The focus today will be on parallel algorithms for reachability and shortest paths. We want to parallelize the work on computer, the problem is there is dependency in the graph. We define the max chain of dependency as the depth of the graph, and amount of things processed as work.

Parallel algorithms run in depth proportional to the hop count between source to sink: that is, the number of edges between them. So a goal in making shortest path algorithms provably parallel is to add edges that provably reduce the number of hops needed. Let $D^{(h)}_{ij}$ represent the min length of a h-hop path from $i$ to $j$. we have

$$D^{(2h)}_{ij} = \min_k (D^{(h)}_{ik} + D^{(h)}_{kj}) \leq 2h$$

There are several ways of doing this: matrix powering is probably the most direct and most powerful. However, it creates dense graphs, thus leading to costs of $n^3$ (that can be accelerated to $n^{2.37...}$ in more restricted cases such as reachability). The depth is $O(\log^3 n)$. The state of the art algorithm can do directed graph $s$ to $t$ parallel shortest path with work $O(m \log^* n)$ and depth $O(m^{1/2+\epsilon})$ [JLS19].

Much better hopsets can be found: consider the following randomized scheme: pick $n$ random pairs of vertices, put in shortest paths between them. We can prove that any two paths of length $\sqrt{n}$ contains a pair of such edges, thus, the hop diameter is reduced to $\sqrt{n}$.

For such a reduced hop count, we can:

1. Compute reachability.
2. Compute $\epsilon$-approximate shortest paths.

In $O(\sqrt{n} / \epsilon)$ depth, and $O(m \log n)$ work.

To build such a hopset more efficiently, we can:

1. Pick $\sqrt{n}$ random vertices, $S$.
2. Compute shortest paths from those vertices to everywhere, for up to $\sqrt{n}$ hops.
3. Compute all pairs shortest path between vertices in $S$ (using matrix powering).

This leads to an $O(\sqrt{n})$ depth, $O(m \sqrt{n})$ work algorithm.

Pick $n$ vertices at random as "distinguished" nodes, then build from them: do $\sqrt{n}$ steps of shortest path.
This is based on the fact that we pick \( s \) distinguished node, a path is unlikely to have a gap with length \( O((n \log n)/s) \) with no distinguished node. Choose \( \sqrt{n} \) as \( s \) gives a failure probability of
\[
\left(1 - \frac{1}{\sqrt{n}}\right)^L
\]
which when we substitute in \( L = c \log n \sqrt{n} \) gives
\[
\left(\frac{1}{e}\right)^{c \log n} = n^{-c}.
\]

All the distances are recorded in the \( \sqrt{m} \log n \)-limited BFS. Can carry out all search in \( o(\sqrt{m} \log n \log^* n) \) time with \( p = m/\sqrt{n} \) processors. Then form a graph, the edge with weight that is minimum path size in \( \sqrt{m} \log n \)-limited BFS. There are \( s = \sqrt{n} \) vertices in the graph, and we compute all distance in \( o(\sqrt{n^3 \log n}) \) time.

To reduce work further, the current state-of-the-art is to either allow approximations, or only deal with reachability. Both constructions make heavy use of recursion.

1 Approximate Hop-sets

There are several ways of constructing \( O(n^\epsilon) \) depth hopsets that give \((1 + \epsilon)\) approximations. The first of these was by Cohen [Coh00], and used a more intricate decomposition scheme known as \((\beta, \omega)\) covers.

We describe a more recent approach using probabilistic low diameter decompositions from [MPVX15]. Recall these decompositions cut the graph into pieces with diameter at most \( d \), while guaranteeing that the probability of an edge being cut is at most \( O(d/\log n) \).

Then the recursive scheme is:
1. Do low diameter decomposition.
2. Connect together centers of all large pieces with true short-cut edges.
3. Recurse inside the smaller pieces.

The key is the difference between increases in cutting probabilities, and the sizes of the clusters. For simplicity, we consider a two-layer scheme on a unit weighted graph. We first cut the graph into pieces of diameter at most \( n^{0.9} \), and then build short-cutting edges between all centers with size at most \( n^{0.9} \), as well as the exact hopsets mentioned earlier on the smaller clusters.

The costs are:
1. BFS from \( n^{0.1} \) centers, for distance of up to \( n^{0.9} \): \( O(n^{0.9}) \) depth and \( O(n^{1.1}) \) work.
2. Constructing hopsets on graphs of size at most \( n^{0.9} \), but total size about \( n \): total of this can be bounded by \( n^{1.45} \).
Balancing parameters further on this actually gives us \( n^{1.33} \). The reason why it works is that we have only \( O(n^{0.1}) \) big pieces. Compute the distance from big piece center to other vertices is \( O(n^{1.1}) \). For hopsets on small pieces, size of piece is at most \( O(n^{0.9}) \), so total size is bounded by \((\text{size}^{1.5}) \leq n^{1.45}\).

For the correctness, then the main idea is that if a path goes through one of the large clusters, it can incur distortion at most a \( \log n \) factor larger than its diameter. We can consider the distance between the first large piece and the last large piece.

This schema can in fact be generalized to \( c \) layers, giving a depth/work tradeoff of of \( n^{\epsilon} \) depth for and \( n \log^{O(1/\epsilon)} \) work.

## 2 Short Cuts for Directed Reachability

Note that for eachability, undirected reachability is ‘easy’ due to greedy contraction type routines. So we will focus on directed connectivity.

Here a recent result by Fineman [Fin18], plus an improvement by Jambulapati, Liu, and Sidford [JLS19] led to hopsets with about \( \sqrt{m} \) edges. The main recursion idea is to pick a random vertex \( u \), and partition the vertices into:

1. Those reachable from \( u \)
2. Those that can reach \( u \)
3. Those that cannot touch \( u \).

The main thing to prove is that any \( s \to t \) path gets broken up about \( o(m) \) times, in expectation.

## References


