Brief Announcement: MapReduce Algorithms for Massive Trees

MohammadHossein Bateni
Google Research, New York

Soheil Behnezhad
University of Maryland

Mahsa Derakhshan
University of Maryland

MohammadTaghi Hajiaghayi
University of Maryland

Vahab Mirrokni
Google Research, New York

Abstract

Solving large-scale graph problems is a fundamental task in many real-world applications, and it is an increasingly important problem in data analysis. Despite the large effort in designing scalable graph algorithms, many classic graph problems lack algorithms that require only a sublinear number of machines and space in the input size. Specifically when the input graph is large and sparse, which is indeed the case for many real-world graphs, it becomes impossible to store and access all the vertices in one machine – something that is often taken for granted in designing algorithms for massive graphs. The theoretical model that we consider is the *Massively Parallel Communications* (MPC) model which is a popular theoretical model of MapReduce-like systems. In this paper, we give an algorithmic framework to adapt a large family of dynamic programs on MPC. We start by introducing two classes of dynamic programming problems, namely "(poly log)-expressible" and "linear-expressible" problems. We show that both classes can be solved efficiently using a sublinear number of machines and a sublinear memory per machine. To achieve this result, we introduce a series of techniques that can be plugged together. To illustrate the generality of our framework, we implement in $O(\log n)$ rounds of MPC, the dynamic programming solution of fundamental problems such as minimum bisection, k-spanning tree, maximum independent set, longest path, etc., when the input graph is a tree.

2012 ACM Subject Classification Theory of computation → MapReduce algorithms, Theory of computation → Massively parallel algorithms, Theory of computation → Dynamic programming

Keywords and phrases MapReduce, Trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.162

Acknowledgements Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.

1 Introduction

With the inevitable growth of the size of datasets to analyze, the rapid advance of distributed computing infrastructure and platforms (such as MapReduce, Spark [11], Hadoop [10], Flume [6], etc.), the need for developing better distributed algorithms is felt far and wide nowadays. The past decade has seen a lot of progress in studying important computer science problems in the large-scale setting, which led to either adapting the sequential algorithms to modern parallel settings or at times designing from scratch algorithms for these problems [1, 2, 4, 7, 5]. Despite this trend, we still have limited theoretical understanding of the status of several fundamental problems when it comes to designing large-scale algorithms. In fact, even simple and widely used techniques such as the greedy approach or dynamic programming seem to suffer from an inherent sequentiality that makes them difficult to adapt in parallel or distributed settings on the aforementioned platforms. Finding methods to run generic greedy algorithms or dynamic programming algorithms on MapReduce, for instance, has broad applications. This is the main goal of this paper.

We consider the Massively Parallel Communication ($\mathcal{MPC}$) model among previously studied MapReduce-like models [9, 8, 3]. Let n denote the input size and let m denote the number of available machines which is given in the input. At each round, every machine can use a space of size $s = \tilde{\mathrm{O}}(n/m)$ and run an algorithm that is preferably linear time (but at most polynomial time) in the size of its memory. Machines may only communicate between the rounds, and no machine can receive or send more data than its memory.

In this paper, we give an algorithmic framework that could be used to simulate many natural dynamic programs on trees. Indeed we formulate the properties that make a dynamic program (on trees) amenable to our techniques. These properties, we show, are natural and are satisfied by many known algorithms for fundamental optimization problems. To illustrate the generality of our framework, we design $\mathrm{O}(\log n)$ round algorithms for well-studied graph problems on trees, such as, minimum bisection, minimum k-spanning tree, maximum weighted matching, etc.

2 Main Results

We introduce a class of dynamic programming problems which we call f-expressible problems. Here, f is a function and we get classes such as $(\operatorname{poly}\log)$-expressible problems or linear-expressible problems. Roughly speaking, f is proportional to the amount of data that each node of the tree stores in the dynamic program. Thus, linear-expressible problems are generally harder to solve than $(\operatorname{poly}\log)$-expressible problems.

(poly log)-Expressible Problems. Many natural problems can be shown to be $(\operatorname{poly}\log)$-expressible. For example, the following graph problems are all $(\operatorname{poly}\log)$-expressible if defined on trees: maximum (weighted) matching, vertex cover, maximum independent set, dominating set, longest path, etc. Intuitively, the dynamic programming solution of each of these problems, for any vertex v, computes at most a constant number of values. Our first result is to show that every $(\operatorname{poly}\log)$-expressible problem can be efficiently solved in $\mathcal{MPC}$. As a corollary of that, all the aforementioned problems can be solved efficiently on trees using the optimal total space of $\tilde{\mathrm{O}}(n)$.

▶ **Theorem 1.** *For any given m, there exists an algorithm that w.h.p. solves any $(\operatorname{poly}\log)$-expressible problem in $\mathrm{O}(\log n)$ rounds of $\mathcal{MPC}$ using m machines while each machine uses a space of size at most $\tilde{\mathrm{O}}(n/m)$.*

Proof sketch. The first problem in solving dynamic programs on trees is that there is no guarantee on the depth of the tree. If the given tree has only logarithmic depth one can obtain a logarithmic round algorithm by simulating a bottom-up dynamic program in parallel, where nodes at the same level are handled in the same round simultaneously. This is reminiscent of certain parallel algorithms whose number of rounds depends on the diameter of the graph. Unfortunately the input tree might be quite unbalanced, with superlogarithmic depth. An extreme case is a path of length n. In this case we can partition the path into equal pieces (despite not knowing a priori the depth of each particular node), handling each piece independently, and then stitching the results together. Complications arise, because the subproblems are not completely independent. Things become more nuanced when the input tree is not simply a path. To resolve this issue, we adopt a celebrated *tree contraction* method of parallel computing to our model. Roughly speaking, the algorithm decomposes the tree into pieces of size at most $\tilde{\mathrm{O}}(m)$ (i.e., we can fit each component completely on one machine), with small interdependence. Omitting minor technical details, the latter property allows us to almost independently solve the subproblems on different machines. This results in a partial solution that is significantly smaller than the whole subtree; therefore, we can send all these partial solutions to a master machine in the next round and merge them.

Linear-Expressible Problems. Although many natural problems are indeed (poly log)-expressible, there are instances that are not. Consider for example the *minimum bisection problem*. In the natural dynamic programming solution of this problem, for a subtree T of size n_T, we store $\mathrm{O}(n_T)$ different values. That is, for any $i \in [n_T]$, the dynamic program stores the weight of the optimal coloring that assigns blue to i vertices and red to the rest of $n_T - i$ vertices. This problem is not necessarily (poly log)-expressible unless we find another problem specific dynamic programming solution for it. However, it can be shown that minimum bisection, as well as many other natural problems, including k-spanning-tree, k-center, k-median, etc., are linear-expressible. It is notoriously more difficult to solve linear-expressible problems. However, using a slightly more total memory, we show that it is still possible to obtain the same result.

▶ **Theorem 2** (Main Result). *For any given m, there exists an algorithm that w.h.p. solves any linear-expressible problem that is splittable in $\mathrm{O}(\log n)$ rounds of $\mathcal{MPC}$ using m machines that each uses a space of size $\tilde{\mathrm{O}}(n^{4/3}/m)$*

Proof sketch. Recall that linear-expressibility implies that the dynamic programming data on each node, can be as large as the size of its subtree (i.e., even up to $\mathrm{O}(n)$). Therefore, even by using the tree decomposition technique, the partial solution that is computed for each component of the tree can be linear in its size. This means that the idea of sending all these partial data to one master machine, which worked for (poly log)-expressible problems, does not work here since when aggregated, they take as much space as the original input. Therefore we have to distribute the merging step among the machines.

Assume for now that each component that is obtained by the tree decomposition algorithm is contracted into a node and call this *contracted tree*. The tree decomposition algorithm has no guarantee on the depth of the contracted tree and it can be super-logarithmic; therefore a simple bottom-up merging procedure does not work. However, it is guaranteed that the contracted tree itself (i.e., when the components are contracted) can be stored in one machine. Using this, we send the contracted tree to a machine and design a *merging schedule* that informs each component about the round at which it has to be merged with each of its neighbours. The merging schedule ensures that after $\mathrm{O}(\log n)$ phases, all the components are merged together. The merging schedule also guarantees that the number of neighbours of the components, after any number of merging phases, remains constant. This is essential to allow (almost) independent merging for many linear-expressible problems such as minimum bisection. Observe that after a few rounds, the merged components grow to have up to

$\Omega(n)$ nodes and even the partial data of one component cannot be stored in one machine. Therefore, even merging the partial data of two components has to be distributed among the machines. For this to be possible, we use a *splitting* technique of independent interest that requires a further *splittability* property on linear-expressible problems. Indeed we show that the aforementioned linear-expressible problems have the splittability property, and therefore Theorem 2 implies they can also be solved in $O(\log n)$ rounds of $\mathcal{MPC}$.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012. URL: `http://portal.acm.org/citation.cfm?id=2095156&CFID=63838676&CFTOKEN=79617016`.

2 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 574–583. ACM, 2014.

3 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 273–284. ACM, 2013.

4 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1326–1344, 2016. `doi:10.1137/1.9781611974331.ch92`.

5 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1234–1251, 2015. `doi:10.1137/1.9781611973730.82`.

6 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

7 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1217–1233, 2015. `doi:10.1137/1.9781611973730.81`.

8 Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

9 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

10 Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

11 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, 2010. URL: `http://dl.acm.org/citation.cfm?id=1863103.1863113`.