UPDATING MINIMUM WEIGHTED SPANNING TREES IN PARALLEL

Sriram Srinivasan

In Collaboration With

Sanjukta Bhowmick and Sajal Das

University of Nebraska at Omaha

Missouri University of Science and Technology

SIAM CSE2017

Feb 28 2017

Minimum Weighted Spanning Tree (MST)

- Select a subset of edges from an undirected weighted graph (V,E), such that
- (i) all the vertices are connected
- (ii) the sum of the total edges is minimized

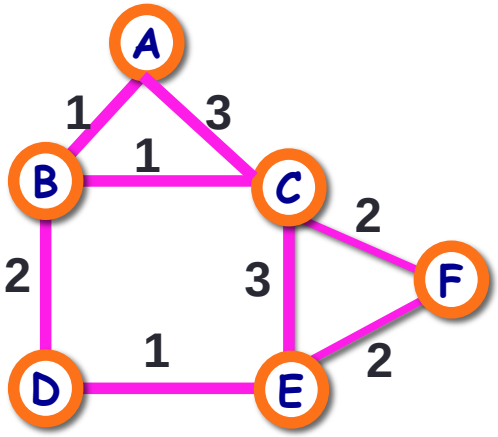
- Sequential Algorithms: Kruskal's *O(ElogV)* Prim's *(E+VlogV)*
- Parallel Algorithms: Boruvka's.

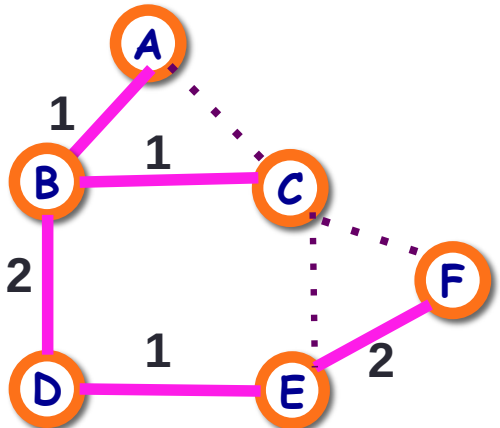
- Applications: Cluster Analysis, Circuit Design, Approximating TSP

- **Our goal is to develop a parallel algorithm for updating MST as new edges are added and old edges are deleted.**

Graph Sparsification

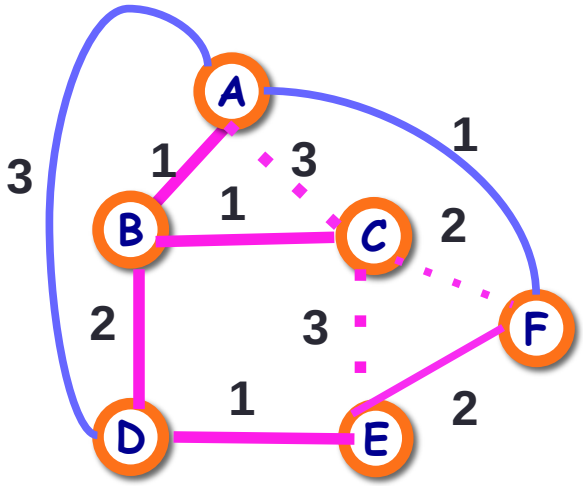
- Issue: Massive size of the graphs makes it difficult to identify which portions to update
- Solution: Use sparsification to identify only the edges that are important to the property under consideration
  - Key Edges: Edges pertaining to the property (here edges in MST)
  - Remainder Edges: Remaining edges (here everything but MST)



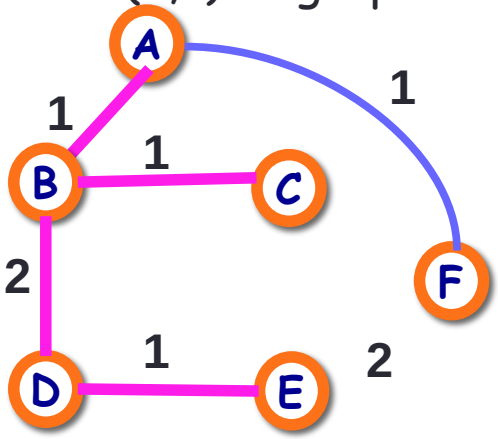


Insertion Operation

- Edge (u,v) with weight W to be inserted
- Find path in MST from vertex u to vertex v
- Find the maximum weighted edge (x,y) in the path (wt=maxW)
- If (maxW > W):
  - Add (u,v) to MST; Delete (x,y)
- Else:
  - Add (u,v) to graph but not MST



Add edges (A,F:1) and (A,D:3)

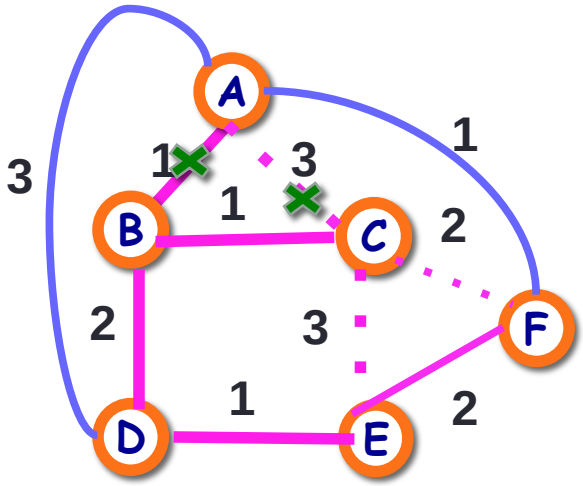


Heaviest Edge in Path A—F in MST is B-D:2 or E-F:2
Replace either with A-F:1

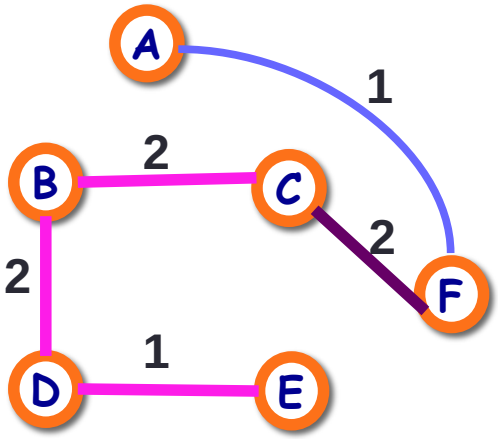
Heaviest Edge in Path A—D in MST is B-D:2
Do not replace A-D:3

Deletion Operation

- Deletion Operation
  - Delete Edge (u,v) from the graph
  - Reconnect the tree (if possible) by finding minimum weighted edge connecting the two parts.



Deleted edge (A,B:1) from MST
Deleted edge (A,C:3) from remainder



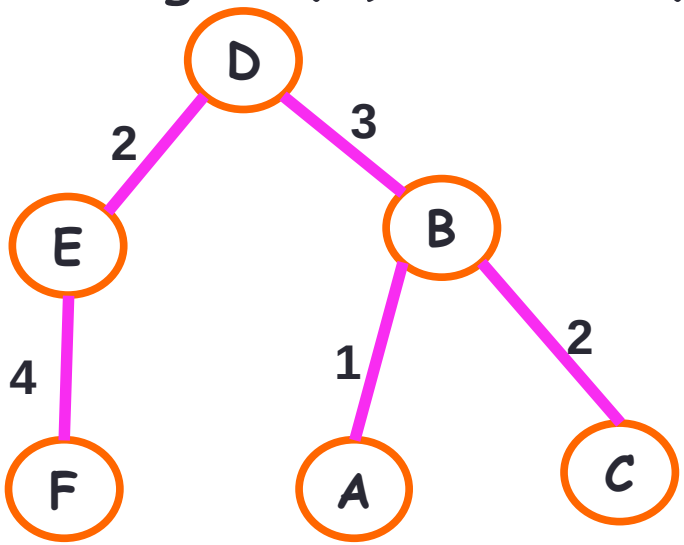
Deleted edge A-B:1
Added C-F:2 from remainder to rejoin the tree

Issues with Insertion-I

- Finding the path between (u,v) for insertion—worst case complexity *O(V+E)*
- Complexity of simply re-doing the MST *O(ELogV)*
- Therefore over multiple insertions time to update will be more than time to re-compute MST
- Solution: Store the paths (or maximum weighted edges) between vertex pairs. Requires $O(V^2)$ storage

Finding Maximum Weighted Edges

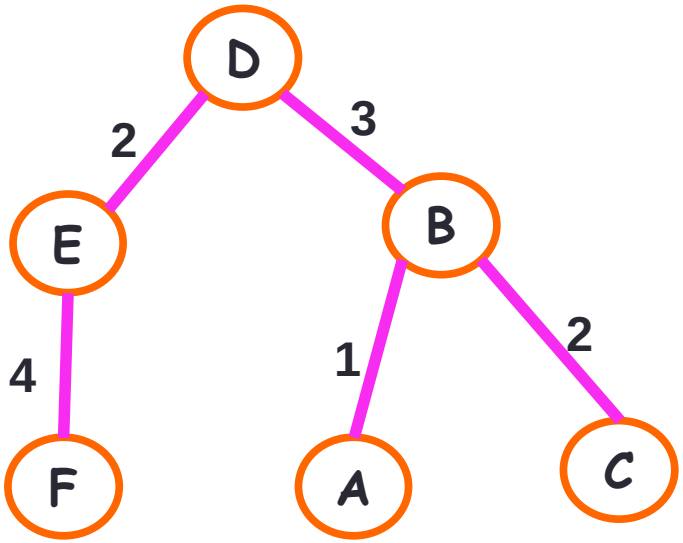
- Find path from a designated root to all other vertices
- Mark the edges that have maximum weight in these paths
- Storage *O(V)*; Time *O(V+E)*



Root D

D:B (D-B)3
D:C (D-B) 3
D:A (D-B) 3
D:E (D-E) 2
D:F (E-F) 4

Finding Maximum Weighted Edges



Case 1: (F:C) Max Weight Edges are Different
Max Weight from F:D (E-F) 4
Max Weight from C:D (B-D) 3

Pick the highest weight edge (E-F) 4

Case 2: (A:C) Max Weight Edges are Same
Max Weight from A:D (B-D) 3
Max Weight from C:D (B-D) 3

Find path from A-C and then find max weighted edge B:C 2

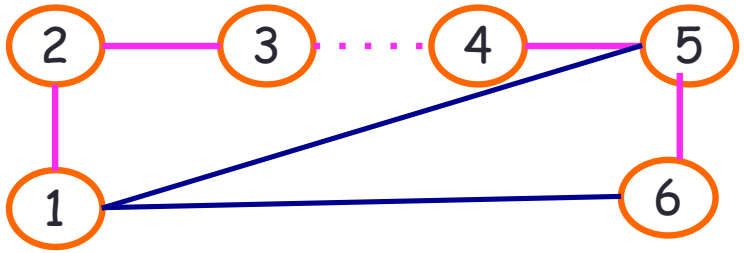
If we keep track of the parent, the complexity of this at most O(h);
h=height of the tree

Selection of Root

- We need to select the root such that height of tree is minimized
- Assume we have no control over the original MST
- Find the longest path in the tree. Then select the vertex in the center of the path as the root
- Best Case $O(\log_k V)$: k=average branching
- Worst Case $O(V/2)$ but 50% chance that the max weighted edges will be different

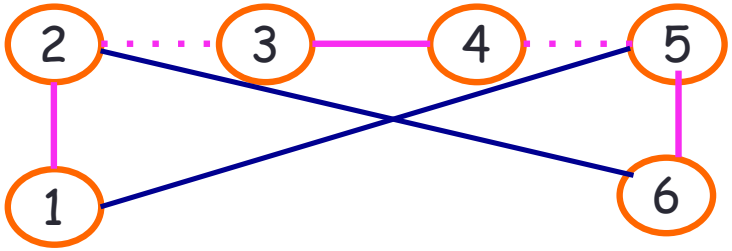
Issues with Insertion-II

Inserting edges in parallel can lead to cycles



Case 1: Edges 1-5 and 1-6 are inserted in parallel
Both find 3-4 as the edges to be deleted
If added without synchronization creates a cycle

Solution: Mark 3-4 with id of edge replacing it.
Only one id is possible



Case 2: Edges 1-5 and 1-6 are inserted in parallel
1-5 replaces 2-3 and 2-6 replaces 4-5
If added without synchronization creates a cycle

Solution: 2-3 and 4-5 must have the same weight.
Break ties by selecting edges with lower vertex ids.
Reduces to Case 1

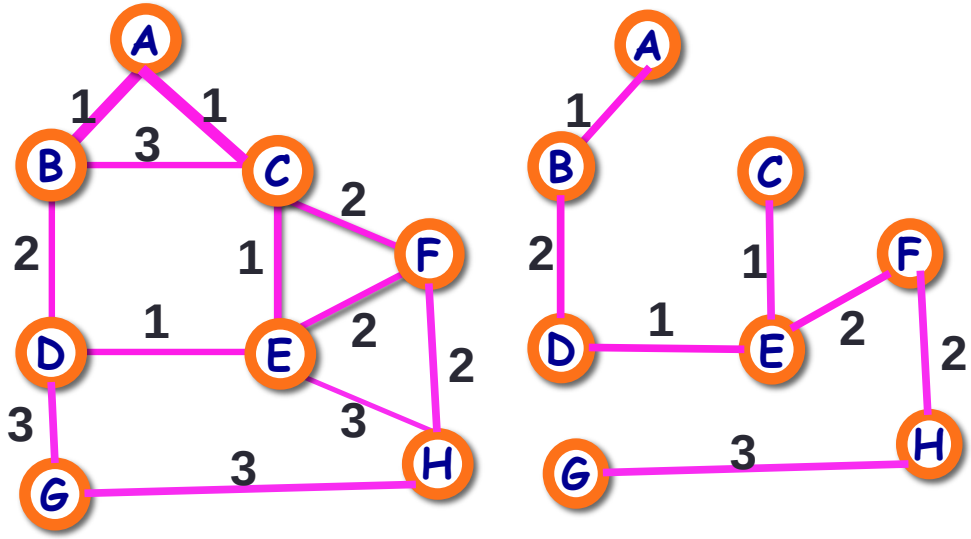
Issues with Deletion

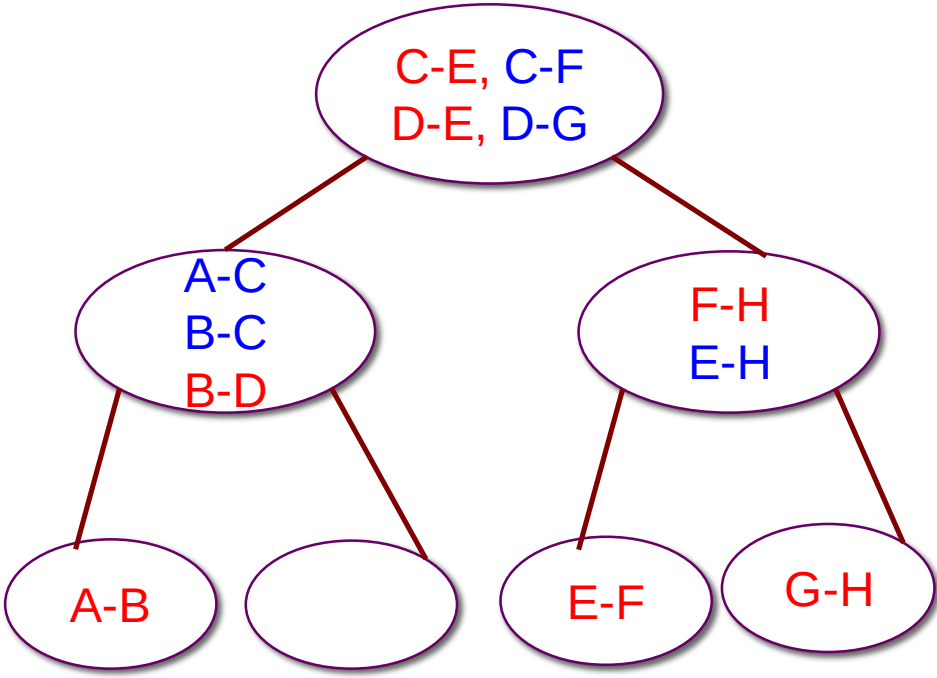
- Deletion can be done in parallel by simply marking the edge as deleted
- Finding edge to recombine the broken trees is expensive
- May need to search all remainder edges *O(E)*

- Solution: Number of deletions in MST is less (4-5% of # of changes). Use Boruvka to update
- Keep remainder edges in min-heap
- Reduce the number of edges to search by using sparsification tree

Sparsification Tree

- Introduced by Eppstein in 1997
  - Sparsification—a technique for speeding up dynamic graph algorithms by Eppstein et. al. JACM 1997
- Divided the edges of the graph into a binary tree
- Each node in the tree represents a subgraph



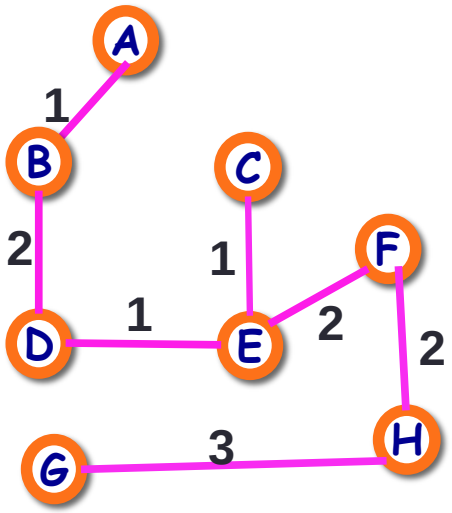


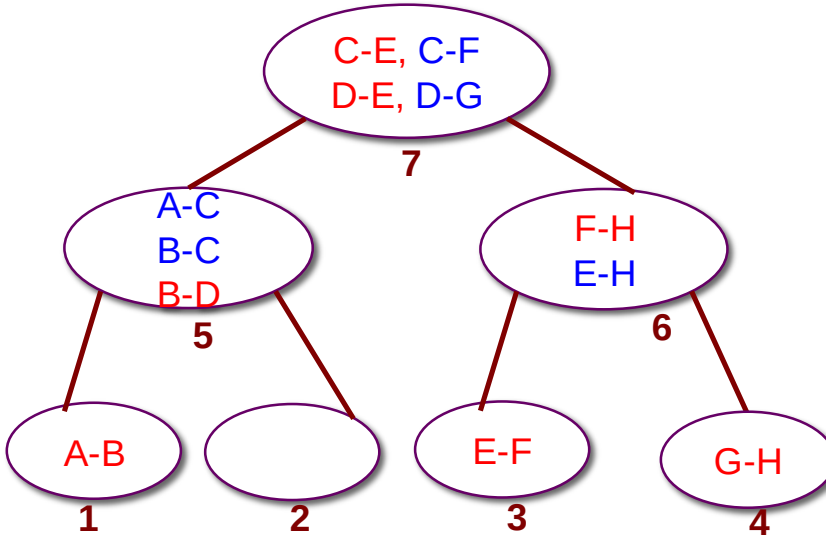
Sparsification Tree

- Sparsification tree reduce the number of edges we need to consider

- For Deletion (u,v)
  - Only consider edges from the from the node where (u,v) are in the same component to leaf
  - Delete (B-D): Search in nodes 5,1,2

- For Insertion (u,v)
  - Only traverse the subgraph from the node where (u,v) are in the same component to leaf
  - Insert (G-F): Traverse through subgraph at nodes 6,3,4





Putting It All Together

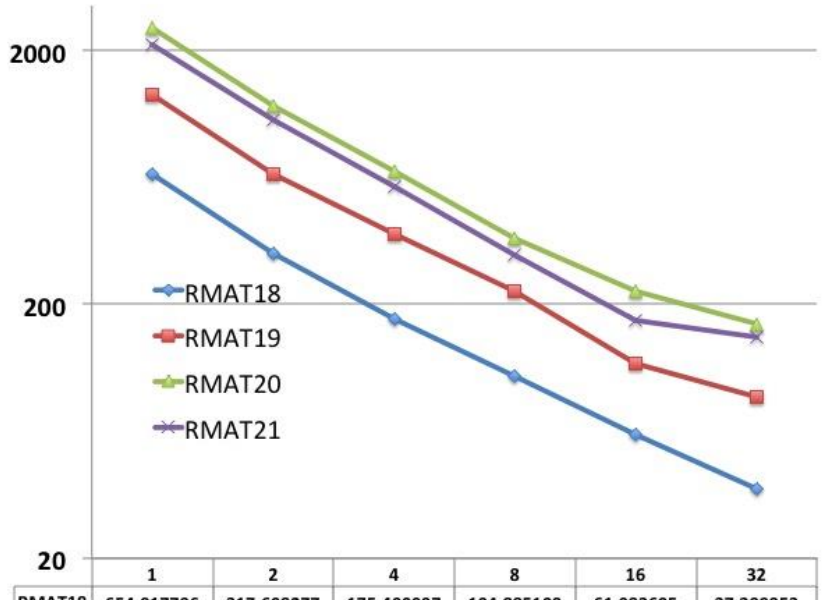
- Input: MST, Original Graph, Set of Changed Edges
- Output: Updated MST
- Create Sparsification Tree
- Place key edges and remainder edges in sparsification tree
- Select root of MST and find distances of all vertices
- Each changed edge is processed in parallel
  - Insertion (u,v)
    - Find maximum weighted edge in path from (u,v)
    - Replace as necessary
  - Deletion (u,v)
    - Delete edge from MST
    - Rejoin by checking remainder edges

Experimental Setup

- Datasets
  - Experiments on random graphs created using RMAT
  - Vertices: 2^18—2^21
  - Edges: 8XVertices

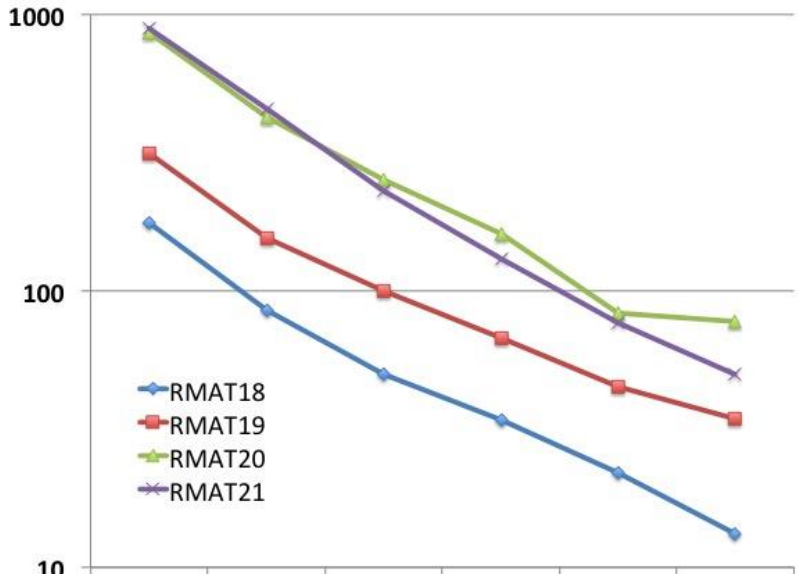
- Machine
  - Tusker at Holland Computing Centre
  - 6,784 cores interconnected with Mellanox QDR Infiniband along with 523TB of Lustre storage. Each compute node has 256 GB RAM and 4 Opteron 6272 (2.1 GHz) processors.
  - Shared memory implementation using OpenMP

Scalability Results (repeated traversals-no rooted tree)



	1	2	4	8	16	32
RMAT18	654.917796	317.608277	175.400097	104.885109	61.083695	37.399953
RMAT19	1329.310264	649.498618	378.030749	224.743045	116.380785	85.929262
RMAT20	2466.222346	1210.160605	669.860358	366.364035	225.403517	167.52635
RMAT21	2099.893274	1067.021843	579.958594	311.885123	172.921612	148.483177

10K changes insertions and deletions mixed

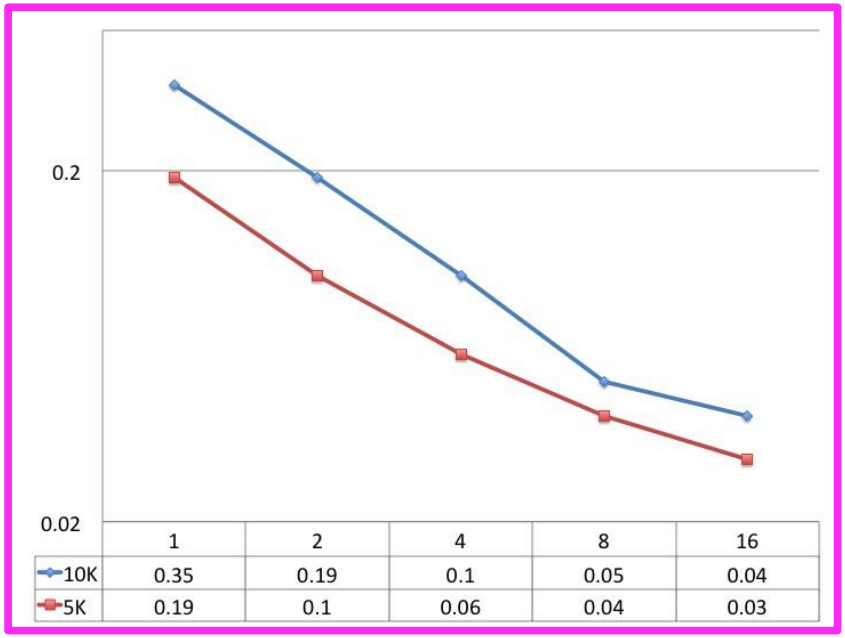


	1	2	4	8	16	32
RMAT18	176.066872	85.25837	50.169395	34.008163	21.945669	13.254299
RMAT19	313.008632	155.446662	99.563333	67.572201	44.983738	34.469839
RMAT20	855.978981	424.648862	252.308515	160.601264	83.410592	77.497536
RMAT21	887.986925	453.086756	228.74054	131.115391	76.700702	49.92099

5K changes insertions and deletions mixed

Time in seconds

Scalability Results (Rooted Tree)



Time to create rooted tree= 0.20 seconds (sequential)

Total time =.20 +time to update

	Rooted	Traversal
1	.57	654.91
2	.41	317.6
4	.32	175.4
8	.27	104.4

Conclusions

- The first parallel algorithm for updating MST
- Sparsification tree reduces amount of traversal
- Rooted tree method much faster than traversal—but need to make rooting parallel

- We can now handle updates for weighted trees.
- In lookout for other interesting properties to update

- Current code available at https://graphsparsification.herokuapp.com/
- Come to the Poster tomorrow

Funded by NSF