

Parallel Framework for Updating Large Scale Dynamic Networks

SUDHARSHAN SRINIVASAN, University of Oregon, USA

AASHISH PANDEY, University of North Texas, USA

ARINDAM KHANDA, Missouri University of Science and Technology, USA

SRIRAM SRINIVASAN, Virginia Commonwealth University, & National Radiation Oncology Program, Department of Veteran Affairs, USA

SAJAL K. DAS, Missouri University of Science and Technology, USA

Additional Key Words and Phrases: Graph Theory, Parallel Algorithm, Dynamic Network

ACM Reference Format:

Sudharshan Srinivasan, Aashish Pandey, Arindam Khanda, Sriram Srinivasan, and Sajal K. Das. 2021. Parallel Framework for Updating Large Scale Dynamic Networks.

1 INTRODUCTION

Network analysis has become an important tool for studying large-scale systems of interacting entities that arise in diverse domains such as bioinformatics, sociology, and epidemiology. Properties of networks (or graphs), such as centrality metrics, communities, can provide larger insights into the characteristics of the underlying systems.

Since the networks are extremely large, parallel algorithms are the need for timely analysis. However, developing scalable parallel algorithms for large-scale graphs is very challenging. This is because graph traversal is the primary component of many network algorithms. Traversal over unstructured data, such as networks, lead to irregular memory accesses resulting in low scalability and high computation costs. The problem is even more difficult when the networks are dynamic, that is, their structure changes with time.

In this poster, we present a parallel framework for creating scalable parallel algorithms for updating the properties of dynamic networks.

2 KEY CONTRIBUTIONS

Our key contribution is that we create a parallel framework for analyzing key properties of dynamic networks. In a dynamic network, when graph structure changes with time, our approach updates the existing property of previous time instance, and avoids re-computation of the property from scratch. Except Page Rank, we have seen that re-computation of the property is expensive and our proposed framework updates the network faster. Earlier researches [3–7] considered developing algorithms on a case-by-case basis. In our poster, we show that our framework can produce scalable algorithms on shared memory, distributed platform, and GPU machines for updating Connected Paths,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

Manuscript submitted to ACM

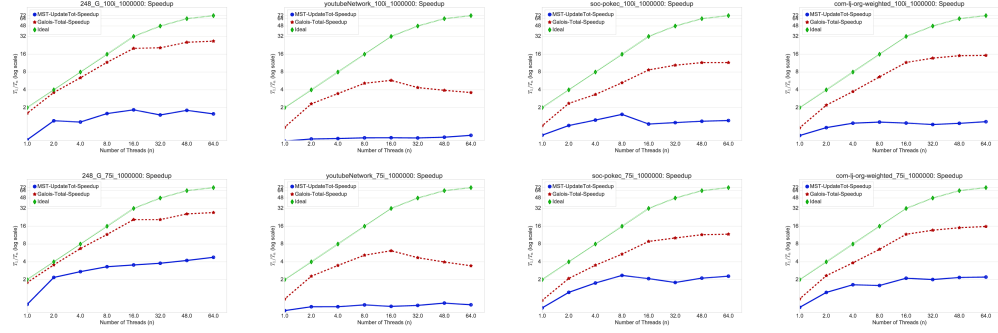


Fig. 1. Scalability Plot for our Proposed MST Implementation

Minimum Weighted Spanning Tree, Single Source Shortest Path, Strongly Connected Components, and Page Rank in large networks.

Overview of the Framework: The main target of our framework is to reduce the required execution time and computing resources by confining the computation only on affected subgraph. Here, the affected subgraph implies the part of the graph for which the graph property changes due to graph structure change. In case of Minimum Spanning Tree (MST) or Single Source Shortest Path (SSSP) problem, graph sparsification helps to find the affected subgraph. More specifically, a sparsification tree is generated for the original graph and the graph property is updated only when an incoming change affects the sparsification tree. E.g. in SSSP problem a new edge can affect the shortest path property, only when the edge becomes part of the Single Source Shortest Path Tree (SSSP Tree), i.e., this new edge can reduce the distance of a vertex from source. Results for SSSP update are included in the poster.

Minimum Spanning Tree (MST) Update: MST can be updated using a sparsification based approach. Our shared memory implementation achieves around 10X speedup on 16 processors (see Figure 1). To the best of our knowledge, this is the first parallel algorithm for updating a minimum spanning tree.

[7]

In case of Strongly Connected Components (SCC), our approach works on a simplified (and smaller) graph presented as meta-graph while updating the property.

Detecting SCCs on Dynamic Networks:

The framework implements a new hybrid algorithm to detect SCCs on distributed networks. The approach is considered hybrid as it is enabled with distributed, multithreaded CPU and GPU parallelism. The algorithm comprises three phases. In the first phase, we locally perform SCC over all partitions, while in the second phase, we update the partial results across the partitions in a reduced graph form. Lastly, we reapply SCC over the reduced graph at all partitions. The computations within individual partitions are highly scalable using both shared-memory CPU and GPU threads. Due to its distributed nature, our algorithm allows for the analysis of much larger networks than is previously possible. Here, the recomputation for dynamic networks are isolated to new batch of edges along with border vertices of meta-nodes.

In figure 2, we can see that the original graph on the left is partitioned across processes P1, P2, and P3. The yellow vertices represent ghost vertices(vertices that don't belong to that partition), while the green circles represent SCCs containing vertices. The thin red arrows indicate inter-process edges, while the solid white arrows represent the flow

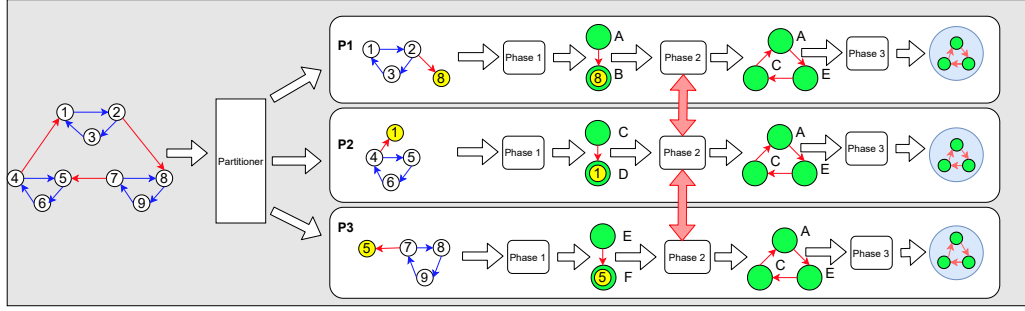


Fig. 2. Workflow of the dynamic SCC algorithm. The graph data moves from left to right as it goes through each phase of execution to produce the final strongly connected components

of execution. The subgraph at each process is transformed to a partial meta-graph in phase 1 and goes through the communication phase 2, where it is transformed to a complete meta-graph. This, in turn, goes through phase 3 to give the global SCC as shown in the last blue circles. In-progress results for SCC are included in the poster.

References

ACKNOWLEDGMENTS

This work is supported by the NSF OAC grants 2104078 and 1725755.

REFERENCES

- [1] David A Bader, Jonathan Berry, Adam Amos-Binks, Daniel Chavarría-Miranda, Charles Hastings, Kamesh Madduri, and Steven C Poulos. 2009. Stinger: Spatio-temporal interaction networks and graphs (sting) extensible representation. *Georgia Institute of Technology, Tech. Rep* (2009).
- [2] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzeig. 1997. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)* 44, 5 (1997), 669–696.
- [3] Arindam Khanda, Federico Coro, F Betti Sorbelli, Cristina M Pinotti, and Sajal K Das. 2021. Efficient Route Selection for Drone-based Delivery Under Time-varying Dynamics. In *18th International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*. IEEE.
- [4] Arindam Khanda, Sriram Srinivasan, Sanjukta Bhowmick, Boyana Norris, and Sajal K Das. 2021. A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks. *IEEE Transactions on Parallel and Distributed Systems* (2021).
- [5] Steven J Plimpton and Tim Shead. 2014. Streaming data analytics via message passing with application to graph algorithms. *J. Parallel and Distrib. Comput.* 74, 8 (2014), 2687–2698.
- [6] Sriram Srinivasan, Sanjukta Bhowmick, and Sajal Das. 2016. Application of graph sparsification in developing parallel algorithms for updating connected components. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 885–891.
- [7] Sriram Srinivasan, Samuel Pollard, Sajal K Das, Boyana Norris, and Sanjukta Bhowmick. 2018. A shared-memory algorithm for updating tree-based properties of large dynamic networks. *IEEE Transactions on Big Data* (2018).