Brief Announcement: Massively Parallel Approximate Distance Sketches

Michael Dinitz
Johns Hopkins University, Baltimore, MD, United States
mdinitz@cs.jhu.edu

Yasamin Nazari
Johns Hopkins University, Baltimore, MD, United States
ynazari@jhu.edu

Abstract

Data structures that allow efficient distance estimation have been extensively studied both in centralized models and classical distributed models. We initiate their study in newer (and arguably more realistic) models of distributed computation: the Congested Clique model and the Massively Parallel Computation (MPC) model. In MPC we give two main results: an algorithm that constructs stretch/space optimal distance sketches but takes a (small) polynomial number of rounds, and an algorithm that constructs distance sketches with worse stretch but that only takes polylogarithmic rounds. Along the way, we show that other useful combinatorial structures can also be computed in MPC. In particular, one key component we use is an MPC construction of the hopsets of [2]. This result has additional applications such as the first polylogarithmic time algorithm for constant approximate single-source shortest paths for weighted graphs in the low memory MPC setting.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases Distance Sketches, Massively Parallel Computation, Congested Clique

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.42

Related Version The full version of this paper is available at `https://arxiv.org/abs/1810.09027`.

Funding Supported in part by NSF awards CCF-1464239 and CCF-1535887.

1 Introduction

A common task when performing graph analytics is to compute distances between vertices. This has motivated the study of shortest path algorithms in essentially every interesting model of computation. We focus on two models which correspond to modern big-data graph analytics: Congested Clique [6] and Massively Parallel Computation (MPC) [4]. The MPC model in particular has recently received significant attention, as it captures many modern data analytics frameworks such as MapReduce, Hadoop, and Spark. Since these are important models of distributed storage and computation, and computing distances in graphs is an important primitive, we have an obvious question: in MPC or Congested Clique, can we compute distances between nodes sufficiently quickly to support important graph analytics? While one side effect of our techniques is indeed a state of the art algorithm for shortest paths in MPC, the focus of this paper is on getting around the limitations of these models by allowing preprocessing of the (distributed) graph. We will build a data structure known as *approximate distance sketches*, which will then let us (approximately) answer any distance query using at most two rounds of network communication. So our focus is on how to compute these data structures efficiently, since once they are computed distance estimates become fast and easy. We show that in both the Congested Clique and the MPC models, we can compute oracles/sketches which essentially match the best centralized bounds in time that is only a small polynomial. In MPC, we can go even further and compute slightly suboptimal sketches in time that is only polylogarithmic.

Distance Oracles and Sketches. Even in many centralized applications, the time it takes to compute exact distances in graphs is undesriable, and similarly the memory that it would take to store all $\binom{n}{2}$ distances is also undesirable. This motivated Thorup and Zwick [5] to propose a space-efficient data structure which can quickly report an approximation of the true distance for any pair of vertices. In other words, by spending some time up front to compute this data structure, any algorithm used in the future can quickly obtain provably accurate distance estimates. More formally, an approximate distance oracle is said to have *stretch* t if, when queried on $u, v \in V$, it returns a value $d'(u,v)$ such that $d(u,v) \leq d'(u,v) \leq t \cdot d(u,v)$ for all $u, v \in V$, where $d(u,v)$ denotes the shortest-path distance between u and v. For any constant k, Thorup and Zwick's (centralized) construction has expected size $O(kn^{1+1/k})$, stretch $(2k-1)$, query time $O(k)$, and preprocessing time $O(kmn^{1/k})$. Also, this data structure can be "broken up" into n pieces, each of size $O(kn^{1/k}\log n)$, so that the estimate $d'(u,v)$ can be computed just from the piece for u and the piece for v. These are called *distance sketches*.

Model. Our main focus is the *Massively Parallel Computation*, or MPC model. In this model there is an input of size N which is arbitrarily distributed over N/S machines, each of which has $S = N^\epsilon$ memory for some $0 < \epsilon < 1$. Every machine can communicate with every other machine in the network, but each machine in each round can have total I/O of at most S. Specifically, for graph problems the total memory N is $O(|E|)$ words. The low memory setting is the more challenging (but arguably more realistic) setting in which each machine has $O(n^\gamma), \gamma < 1$ memory, which we denote by MPC(n^γ).

2 Our Results

We initiate the study of distance sketches in the MPC model. Our techniques also extends to the Congested Clique and streaming models. Exact results can be found in the full paper. We first show that distance sketches with the same guarantees as the centralized Thorup-Zwick distance oracles can be implemented in MPC, but with a polynomial (sublinear) round complexity. Since such a high round complexity is generally considered impractical, so we also give a different (but related) algorithm which achieves polylogarithmic round complexity at the price of larger stretch. More formally,

▶ **Theorem 1.** *Consider a graph $G = (V, E)$ where $m = \Omega(kn^{1+1/k}\log n)$, for some integer $k \geq 2$. Then there is an algorithm in MPC(n^γ) (with $0 < \gamma < 1$) that constructs Thorup-Zwick distance sketches with stretch $O(k^2)$ and size $O(kn^{1/k}\log n)$ and with high probability completes in $O(\frac{k}{\gamma} \cdot (\log^3 n \cdot \log^3 k)^{2\log k})$ rounds.*

As a side effect of our techniques, we immediately get an algorithm for computing approximate single-source shortest paths (SSSP). We show that we can compute an $O(1)$-approximation in only polylogarithmic time under a certain assumption on the density of the input graph.

3 Techniques

Our main approach is to combine constructions of *hopsets* with efficient distributed constructions of Thorup-Zwick distance oracles/sketches. In particular, Das Sarma et al. [1] showed that Thorup-Zwick sketches could be computed in the CONGEST model, but the time depended on the graph diameter. Roughly speaking, we use hopsets to reduce the diameter of

the graph while preserving distances by adding in a carefully chosen set of weighted "shortcut" edges. We use a hopset construction proposed by Elkin and Neiman [2]. To implement their algorithm in the MPC model, we need to handle some technical difficulties particularly when the space per machine is $o(n)$. Not surprisingly, both [1] and [2] use as a fundamental primitive a "restricted" version of the classical Bellman-Ford shortest-path algorithm that ends early. Hence the first step for us is implementing this restricted Bellman-Ford in the MPC model. When implementing restricted Bellman-Ford in low-memory MPC, the main difficulty is that since the memory at each server is $o(n)$, a single server cannot "simulate" a node in Bellman-Ford. It takes many machines to store the edges incident on any particular node. We first show that it is possible to implement Bellman-Ford in low-memory MPC with very little additional overhead. Once we develop this tool, we argue that the hopsets of [2]) can be constructed in MPC. Our implementation of Bellman-Ford and this hopset construction, as well as a few other primitives we develop for low-memory MPC (e.g., finding minimum or broadcasting on a range of machines), may be of independent interest.

Directly implementing the hopset algorithm of [2] requires a polynomial number of rounds to obtain polylogarithmic hopbound. Even after using hopsets, we would still need polynomial time to construct constant stretch distance sketches. We overcome this issue and improve the running time using two ideas. First, we show that by relaxing the model to allow small additional total memory (either through extra space per machine or additional machines), we can run our algorithms in polylogarithmic number of rounds. In other words, the MPC model is very delicate: a small polynomial amount of extra space allows us to decrease running times not just by that polynomial, but from polynomial to polylogarithmic. So we just need to argue that there is a way of obtaining extra memory without actually changing the model assumptions. This is our second idea: by constructing a spanner we can sparsify the graph while keeping the memory per machine and number of machines the same. Thus from the perspective of the spanner, it will appear that we do indeed have "extra" memory. The idea of sparsifying the input to obtain extra resources has already proved to be powerful in related contexts (for example, [3] recently used spanners to give a work-efficient PRAM metric embedding algorithm). To the best of our knowledge, though, this idea has not yet appeared in the MPC graph algorithms literature.

References

1 M. Dinitz A. Sarma and G. Pandurangan. Efficient distributed computation of distance sketches in networks. *Distributed Computing*, 2015.

2 M. Elkin and O. Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *FOCS*, 2016.

3 S. Friedrichs and C. Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. *Journal of the ACM (JACM)*, 2018.

4 P. Koutris P. Beame and D. Suciu. Communication steps for parallel query processing. In *PODS*, 2013.

5 M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 2005.

6 E. Pavlov Z. Lotker, B. Patt-Shamir and D. Peleg. Minimum-weight spanning tree construction in O (log log n) communication rounds. *SIAM Journal on Computing*, 2005.