ECS-289A Special Topics in TCS, FQ 2022

Project Report

Lecturer: Slobodan Mitrović; TA: Kai Lan *Komal Bakshi and Saif Lakhani*

A Report on:
Dynamic Graph Algorithms with Batch Updates in the Massively Parallel Computation Model

1 Introduction & Motivation

This seminal paper, authored by Krzysztof Nowicki and Krzysztof Onak, provides algorithms that can efficiently handle batch updates (insertions and deletions) on well known static graph algorithms. The paper considers dynamic graph algorithms, which means the dataset is that of an evolving graph, where the change from the ith to the $(i+1)$th graph is a set of batch edge insertions and deletions.

The paper focuses on using the solution obtained for the ith graph as a precursor to solving the $(i+1)$th data set. This negates the need to compute the $(i+1)$th dataset from scratch.

The design of the algorithms emphasizes on achieving the desired result in minimum number of computation rounds for the given sublinear memory regime.

The paper highlights the following algorithms in MPC:

- Minimum Spanning Tree Problem
- 2-Edge Connected Components Problem
- Maximal Matching Problem

For the purposes of this report we show how the Minimum Spanning Tree Problem can be processed in batches of updates of size of $\Theta(S)$ where S is the space on each machine. We also show how this is achieved using a special data structure called **Top Trees**. We explain how this variant of top trees is an efficient way to perform operations in batch updates.

The motivation behind this paper is to address to substantial gap between the computation rounds amongst the two variants of the MPC model: $S \epsilon \mathcal{O}(n^{\alpha})$ and $S \epsilon \mathcal{O}(n)$. The dynamic algorithms obtained in this paper have provable lower computational complexities than their static algorithm counterparts; achieving both smaller local memory and lower round complexity.

2 Past Work

2.1 Minimum Spanning Forest Problem in MPC for static graphs

Modified variants of Boruvka's algorithm in sublinear memory regime, solves the problem in $O(\log n)$ rounds. The two cycle conjecture states that, for static graphs, the local memory bounded by $O(n^{\alpha})$ for some $\alpha < 1$, and at most polynomial number of machines, the task that can be solved with Minimum Forest algorithm has computational complexity of $\Omega(\log n)$. This is known as the famous 2-cycle problem.

3 Key Ideas

3.1 Intuition for approaching dynamic algorithms

Take MST for example. To solve this problem in MPC we take the following approach to improve upon the existing static graph algorithm, we have to ensure that we do not start the problem from scratch for

every updation. Instead we do the following:

1. G is the original graph before applying any updates
2. U is a set of external updates $\{u_1, u_2...u_k\}$
3. The minimum spanning tree of the original graph is represented by F.
4. The minimum spanning tree of the graph after applying the first k updates is F_x

We develop the set of updations that will transform the original spanning tree F into F_x rather than calculating F_x as a separate static algorithm.

3.2 Top Trees and their usefulness in MPC for batch updates.

In order to fit the dynamically changing graph into the restricted memory regime, a variant of the Top Tree data structure originally proposed by Tarjan et. al is used. Each top tree node is characterized as a cluster of a specific number of nodes in its underlying graph. The authors propose $\Theta(n^{\alpha/2}) - ary$ trees.

A diagram is shown below to help understand this structure.

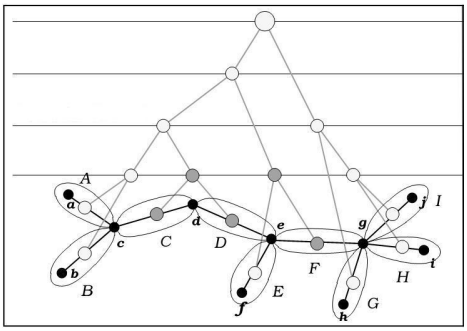

Fig.1 Top Tree

As shown in the above diagram, there are a few pre-requisites to understand:

- A and C are clusters, that are characterized by a common vertex c. They are therefore siblings.
- Each edge is a top tree node with rank 0.
- **Important** Since the the space on each machine is S, we select $\Theta(n^{\alpha/2}) - ary$ trees so that a node can fit itself, its children and its grandchildren in the same machine. This is particularly useful when the tree is imbalanced.

3.2.1 Top Tree Operations: Rebalancing Step 1

After an insertion or deletion is performed, the Top Tree is left imbalanced. This means that there are nodes in the top tree which have rank r, but have children with rank lower than $r - 1$. To perform rebalancing therefore, we need to merge a sibling of a node with lower rank into the sibling with higher rank. This however, is subject to the constraint that the siblings must share a common vertex, also known as a *boundary vertex*.

3.2.2 Top Tree Operations: Rebalancing Step 2

After Step 1 is performed, we will have no nodes that are underloaded, i.e, have rank less than $r-1$, however, we may have nodes that are *overloaded*. For this step, we gather all the children of the current node and its grandchildren (remember that they fit on a single machine). So to balance this tree, we disconnect the grandchild from the intermediate parent and attach it to the current node. This step can be performed on a single machine.

At the end of rebalancing on all levels, the number of children could increase by at most $O(n^{\alpha/2}$, which does not violate our size constraint.

3.3 Boruvka's Algorithm

3.3.1 Static implementation of Bouruvka's Algorithm.

Algorithm 1 Boruvka's Static Algorithm

Break the graph into a several unconnected trees. In this first step, we make each vertex its own tree.
For each vertex v:
  Find the minimum weighted edge
  Connect the corresponding unconnected trees with that edge.
  If the trees are already connected, skip v.

We have also included a diagram for Boruvka's static algorithm herewith.

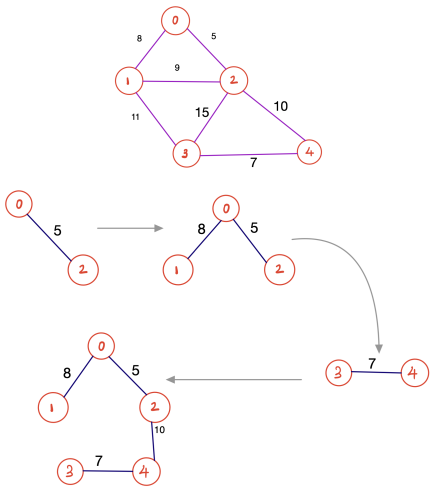

Fig. 2 Boruvka's Static Algorithm

3.3.2 Dynamic implementation for Boruvka's Algorithm for MPC

Assume we begin with a MST for an existing subgraph of the actual graph. When a new edge is added or deleted in the underlying graph, we compute the batch update steps as given below:

Growing the Forest

1. Each vertex in the components of the growing Minimum spanning forest flips a coin.
2. All vertices with heads in one component are connected with all vertices with tails in the other component. This is called merging.

3. When this merging takes place, the top trees of the corresponding components are merged as well.
4. After this operation is performed, it may be possible that the graph will have formed a cycle.
5. In order not to violate the cycle property, we then proceed to remove the edge with the heaviest weight in the cycle.
6. This cycle can be detected by looking at the top tree of the recently merged components.
7. It should be noted, however, that for the jth insert operation, we do not really know what the spanning forest would be, after the $(j-1)th$ update operation, and therefore, each iteration of this merge must be performed at each updation (insertion or deletion) of the underlying graph.

4 Conclusion

The algorithm proposed in the paper for batch dynamic MST requires only $O(\log n)$ round complexity for computation when memory is limited to $O(n^{\alpha})$ This round complexity is achieved by the following steps:

1. The merging algorithm sorts all the subtrees of a single star component in $O(1/\alpha)$
2. The merging operation on the corresponding top trees will also take $O(1/\alpha)$ rounds.
3. This means that a single round of Boruvka's dynamic algorithm takes $O(1/\alpha^2)$ rounds.
4. We know that Boruvka's algorithm with high probability requires only $O(\log n)$ rounds, and therefore, if $(1/\alpha^2)$ is a constant, the total time complexity of batch dynamic MST is $O(\log n)$

5 Future Work

- The motivation behind using the Boruvka's algorithm, is that it loops through each vertex unlike other MST algorithms like Kruskal's, which requires sorting of the graph nodes at every iteration. However, in the MPC realm, this step can be parallelized. This means that Boruvka's doesn't have any real advantages over Boruvka's in practice. A future direction that researchers can take is the implementation of other, more well known, MST algorithms like Prim's and Kruskal's for dynamic batch updates in the sublinear memory regime. Durbhakula et. al in [3] creates exactly such an approach using lock variables.
- The authors used a variation of top-trees that required merging to occur only if two clusters shared a common vertex. However, as graphs get more sparse, especially in real world scenarios like cliques, this occurence gets more rare, and therefore requires more rounds for merging and balancing. A future direction could be the development of a more efficient data structure for storing graph clusters.

6 References

1. Nowicki, Krzysztof, and Krzysztof Onak. "Dynamic graph algorithms with batch updates in the massively parallel computation model." Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). Society for Industrial and Applied Mathematics, 2021.
2. Tarjan, Robert Endre, and Renato Fonseca F. Werneck. "Self-adjusting top trees." SODA. Vol. 5. 2005.
3. Durbhakula, Suryanarayana Murthy. "Parallel Minimum Spanning Tree Algorithms and Evaluation." arXiv preprint arXiv:2005.06913 (2020).