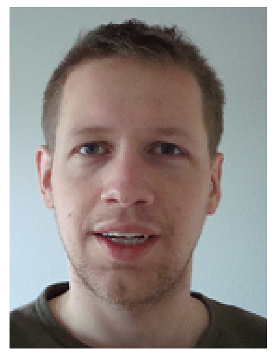
MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS



Claus Andersen
Master Student
University of Aarhus
Supervisor: Gerth S. Brodal

An optimal minimum spanning tree algorithm

What is a minimum spanning tree (MST)

Consider a connected graph with n vertices and m edges. A graph could be a model of

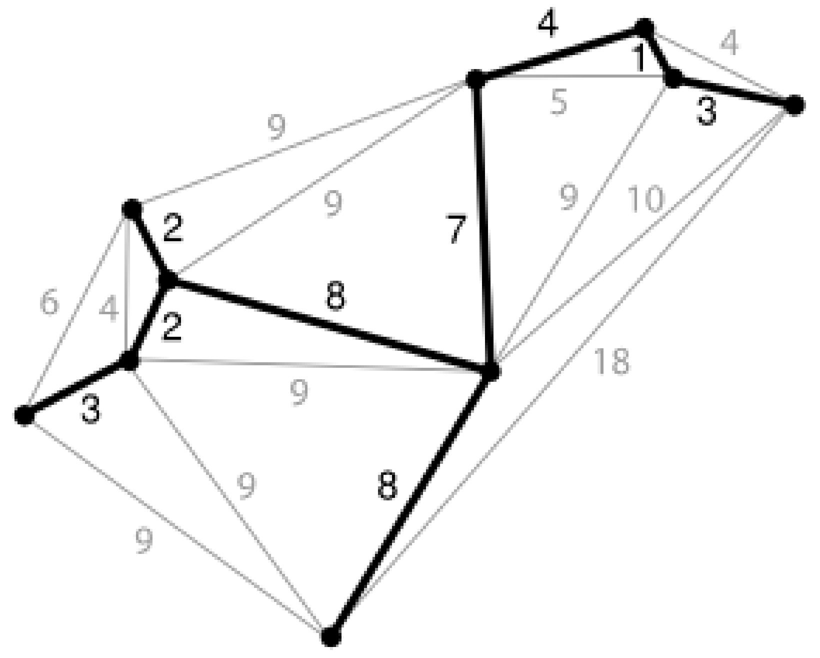
- Cities (vertices) and distances between cities (edge wieghts)
- Computers/routers on a (inter)network (vertices) and the average latencies (edges weights)
- Etc.

In general the edge weights are some kind of cost function.

Let us assume the edge weights are distinct. A spanning tree of a graph, is a subset of edges, which form a tree and connects all vertices. The tree has n-1 edges. The MST of a graph is the spanning tree which have the minimum total cost. Suppose we wish to minimize the total cable length when connecting citites with power or data lines. The MST of the graph is the optimal way to lay out these cables without redundancy (without cycles).

Introduction

The classical text book algorithms solving the MST problem have worst case running time *O(m log(n))*.
This algorithm [Seth Pettie, Vijaya Ramachandran] solves the problem in time proportional to *T*(m,n)*, which is the optimal number of edge comparisons needed to determine the MST of any graph with n vertices and m edges. I.e. The running time is *O(T*(m,n))*.
It is obvious that *T*(m,n) ≥ m*, since we need to consider all edges. The idea is to use optimal decision trees for sufficiently small subgraphs of the input graph. Because the decision trees are sufficiently small, we can afford to build them. The subgraphs are found in linear time using the soft heap [Chazelle].



```
function OptimalMST($G$)
    if $E(G) = \emptyset$ then
        return $\emptyset$
    $r \leftarrow \lceil \log^{(3)} |V(G)| \rceil$
    $(M, C) \leftarrow$ Partition( $G$, $r$, $\epsilon$ )
    $F \leftarrow$ DecisionTree( $C$ )
    Let $k \leftarrow |C|$ and $F = \{F_1, \ldots, F_k\}$
    $G_a \leftarrow G \setminus (F_1 \cup \ldots \cup F_k) - M$
    $F_0 \leftarrow$ DenseCase( $G_a$ )
    $G_b \leftarrow F_0 \cup F_1 \cup \ldots \cup F_k \cup M$
    $(F', G_c) \leftarrow$ Boruvka2( $G_b$ )
    $F \leftarrow$ OptimalMST( $G_c$ )
    return $(F \cup F')$
```

Status

Because *log$^{(3)}$(n) ≤ 3* for all *n ≤ 2^{256}*, the use of decision trees is impractical, since the MST of a graph with ≤ 3 vertices can easily be found with *O(m)* comparisons. The number 2^{256} should be compared with the number of atoms in the universe wihch is around 2^{265}.

All helper functions except "DecisionTree" are implemented, tested and partly documented (i.e. Partition, DenseCase, Boruvka2). It remains to glue these steps together, run tests on the full algorithm, and of course to document this :)

The algorithm

The algorithm works recursively by first removing some non-candidate edges, and then find some real MST edges and contracting the graph along these. This procedure runs until there is only one vertex left. The head points of the procedure is:

- Grow DJP-contractable partitions of the graph with vertex bound *log$^{(3)}$(n) = log(log(log(n)))*
  With a soft heap as priority queue with a constant error rate ε, this takes *O(m)* time
  The soft heap corrupts some edge weights by raising their priority. These edges are used later.
- Find the MST of each partition with a optimal number of edge comparisons, using decision trees.
- Contract all partitions in the graph and remove the corrupted edges. Find the MST of this dense graph in *O(m)* time using the "DenseCase" algorithm by [Fredman and Tarjan].
- Build the union of the previously found MST's and the corrupted edges. Run two Borukvka phases on this graph.
- The two Boruvka phases identifies some real MST edges and contracts the graph into a graph with less than m/2 edges and n/4 vertices. This takes *O(m)* time.
- Call recursively on the Boruvka-contracted graph.
- Return real MST edges: The union of the recursive call and the MST edges found in the Boruvka phases.

Total running time: Each point takes *O(m)* time, except the decision tree point which makes an optimal number of comparions. We recurse on a graph where the number edges and vertices are reduced geometrically, so this will not hurt the running time. Therefore the total running time is proportional to the optimal number of edge comparisons.