Brief Announcement: Improved Work Span Tradeoff for Single Source Reachability and Approximate Shortest Paths

Nairen Cao
nairen@ir.cs.georgetown.edu
Georgetown University
Washington D.C., USA

Jeremy T. Fineman
jfineman@cs.georgetown.edu
Georgetown University
Washington D.C., USA

Katina Russell
katina.russell@.cs.georgetown.edu
Georgetown University
Washington D.C., USA

ABSTRACT

This brief announcement presents parallel algorithms with a tradeoff between work and span for single source reachability and approximate shortest paths on directed graphs. Both algorithms have $\tilde{O}(m\rho^2 + n\rho^4)$ work and achieve $n^{1/2+o(1)}/\rho$ span for all $\rho \in [1, \sqrt{n}]$.

CCS CONCEPTS

• Theory of computation → Design and analysis of algorithms; **Shortest paths**; **Parallel algorithms**.

KEYWORDS

Parallel algorithm; shortest paths; reachability; shortcuts.

ACM Reference Format:
Nairen Cao, Jeremy T. Fineman, and Katina Russell. 2020. Brief Announcement: Improved Work Span Tradeoff for Single Source Reachability and Approximate Shortest Paths. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3350755.3400222

1 INTRODUCTION

Given a directed graph and designated source vertex s, the single-source reachability problem is that of identifying the set of vertices reachable from s. Given a directed graph with nonnegative edge weights and source vertex s, the single-source shortest paths problem is that of finding the shortest-path distances $d(s, v)$ from s to all other vertices v. These problems are easily solvable in linear or nearly linear time by sequential algorithms, but they are notoriously difficult to solve efficiently in parallel.

Fineman [2] gave the first parallel algorithm for single-source reachability on directed graphs with nearly linear work and sublinear span[1]. Specifically, his algorithm has $\tilde{O}(m)$ work and $\tilde{O}(n^{2/3})$ span, where n and m denote the number of vertices and edges, respectively, in the input graph and $\tilde{O}$ hides logarithmic factors. Jambulapati et al. (JLS) [3] improved upon this result, also achieving nearly linear work but improving the span to $n^{1/2+o(1)}$. More recently, Cao et al. (CFR) [1] extended the JLS algorithm to solve $(1+\epsilon)$-approximate single-source shortest paths, where the distance estimates $\hat{d}(s, v)$ returned by the algorithm satisfy $d(s, v) \leq \hat{d}(s, v) \leq (1+\epsilon)d(s, v)$. Though it would be ideal to reduce the span of these algorithms further without sacrificing the work bound, $\sqrt{n}$ span seems to be an inherent bottleneck in Fineman's general approach, at least when limited to $\tilde{O}(m)$ work. But what if the work bound is relaxed? What are the best work-span tradeoffs that we can achieve? This brief announcements argues that CFR [1] can be tuned to achieve a work-span tradeoff. In particular, for any choice of parameter $\rho \in [1, \sqrt{n}]$, the algorithm has $\tilde{O}(m\rho^2 + n\rho^4)$ work and $n^{1/2+o(1)}/\rho$ span.

The idea of studying work-span tradeoffs for these problems in general is not new. The contribution of this work is that the algorithm achieves polynomially better tradeoffs than previous algorithms when the work bound is kept relatively low (e.g., $\rho < 1/8$) and the graph is not too dense (e.g., $m < n^{7/4}$).

Previous work. The two previously best work-span tradeoffs for single-source reachability are the following. Ullman and Yannakakis (UY) [6] provide an algorithm with $\tilde{O}(m\rho + \rho^4/n)$ work and (n/ρ) span, for any $\rho \in [1, n]$. Spencer's algorithm [5] runs in $\tilde{O}(m + n\rho^2)$ work and $\tilde{O}(n/\rho)$ span, for any $\rho \in [1, n]$. Both algorithm also solve unweighted single-source shortest paths (exactly), and Spencer's algorithm solves the weighted version. Klein and Subramanian [4] extended UY to solve exact shortest paths for integer-weighted graphs, but they do so only for the specific setting of $\rho = \sqrt{n}$.

Results. The main result of this work is the captured by the following theorem.

THEOREM 1.1. *For all $\rho \in [1, \sqrt{n}]$, there exists a randomized parallel algorithm that solves single-source reachability in $\tilde{O}(m\rho^2 + n\rho^4)$ work and $n^{1/2+o(1)}/\rho$ span, with high probability, where n and m denote the number of vertices and edges in the graph, respectively.*

The algorithm also extends to solve single-source approximate shortest paths with the same performance bounds.

For $\rho = 1$, the algorithm is identical to JLS (for single-source reachability) or CFR (for approximate shortest paths) and has work $\tilde{O}(m)$ and span $n^{1/2+o(1)}$.

The main advantage of our result relative to UY or Spencer's is that the baseline span (for $\rho = 1$) is $n^{1/2+o(1)}$, whereas theirs is $\tilde{O}(n)$. Let us first compare our result with UY by calculating the work necessary to achieve a span of $\tilde{O}(n^{1/2-\epsilon})$ for any constant $0 \leq \epsilon < 1/2$. For this target span, UY has $\tilde{O}(mn^{1/2+\epsilon} + n^{1+4\epsilon})$ work, whereas ours has $mn^{2\epsilon+o(1)} + n^{1+4\epsilon+o(1)}$. In general, these bounds

[1]The span of a parallel algorithm is the length of the longest chain of sequential dependencies.

This work was supported in part by NSF grants CCF-1718700 and CCF-1918989.

are incomparable, but ours is polynomially better if $mn^{1/2+\epsilon}$ is polynomially larger than $n^{1+4\epsilon}$. This condition always holds for $\epsilon < 1/6$, and it also holds for larger ϵ as long as the graph has sufficient density. Let us next compare with Spencer's algorithm, which has work $\tilde{O}(n^{2+2\epsilon})$ to achieve $\tilde{O}(n^{1/2-\epsilon})$ span. Our work bound again represents a polynomial improvement as long as $m \le n^{2-\alpha}$ for constant $\alpha > 0$. When $m = \Theta(n)$ in particular, our algorithm has work $n^{1+4\epsilon+o(1)}$ compared to Spencer's $\tilde{O}(n^{2+2\epsilon})$.

The main disadvantage of our algorithm compared to the others is that our algorithm can only compute approximate shortest paths, whereas UY can compute exact shortest paths for unweighted graphs, and Spencer's algorithm can compute exact shortest paths for weighted graphs.

Outline of the paper. This brief announcement focuses on the simpler single-source reachability problem, but since we are extending CFR [1] the result readily extends to approximate shortest paths. The key component of prior work [1–3] is an algorithm that finds a set of "shortcut" edges that, when added to the graph, reduce the directed diameter of the graph. Given a graph with diameter D, reachability can be solved in $\tilde{O}(m)$ work and $\tilde{O}(D)$ span using parallel BFS. (See e.g., [2] for more discussion.) Fineman's main contribution [2] was the first efficient sequential algorithm for shortcutting, and he also showed how to parallelize the algorithm. The main focus of this brief announcement (Section 3) a sequential algorithm for shortcutting with a tradeoff between diameter and running time, achieving diameter $n^{1/2+o(1)}/\rho$ in time $\tilde{O}(m\rho^2)$. The algorithm is a natural simplification of CFR [1] for reachability; the main contribution is the observation that the tradeoff can be achieved. Section 5 briefly discusses the parallelization.

2 PRELIMINARIES

For a weighted directed graph $G = (V, E)$, the number of vertices is $n = |V|$ and the number of edges is $m = |E|$. For any subset of vertices $V' \subset V$, the induced subgraph on V' is denoted $G[V']$. For two nodes u, v, define the relation $v \le u$ if and only if there is a path from v to u. Denote $R^+(G, v) = \{u | v \le u\}$ to be the set of nodes that are reachable from v in G. Similarly, $R^-(G, v) = \{u | u \le v\}$ denotes the set of nodes that can reach v in G. This notation is adopted from Fineman [2] and CFR [1].

3 ALGORITHM

In this section we show a sequential algorithm for adding shortcuts to a directed unweighted graph with a diameter-runtime tradeoff. This algorithm is based on CFR [1], which in turn builds off JLS [3].

Now we will describe the main algorithm SC(G). The algorithm takes a directed unweighted graph as input and outputs a set of edges that when added to G reduce the diameter of G. In the algorithm, vertices have two roles, pivots and shortcutters. Shortcutters will search and add shortcuts, while pivots will search to partition the graph for recursion. How SC(G) assigns these roles will be explained later. SC(G) repeats assigning roles to vertices, and calling the recursive subroutine, shown in Algorithm 1, SCRecurse($G, r = 0$) $\lambda \log n$ times to increase the probability of success, where the parameter $\lambda > 8$ is a constant. SCRecurse($G, r = 0$) runs as follows. Each pivot v searches forwards and backwards adding v^{Des} and v^{Anc} labels to any vertices it reaches forwards and backwards respectively. Any vertex that is reached in both directions by v, is given the label X. Next any vertex with label X is removed, and the graph is partitioned such that vertices u and v are in the same group V_i if and only if u and v received the identical set of labels from all the pivots. Next, each shortcutter searches forwards and backwards and adds shortcuts to the vertices it reaches. Finally, SCRecurse($G, r = 0$) recurses on each group V_i with level of recursion $r + 1$. The shortcuts added throughout the entire execution are returned to the main algorithm as output.

The shortcutter and pivot roles are assigned as follows. Each vertex v is assigned level $\ell(v) = i$ with probability $\lambda k^{i+1} \log n/n$, for $i \in [0, \log_k n]$. If multiple settings are successful, the minimum level is taken. In a level of recursion r, each vertex u where $\ell(u) = r$ is a pivot, and each vertex v where $\ell(v) \le r + L$ is a shortcutter.

Algorithm 1 Recursive subroutine for shortcut algorithm. The input is a directed, unweighted graph G, level of recursion r. There is an assignment of levels $\ell(v)$ to vertices v. L and k are parameters.

```
1:  function SCRecurse($G, r$)
2:      for each $v \in V$ with $\ell(v) = r$
3:          for each $u \in R^+(G, v)$ add label $v^{Des}$ to $u$
4:          for each $u \in R^-(G, v)$ add label $v^{Anc}$ to $u$
5:          for each $u \in R^+(G, v) \cap R^-(G, v)$ add label $X$ to $u$
6:      for each $v \in V$ with $\ell(v) \le r + L$
7:          for each $u \in R^+(G, v)$ add edge $(v, u)$ to $H$
8:          for each $u \in R^-(G, v)$ add edge $(u, v)$ to $H$
9:      for each $u \in V$ that has a $X$ label, remove $u$
10:     $V_1, V_2, ..., V_t \leftarrow$ partition based on labels
11:     for each $i \in [1, t]$
12:         $H \leftarrow H \cup$ SCRecurse($G[V_i], r + 1$)
13:     return $H$
```

4 ANALYSIS SKETCH

Our goal is to show that the sequential algorithm in Section 3 exhibits a tradeoff between the work and the diameter. We use ρ as a tradeoff parameter. Recall that L is a parameter of the algorithm that determines how many shortcutters are oversampled. A larger L will result in more work but a smaller diameter. We set $L = max\{0, 2\log_k \rho - 1\}$ to achieve the desired tradeoff. The following Theorem and its proof come from CFR [1] Theorem 1.

Theorem 4.1. *There exists a randomized algorithm for producing shortcuts for unweighted directed graphs that has runtime $O(m\rho^2 \log^3 n)$ and when the shortcuts are added to the graph reduces the diameter to at most $n^{1/2+O(1/\log\log n)}/\rho$ with probability $1 - n^{2-\lambda}$, where $n = |V|$, $m = |E|$, and $\rho \in [1, \sqrt{n}]$ is a tradeoff parameter.*

The algorithm is a simplified version of CFR [1]. From Theorem 2 in CFR, we have the following Lemma about the running time of the recursive subroutine. The subsequent Corollary follows from Lemma 4.2, the number of executions of SC(G) and the setting of L.

Lemma 4.2. *One execution of SCRecurse(G, r) with parameters L and k, where $n = |V|$, $m = |E|$, runs in $O(mk^{L+1} \log^2 n)$ time and adds $O(nk^{L+1} \log^2 n)$ shortcuts to the graph.*

Corollary 4.3. *One execution of $SC(G)$ runs in $O(m\rho^2 \log^3 n)$ time and adds $O(n\rho^2 \log^3 n)$ shortcuts to the graph.*

Next we will sketch the diameter argument. The high level idea is the same as the previous papers [1–3]. The analysis considers a path where the length is greater than the desired diameter. The goal is to show by the end of the execution of the algorithm the path has been shortcutted to be at most $n^{1/2+O(1/\log k)} k \log^2 n/\rho$. A good shortcutter would be one that is reachable from the beginning of the path and reaches the end of the path. That way it adds edges to the graph that shortcuts the path to two hops. In the beginning it may not be likely to get a good shortcutter, however the previous papers have shown as the algorithm recurses the probability of getting a good shortcutter increases [1–3]. The problem is that the path is split into subpaths with each level of recursion. CFR showed that after r levels of recursion, a path will be split to at most $O((2k)^{(r+1)/2} \log n)$ subproblems. Our new observation is that we can oversample the number of shortcutters, which increases the likelihood of getting a good shortcutter. Oversampling also means that the recursion can be stopped earlier. After $r - L$ levels of recursion, every vertex has either been removed or is a shortcutter, and so each subpath has been shortcutted to two hops. Since r is at most $\log_k n$, and $L = max\{0, 2\log_k \rho - 1\}$, the diameter is at most $O((2k)^{(r+1-L)/2} \log n) = n^{1/2+O(1/\log k)} k \log^2 n/\rho$. We will use $k = \Theta(\log n)$. The following Lemma is similar to Lemma 12 from CFR [1], and the proof is essentially the same.

Lemma 4.4. *Consider any graph $G' = (V, E)$ and an execution of $SC(G')$ with parameter λ. With probability $1 - n^{-\lambda+2}$, the diameter produced is $n^{1/2+O(1/\log k)} k \log^2 n/\rho$.*

5 PARALLEL ALGORITHM

As in previous work [1–3], the main idea of the parallel algorithm is to limit the searches to some distance $O(\beta)$ (corresponding to the span of the algorithm). One of the issues with limited searches is that it is no longer possible to shortcut a long path, but instead the algorithm only shortens subpaths of length $O(\beta)$ down to β. The algorithm must therefore be iterated a polylogarithmic number of times, adding more shortcuts to the graph on each iteration. The extra shortcuts also increase the work of the algorithm, but not by too much, as stated by the following lemma. In particular, the number of edges increases to $m' = \tilde{O}(m + n\rho^2)$, which impacts the work in all subsequent iterations. The following Lemma is a restatement of CFR [1] Theorem 18.

Lemma 5.1. *For a directed unweighted graph G, the parallel shortcut algorithm does $O(m\rho^2 \log^4 n + n\rho^4 \log^8 n)$ work, and the number of shortcuts added is $O(n\rho^2 \log^4 n)$.*

REFERENCES

[1] Nairen Cao, Jeremy T. Fineman, and Katina Russell. 2019. Efficient Construction of Directed Hopsets and Parallel Approximate Shortest Paths. arXiv:cs.DS/1912.05506

[2] Jeremy T. Fineman. 2018. Nearly Work-Efficient Parallel Algorithm for Digraph Reachability. In *Proceedings of the 50th Annual ACM SIGACT Symposium on the Theory of Computation.* 457–470.

[3] Arun Jambulapati, Yang P. Liu, and Aaron Sidford. 2019. Parallel Reachability in Almost Linear Work and Square Root Depth. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 1664–1686. https://doi.org/10.1109/FOCS.2019.00098

[4] Philip N Klein and Sairam Subramanian. 1997. A Randomized Parallel Algorithm for Single-Source Shortest Paths. *J. Algorithms* 25, 2 (Nov. 1997), 205–220. https://doi.org/10.1006/jagm.1997.0888

[5] Thomas H. Spencer. 1997. Time-work Tradeoffs for Parallel Algorithms. *J. ACM* 44, 5 (Sept. 1997), 742–778. https://doi.org/10.1145/265910.265923

[6] Jeffrey D. Ullman and Mihalis Yannakakis. 1991. High Probability Parallel Transitive-closure Algorithms. *SIAM J. Comput.* 20, 1 (Feb. 1991), 100–125. https://doi.org/10.1137/0220006