2fC_\phi^{5/3}$. Putting everything together gives

$$\begin{aligned}\phi^f &= \phi_1^f C_\phi + \eta + f \leq (\phi_1 + 2fC_\phi^{5/3} + f)C_\phi + \eta + f \\ &= \phi + 2fC_\phi^{7/3} + fC_\phi + f \leq \phi + 3fC_\phi^{7/3} \quad \text{for } C_\phi \geq 2.\end{aligned}$$

Taking the larger of the two cases for each term proves the claim. ☐

4.8. Progress on path-active vertices. This section analyzes the progress with respect to potential. The goal is to argue that after $(2/3)\lg n + o(\log n)$ rounds (or levels in the tree), the number of active vertices drops below $D/2$ with constant probability (or, conversely, to choose D so that this statement is true).

All progress arguments are made with respect to the first partly path-related pivot x selected. Nevertheless, multiple pivots must be considered with respect to the impact of fringe nodes.

Fringe vertices. To separate the analysis of the fringe nodes from the analysis of core nodes, it is convenient to track the provenance of vertices and consider fringe vertices at the end. To that end, a path-active vertex v is said to be *core activated* by the execution of a specific subproblem if (1) v is still path active in a child subproblem, and (2) v was not added to that child as a fringe vertex. For example, if v is path active in some subproblem $G[V_{F,j} \cup F_j^+]$, but $v \in F_j^+$, then v is *not* considered core activated.

Since fringe vertices are only added to the forward/backward subproblems, an active vertex in the unrelated subproblem is always core activated.

Finally, observe that each additional pivot can only reduce the number of core-activated vertices further by knocking out vertices that would have otherwise been active in the unrelated subproblem.

LEMMA 4.19. *Consider any subproblem (G, P). Let $A = Anc_{d_{\min}D}(G, P)$ be the fully related ancestors of path P. The following is true for any search distance $dD \geq d_{\min}D$.*

Let x be the first fully related pivot selected in the current iteration, and suppose that x is drawn uniformly at random from A. Let $\alpha = |A|$ and let α' denote the total number of core-activated vertices in A. Then $E[\alpha'] < \alpha/2$.

Proof. The proof is similar to that of Lemma 3.4, except that a subset of vertices is considered, and all relationships are with respect to $\preceq_{dD}$ instead of $\preceq$. As before, the goal is to show that the preserves relation is antisymmetric for all $u, v \in A$.

Note that it is possible for $u \in A$ to be a dD-limited bridge. (It is not a bridge at distance $d_{\min}D$, but it could be a bridge at greater distances.) By Lemma 4.5, this case would lead to $\alpha' = 0$ as there are no path-relevant subproblems.

Consider any pair of vertices $u, v \in A$ such that u preserves v. The logic follows the proof of Lemma 3.4 with the same two cases, summarized briefly here. If $u \preceq_{dD} v$, then v cannot preserve u. If u and v are dD-unrelated, then consider the earliest vertices v_a with $u \preceq_{dD} v_a$ and v_b with $v \preceq_{dD} v_b$ on the path. For u to preserve v, it must be that $b < a$, so v cannot also preserve u. ∎

The next lemma pulls together the various properties of the potential to argue that expected reduction on $\phi(s)$ is still almost as good as previously.

LEMMA 4.20. *Consider any path-relevant subproblem $s = (G, P)$. Suppose that no neighborhood failure occurs for this iteration. Let $s_1, s_2, \ldots$ be random variables denoting its child subproblems in the flattened path-relevant tree, and suppose $N_L \geq C_\phi^3 \log n = \Omega(\log^{5.5} n)$. Then $E[\phi(s_1) + \phi(s_2) + \cdots] \leq (\phi(s)/\sqrt{2})(1 + O(1/\log n))$.*

Proof. Let i denote the iteration during which at least one pivot x_j that is partly related to the path P is selected. For any earlier iterations, the potential can only decrease according to Definition 3.8. Let $\alpha = |Anc_{d_{\max}D}(G, P)|$ denote the number of partly path-related vertices in the graph G at the start of the iteration. Define β and δ similarly for bridges and descendants, respectively. Let $\bar{\alpha} = |Anc_{d_{\min}D}(G, P)|$ denote the number of fully path-related ancestors at the start. Similarly for $\bar{\beta}$ and $\bar{\delta}$.

If at least one of the pivots is a bridge, there are no path-relevant subproblems (Lemma 4.5) and the potential is 0. Suppose for the remainder that the first pivot has not yet been revealed, but all subsequent pivots are not bridges.

Consider the pivots and corresponding partition steps in order, as in Lemma 4.5. Let s_1 denote the recursive subproblem generated by the first pivot, and let $s_{1,U}$ denote the nonrecursive subproblem corresponding to vertices not found by the core searches. The second pivot partitions $s_{1,U}$ into recursive problem s_2 and remainder $s_{2,U}$. The third pivot partitions $s_{2,U}$, and so on. For k pivots, the subproblems are $s_1, s_2, \ldots, s_r, s_{k+1}$, where $s_{k+1} = s_{k,U}$.

The goal is to bound $E[\sum_{i=1}^{k+1} \phi(s_i)]$ given that there is at least one partly path-related pivot. For each of the subproblems, let $\bar{\alpha}_i$, $\bar{\beta}_i$, and $\bar{\delta}_i$ denote the number of fully path-related vertices that are core activated, i.e., part of core searches. Only the fully related vertices need be considered as these are the only ones that can be active at the next level. Also consider the result $\bar{\alpha}_{1,U}$, $\bar{\beta}_{1,U}$, and $\bar{\delta}_{1,U}$ of the first search, and let $\bar{\alpha}' = \bar{\alpha}_1 + \bar{\alpha}_{1,U}$ be the number of ancestors core activated by just the first pivot. (Similarly for bridges and descendants.) Finally, let f_i denote the number of path-active fringe nodes added, and let $f = \sum_{i=1}^{r+1} f_i$. We bound $\sum_i \phi(s_i)$ as follows:

$$\begin{aligned}
\sum_{i=1}^{k+1} \phi(s_i) &\leq \phi(s_1) + \sum_{i=2}^{k+1} (\phi_1(s_i) + \phi_2(s_i)) \\
&\leq \psi(\bar{\alpha}_1, \bar{\beta}_1 + f_1, \bar{\delta}_1) + \sum_{i=2}^{k+1} \psi(\bar{\alpha}_i, \bar{\beta}_i + f_i, \bar{\delta}_i) && \text{by Definition 3.8(i)} \\
&\leq \psi(\bar{\alpha}_1, \bar{\beta}_1 + f_1, \bar{\delta}_1) + \psi\left(\bar{\alpha}_{1,U}, \bar{\beta}_{1,U} + \sum_{i=2}^{k+1} f_i, \bar{\delta}_{1,U}\right) && \text{by Definition 3.8(iv)} \\
&\leq \psi(\bar{\alpha}', \bar{\beta}' + f, \bar{\delta}') && \text{by Definition 3.8(iv)} \\
&\leq (1 + O(1/\log n)) \cdot \psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}') + 3fC_\phi^{7/3} && \text{by Lemma 4.18.}
\end{aligned}$$

Note that at this point, no expectations have been applied yet; all manipulations thus far are just algebra on the random variables.

The next step is to bound $E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')]$, i.e., the expected potential including just the impact of nodes core activated by the first pivot. There are two cases, depending on how the pivot is classified. Let $p = (\bar{\alpha} + \bar{\beta} + \bar{\delta})/(\alpha + \beta + \delta)$ be the fraction of partly path-related vertices that are fully related. Let R be the event that the pivot is fully related. If R does not occur, then we simply assume that all fully path-related vertices survive. Then

$$\begin{aligned}
E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')] &= \Pr\{R\} \cdot E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')|R] + (1 - \Pr\{R\}) E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')|\neg R] \\
&\leq p \cdot E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')|R] + (1-p) \cdot \psi(\bar{\alpha}, \bar{\beta}, \bar{\delta}) \\
&\leq p \cdot \Bigg(\left(\frac{\bar{\alpha}}{\bar{\alpha} + \bar{\beta} + \bar{\delta}}\right) E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')|x \in Anc_{d_{\min} D}(G, P)] \\
&\qquad + \left(\frac{\bar{\delta}}{\bar{\alpha} + \bar{\beta} + \bar{\delta}}\right) E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')|x \in Desc_{d_{\min} D}(G, P)] \Bigg) \\
&\qquad + (1-p) \cdot \psi(\bar{\alpha}, \bar{\beta}, \bar{\delta}) \\
&\leq p \cdot c \cdot \psi(\bar{\alpha}, \bar{\beta}, \bar{\delta}) + (1-p) \cdot \psi(\bar{\alpha}, \bar{\beta}, \bar{\delta}) && (*) \\
&\leq c \cdot \psi(\alpha, \beta, \delta) = c \cdot \phi(s) && \text{by balance (Lemma 4.17),}
\end{aligned}$$

where (*) follows by the same algebraic manipulations, for a well-behaved c-reducing function, as Lemma 3.9.

Substituting back in the bound bound for $E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')]$, we have

$$\begin{aligned}
E\left[\sum_{i=1}^{k+1}\phi(s_i)\right] &\le (1+O(1/\log n))\cdot E[\psi(\bar{\alpha}', \bar{\beta}', \bar{\delta}')] + 3C_\phi^{7/3}E[f] \\
&\le c\cdot(1+O(1/\log n))\cdot\phi(s) + 3C_\phi^{7/3}E[f] \\
&\le c\cdot(1+O(1/\log n))\cdot\phi(s) + 3C_\phi^{7/3}O((\alpha+\beta+\delta)\log n/N_L) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Lemma 4.9)} \\
&\le c\cdot(1+O(1/\log n))\cdot\phi(s) + O(1/\log n)(\alpha+\beta+\delta) \\
&\le c\cdot(1+O(1/\log n)\cdot\phi(s) && (\phi(s)\ge\alpha+\beta+\delta) \\
&\le (1/\sqrt{2})(1+O(1/\log n))^2\cdot\phi(s) && \text{(Lemma 4.16)} \\
&= (\phi(s)/\sqrt{2})(1+O(1/\log n)).
\end{aligned}$$
∎

4.9. Analyzing the layers in the tree and proving Lemma 4.1. Define the total potential Φ of a level as follows:

$$\Phi(I_r) = (1+c_\Phi/\lg n)^{\lg n - r}\sum_{s\in I_r}\phi(s)\ ,$$

where I_r is the collection of subproblems corresponding to level r in the flattened tree and c_Φ is a constant to be set later.

COROLLARY 4.21. *Suppose $C_\phi = \Theta(\lg^{1.5} n)$ and $N_L = \Omega(\lg^{5.5} n)$ Then there exists a large enough constant c_Φ such that $E[\Phi(I_r)|I_{r-1}] \le \Phi(I_{r-1})/\sqrt{2}$.*

Proof. Choose c_Φ large enough so that $1 + c_\Phi/\lg n$ is greater than the $1 + O(1/\log n))$ term in Lemma 4.20. The claim then follows from linearity of expectation over subproblems. ∎

The real purpose of the extra $(1 + c_\Phi/\lg n)^{\lg n - r}$ factor is to offset any potential increases to the subproblem potentials ϕ. With an unlucky number of active fringe vertices, it is possible that Φ increase when going from one row to the next. Such an increase, called a *fringe failure*, would preclude the application of Theorem 2.1. The next lemma shows that fringe failures are unlikely.

LEMMA 4.22. *There exist constants c_Φ and c_L such that, for $C_\phi = \Theta(\lg^{1.5} n)$ and $N_L \ge c_L C_\phi^3 \lg^2 n = \Omega(\log^{6.5} n)$, $\Pr\{\Phi(I_{r+1}) > \Phi(I_r)\} \le 1/(10\lg n)$.*

Proof. Even if all active vertices are preserved, well-behavedness implies that the subproblem potentials $\sum_s \phi(s)$ cannot increase without the addition of fringe nodes. The active fringe nodes themselves have two contributions (see Lemma 4.18): a multiplicative $(1+O(1/\log n))$ overhead, and an additive $3C_\phi^{7/3}f$. The former does not depend on the number of fringe nodes, so choose $c_\Phi/\lg n$ to be, say, twice as large as the $O(1/\log n)$ term. Thus for Φ to increase would require that the total contribution from f fringe nodes exceed $f \ge c_\Phi/(2\lg n)\sum_{s\in I_r}\phi(s)$. For large enough N_L, the expected number of fringe nodes is $E[f] = O(\sum_{s\in I_r}\phi(s)\log n/N_L) \le \sum_{s\in I_r}\phi(s)/(c_L 3C_\phi^3 \log n)$, giving an expected potential contribution of $\sum_{s\in I_r}\phi(s)/(c_L\log^2 n)$. For large enough c_L, this expectation is at most $c_\Phi/(20\lg^2 n)\sum_{s\in I_r}\phi(s)$. Reaching the target threshold would require there to be $10\lg n$ times the expectation, which occurs with probability at most $1/(10\lg n)$ by Markov's inequality. ∎

To prevent Φ from increasing at all, instead define Φ' to be equal to Φ, except that it drops to 0 when a fringe failure occurs. It follows that (1) Φ' never increases, and (2) $E[\Phi'(I_{r+1}|I_r)] \leq E[\Phi(I_{r+1}|I_r)]$. Theorem 2.1 can now be applied.

We are now ready to prove Lemma 4.1.

Proof of Lemma 4.1. Let $D = \Theta(n^{2/3}\log n)$, $C_\phi = \Theta(\log^{3/2} n)$, $N_L = \Theta(\log^{6.5} n)$, and $\epsilon_\pi = O(1/\log^3 n)$, to be consistent with previous lemmas.

The starting value of $\Phi'(I_0) \leq (1 + c_\Phi/\lg n)^{\lg n}(C_\phi + 1)n = O(n\log^{3/2} n)$. For large enough constant w, Theorem 2.1 bounds the probability that the potential stays too high as $\Pr\left\{\Phi'(I_{r+w}) > (1/\sqrt{2})^r O(n\lg^{3/2} n)\right\} < 1/10$. Then for $r = (2/3)\lg n + \lg\lg(n) + \Theta(1)$, this expression reduces to $\Pr\left\{\Phi'(I_{r+w}) > cn^{2/3}\lg n\right\} < 1/10$ for some constant c.

If a fringe failure occurs, the bound on Φ' is meaningless. The probability of a fringe failure is at most the union bound over $r < \lg n$ levels of the failure probability $1/(10\lg n)$ from Lemma 4.22, which reduces to $1/10$. If neither of these failures occurs, the bound on Φ' implies a bound on active unshortcut subpaths, as all path vertices counted as bridges towards Φ'. Thus, with failure probability $1/4$, the total length of all subpaths in path-relevant subproblems in level $(r+2)$ is at most $O(n^{2/3}\lg n)$.

Finally, consider the concatenations and shortcut leaves via Lemma 4.12. With failure probability $1/10$, the total shortcut length is thus $O(n^{2/3}\lg n)$. Choose D to be a constant factor larger than the constant hidden inside the big-O. ∎

4.10. Proof of Lemma 4.2. This section proves bounds on the number of shortcuts and overall work performed. The main goal is to show that, with probability at least 9/10, the number of vertices (and hence shortcuts) and arcs visited by searches is consistent with Lemma 4.1. Thus, with probability at least 1/2, Lemmas 4.1 and 4.2 both hold.

The settings used are $D = \Theta(n^{2/3}\log n)$, $C_\phi = \Theta(\log^{1.5} n)$, $N_L = \Theta(\log^{6.5} n)$, $\epsilon_\pi = \Theta(1/\log^3 n)$, and $N_k = \Theta(\log^4 n)$, as dictated by constraints offered in previous lemmas. The maximum search distance is immediate: it is at most $hDN_kN_L = O(\log n \cdot n^{2/3}\log n \cdot \log^4 n \cdot \log^{6.5} n) = O(n^{2/3}\log^{12.5} n)$.

Consider each level of recursion in Algorithm 5. Lemma 4.8 holds with high probability, so assume that it holds for every node at every iteration. Consider any vertex in the iteration in which it is visited by a core search. By assumption of Lemma 4.8, the vertex, and hence its incident arcs, is visited by at most $O(\log n)$ searches.

The number of vertices (and hence arcs) may increase with each level in the tree due to fringe searches. The final step of the proof is to argue that the total size of all subproblems in the final level of recursion is $O(n)$ vertices and $O(m)$ arcs, and hence the total cost of all levels is $O(n\log^2 n)$ vertices and $O(m\log^2 n)$ arcs.

By Lemma 4.9 with $N_L = \log^{6.5} n$, the number of vertices and arcs increases with each level of recursion by at most an additive $O(n'/\log^2 n)$ and $O(m'/\log^2 n)$ in expectation, where n' and m' are the current numbers at that level. Thus, by Markov's inequality, with probability at most $1/(10\lg n)$, the increases is not more than a multiplicative $1 + O(1/\log n)$. Since there are $\lg n$ levels of recursion, this results in probability at most $1/10$ of exceeding a total of $n(1+O(1/\log n))^{\lg n} = O(n)$ vertices and $m(1 + O(1/\log n))^{\lg n} = O(m)$ arcs. ∎

5. Parallel version. This section briefly discusses the parallel version of Algorithms 5 and 6. This section assumes the reader is comfortable enough with parallel algorithms to infer the details and instead focuses 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* $n = |\hat{V}|$, *let* $m = |\hat{E}|$, *and without loss of generality assume* $m \geq n/2$. *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* $O(n^{2/3} \log^{19.5} n)$ *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 concurrent-read exclusive-write (CREW) parallel algorithm for digraph reachability that has* $O(m \log^6 n + n \log^{10} n)$ *work and* $O(n^{2/3} \log^{19.5} n)$ *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 5 is the dD-limited searches to find, e.g., R_j^+. One might simply replace the foreach loops with parallel loops, but the question is how the bookkeeping should be performed.

[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.

Ordinarily, a BFS keeps track of already-visited vertices either by 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 parallel realization is Lemma 4.8—with high probability, no vertex is visited by more than $O(\log n)$ parallel searches. The algorithm 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 m_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 $O(m_e \log^2 n + |X_i| \log n)$ work and $O(n^{2/3} \log^{13.5})$ 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, initially 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. For each 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. For each 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 next $O(\log n)$ 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. For each vertex v in core searches, in parallel, identify the lowest ID pivot 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 5. To make the work (and shortcut) bound deterministic, Algorithm 6 needs the ability to abort any runs of Algorithm 5 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 4.2 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 both on how efficiently the program is scheduled and on the number of processors employed.

There is a solution in the work-span model: logically implement the recursive steps of the algorithm as a BFS. That is, maintain an array of subproblems, initially just `ParSC(`$\hat{G}, \lg n$`)`. To implement a level of recursion, perform prefix sums to add up the total number of vertices across all subproblems, and give each vertex (pivot) a specific slot in which to put its recursive subproblem. Instead of launching the recursive subproblems immediately, simply record them in the appropriate slot. When all subproblems at this level of recursion complete, launch all problems at the next level (in parallel).

The work bound can only be exceeded if the total number of arcs in the next set of subproblems grows too large. This number can be counted with a parallel reduce after each level of recursion completes. None of these steps increases the work or span asymptotically.

Proof of Theorem 5.1. The diameter and shortcut bounds are taken directly from Theorem 4.3. Multiplying the cost per arc of Lemma 5.3 with the number of arcs searched in Theorem 4.3 gives the work bound. Shortcuts can be gathered and larger graphs built for each iteration of Algorithm 6 by sorting, but that is dominated by the other work performed.

The span bound is obtained by multiplying the maximum search distance of $O(n^{2/3} \log^{12.5} n)$ by the $O(\log n)$ span per BFS round, the $N_k = O(\log^4 n)$ iterations in a call, the $O(\log n)$ levels of recursion in a run of Algorithm 5, and the $O(\log n)$ iterations of the outer loop of Algorithm 6. Note that the inner loop of Algorithm 6 can be implemented in parallel. Altogether, that gives $O(n^{2/3} \log^{19.5} n)$ span.

6. Building a directed spanning tree.

This section discusses how to augment the algorithm to produce a directed spanning tree. It is not immediately obvious how to do so even for the sequential algorithm of section 3. To illustrate the issues, consider the following graph: $s \to u \to v \rightleftarrows w$. If w is selected as a pivot first, then a shortcut (s, w) is added, and a BFS from s in the shortcut graph may discover the following path: $s \rightsquigarrow w \to v$. Simply splicing in the path corresponding to the shortcut would result in $s \to u \to v \to w \to v$, which is no longer a simple path. The goal is to do this splicing, but in a way that avoids repeated vertices. The situation is slightly more challenging in the case of Algorithm 6 because the arcs that are shortcut could themselves be shortcuts, but the result is just that several iterations are needed.

The algorithm for building the directed spanning tree is a postprocessing step performed after the full execution of Algorithm 6. The algorithm references the BFS trees used to build shortcuts, however, so all BFS trees need to be saved as Algorithm 6 executes. Each shortcut must also be augmented with a reference to the BFS tree that produced it. The forward-search BFS trees are directed out from the root, whereas the backward-search BFS trees are directed towards the root. In this way, the BFS trees correspond to arcs in some graph.

Let $G_0, G_1, \ldots, G_{k=\Theta(\log n)}$, where $G_0 = \hat{G}$, denote the sequence of graphs built after each iteration of the outer loop of Algorithm 6. Running BFS on the resulting graph G_k yields a directed spanning tree T_k in G_k. This section describes how to transform a directed spanning tree T_i in graph G_i to a directed spanning tree T_{i-1} in G_{i-1}. Iterating $\Theta(\log n)$ times gives a spanning tree in the original graph.

Start each iteration by labeling every vertex v in the tree with label $\mathit{low}(v) = 0$ and $\mathit{high}(v)$, where $\mathit{high}(v)$ is v's distance from the root in T_i. This step can be accomplished in linear work and logarithmic span using the Euler-tour method [21].

For the next step, the shortcuts in both directions are treated differently. The goal is to essentially splice in the paths, which results in vertices appearing multiple times. This multiplicity will be resolved afterwards.

For each vertex $u \in T_i$ in parallel, traverse all forward-search BFS trees created in iteration i and rooted at u. Label those vertices v with $\mathit{high}(v) = \mathit{high}(u)$, and $\mathit{low}(v)$ is v's depth (or distance from u) in the tree. Note that these labelings should be performed on the BFS trees themselves, not on T_i or G_i, as each vertex may belong to multiple trees and may otherwise be labeled multiple times concurrently.[9]

For the backward direction, consider all arcs (u, v) in T_i in parallel. If (u, v) is a shortcut on a backward-search BFS tree rooted at v, traverse the path from u to v in the BFS tree and label each vertex w on the path by $\mathit{high}(w) = \mathit{high}(u)$. Also label $\mathit{low}(w)$ with w's distance from u on the path.

Finally, sort all arcs (u, v) in the collection of BFS trees traversed in the above process, as well as the arcs in T_i that also exist in G_{i-1}, by three values: v's ID (most significant), $\mathit{high}(u)$, and $\mathit{low}(u)$ (least significant). For each arc (u, v) in the sorted list in parallel, if (u, v) is the first arc directed toward v according to the sorted order, then include the arc (u, v) in T_{i-1}.

[9]The Euler-tour technique could be applied to each tree, but a parallel BFS is sufficient here as the trees have depth $O(n^{2/3} \log^{4/3} n)$ by construction; the work and span would be at most the work and span of constructing the tree in the first place.

Proof of correctness is by induction over i, in decreasing order. The base case is trivial: T_k is a directed spanning tree of G_k. The following lemma proves the inductive step; namely, that given a directed spanning tree T_i for G_i, the algorithm correctly produces a directed spanning tree T_{i-1} for G_{i-1}.

LEMMA 6.1. *Suppose that T_i is a directed spanning tree for G_i rooted at vertex s. Then the algorithm produces a directed spanning tree T_{i-1} rooted at s for the graph G_{i-1}.*

Proof. Since only arcs present in G_{i-1} are considered in the last step of the algorithm, T_{i-1} is a subgraph of G_{i-1}. It is not obvious, however, that it is a tree, nor is it obvious that it spans.

The first step is to show that every vertex, except s, has an incoming arc in T_{i-1}. Consider a vertex v and its incoming arc (u, v) in T_i. If (u, v) is present in G_{i-1} as well, then it is in consideration the last step, so v must select an arc. If (u, v) is a shortcut, then it corresponds to some path in a BFS tree. All arcs in that path, and specifically the arc directed toward v, are also in consideration. Thus, v has an incoming arc.

For each vertex, let the final label be the lowest label associated with any of its copies. If all arcs go from lower label to higher label, then there are no cycles. To prove this is the case, the claim is that every copy of each vertex other than the source (and in particular the lowest-label copy) has an incoming arc from a vertex with a lower label. Since the minimum incoming arc is the one used, that would imply that all arcs are from lower to higher label.

To prove the claim, consider a copy of vertex v. There are three cases. If v is in a forward-search BFS tree and not the root, then v has depth (and hence $low(v)$ label) one higher than its parent in the tree. If v is in a backward-search path and not the source, the same argument holds.

Otherwise, v's label is the same as in T_i. In T_i, v's incoming arc (u, v) satisfies $high(u) < high(v)$ by construction. If (u, v) is in G_{i-1}, then this arc satisfies the claim. Otherwise, v is the nonroot of a BFS tree with a strictly lower *high* value, and hence one of the first two cases applies. ∎

7. 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.

REFERENCES

[1] G. E. Blelloch, Y. Gu, J. Shun, and Y. Sun, *Parallelism in randomized incremental algorithms*, in Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, 2016, pp. 467–478, https://doi.org/10.1145/2935764.2935766.

[2] R. P. Brent, *The parallel evaluation of general arithmetic expressions*, J. ACM, 21 (1974), pp. 201–206.

[3] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785, https://doi.org/10.1137/0217049.

[4] D. Coppersmith, L. Fleischer, B. Hendrickson, and A. Pinar, *A Divide-and-Conquer Algorithm for Identifying Strongly Connected Components*, Tech. Report RC23744, IBM Research, 2005.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[6] S. Fortune and J. Wyllie, *Parallelism in random access machines*, in Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 114–118, https://doi.org/10.1145/800133.804339.

[7] P. B. Gibbons, *A more practical pram model*, in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 158–168, https://doi.org/10.1145/72935.72953.

[8] L. M. Goldschlager, *A unified approach to models of synchronous parallel machines*, in Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 89–94, https://doi.org/10.1145/800133.804336.

[9] W. Hesse, *Directed graphs requiring large numbers of shortcuts*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 665–669, http://dl.acm.org/citation.cfm?id=644108.644216.

[10] S.-E. Huang and S. Pettie, *Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts*, in the 16th Scandinavian Symposium and Workshops on Algorithm Theory, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 26:1–26:12, https://doi.org/10.4230/LIPIcs.SWAT.2018.26.

[11] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[12] M.-Y. Kao and P. N. Klein, *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, 1990, pp. 181–192, https://doi.org/10.1145/100216.100237.

[13] R. M. Karp, *Probabilistic recurrence relations*, J. ACM, 41 (1994), pp. 1136–1150, https://doi.org/10.1145/195613.195632.

[14] R. M. Karp and V. Ramachandran, *A Survey of Parallel Algorithms for Shared-Memory Machines*, Tech. Report UCB/CSD-88-408, EECS Department, University of California, Berkeley, 1988, http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html.

[15] P. N. Klein, *Parallelism, preprocessing, and reachability: A hybrid algorithm for directed graphs*, J. Algorithms, 14 (1993), pp. 331–343, https://doi.org/10.1006/jagm.1993.1017.

[16] P. N. Klein and S. Subramanian, *A randomized parallel algorithm for single-source shortest paths*, J. Algorithms, 25 (1997), pp. 205–220, https://doi.org/10.1006/jagm.1997.0888.

[17] F. Le Gall, *Powers of tensors and fast matrix multiplication*, in Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, 2014, pp. 296–303, https://doi.org/10.1145/2608628.2608664.

[18] W. J. Savitch and M. J. Stimson, *Time bounded random access machines with parallel processing*, J. ACM, 26 (1979), pp. 103–118, https://doi.org/10.1145/322108.322119.

[19] W. Schudy, *Finding strongly connected components in parallel using $O(\log^2 n)$ reachability queries*, in Proceedings of the 20th Annual Symposium on Parallelism in Algorithms and Architectures, 2008, pp. 146–151, https://doi.org/10.1145/1378533.1378560.

[20] T. H. Spencer, *Time-work tradeoffs for parallel algorithms*, J. ACM, 44 (1997), pp. 742–778, https://doi.org/10.1145/265910.265923.

[21] R. E. Tarjan and U. Vishkin, *Finding biconnected components and computing tree functions in logarithmic parallel time*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 12–20, https://doi.org/10.1109/SFCS.1984.715896.

[22] J. Tassarotti, *Probabilistic Recurrence Relations for Work and Span of Parallel Algorithms*, preprint, https://arxiv.org/abs/1704.02061, 2017.

[23] M. Thorup, *On shortcutting digraphs*, in Graph-Theoretic Concepts in Computer Science, E. W. Mayr, ed., Springer, 1993, pp. 205–211.

[24] J. D. Ullman and M. Yannakakis, *High-probability parallel transitive-closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125, https://doi.org/10.1137/0220006.