A Note on Enumeration of 3-Edge-Connected Spanning Subgraphs in Plane Graphs

Yasuko MATSUI^{†a)}, Member and Kenta OZEKI^{††}, Nonmember

SUMMARY This paper deals with the problem of enumerating 3-edge-connected spanning subgraphs of an input plane graph. In 2018, Yamanaka et al. proposed two enumeration algorithms for such a problem. Their algorithm generates each 2-edge-connected spanning subgraph of a given plane graph with n vertices in $O(n)$ time, and another one generates each k -edge-connected spanning subgraph of a general graph with m edges in $O(mT)$ time, where T is the running time to check the k -edge connectivity of a graph. This paper focuses on the case of the 3-edge-connectivity in a plane graph. We give an algorithm which generates each 3-edge-connected spanning subgraph of the input plane graph in $O(n^2)$ time. This time complexity is the same as the algorithm by Yamanaka et al., but our algorithm is simpler than theirs.

key words: enumeration, algorithm, spanning subgraph, edge-connectivity

1. Introduction

In this paper, we present a simple algorithm that enumerates all the 3-edge-connected spanning subgraphs of a given plane graph with n vertices in $O(n^2)$ time, using reverse search method by Avis and Fukuda [1]. For this problem, Yamanaka et al. proposed the algorithm that enumerates all the k -edge-connected spanning subgraphs of a given general graph with n vertices and m edges. Their algorithm generates each k -edge-connected spanning subgraph of the input graph in $O(mT)$ time, where T is the running time to check the k -edge-connectivity of a graph. From the result by Gabow [3], it can be observed that $T = O(m + k^2n \log(n/k))$ holds. So, their algorithm enumerates all the 3-edge-connected spanning subgraphs in an input plane graph and requires $O(n^2 \log n)$ time per each. Note that $m \leq 3n - 6$ holds for a plane graph. On the other hand, this paper also gives an algorithm that generates each 3-edge-connected spanning subgraph of an input plane graph in $O(n^2)$ time by using an algorithm which can check 3-edge-connectivity in $O(n)$ time for a given plane graph. In 2017, for a given graph, Mehlhorn et al. already proposed an algorithm for checking 3-edge-connectivity in $O(n)$ time [5]. So, both algorithms require the same time complexity. However, ours is very simple and short. It has an advantage.

2. Preliminary

Let $G = (V(G), E(G))$ be an undirected unweighted graph with vertex set $V(G)$ and edge set $E(G)$. We always denote $|V(G)|$ and $|E(G)|$ by n and m , respectively. A graph G is *simple* if G has no multi-edges and no self-loop. Throughout this paper, we suppose that graphs are simple unless otherwise noted. An edge set S of a graph G is called a *bond* if $G - S$ is not connected, while $G - S'$ is connected for any proper subset S' of S . A graph G is *k-edge-connected* if G does not have any bond of size at most $k - 1$. Let us remark that, for $k = 1$, a 1-edge-connected graph is just a connected graph. A graph $H = (V(H), E(H))$ is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ hold. A subgraph $H = (V(H), E(H))$ of G is *spanning* if $V(H) = V(G)$. Throughout this paper, we assume that the edges in $E(G)$ are labeled such as $E(G) = \{e_1, e_2, \dots, e_m\}$. Let e_i and e_j , $i < j$, be two edges in G . We say that e_i is *smaller* than e_j , denoted by $e_i < e_j$. Let e be an edge of G . For a subgraph H of G , we denote by $H - e$ the graph obtained from H by removing e . We denote by $H + e$ the graph obtained from H by inserting e . A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph G divides the plane into connected regions called *faces*. We put a vertex f^* in each face f of G , and for each edge e in G , we connect by the edge e^* the vertices f^* and g^* in G^* , where f and g are the faces of G whose contours share the edge e . Then we call the obtained graph the *dual graph* of G , and is denoted by G^* . For an edge set S in G , let $S^* = \{e^* | e \in S\}$. A *cycle* is a circuit in which the only repeated vertices are the first and last vertices. The *length* of a cycle is the number of edges involved. The following is a well-known lemma, which will be used for our algorithm.

Lemma 1: An edge set S in a connected plane graph G is bond if and only if S^* forms a cycle in G^* .

3. Enumerating All k-Edge-Connected Spanning Subgraphs in Graphs

In this section, we introduce results by Yamanaka et al. [7], [8]. They proposed the algorithm for generating all k -edge-connected spanning subgraphs. The algorithm based on the reverse search [1] traverses the tree structure in depth-first

Manuscript received March 26, 2020.

Manuscript revised August 12, 2020.

Manuscript publicized October 7, 2020.

[†]The author is with the School of Science, Tokai University, Hiratsuka-shi, 259-1292 Japan.

^{††}The author is with the Faculty of Environment and Information Sciences, Yokohama National University, Yokohama-shi, 240-8501 Japan.

a) E-mail: yasuko@tokai-u.jp

DOI: 10.1587/transinf.2020FCL0002

search manner.

Let G be a k -edge-connected graph with $k \geq 1$. We denote by $S_k(G)$ the set of k -edge-connected spanning subgraphs of G . Note that G itself is in $S_k(G)$. To see a tree structure, we define the parent for each k -edge-connected spanning subgraph of G except G . Let H be a k -edge-connected spanning subgraph in $S_k(G) \setminus \{G\}$. Let $\text{sm}(H)$ be the smallest edge in $E(G) \setminus E(H)$. Then, we define $\text{par}(H) := H + \text{sm}(H)$. From the definition, it is easy to observe that $\text{par}(H)$ is also k -edge-connected spanning subgraph of G . By repeatedly finding the parents starting from H , we obtain a sequence $H, \text{par}(H), \text{par}(\text{par}(H)), \dots$ of k -edge-connected spanning subgraphs. We call such a sequence the *appending sequence* of H . The sequence starts with H and ends with G . In [7] and [8], they proved the following lemma and proposed two algorithms for $k = 2$ and a general case.

Lemma 2: Let $H \neq G$ be a k -edge-connected spanning subgraph of a graph G . Then, the appending sequence of H always ends up with G .

Theorem 1: Let G be a plane graph with n vertices. One can generate every 2-edge-connected spanning subgraph of G in $O(n)$ time for each.

Theorem 2: Let G be a graph. One can generate each k -edge-connected spanning subgraph of G in $O(mT)$ time for each, where T is the running time to check whether a graph is k -edge-connected.

From the result by Gabow [3], it can be observed that $T = O(m + k^2 n \log(n/k))$ holds.

Here, we focus on the case of $k = 3$. In 2017, Mehlhorn et al. proposed an algorithm for checking 3-edge-connectivity of an input graph [5]. The time complexity of their algorithm is a linear time. Therefore, by using their checking algorithm, the second algorithm enumerates all the 3-edge-connected spanning subgraphs in an input plane graph and requires $O(n^2)$ time per each.

4. Enumerating All 3-Edge-Connected Spanning Subgraphs in Plane Graphs

In this section, we focus on 3-edge-connected spanning subgraphs in plane graphs. For this problem, as mentioned above, the algorithm obtained by Yamanaka et al.'s one and Mehlhorn et al.'s one generates all 3-edge-connected spanning subgraphs in $O(n^2)$ time per each. Our algorithm also requires the same time complexity per each and bases on Yamanaka et al.'s algorithm whose case is $k = 2$. The key idea is Lemma 1.

To generate all 2-edge-connected subgraphs in a plane graph G , Yamanaka et al. used in the proof of Theorem 1 the dual graph G^* of G . In their algorithm, they check multiple edges in G^* , whose corresponding edges in G cannot be deleted to generate 2-edge-connected subgraphs of G by Lemma 1.

In the case of 3-edge-connected subgraphs, we focus

on a cycle of length 3 in the dual graph G^* of G to find those edges that cannot be deleted to keep 3-edge-connectedness.

In the following, we give our main theorem.

Theorem 3: Let G be a plane graph with n vertices. One can generate each 3-edge-connected spanning subgraph of G in $O(n^2)$ time for each.

Proof. Let us estimate the running time required for a subgraph $H \in S_3(G)$ in the family tree. When H is generated from its parent $\text{par}(H)$, the dual graph is updated. This takes $O(n)$ time. To generate a child of H , for each edge e in H with $e < \text{sm}(H)$, we check 3-edge-connectivity of $H - e$.

Recall that, from Lemma 1, it is sufficient to check cycles whose length is 3 in the dual graph G^* of G . To find such cycles efficiently, we check whether or not two end points of each edge e^* have a common neighbor in G^* , which requires $O(n)$ time for each edge e^* in G^* . If there exists such a vertex, then e^* is contained in a cycle of length 3, and hence we cannot delete the corresponding edge e from G . Thus, this check can be done in $O(n)$ time for each and $O(m)$ time in total. Therefore, our algorithm requires $O(n^2)$ time for each 3-edge-connected spanning subgraph in G , since $m \leq 3n - 6$ holds in a plane graph. Q.E.D.

5. Conclusion

We have proposed the algorithm for enumerating 3-edge-connected spanning subgraphs. Our algorithm enumerates all the 3-edge-connected spanning subgraphs of a given plane graph with n vertices in $O(n^2)$ time for each. On checking 3-edge-connectivity, although it is the same time complexity of the algorithm by Mehlhorn et al. [5], our algorithm is extremely simple, which is also an advantage.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP18K03391 and JP20K04973.

References

- [1] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Appl. Math.*, vol.65, no.1-3, pp.21–46, 1996.
- [2] A. Conte, M.M. Kanté, Y. Otachi, T. Uno, and K. Wasa, "Efficient enumeration of maximal k -degenerate subgraphs in a chordal graph," *Proc. 23th Annual International Computing and Combinatorics Conference, Lecture Notes in Computer Science*, vol.10392, pp.150–161, Springer, 2017.
- [3] H.N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences," *J. Computer and System Sciences*, vol.50, no.2, pp.259–273, 1995.
- [4] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, and K. Makino, "Enumerating spanning and connected subsets in graphs and matroids," *Proc. 14th Annual European Symposium on Algorithms, Lecture Notes in Computer Science*, vol.4168, pp.444–455, Springer, 2006.
- [5] K. Mehlhorn, A. Neumann, and J.M. Schmidt, "Certifying 3-edge-connectivity," *Algorithmica*, vol.77, pp.309–335, 2017.
- [6] R.C. Read and R.E. Tarjan, "Bounds on backtrack algorithms for listing cycles, paths, and spanning trees," *Networks*, vol.5, no.3,

- pp.237–252, 1975.
- [7] K. Yamanaka, Y. Matsui, and S. Nakano, “More routes for evacuation,” Proc. 24th International Computing and Combinatorics Conference (COCOON), Lecture Notes in Computer Science, vol.10976, pp.76–83, Springer, July 2018.
- [8] K. Yamanaka, Y. Matsui, and S. Nakano, “Enumerating highly-edge-connected spanning subgraphs,” IEICE Trans. Fundamentals, vol.E102-A, no.9, pp.1002–1006, Sept. 2019.
-