A Linear Time Algorithm for the Minimum Spanning Tree Problem on a Planar Graph

Tomomi MATSUI

(January 1994)

Department of Mathematical Engineering and Information Physics
Faculty of Engineering, University of Tokyo
Bunkyo-ku, Tokyo 113, Japan

Abstract: In this paper, we propose a linear time algorithm for finding a minimum spanning tree on a planar graph.

Keywords: Combinatorial problems; graphs; spanning trees; planar graphs

1 Introduction

Finding a spanning tree of minimum weight is one of the best known graph problems. Several efficient algorithms exist for solving this problem [1, 3, 4, 5, 6, 9, 11, 13]. This paper presents a liner time algorithm for the minimum spanning tree problem on a planar graph. In [1], Cheriton and Tarjan have proposed a linear time algorithm for this problem. The time complexity of our algorithm is the same as that of Cheriton and Tarjan's algorithm. Different from Cheriton and Tarjan's algorithm, our algorithm does not require the *clean-up* activity. So, the implementation of our algorithm is very easy.

Our algorithm maintains a pair of a planar graph and its dual graph and breeds both a minimum spanning tree of original graph and a maximum spanning tree of a dual graph. In each iteration of our algorithm, either the number of edges decreases or a vertex of the planar graph or its dual graph is deleted. By employing a simple bucket structure, we can save the time complexity of every iteration to $O(1)$.

2 Main Framework

In this section, we give some definitions and describe a main framework of our algorithm.

Let us consider an undirected graph $G = (V, E)$ with the vertex set V and the edge set E. For any vertex v of G, $\delta_G(v)$ denotes the set of edges in G incident to v. For any edge subset $E' \subseteq E$, the graph (V, E') is called a *spanning subgraph* of G. In this paper, we present a spanning subgraph as its edge set. A spanning subgraph of G is called a *spanning forest* of G when the graph does not contain any cycle. A spanning forest of G is called a *spanning tree* when it is connected. A graph G contains a spanning tree if and only if G is connected. Given a graph G and its edge e, $G \backslash e$ denotes the graph obtained by deleting the edge e and G/e denotes the graph obtained by contracting e. For each edge $e \in E$, $w(e)$ denotes the weight of the edge e. The weight of a spanning subgraph F, denoted be $w(F)$, is the sum of the weights of edges in F. A maximal spanning forest F is called a *minimum (maximum) weight spanning forest*, when F minimizes (maximizes) the weight $w(F)$.

A graph is called *planar* if it can be drawn in the plane so that its edges intersect only at their ends. Given a graph $G = (V, E)$, a graph $G^* = (V^*, E)$ with common edge set is called a *dual graph* of G if it satisfies the condition that an edge subset $C \subseteq E$ is a cycle of G if and only if C is a cut-set of G^*. If G^* is a dual graph of G, then G is a dual graph of G^* (see [7, 10] for example). It is known that a graph is planar if and only if it has a dual graph [12]. If we have a planar embedding of a graph G, it is easy to construct a dual graph of G geometrically (see [7, 10]). In [8], Hopcroft and Tarjan proposed a linear time algorithm for embedding a planar graph in the plane.

In this paper, we propose an algorithm for finding a minimum (maximum) weight spanning forest of a planar graph G. Clearly, if the given graph is connected, this problem is equivalent to the ordinary minimum (maximum) spanning tree problem. It is well-known that an edge subset $T \subseteq E$ is a maximal spanning forest of G if and only if $E \setminus T$ is a maximal spanning forest of G^*. It implies that an edge subset $T \subseteq E$ is a minimum weight spanning forest of G if and only if $E \setminus T$ is a maximum weight spanning forest of G^*. Thus, the problem for finding a minimum weight spanning forest of G is essentially the same with the problem for finding a maximum weight spanning forest of G^*.

Now we describe a main framework of our algorithm. Each iteration of the algorithm is similar to that of Prim's Algorithm [11] for minimum (maximum) spanning tree problem.

Algorithm A

Input: A planar graph $G = (V, E)$, a dual graph $G^* = (V^*, E)$ of G and edge weights w.

Output: A minimum weight spanning forest T of G and a maximum weight spanning forest T^* of G^*.

Step 0: Set $G_1 := G$, $G_1^* := G^*$, $T := \emptyset$ and $T^* := \emptyset$.

Step 1: If both G_1 and G_1^* are empty graph, then output (T, T^*) and stop.

Step 2: Choose a vertex v in G_1 or G_1^*.

If v is an isolated vertex, then delete the vertex and go to Step 1.

Else if v is a vertex of G_1 and $\delta_{G_1}(v)$ contains a self-loop, then go to Step 3.

Else if v is a vertex of G_1 and $\delta_{G_1}(v)$ contains no self-loop, then go to Step 4.

Else if v is a vertex of G_1^* and $\delta_{G_1^*}(v)$ contains a self-loop, then go to Step 5.

Else if v is a vertex of G_1^* and $\delta_{G_1^*}(v)$ contains no self-loop, then go to Step 6.

Step 3: (v is a vertex of G_1 and $\delta_{G_1}(v)$ contains a self-loop)

Let f be a self-loop of G_1 incident to v.

Set $G_1 := G_1 \backslash f$, $G_1^* := G_1^* \backslash f$ and $T^* := T^* \cup \{f\}$. Go to Step 1.

Step 4: (v is a vertex of G_1 and $\delta_{G_1}(v)$ does not contain any self-loops)

Find an edge e in $\delta_{G_1}(v)$ which attains the value $\min\{w(e') \mid e' \in \delta_{G_1}(v)\}$.

Set $G_1 := G_1/e$, $G_1^* := G_1^* \backslash e$ and $T := T \cup \{e\}$. Go to Step 1.

Step 5: (v is a vertex of G_1^* and $\delta_{G_1^*}(v)$ contains a self-loop)

Let f be a self-loop of G_1^* incident to v.

Set $G_1 := G_1 \backslash f$, $G_1^* := G_1^* \backslash f$ and $T := T \cup \{f\}$. Go to Step 1.

Step 6: (v is a vertex of G_1^* and $\delta_{G_1^*}(v)$ does not contain any self-loops),

Find an edge e in $\delta_{G_1^*}(v)$ which attains the value $\max\{w(e') \mid e' \in \delta_{G_1^*}(v)\}$.

Set $G_1 := G_1 \backslash e$, $G_1^* := G_1^*/e$ and $T^* := T^* \cup \{e\}$. Go to Step 1.

In the above algorithm, we can symmetrize Step 3 and Step 5, when we replace the operation $G_1^* := G_1^* \backslash f$ in Step 3 by $G_1^* := G_1^*/f$ and the operation $G_1 := G_1 \backslash f$ in Step 5 by $G_1 := G_1/f$. However, the edge contraction procedure is time consuming. In addition, to construct a linear time algorithm, we have to delete the edge f from both graphs in Step 3 and Step 5. We will discuss this problem in the next section.

Now we show the correctness of the algorithm briefly. It is well-known that for any edge $e \in E$, the graph $G \backslash e$ is a dual graph of G^*/e and the graph G/e is a dual graph of $G^* \backslash e$ (see [10] for example). If $e \in E$ is a self-loop of the graph G, it is easy to show that $G \backslash e$ is a dual graph of $G^* \backslash e$. This property directly implies the following.

Claim 1 *Throughout the iterations of Algorithm A, G_1^* is a dual graph of G_1.*

Then we show the correctness of the algorithm.

Theorem 2 *If Algorithm A terminates, it correctly finds a minimum weight spanning forest T of G and a maximum weight spanning forest T^* of G^*.*

Proof. We only need to show that at the entrance of each iteration, (1) for any minimum weight spanning forest T_1 of G_1, $T_1 \cup T$ is a minimum weight spanning forest of G and (2) for any maximum weight spanning forest T_1^* of G_1^*, $T_1^* \cup T^*$ is a maximum weight spanning forest of G^*.

When an isolated vertex is chosen at Step 2, it is obvious. Consider the case that the algorithm executes Step 3. Since f is a self-loop of G_1, any spanning forest of G_1 does not contain the edge f. From the definition of the dual graph, $\{f\}$ is a cut-set of G_1^* and so, each spanning forest of G_1^* contains the edge f. When Step 4 is executed, the edge set $\delta_{G_1}(v)$ is a cut-set of G_1 and also a cycle of G_1^*. Let e be a minimum weight edge in $\delta_{G_1}(v)$. Then it is easy to show that (i) there exists a minimum weight spanning forest of G_1 containing e and (ii) there exists a maximum weight spanning forest of G_1^* excluding the edge e. The proof is similar to that of the correctness of Prim's algorithm for minimum spanning tree problem (see [11] or [10] for example). We can show the correctness of Step 5 and Step 6 in the same way. //

In each iteration of Algorithm A, either a vertex or an edge is removed. It implies the following result.

Claim 3 *The number of iterations of Algorithm A is bounded by $|V| + |V^*| + |E|$.*

In the next section, we describe a technique to save the time complexity of each iteration to O(1).

3 Linear Time Algorithm

For any vertex v of a graph G, $d_G(v)$ denotes the degree of the vertex of G (here we assume that each self-loop is counted twice). The following property gives an idea to save the time complexity of each iteration of Algorithm A to O(1).

Lemma 4 *Let $G = (V, E)$ be a planar graph and $G^* = (V^*, E)$ a dual graph of G. Then either G or G^* contains a vertex whose degree is less than four.*

Proof. From Euler's formula, $|V| - |E| + |V^*| = \kappa + \kappa^*$ where κ (κ^*) denotes the number of components of G (G^*). It is enough to show that the mean value z of degrees of the vertices in G or G^* is less than four. From Euler's formula, it is obvious that $z = (\sum_{v \in V} d_G(v) + \sum_{v \in V^*} d_{G^*}(v))/(|V| + |V^*|) = 4|E|/(|E| + \kappa + \kappa^*) < 4.//$

In the rest of this section, we construct and discuss an algorithm which choses a vertex whose degree is less than four at Step 2 of Algorithm A. At first, we show that by employing the above strategy, the time complexity of each iteration of Algorithm A is bounded by O(1). Next, we show that by employing a bucket technique, we can choose a desired vertex at Step 2 of Algorithm A in O(1) time.

Now assume that Algorithm A choses a vertex v whose degree is less than four at Step 2. In the rest of this section, we assume that we maintain two graphs G_1 and G_1^* by corresponding adjacency lists. Clearly, both Step 1 and Step 2 require constant time. Since we maintain each graph by an adjacency list, we can delete an edge in constant time. Thus, the time complexities of Step 3 and Step 5 are O(1). Consider the case that Algorithm A executed Step 4. Since the degree of v is less than four, we can find a minimum weight edge e in $\delta_{G_1}(v)$ in constant time. If we contract the edge e connecting two vertices v and u, it requires $O(\min\{d_{G_1}(v), d_{G_1}(u)\})$ time (see [2] for example). Since $d_{G_1}(v) < 4$, the edge contraction procedure in Step 4 requires O(1) time. Similarly, we can show that the time complexity of Step 6 is O(1).

The above discussion implies that when we can choose a vertex v whose degree is less than four at Step 2, the time complexities of Steps 1-6 are O(1). Next, we show how to choose such a vertex in constant time. We prepare a bucket which contains all the vertices whose degrees are less than four. Then, we can choose a desired vertex from the bucket at Step 2 in constant time. In the following, we describe a method to update the bucket.

When an edge e is deleted from a graph, the degrees of two end vertices of e decrease by 1 and degrees of other vertices do not change. So, in the worst case, we have to throw two vertices into the bucket. Now consider the case that an edge e is contracted. The new vertex obtained by identifying two ends of e is denoted by r. At first we remove the end vertices of e and if the degree of the new vertex r is less than four, we throw the vertex into the bucket. The degrees of other vertices do not change. From the above, in each iteration of Algorithm A, we remove at most two vertices from the bucket and throw at most four vertices into the bucket in each iteration. So, we can update the bucket in constant time.

By employing the above bucket technique, we can save the time complexity of each iteration of Algorithm A to $O(1)$ time. Claim 3 shows that the number of iterations is bounded by $|V| + |V^*| + |E|$. Clearly, Step 0 requires $O(|V| + |V^*| + |E|)$ time and when we start the algorithm we can calculate the degrees of all vertices and set-up the bucket in $O(|V| + |V^*| + |E|)$ time. Thus, the overall time complexity is $O(|V| + |V^*| + |E|)$.

4 Discussions

In this paper, we proposed a liner time algorithm for the minimum weight spanning forest problem on a planar graph. In each iteration, our algorithm picks a vertex whose degree is less than four and executes Prim's algorithm. Lemma 4 shows the existence of such an vertex.

Similarly to Lemma 4, we can prove the following property easily.

Lemma 5 *Let $G = (V, E)$ be a planar graph and $G^* = (V^*, E)$ a dual graph of G. Then either G has a vertex whose degree is less than six or G^* has a vertex whose degree is less than three.*

If we employ this lemma, we can construct another linear time algorithm. Now assume that we maintain the original graph as a planar embedding and its the geometric dual graph. If the degree of a vertex of the dual graph is one, unique adjacent edge corresponds to a facial self-loop of the planar embedding of the original graph. When the degree of a vertex of the dual graph is two, we can identify the pair of emanating edges as a facial parallel edges of the planar embedding. So, it is possible to describe the algorithm without using the notion of dual graphs. However, in that case, we have to maintain the face structure of the planar embedding of original graph.

Acknowledgements

I am grateful to Takao Nishizeki for many useful suggestions.

References

[1] D. Cheriton and R.E. Tarjan. Finding minimum spanning trees. *SIAM J. Computing*, 5, pp.724–742, 1976.

[2] N. Chiba, T. Nishizeki, and N. Saito. Efficient algorithms for graph alterations (in Japanese). *Trans. IECE*, J64-D, pp.934–939, 1981.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, pp.269–271, 1959.

[4] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization problems. *J. of ACM*, 34, pp.596–615, 1987.

[5] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected trees in undirected and directed graphs. *Combinatorica*, 6, pp.109–122, 1986.

[6] R.L. Graham and O. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7, pp.43–57, 1985.

[7] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[8] J. E. Hopcroft and R.E. Tarjan. Efficient planarity testing. *J. of ACM*, 21, pp.549–568, 1974.

[9] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. American Math. Society*, 2, pp.48–50, 1956.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[11] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Tech. J.*, 36, pp.1389–1401, 1957.

[12] H. Whitney. On the abstract properties of linear dependence. *Amer. J. Math.*, 57, pp.509–533, 1935.

[13] A. Yao. An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees. *Information Processing Letters*, 4, pp.21–23, 1975.