Chapter 65

Parallel MST Algorithms

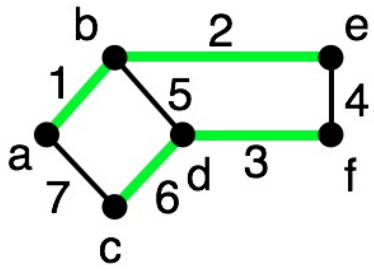
This chapter presents a parallel algorithm, due to Boruvka, for computing minimum spanning trees in parallel. As with all parallel algorithms, the algorithm is trivially a sequential algorithm also, and in fact sequential version of the algorithm are known to perform well in practice.

1 Boruvka's Algorithm

As discussed in previous sections, Kruskal and Prim's algorithm are sequential algorithms. In this section, we present an MST algorithm that runs efficiently in parallel using graph contraction. This parallel algorithm is based on an approach by Boruvka. As Kruskal's and Prim's, Boruvka's algorithm constructs the MST by inserting light edges but unlike them, it inserts many light edges at once.

Vertex Bridges. To see how we can select multiple light edges, recall that by Lemma 63.3 all light edges that cross a cut must be in the MST. Consider now a cut that is defined by a vertex v and the rest of the vertices in the graph. The edges that cross this cut are exactly the edges incident on v. Therefore, by the light edge rule, for v, the minimum weight edge between it and its neighbors is in the MST. Since this argument applies to all vertices at the same time, the minimum weight edges incident on any vertex is in the MST. We call such edges ***vertex-bridges*** or more simply as ***bridges***.

Example 65.1. The vertex bridges of the graph are highlighted.



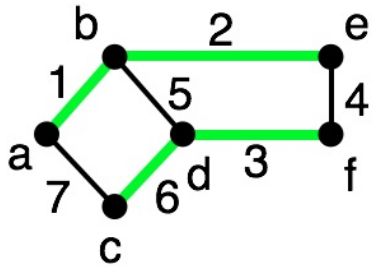
1.1 Algorithm Idea

Let's start by finding the bridge for each and every vertex in the graph. We know that the bridges are all in the MST and thus we can insert them into the MST in parallel. At this point, we might be done—we might have already selected all the MST edges and we can stop. But in most cases, we will not have a spanning tree.

To see how we can proceed, note that the bridges define a partition of the graph, because each and every vertex is the end-point of some bridge and thus is in a block. Consider now the edges that remain internal to a block but are not bridges. Such an edge cannot be in the MST, because inserting it into the MST would create a cycle. The edges that cross the blocks, however, could be in the MST. We therefore want to eliminate the internal edges from consideration, but keep the cross edges. We can do so by performing a graph contraction based on the partition defined by the bridges.

Boruvka's algorithm iterates this approach until the graph is reduced to a single vertex.

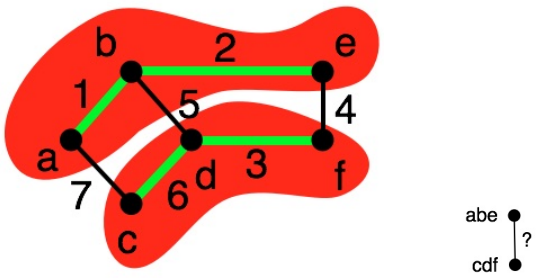
Example 65.2 (One Round of Boruvka's Algorithm)**.** Consider the graph below and the highlighted vertex-bridges.



- The vertices `a` and `b` both pick edge $\{\texttt{a}, \texttt{b}\}$;
- vertex `c` picks $\{\texttt{c}, \texttt{d}\}$, `d`;
- the vertices `d` and `f` both pick $\{\texttt{d}, \texttt{f}\}$, and
- `e` picks $\{\texttt{e}, \texttt{b}\}$.

The edge (e, f), which is in the MST, is not selected (neither e nor f pick it).

To proceed, we can take the partitions defined by bridges and contract them by using graph contraction. The figure below illustrates such a contraction. After the contraction completes, we obtain multiple edges between the the resulting partitions.



Redundant Edges. When performing graph contraction, we have to be careful about redundant edges. In our discussion of graph contraction of unweighted graphs, we mentioned that we may treat redundant edges differently based on the application. In unweighted graphs, the task is usually simple because we can keep any one of the redundant edges, and it usually does not matter which one. When the edges have weights, however, we have to decide to keep all the edges or select some of the edges to keep. For the purposes of MST, in particular, we can keep all the edges or keep just the edge with the minimum weight, because the others, cannot be in the MST. In the example above, we would keep the edge with weight 4.

Summary. What we just covered is exactly Boruvka's idea. He did not discuss implementing the contraction in parallel. At the time, there were not any computers let alone parallel ones. In summary, Boruvka's algorithm can be described as follows.

Algorithm 65.1 (Boruvka)**.** While there are edges remaining:

1. select the minimum weight edge out of each vertex and contract each part defined by these edges into a vertex;
2. remove self edges, and when there are redundant edges keep the minimum weight edge; and
3. add all selected edges to the MST.

1.2 Boruvka's Algorithm with Tree Contraction

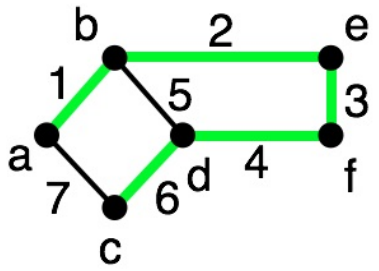
We can implement Boruvka's algorithm as described in Section 1.1 by using tree contraction. In this section we describe how to do this and analyze its cost.

Tree Contraction. To contract the partition defined by the vertex-bridges, we cannot use edge or star contraction, because the blocks may not correspond to an edge or a star. The blocks in general are trees, because each vertex selects exactly one bridge. To contract the block that is induced by the bridges in the block, it suffices to contract along the tree formed by the tree edges. We can do this by removing all non-bridge edges from a block and contracting the block by applying star contraction to it. Because each round of star contraction applied on a tree yields another tree, the number of edges goes down with the number of vertices.

Therefore the total work to contract all the partitions is bounded by $O(n)$ if using array sequences. The span remains $O(\log^2 n)$.

After contracting each tree, we have to update the edges, by re-routing the cut edges between blocks to their new endpoints. This can lead to multiple edges between two vertices, effectively giving us a multi-graph. This can be an effective solution, and allows the updating the edges in $O(m)$ work.

Example 65.3. An example where finding the bridges for all vertices yields a tree that is not a star graph. Note that the selected bridges form a minimum spanning tree.



Cost of Boruvka by Using Tree Contraction. Let's first bound the number of rounds of contraction. Observe that contracting a bridge removes exactly one vertex (contraction of an edge can be viewed as folding one endpoint into the other). Therefore, if k bridges are selected then k vertices are removed.

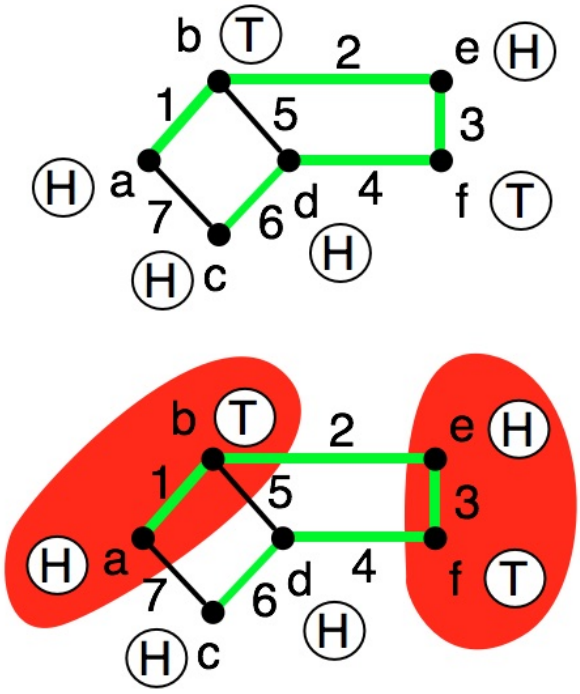
Because each vertex picks a vertex bridge independently in parallel, it is possible that $k = n$. In this case, we would be able to fold all the vertices in one round. In the general case, however, one edge can be chosen by two vertices as vertex bridges. Therefore at least $n/2$ vertex bridges are picked and thus $n/2$ vertices will be removed. Consequently, Boruvka's algorithm will take at most $\lg n$ rounds of selecting bridges and contracting by using 1.2.

Because updating all cut edges requires $O(m)$ work and because there are $\log n$ rounds, Boruvka's algorithm takes $O(m \log n)$ work and $O(\log^3 n)$ span.

1.3 Boruvka's Algorithm with Star Contraction

Star Partition on Bridges. We now describe how to improve the span of Boruvka by a logarithmic factor by interleaving steps of star contraction with steps of finding the vertex-bridges. The idea is to apply star contraction to the subgraph of the graph induced by the bridges. The key observation is that because each non-isolated vertex has a bridge, the subraph is large enough to give us a constant contraction ratio. Specifically, we will prove that the technique reduces the number of vertices by a constant factor (in expectation), leading to logarithmic number of total rounds. Consequently, we will reduce the overall span for finding the MST from $O(\log^3 n)$ to $O(\log^2 n)$ and maintain the same work.

Example 65.4. An example of Boruvka with star contraction.



Algorithm 65.2 (Star Partition along Bridges)**.** Given a function `vertexBridges` (G) that finds the vertex-bridges out of each vertex in G, the function `bridgeStarPartition` performs star contraction along the vertex bridges. To apply star contraction, the algorithm modifies standard `starContract` function so that after flipping coins, we only contract edges which are vertex-bridges. In the code w denotes the weight of the edge (u, v).

```
1  bridgeStarPartition (G = (V, E), i) =
2    let
3      Eb = vertexBridges (G)
4      P = {u ↦ (v, w) ∈ Eb | (flips (u) = T) ∧ (flips (v) = H)}
5      V' = V \ domain(P)
6    in
7      (V', P)
8    end
```

Contraction Ratio. Before we go into details about how we might keep track of the MST and other information, let us try to understand what effects this change has on the number of vertices contracted away. If we have n non-isolated vertices, the following lemma shows that the algorithm $bridgeStarPartition$ still selects $n/4$ satellities in expectation on each step, and this contracting the graph along these edges will lead to a $1/4$ factor reduction in the number of vertices. The lemma thus implies that this MST algorithm will take only $O(\log n)$ rounds, just like the original star contraction algorithm.

Lemma 65.1 (Number of Bridged Satellites)**.** For a graph G with n non-isolated vertices, let X_n be the random variable indicating the number of vertices removed by `bridgeStarPartition` (G, r). Then, $\mathbf{E}[X_n] \geq n/4$.

Proof. The proof is pretty much identical to our proof for $starContract$ except here we're not working with the whole edge set, only a restricted one Eb. Let $v \in V(G)$ be a non-isolated vertex. Like before, let H_v be the event that v comes up heads, T_v that it comes up tails, and R_v that $v \in domain\ (P)$ (i.e, it is removed). Since v is a non-isolated vertex, v has neighbors—and one of them has the minimum weight, so there exists a vertex u such that $(v, u) \in Eb$. Then, we have that $T_v \wedge H_u$ implies R_v since if v is a tail and u is a head, then v must join u. Therefore, $\mathbf{P}[R_v] \geq \mathbf{P}[T_v]\,\mathbf{P}[H_u] = 1/4$. By the linearity of expectation, we have that the number of removed vertices is

$$\mathbf{E}\left[\sum_{v:v \text{ non-isolated}} \mathbb{I}\{R_v\}\right] = \sum_{v:v \text{ non-isolated}} \mathbf{E}[\mathbb{I}\{R_v\}] \geq n/4$$

since we have n vertices that are non-isolated. □

Exercise 65.1. Compare the proof the bridged-satellites lemma to the original star-partition lemma. What remains the same, what has changed?

Tracking Edges. There is a little bit of trickiness in constructing the result MST. As the graph contracts, the endpoints of each edge changes. Therefore, if we want to return the edges of the minimum spanning tree, then we might need to keep track of changes in how edges are re-routed between vertices. To avoid this, we associate a unique label with every edge and return the tree as a set of labels (i.e. the labels of the edges in the spanning tree). We also associate the weight directly with the edge. The type of each edge is therefore (`vertex` × `vertex` × `weight` × `label`), where the two vertex endpoints can change as the graph contracts but the weight and label stays fixed. This leads to the slightly-updated version of `bridgeStarPartition` that appears in the algorithm given below.

Algorithm 65.3 (Boruvka's based on Star Contraction)**.** The function $vertexBridge(G)$ finds the minimum edge out of each vertex v and maps v to the pair consisting of the neighbor along the edge and the edge label. To this end, the function makes a singleton table for each edge and then merge all the tables with a function to resolve collisions, which favors lighter edge.

The function $bridgeStarPartition$ performs star contraction on the subgraph induced by the bridges. It starts by selecting the bridges and then in Line 12 it picks from bridges the edges that go from a tail to a head. It then generates a mapping from tails to heads along

minimum edges, creating stars. Line 13 removes all vertices that are in this mapping to star centers.

The function *bridgeStarPartition* is ready to be used in the MST code, similar to the `graphContract` code studied last time, except we return the set of labels for the MST edges instead of the remaining vertices. The code is given below. The MST algorithm is called by running $\texttt{MST}(G, \emptyset, 0)$. As an aside, we know that T is a spanning forest on the contracted nodes.

```
1   vertexBridges E =
2     let
3       ET = {(u,v,w,l) ↦ {u ↦ (v,w,l)} : (u,v,w,l) ∈ E}
4       select ((v1,w1,l1),(v2,w2,l2)) =
5         if (w1 ≤ w2) then (v1,w1,l1) else (v2,w2,l2)
6     in
7       reduce (union select) {} ET
8     end
9   bridgeStarPartition (G = (V,E)) =
10    let
11      Eb = vertexBridges G
12      P = {(u ↦ (v,w,ℓ)) ∈ Eb | (flip(u) = T) ∧ (flip(v) = H)}
13      V' = V \ domain(P)
14    in
15      (V',P)
16    end
17  MST (V,E) T =
18    if (|E| = 0) then T
19    else
20      let
21        (V',PT) = bridgeStarPartition (V,E)
22        P = {u ↦ v : u ↦ (v,w,ℓ) ∈ PT} ∪ {v ↦ v : v ∈ V'}
23        T' = {ℓ : u ↦ (v,w,ℓ) ∈ PT}
24        E' = {(P[u],P[v],w,l) : (u,v,w,l) ∈ E | P[u] ≠ P[v]}
25      in
26        MST (V',E') (T ∪ T')
27      end
```

Remark. Even though Boruvka's algorithm is not the only parallel algorithm, it was the earliest, invented in 1926, as a method for constructing an efficient electricity network in Moravia in the Czech Republic. It was re-invented many times over.