A parallel algorithm for minimum spanning tree on GPU - Proofs of Theorems

Jucele França de Alencar Vasconcellos,
Edson Norberto Cáceres,
Henrique Mongelli and
Siang Wun Song

September 22, 2017

1 Discussion of the Algorithm

In this section we address the correctness of the Algorithm 1. We now assume, without loss of generality, that all the weights of the edges $e \in E$ of the input graph $G = (V, E)$ to be different.

Lemma 1. *Consider a connected graph $G = (V, E)$. Let $H = (V, U, E')$ be the transformed bipartite graph and S be a strut in U, obtained at step 5 of Algorithm 1. Let G' be the graph obtained by adding the edges associated to vertices u (i.e. original_edge(u)) from U such that $d_S(u) \geq 1$ (step 7 of Algorithm 1). Then G' is acyclic. More over, if S contains exactly one zero-difference vertex then G' is a spanning tree of G.*

Proof. By definition, S is a forest in $H = (V, U, E')$ such that there is exactly one edge of E' incident to each vertex $v_i \in V$, being chosen the edge (v_i, u_x) with the smallest weight. Let w be the smallest weight considering all edges of G and, consequently, of H. Considering the strut construction process, there will be at least one vertex $u_t \in U$ that is incident to edges with weight w. For this vertex u_t both its incident edges will be in S, thus S has at least one zero-difference vertex.

Therefore, we can conclude that there will be at most $|V| - 1$ vertices u of U with degree equal to or greater than 1 in S, or $d_S(u) \geq 1$. This means that at most $|V| - 1$ edges will be added to G', one incident edge in each vertex of V, resulting in an acyclic graph. If S has only one zero-difference vertex, the generated graph G' will interconnect all its vertices using $|V| - 1$ edges, resulting in a spanning tree of G. □

Note that if S has more than one zero-difference vertex, the generated graph G' will have less than $|V| - 1$ edges and will not be a spanning tree. Thus, the algorithm will need more iterations to complete the graph G'.

Algorithm 1 Main ideas of the proposed MST algorithm

Input: A connected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices and E is a set of m edges (v_i, v_j), where v_i and v_j are vertices of V. Each edge (v_i, v_j) has a weight denoted by w_{ij}.
Output: A minimum spanning tree of G whose edges are in $SolutionEdgeSet$.

1: $SolutionEdgeSet$:= empty.
2: Transform the given graph G into a bipartite graph $H = (V, U, E')$.
3: *condition* := true.
4: **while** *condition* **do**
5:    Obtain a strut.
6:    **for** every strut-edge (v_i, u_j) **do**
7:       Add the edge (v_i, v_k) to the $SolutionEdgeSet$, where v_i and v_k are adjacent to u_j. In other words, add *original_edge*(u_j) to the $SolutionEdgeSet$.
8:    **end for**
9:    Compute the number of zero-difference vertices.
10:    **if** number of zero-difference vertices = 1 **then**
11:       *condition* := false
12:    **else**
13:       **for** every strut-edge (v_i, u_j) **do**
14:          Compact the two vertices adjacent to u_j thereby producing a new bipartite graph.
15:       **end for**
16:    **end if**
17: **end while**

Before we prove the following theorem, we give some definitions. A cut$(W, V\backslash W)$ of a graph $G = (V, E)$, with $W \subseteq V$, is a partition of V. An edge of E *crosses* the cut$(W, V\backslash W)$ if one of its endpoints is in W and the other endpoint is in $V\backslash W$. Let T be a spanning tree of G. Then the removal of any edge $e \in T$ will result in the components $(W, V\backslash W)$, where one endpoint of e is in W and the other in $V\backslash W$.

Theorem 1. *Consider a connected graph $G = (V, E)$. Let $H = (V, U, E')$ be the transformed bipartite graph and S be a strut in U, obtained at step 5 of Algorithm 1. Let G' be the graph obtained by adding the edges associated to vertices u (i.e. original_edge(u)) from U such that $d_S(u) \geq 1$ (step 7 of Algorithm 1). If S contains exactly one zero-difference vertex then G' is a minimum spanning tree of G.*

Proof. By Lemma 1 it is known that G' is a spanning tree of G. Consider a vertex $v \in V$ and the edge $(v, w) \in E$ such that the weight of (v, w) is the smallest among all edges incident to v. By step 7 of Algorithm 1, edge (v, w) is added to the set of edges of the spanning tree. Consider the cut$(\{v\}, V\backslash\{v\})$. Assume by contradiction that edge (v, w) is not part of the minimum spanning tree. Then there is another edge $(v, z) \in E$, among those edges that cross the cut, that connects v to the minimum spanning tree. However, the weight of edge (v, z) is greater than that of edge (v, w). Therefore, if we remove edge (v, z) and add edge(v, w), the total edge weights would be smaller. This is a contradiction. Therefore, we conclude that G' is a minimum spanning tree of G. $\square$

Lemma 2. *Let V and U be the partitions of H right before the compaction (described in step 14 of Algorithm 1) and let V' and U' be the partitions of the compacted graph H' right after step 14. Let k be the number of zero-difference vertices in the strut S obtained in step 5, then the number of vertices in V' is k.*

Proof. Algorithm 1 adds to the solution set any edge associated to a vertex $u \in U$ such that $d_S(u) \geq 1$. With this, all vertices of V that are interconnected by the added edges will be united or combined into a single component, in the compaction step. Each such component will be a vertex of V'. Thus, each zero-difference vertex of U represents a new vertex of V' in the compacted graph and, therefore, V' will have at least k vertices, i.e. $|V'| \geq k$.

We now prove $|V'| = k$. Suppose, by contradiction, $|V'| > k$. We have k zero-difference vertices in S, that will give rise, after compaction, to k vertices in V'. (Notice that $2k$ vertices of V are required to produce the k zero-difference vertices in S.) If $|V'| > k$, then we have at least one vertex of V' that is formed by vertices of V that are not interconnected to zero-difference vertices of S. This means there must exist x vertices of V, $1 \leq x \leq |V| - 2k$, identified as vertices of the set $V_x = \{v_1, v_2, \ldots, v_x\}$, that are connected in the strut to x vertices of U, identified as vertices of the set $U_x = \{u_1, u_2, \ldots, u_x\}$, where each u_i, $1 \leq i \leq x$, has $d_S(u_i) = 1$. Since in graph H all vertices of U have degree two, the other vertex that is connected to one of the vertices of U_x should be one of V_x.

Consider the vertex in $U_x = \{u_1, u_2, \ldots, u_x\}$ which is incident to edges with the smallest weight. Call this vertex u_a. Let $v_a \in V_x$ and $v_b \in V_x$ be the vertices connected to u_a in H. Let the edge (v_a, u_a) be a strut-edge of S. See Figure 1 (a).

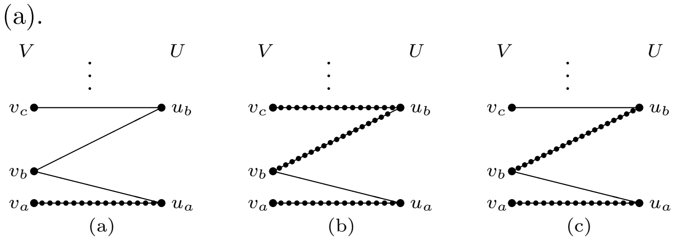

Figure 1: Part of graph H used in the proof of Lemma.

Consider the strut-edge incident to vertex v_b. It cannot be the edge (v_b, u_a), for otherwise u_a would be a zero-difference vertex. Let (v_b, u_b) the strut-edge. Now we have two cases: either u_b is incident to two strut-edges (Figure 1 (b)) or is incident only to one strut edge (Figure 1 (c)). The first case is impossible, since $v_b \in V_x$ cannot be interconnected to zero-difference vertices. The second case implies $u_b \in U_x$. Recall that u_a was chosen to possess the smallest weight in U_x. Then the strut-edge from v_b, by definition of strut, should be (v_b, u_a). This contradiction proves the lemma.

□

Theorem 2. *The number of zero-difference vertices in U' after step 14 is at least divided by 2 in each iteration.*

Proof. Let V and U be the partitions right before step 14 and let V' and U' be the partitions right after step 14. Let k be the number of zero-difference vertices in U. By Lemma 2, the number of vertices in V' is also k.

Since each zero-difference vertex in U' have degree 2, the number of zero-difference in U' is at most $|V'|/2$, i.e., $k/2$. □

The number of $\log n$ iterations needed for the algorithm convergence occurs in the worst case, for a peculiar artificially constructed graph, e.g. a hypercube with adequate weights as illustrated in Figure 2. In practice, as shown in our experiments, the number of iterations required never exceeds six for all the test cases.

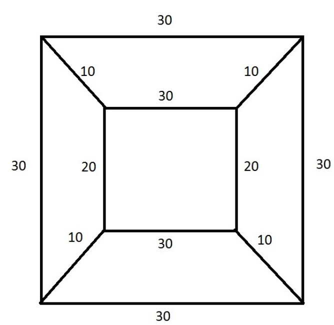

Figure 2: A graph example that needs $\log n$ iterations of the algorithm.