Parallel Algorithms for Updating Large Dynamic Networks using Graph Sparsification

Sriram Srinivasan

University of Nebraska at Omaha, USA

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 insights into the characteristics of the underlying systems.

Since the networks are extremely large, parallel algorithms are essential for timely analysis. However, developing scalable parallel algorithms for networks 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 framework for creating fast and scalable parallel algorithms for updating properties of dynamic networks. Our framework is created using an elegant technique known as ***graph sparsification***. Graph sparsification uses a divide and conquer approach to updating the network properties. Specifically, using graph sparsification, the original network is divided into several small subgraphs over a structure called the sparsification tree. The sparsification tree is formed by recursively dividing the network into two halves until each subgraph represents an edge in the network. These edges form the leaves of the sparsification tree. The non-leaf nodes contain the edges connecting the two child subgraphs. The height of the tree is $O(log(V))$, where V is the number of vertices in the network.

Updating Over the Sparsification Tree. Graph sparsification is based on the observation that the edges that pertain to the property to be computed, here we term them as *key edges*, often form a very small portion of the total edges of the network. Therefore if a new edge is added, we only need to process the key edges to update the property according to the edge. If an edge from the graph is deleted, it is less likely to be part of the key edges, since key edges form a smaller percentage of the total edge. In this case, no update needs to be done. If the edge is part of the key edges, then we can delete it and replace it with a new edge from the graph. Also note that when a new edge is added or deleted, we only need to consider the subgraph to which it belongs. Based on the construction of the sparsification tree, this requires processing over at most *log(V)* nodes, i.e., the height of the tree. Figure 1 shows an example of a network, its corresponding sparsification tree. In the poster, we show several examples of how edges are inserted or deleted over the sparsification tree.

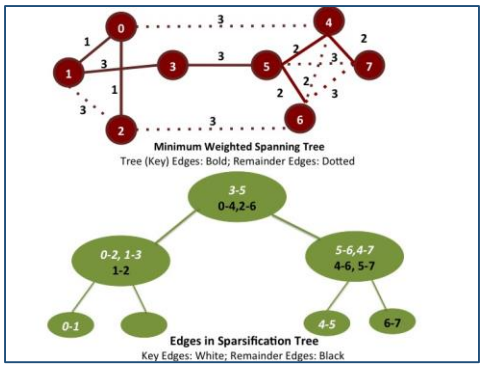

Figure 1: Example of Graph Sparsification

Parallel Implementation. Graph sparsification was introduced by Eppstein et .al.[1] in 1997. However, the process was mostly studied in the theoretical context and recently a few sequential algorithms have been presented for updating dynamic network. We present the first scalable parallel implementation of graph sparsification [2]. First, the sparsification tree and the assignment of the edges to the nodes of the tree are implemented in parallel. The complexity for this step is O(V/p). Second, for each edge, in parallel, we determine in which tree node the edge should be allocated and how the edge should be processed. The complexity for this step is O(E/p). Second, for each edge, in parallel, we determine in which tree node the edge should be

allocated and how the edge should be processed. Finally, we process the tree level by level to update the properties. In this step, nodes at each level can be processed in parallel. The complexity for each step is $O(\sum_{i=0}^{i=\log(v)}(Eav)/p)$, where Eav is the average number of edges per tree node and p, the number of threads. The scalability increases if there are more edges in the lower nodes (which can be processed in parallel) than the upper ones.

Key Contributions. Our key contribution is that we create parallel framework for analyzing key properties of dynamic network using graph sparsification. Using our framework, sequential algorithms for static networks can be converted to parallel algorithms for dynamic networks with minimal user input. In contrast, earlier research [3,4] considered developing algorithms on a case by case basis. In our poster we show that our graph sparsification can produce scalable algorithms on a shared memory machine for updating connected paths and the minimum weighted spanning tree in large networks. Our algorithms achieve around 10X speedup on 16 processors. To the best of our knowledge, this is the first parallel algorithm for updating minimum spanning tree.

The scalability is dependent on how well the edges are distributed in the nodes of the sparsification tree. We developed, *path ordering*, a vertex ordering technique, where the unbranched paths in the tree are numbered first. Thus longer paths are in the lower nodes making the updates more scalable and significantly reducing the runtime. Our future research plans include extending the framework to update SSSP, and different centrality metrics. Our future goal is implementing graph sparsification on other parallel paradigms such as distributed systems, MapReduce and GPUs and measure its performance for analyzing dynamic networks.

References

1. D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification—a technique for speeding up dynamic graph algorithms," *J. ACM JACM*, vol. 44, no. 5, pp. 669–696, 1997.
2. S. Srinivasan, S. Bhowmick, and S. Das, "Application of Graph Sparsification in Developing Parallel Algorithms for Updating Dynamic Networks," presented at the Graph Algorithms Building Blocks.
3. D. A. Bader, J. Berry, A. Amos-Binks, D. Chavarría-Miranda, C. Hastings, K. Madduri, and S. C. Poulos, "Stinger: Spatio-temporal interaction networks and graphs (sting) extensible representation," *Ga. Inst. Technol. Tech Rep*, 2009.
4. J. Plimpton and T. Shead "Streaming data analytics via message passing with application to graph algorithms". Technical Report. http://wsga.sandia.gov/docs/Plimpton_stream.pdf.