
Ahmad Traboulsi
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
ahmadtraboulsi@scs.carleton.ca
February 13, 2015

1 Introduction

The world of computing lead people to the thinking of optimizing all the solutions, i.e. finding the best solution to a problem from the set of all feasible solutions. Many such problems require lots of processing time and power especially with problems consisting of processing big data and sometimes its impractical to do that using serial computing due to the limitation of hardware, however nowadays parallel computers with different architectures (multicore, GPUs, Clusters) are available which allows better performance and efficiency if the hardware was utilized well using parallel programming. One of the classical graph optimization problems is Minimum Spanning Trees(MST) and another related problem this paper will be addressing and considering is the Minimum Bottleneck Spanning Tree Problem (MBST).

Formally the Minimum Spanning Tree problem is defined as follow, let $G = (V, E)$ be an undirected connected graph with a cost function $w$ mapping edges to positive real numbers. A spanning tree is an undirected tree connecting all vertices of $G$. The cost of a spanning tree is equal to the sum of the costs of the edges in the tree. A minimum spanning tree is a spanning tree whose cost is minimum over all possible spanning trees of $G$.

The Minimum Bottleneck Spanning Tree problem is defined as follow, let $G = (V, E)$ be an undirected connected graph with a cost function $w$ mapping edges to positive real numbers. The bottleneck edge of a spanning tree is the edge with the maximum cost among all edges of that tree, there might be more than one bottleneck edge in a spanning tree in which they all have the same cost. A spanning tree $T$ is called a minimum bottleneck spanning tree (MBST) if its bottleneck edge cost is minimum among all possible spanning trees.
Some of the applications of MBST and MST ([14]) are:

- Taxonomy
- Cluster analysis: clustering points in the plane, single-linkage clustering (a method of hierarchical clustering), graph-theoretic clustering, and clustering gene expression data.
- Constructing trees for broadcasting in computer networks. On Ethernet networks this is accomplished by means of the Spanning tree protocol.
- Image registration and segmentation
- Curvilinear feature extraction in computer vision.
- Handwriting recognition of mathematical expressions.
- Circuit design: implementing efficient multiple constant multiplications, as used in finite impulse response filters.
- Regionalisation of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.
- Comparing ecotoxicology data.
- Topological observability in power systems.
- Measuring homogeneity of two-dimensional materials.
- Minimax process control.

In addition to that the MBST and MST are often a key module for solving more complex graph algorithms. I’m planning to present an approach to use concurrent threads on the Reverse-Delete algorithm to solve the MBST problem. Which might be utilized to solve the MST algorithm too.
2 Literature Review

The literature has no parallel algorithms or implementations for the Minimum Bottleneck Spanning Tree problem, however for the MBST sequential algorithms I referred to [2] paper by Camerini presents one algorithm for finding minimum bottleneck spanning tree in a weighted undirected graph and another for finding a minimum bottleneck spanning tree in a directed graph, a second paper that solves the MBST in a directed graph is by Harold and Tarjan [4] which again presents a new algorithm for finding an MBST in a directed graph and a second algorithm a modified Dijkstra algorithm that finds an MBST in a directed Graph. Most of those algorithms are inherently sequential therefore not a good choice to parallelize, however in Kruskal’s algorithm original paper it includes an algorithm called reverse-delete which can be utilized to get an MST or an MBST. On the other hand there are lots of papers about finding the minimum spanning trees in a parallel computing systems with different architectures. However the most targeted algorithm in MST among the three well known algorithms Kruskal’s algorithm, Prim’s algorithm and Boruvka’s algorithms is the latter because of the fact that it is naturally parallel, whilst Kruskal’s and prim’s algorithms are inherently sequential which makes it difficult to parallelize, and as I have observed non of the papers that targeted those algorithms as their approach without the combination or modification has little to no improvements and speed ups. In the upcoming I’ve divided my reviews to several subsections according to the parallel architecture used.

2.1 MST - GPUs

Three papers were reviewed in this architecture starting from the current state-of-the-art approach by Vineet et Al [12] which gives speedup of 30 to 50 times over cpu implementation and in under one second on one quarter of Tesla S1070 GPU an MST is constructed for a graph with 5 million node and 30 million edge, their algorithm is based on Boruvka’s algorithm that uses scalable primitives such as scan, split and segmented scan in addition to efficient data mapping primitives including sort scan and reduce, its basically a recursive approach that uses a series of basic primitives at each step. A second paper by S.Bressan et Al [9] claims to outperform the current state of art algorithm by Vineet et al, in which their approach is based on Prim’s algorithm that also uses the parallel primitives namely prefix-sum, stream compaction and sorting as intermediate components of their algorithms, the algorithm let each processor to try to grow a tree using prim’s algorithm whenever a collision between two trees occur then one of the processors hands over its tree to the other and start building a tree from a new unvisited vertex the idea is somehow similar to an approach in multicore transactional memory approach to find an MST by S.Kang and D.Bader [6]. Finally the third paper by W.Wang et Al [13]similarly uses prim’s algorithm however by not parallelizing the outer loop the algorithm does not performs well when compared to the two previously mentioned papers since it doesn’t try to grow multiple trees and only tries to parallelize the two inner loops finding min weight edge and updating the candidate edge set.

2.2 MST - Multicores

A fast shared memory algorithm [1] by David A. Bader and Guiojing Cong for computing an MST in Sparse Graph gives three variants of Boruvka’s Algorithm plus a new MST algorithm
for Symmetric Multiprocessors (SMP). Their best variant of Boruvka’s algorithm has a time complexity of $O\left(\frac{m+n}{p} \log(n)\right)$ and their new MST algorithm has a time complexity in the worst case similarly $O\left(\frac{m+n}{p} \log(n)\right)$ however their new MST is interesting since it combines Prim’s and Boruvka’s algorithm in a way that the processors starts growing trees as in Prim’s algorithm then contract them as in Boruvka’s finally repeating the whole procedure again recursively but the lock free mechanism used has an excessive overhead. The second paper by David A Bader and S.Kang [6] which is based on the latter algorithm of the previous paper the new MST algorithm. This paper also target sparse graphs and provides a speedup of an average 8 up to 16 times, the algorithm grow trees using Prim’s algorithm and a processor stops growing the tree when it touches another tree, in this case processor 1 hands over its tree to processor 2 and starts all over again from a new unvisited node, the drawback of the algorithm presented is it decreases the utilization of the number of processors therefore less parallelism. The third paper in this section by A. Katsigiannis et al [7] tries to parallelize kruskals algorithm using helper threads, a main thread proceeds as the usual Kruskals algorithm while the helping threads trying to decrease the search space of the main thread. this is done by assigning to each of those helping threads a partition of the list of edges, and each processor keeps looping through its partition tying to test each edge if it would cause a cycle with the current found MST edges by the main thread, as soon as the main thread enters a helper thread partition the helper thread stops, they mentioned that a speed up of 5.5 times of the sequential kruskal’s algorithm. The drawback in this algorithm is that again the utilization of threads and processes decrease as the main thread approaches. So current state-of-the-art is the 2nd paper presented in this section for multicores, however this algorithm requires a costly inter-processor communication to merge subtrees when they do get in contact.

2.3 MST - Clusters

Parallelization of both Prim’s and Kruskal’s algorithms are presented by V.Loncar et al [11] in which a master slave approach in parallelization of Prim’s algorithm my having several processes find the min weight edge in their set of edges and vertices and finally collecting the data and processing the results. This algorithm runs in $O\left(\frac{n^2}{p} + O(n\log(p))\right)$ and a parallelization of Kruskal’s algorithm is also presented which works as follow partitions of the main graph are assigned to the processors each locally computing the MST using Kruskal’s algorithm and merging them, the time complexity of this algorithm is $O\left(\frac{n^2}{p} + O(n^2\log(p))\right)$. The second paper is based on MapReduce in which an approach of how to achieve a very simple Java implementation of Minimum Spanning Tree problem in MapReduce [10]. It only gives the implementation details no analysis was provided. Basically uses Kruskal’s algorithm as a reducer after partitioning the graph into subgraphs.

2.4 MST - abstract Machines

Two abstract machines were considered in the two papers F.Dehne and S.Gotz [5] presenting an algorithm Boruvka’s based that computes the MST by finding local MST by each processing unit then prunes and merges the resulting MSTs into a single one using D-ary tree on a BSP abstract computer. The second paper is by K.W. Chong et al [3] is an optimal time logarithmic time $O(\log n)$ on PRAM EREW abstract computer. It takes $\log(n)$ steps by using multiple threads working on different parts of the search space however as soon as one thread finishes the following thread requires only an $O(1)$ to finish and there
are \( \log(n) \) threads therefore resulting in a time complexity of the order \( O(\text{long}) \), therefore being the state-of-the-art.

### 2.5 MST - architecture Independent methodology

One paper presenting an independent platform algorithm C. da Silva Sousa et al [8] a variant of Boruvka’s algorithm. The implementation is based on a specific design and implementation decisions such as data representation. Claims to outperform all other existing algorithms, however no results were shown that compares it to the state-of-the-art algorithms stated previously. The implementation and the approach taken are interesting, and from the implementation Its obvious to state that it would perform best at a GPU architecture.

The above were material related to the Minimum Spanning Trees problem however the problem I’m resolving has not yet been touched in parallel computing therefore I will be reading more papers and doing more literature review regarding my approach to the Minimum Bottleneck Spanning trees and mainly I need to find more about parallel algorithms for connected components, since its a part of my approach to parallelizing the reverse-delete algorithm.

### References


