FASTER MINIMUM SPANNING TREES IN BOUNDED GENUS GRAPHS

Graph on Surfaces - Summary *Giulio Malavolta*

Introduction

Given a connected and undirected graph, the spanning tree is as a sub-graph that is a tree and connects all the vertices together; in weighted graphs (i.e. where each edge is labeled with a weight) we can define the minimum spanning tree (MST) as the spanning tree for which the sum of all the weight of its edges it's less or equal to the value of every other spanning tree. The first efficient algorithm for computing the minimum spanning tree of a graph was presented in 1926 and from that time on lots of faster algorithm have been developed to cope the complexity of this problem. For the algorithms which will be analyzed we are going to start from the following assumption:
- The weights of all edges are distinct
- The weights are compared with a constant time comparison oracle (in order to avoid dealing with real numbers)

Boruvka's Algorithm

The algorithm of Boruvka represents the oldest idea to calculate the MST of a graph and it starts from the idea to grow a forest from the lightest edges and iterating until we obtain a connected component. Starting from an arbitrary vertex the algorithm iterate all the vertices each time selecting the edge with the lowest weight connecting the current vertex to another node which is not part of the same connected component. The iteration will end when a single connected component is obtained outputting the MST for the graph. Considering *n* as the number of vertices and m as the number of edges the Boruvka's algorithm runs in time $O(m * \log(n))$.

Jarnik's Algorithm

The Jarnik's algorithm's basic idea is to build up a single tree instead of a forest following a greedy approach in the selection of the edges. In particular the iteration starts from an arbitrary vertex and chooses the lightest edge connecting an element of the growing tree to a vertex that is not part of the tree, until all the nodes are reached and the tree spans the whole graph. In the standard implementation this approach doesn't consist in an improvement with respect to Boruvka's design since the running time is still in the order of $O(m * \log(n))$.

Contractive MST Algorithms

Instead of growing tree the contractive approach is based on the concept of contracting edges and keeping track on them. Each iteration of the algorithm takes in consideration all the vertices of the graph and contracts the incident edge with the lowest weight, storing the label of the selected edges. The subsequent step consists in the removal of all the parallel edges in the graph maintaining only the one with the lightest value, this operation is called *flattening*. The cycle ends when the graph is composed by a single node and the resulting vector of stored edges will represent the MST for the graph. The efficiency of this method relies on the fact that we can implement the flattening operation on RAM, being able to perform the contraction in linear time with a 2-pass bucket sort algorithm. Indeed we can state that:

Lemma. *The i^{th} step of the contractive Boruvka's algorithm can be carried out in time $O(m_i)$.*

Without any assumption on the graph, the running time is still fixed to $O(m * \log(n))$ but if we are dealing with simple graph we can state that the number of edges is upper-bounded by $m \leq n^2$, hence each cycle is performed in $O(n^2)$ time. Taking in account that each iteration drops at least $\frac{n}{2}$ vertices we obtain a time for all the cycles of $O(n^2)$, that allows us to assess:

Theorem. *The contractive Boruvka's algorithm finds the MST in the input graph in time $O(\min(n^2, m * \log(n)))$.*

Minor Closed Graphs Classes

As we saw before, making assumptions on particular features of the graph that we are dealing with can allow us to build algorithms that significantly drop the running time, therefore we are going to put some restriction in the input graph in order to assess a greater efficiency on the subsequent designs. In the first instance we fix some definitions:

Definition. *A graph H is a minor of a graph G (written as H≼G) if and only if it can be obtained from a subgraph of G by a sequence of simple graph contractions.*

Definition. *A class C of graphs is minor-closed, when for every G∈C and every minor H of G, graph H lies in C as well. A class C is called non-trivial if at least one graph lies in C and at least one lies outside C.*

Definition. *Let G be a graph and C be a class of graphs. We define the edge density ϱ(G) of G as the average number of edges per vertex, i.e., m(G)/n(G). The edge density ϱ(C) of the class is then defined as the infimum of ϱ(G) over all GεC.*

By definition it holds that:

Theorem. *Every non-trivial minor closed graph class has a finite edge density.*

We can take advantage of this fact performing a reduction on the running time of the contractive algorithm previously presented. We assessed that each step of the iteration is performed in time $O(m)$, but if we assume that our input graph belongs to a non-trivial minor closed graph class then we can express the number of the edge as upper-bounded by $m \leq \varrho(C) * n$. Therefore we have that:

Theorem. *For any fixed nontrivial minor-closed class C of graphs, the contractive Boruvka's algorithm finds the MST of any graph of this class in time $O(n)$.*

Iterated Algorithms

Without any a priori knowledge about the structure of the graph we cannot make any assumption on the edge's density, thus we implement different designs to minimize the running time of the MST search. A feasible approach is represented by Jarnik's iterated algorithm that iterates through all the nodes using a greedy approach to select edges to add to the growing three; the difference with respect to the previous ideas is that this time we deny the possibility of an edge to be chosen by two different vertices keeping track of all the *active* edges. The active edges are defined as the edges connecting any node *v* in the growing tree to any node *u* not part of the growing three: each iteration of the algorithm parses the active edges incident to the considered node of the tree are, it sorts them by weight and the lightest edge is flagged as inactive and added to the tree. The cycle ends when all the vertices are connected drawing the

MST of the graph. The advantage of this technique lies in the data structures implemented for the tracking of active edges: the *Fibonacci Heap* is indeed used to store the active edges as pairs of vertices guaranteeing a speed up of the computation since it holds that:

Theorem. *The Fibonacci Heap performs the following operations with the indicated amortized time complexity:*
Insert (insertion of a new element) in $O(1)$.
Decrease (decrease the value of an existing element) in $O(1)$.
Merge (merging two heaps into one) in $O(1)$.
DeleteMin (deletion of the minimal element) in $O(\log(n))$.
Delete (deletion of an arbitrary element) in $O(\log(n))$.

Overall the all algorithm is comprising of *m* iterations and each cycle executes operations on the Fibonacci Heap on at most n elements, therefore:

Theorem. *Using Fibonacci Heap we can find the MST of the input graph in time* $O(m + n * log(n))$.

This is clearly valid for a general graph but if we have knowledge about the structure of the graph we might improve the time complexity of the algorithm, indeed:

Corollary. *For graphs with edge density Ω(log n), this algorithm runs in linear time.*

Combining MST Algorithms

In order to reach higher peaks of performances in the calculation of the MSTs one of the latest approaches consists in combining together different algorithms, in particular it turned out to be efficient to perform $\log(log(n))$ steps of the contractive Boruvka's algorithm and proceed then to the completion of the MST with the Jarnik's iterated approach. The whole idea is based on the following lemma:

Lemma. *Let G be a weighted graph, e an arbitrary edge of MST(G), G/e the multigraph produced by contracting e in G, and π the bijection between edges of G e and their counterparts in G/e. Then* $MST(G) = \pi\ -1\ [MST(G/e)]$.

The first algorithms runs in time $O(m * \log(\log(n)))$ contracting the graph G into the graph G' which contains $m' \leq m$ edges and $n' \leq n/\log(n)$ vertices. The Jarnik's step processes only G' and we can upper-bound its time complexity on $O(m' + \mathrm{n}' \log(\mathrm{n}')) = O(m)$. The combination of the two outputs can be performed in linear time, therefore it follows that:

Theorem. *The Mixed Boruvka-Jarnik's algorithm finds the MST of the input graph in time* $O(m * log(log(n)))$.

Conclusions

Several approaches have been presented so far and the results for the minor closed graphs are surprisingly efficient since the algorithms can output the MST in deterministic linear time, however it's still an open research field to understand whether it's possible to design an algorithm able to achieve the same time complexity on general graphs, for which the best outcome is still $O(m * \log(\log(n)))$.