ELSEVIER

Discrete Applied Mathematics 58 (1995) 91–94

DISCRETE APPLIED MATHEMATICS

Communication

The minimum spanning tree problem on a planar graph

Tomomi Matsui

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

Communicated by T. Ibaraki
Received 8 March 1994

Finding a spanning tree of minimum weight is one of the best known graph problems. Several efficient algorithms exist for solving this problem [3–5, 7, 9]. This note presents a linear time algorithm for the minimum spanning tree problem on a planar graph. Cheriton and Tarjan [1] 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 the 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).

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 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. In this note, we present a spanning forest as its edge set. 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 an edge subset F, denoted by $w(F)$, 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

SSDI 0166-218X(94)00095-6

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^*. It is known that a graph is planar if and only if it has a dual graph [10]. If we have a planar embedding of a graph G, it is easy to construct a dual graph of G geometrically (see, [8] for example). In [6], Hopcroft and Tarjan proposed a linear time algorithm for embedding a planar graph in the plane.

In this note, 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^*. Thus, the problem of finding a minimum weight spanning forest of G is essentially the same as the problem of finding a maximum weight spanning forest of G^*.

Now we describe the main framework of our algorithm.

Algorithm 1

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 graphs, 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: Let f be a self-loop of G_1 incident to v.
Set $G_1 := G_1 \setminus f$, $G_1^* := G_1^* \setminus f$ and $T^* := T^* \cup \{f\}$. Go to Step 1.

Step 4: Find an edge e in $\delta_{G_1}(v)$ which attains the value $\min\{w(e') \mid e' \in \delta_{G_1}(v)\}$.
Set $G_1 := G_1/e$, $G_1^* := G_1^* \setminus e$ and $T := T \cup \{e\}$. Go to Step 1.

Step 5: Let f be a self-loop of G_1^* incident to v.
Set $G_1 := G_1 \setminus f$, $G_1^* := G_1^* \setminus f$ and $T := T \cup \{f\}$. Go to Step 1.

Step 6: Find an edge e in $\delta_{G_1^*}(v)$ which attains the value $\max\{w(e') \mid e' \in \delta_{G_1^*}(v)\}$.
Set $G_1 := G_1 \setminus e$, $G_1^* := G_1^*/e$ and $T^* := T^* \cup \{e\}$. Go to Step 1.

In the above algorithm, we can symmetrize Steps 3 and 5, when we replace the operation $G_1^* := G_1^* \setminus f$ in Step 3 by $G_1^* := G_1^*/f$ and the operation $G_1 := G_1 \setminus f$ 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 Steps 3 and 5. We will discuss this problem later.

It is easy to show that throughout the iterations of Algorithm 1, G_1^* is a dual graph of G_1 (see [8] for example). Then it directly implies the correctness of Algorithm 1.

Theorem 1. *If Algorithm* 1 *terminates, it correctly finds a minimum weight spanning forest* T *of* G *and a maximum weight spanning forest* T^* *of* G^*.

In each iteration of Algorithm 1, either a vertex or an edge is removed. It implies the following result.

Claim 2. *The number of iterations of Algorithm* 1 *is bounded by* $|V| + |V^*| + |E|$.

In the following, we describe a technique to save the time complexity of each iteration to O(1). 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). It is well-known that Euler's formula implies the following property.

Lemma 3. *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.*

In the rest of this note, we construct and discuss an algorithm which chooses a vertex whose degree is less than four at Step 2 of Algorithm 1. First, we show that by employing the above strategy, the time complexity of each iteration of Algorithm 1 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 1 in O(1) time.

Now assume that Algorithm 1 chooses a vertex v whose degree is less than four at Step 2. Algorithm 1 maintains two graphs G_1 and G_1^* by corresponding adjacency lists. Then it is obvious that both Steps 1 and 2 require constant time. Since we maintain each graph by an adjancency list, we can delete an edge in constant time. Thus, the time complexities of Steps 3 and 5 are O(1). Consider the case that Algorithm 1 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 $e = (u, v)$, it requires $O(\min\{d_{G_1}(v), d_{G_1}(u)\}) = O(3) = O(1)$ time (see [2] for example). 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 a vertex whose degree is less than four at Step 2 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. Now we describe a method to update the bucket. When an edge e is deleted from a graph, the degrees of two endvertices of e decrease by 1 and degrees of other vertices do not change. Consider the case that an edge e is contracted and a new vertex, denoted by r, is obtained by identifying two ends of e. Then we remove the endvertices 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 1, 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 1 to O(1) time. Claim 2 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 $O(|V| + |V^*| + |E|)$.

Acknowledgement

I am grateful to Takao Nishizeki for many useful suggestions.

References

[1] D. Cheriton and R.E. Tarjan, Finding minimum spanning trees, SIAM J. Comput. 5 (1976) 724–742.
[2] N. Chiba, T. Nishizeki and N. Saito, Efficient algorithms for graph alterations, Trans. IECE J64-D (1981) 934–939 (in Japanese).
[3] E.W. Dijkstra, A note on two problems in connection with graphs, Numer. Math. 1 (1959) 269–271.
[4] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization problems, J. ACM 34 (1987) 596–615.
[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 (1986) 109–122.
[6] J.E. Hopcroft and R.E. Tarjan, Efficient planarity testing, J. ACM 21 (1974) 549–568.
[7] J.B. Kruskal, On the shortest spanning subtree of a graph and the travelling salesman problem, Proc. Amer. Math. Soc. 2 (1956) 48–50.
[8] E.L. Lawler, Combinatorial Optimization: Networks an Matroids (Holt, Rinehart and Winston, New York, 1976).
[9] R.C. Prim, Shortest connection networks and some generalizations, Bell System Tech. J. 36 (1957) 1389–1401.
[10] H. Whitney, On the abstract properties of linear dependence, Amer. J. Math. 57 (1935) 509–533.