Faster minimum spanning trees in bounded genus graphs

Presented by Blaga Davidova

13.12.2012

Content

History

- the history of MST problem begins in 1926 with the work of Boruvka, who gave an efficient algorithm for the problem.
- Boruvka’s work was further extended by Jarnik, in mostly geometric setting.
- Unfortunately, after 1950 Jarnik's algorithm had to be rediscovered several times.
- In the next 50 years, several significantly faster algorithms were discovered
- The current speed record is held by Chazelle and Pettie which achieve time complexity $-O(m*\alpha(m, n))$, where m and n are the number of vertices and edges of the graph and $\alpha(m, n)$ is an inverse of the Ackermann's function

Content

- 1. History
- 2. What is Comparison Oracle?
- 3. Classical MST Algorithms
- 4. Contractive MST Algorithms
- 5. Minor Closed Graph Classes
- 6. Iterated Algorithms
- 7. Combining MST Algorithms

Comparison Oracles

When we analyze and describe MST algorithms we will make the following assumptions:

- The weights of all edges are distinct
- Instead of numeric weights we are given a comparison oracle
- The oracle is a function that answers questions of type “Is $w(a)<w(b)$?” in constant time

This will conveniently shield us from problems with representation of real numbers in algorithms

Content

Boruvka's Algorithm

- The oldest MST algorithm
- Idea: grow a forest in a sequence of iterations until it becomes connected
- We start with a forest of isolated vertices
- In each iteration we let each tree of the forest select the lightest edge of those having exactly one endpoint in the tree (we will call such edges the neighboring edges of the tree)
- We add all such edges to the forest and proceed with the next iteration
- Running time: O(mlogn)

Boruvka's Algorithm

Input: A graph G with an edge comparison oracle.

1. T←a forest consisting of vertices of G and no edges.
2. While T is not connected:
3. For each component T_i of T , choose the lightest edge e_i from the cut separating T_i from the rest of T .
4. Add all e_i 's to T .

Output: Minimum spanning tree T .

Jarnik's Algorithm

- discovered independently by Jarnik, Prim and Dijkstra
- Similar to the Boruvka's algorithm
- instead of the whole forest it concentrates on a single tree
- Idea: starts with a single vertex and it repeatedly extends the tree by the lightest neighboring edge until the tree spans the whole graph.
- Running time: using binary heaps O(mlogn)

  using Fibonacci heaps O(m + nlogn)

Jarnik's Algorithm

Input: A graph G with an edge comparison oracle.

1. T$\leftarrow$a single-vertex tree containing an arbitrary vertex of G.
2. While there are vertices outside T :
3. Pick the lightest edge *uv* such that $u \in V(T)$ and $v \notin V(T)$
4. T$\leftarrow$T + *uv*

Output: Minimum spanning tree T .

Content

- 1. History
- 2. What is Comparison Oracle?
- 3. Classical MST Algorithms
- 4. Contractive MST Algorithms
- 5. Minor Closed Graph Classes
- 6. Iterated Algorithms
- 7. Combining MST Algorithms

Contractive algorithms

- classical algorithms - based on growing suitable trees
- Idea – reformulated them in terms of edge contraction
- Instead of keeping a forest of trees, we can keep each tree contracted to a single vertex
- Potentially speeding up the calculation at the expense of having to perform the contractions.

Contractive version of the Boruvka's algorithm

Input: A graph G with an edge comparison oracle.

1. $T \leftarrow \emptyset$
2. $l(e) \leftarrow e$ for all edges e. (Initialize the labels.)
3. While $n(G) > 1$:

   For each vertex v_k of G, let e_k be the lightest edge incident to v_k

   $T \leftarrow T \cup \{ l(e_1), \ldots, l(e_n)\}$ (Remember labels of all selected edges.)

6. Contract all edges e_k, inheriting labels and weights
7. Flatten G (remove parallel edges and loops).

Output: Minimum spanning tree T .

Running Time?

Lemma. The i-th Boruvka step can be carried out in time $O(m_i)$.

Proof.

- contractions can be performed in linear time
- Flattening on RAM:
  - sort edges lexicographically by 2-pass bucket sort;
  - coping with sparse arrays

Running Time?

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

Proof.

Why $O(m\log n)$?

- We have $O(\log n)$ number of iteration
- For each iteration we waste $O(m)$ time

Why $O(n^2)$?

- In each iteration the number of vertices drops at least by a factor of 2
- Therefore $n_i \leqq n/2^i$.
- we have $m_i \leqq (n_i)^2$ as the graphs G_i are simple
- total time spent in all iterations is $O(\sum_i n_i^2)=O(\sum_i n^2/4^i)=O(n^2)$

Content

- 1. History
- 2. What is Comparison Oracle?
- 3. Classical MST Algorithms
- 4. Contractive MST Algorithms
- 5. Minor Closed Graph Classes
- 6. Iterated Algorithms
- 7. Combining MST Algorithms

Minor-Closed Graph Classes

Definition. A graph H is a *minor* of a graph G (written as $H \preccurlyeq G$) iff 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 $G \in C$ 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.

Example. Non-trivial minor-closed classes include:

- planar graphs
- graphs embeddable in any fixed surface
- graphs of bounded tree-width or path-width.

Minor-Closed Graph Classes

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

Theorem. Every non-trivial minor-closed class of graphs has finite edge density.

Minor-Closed Graph Classes

Theorem. (MST on minor-closed classes) For any fixed non-trivial minor-closed class C of graphs, the Contractive Boruvka's algorithm finds the MST of any graph of this class in time O(n).

Proof.

- The i-th phase runs in time $O(m_i)$
- we have $n_i \leq n/2^i$,
- each G_i is produced from G_{i-1} by a sequence of edge contractions, thus G_i's are minors of the input graph
- Each G_i belong to C and by the Density theorem $m_i \leq \varrho(C) n_i$
- The time complexity is $\sum_i O(m_i) = \sum_i O(n_i) = O(\sum_i n/2^i) = O(n)$.

Content

- 1. History
- 2. What is Comparison Oracle?
- 3. Classical MST Algorithms
- 4. Contractive MST Algorithms
- 5. Minor Closed Graph Classes
- 6. Iterated Algorithms
- 7. Combining MST Algorithms

Iterated Algorithms

Idea - we will remember the vertices adjacent to T and for each such vertex *v* we will maintain the lightest edge *uv* such that u lies in T;

We will call these edges active edges and keep them in a Fibonacci heap, ordered by weight

When we want to extend T by the lightest edge, it is sufficient to find the lightest active edge *uv* and add this edge to T together with the new vertex *v*

Update active edges

Iterated Algorithms

Input: A graph G with an edge comparison oracle.

1. $v_0 \leftarrow$ an arbitrary vertex of G.

2. $T \leftarrow$ a tree containing just the vertex v_0.

3. $H \leftarrow$ a Fibonacci heap of active edges stored as pairs (u, v) where $u \in T$, $v \notin T$, ordered by the weights w(uv), and initially empty.

4. $A \leftarrow$ a mapping of vertices outside T to their active edges in the heap; initially all elements undefined.

5. Insert all edges incident with v_0 to H and update A accordingly.

Iterated Algorithms

6. While H is not empty:
7. (u, v)⇽DeleteMin(H).
8. T⇽T + uv.
9. For all edges vw such that w∉T :
10. If there exists an active edge A(w):
11. If vw is lighter than A(w), Decrease A(w) to (v, w) in H.
12. If there is no such edge, then Insert (v, w) to H and set A(w).

Output: Minimum spanning tree T .

Running Time

Theorem. *(Fibonacci heaps)*

The Fibonacci heap performs the following operations with the indicated amortized time complexities:

Insert (insertion of a new element) in O(1)

Decrease (decreasing the value of an existing element) in O(1)

Merge (merging of 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)

Running Time

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

Proof.

- The time complexity is O(m) plus the cost of the heap operations.
- The algorithm performs at most one Insert or Decrease per edge and exactly one DeleteMin per vertex.
- There are at most n elements in the heap at any given time
- Using the previous theorem the operations take O(m+nlogn) time in total.

Running Time

Corollary. For graphs with edge density $\Omega(\log n)$, this algorithm runs in linear time.

Content

Combining MST algorithms

- the improved Jarnik’s algorithm runs in linear time for sufficiently dense graphs.
- In some cases, it is useful to combine improved Jarnik's algorithm with another MST algorithm, which identifies a part of the MST edges and contracts them to increase the density of the graph.
- For example, we can perform several Borvka steps and then find the rest of the MST by the Active Edge Jarnik's algorithm.

Contraction of MST Edges

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) = π^{-1} [mst(G/e)] + e.

Mixed Boruvka-Jarnık's Algorithm

Input: A graph G with an edge comparison oracle.

1. Run loglogn Boruvka steps, getting a MST T1 .

2. Run the Active Edge Jarnık's algorithm on the resulting graph getting a MST T2 .

3. Combine T1 and T2 to T as in the Contraction lemma

Output: Minimum spanning tree T

Running Time

Theorem. The Mixed Boruvka-Jarnik's algorithm finds the MST of the input graph in time O(mloglogn).

Proof.

- The first step takes O(mloglogn) time and it gradually contracts G to a graph G′ of size m′≤m and n′≤n/ logn
- The second step then runs in time O(m′+n′logn′) =O(m)
- Both trees can be combined in linear time

Conclusion

- We presented algorithm for the MST problem which run in deterministic linear time for any class of graphs closed on graph minors
- We presented algorithm for the MST problem which run in O(mloglogn) time for any graph
- But the question for the general version of the problem is still open – can we find MST algorithm which runs in linear time for any graph?

References

- *Graph Algorithms - Martin Mares*
- *Two Linear Time Algorithms for MST on Minor Closed Graph Classes – Martin Mares*