SUMMARY In this paper, we consider the problem of enumerating spanning subgraphs with high edge-connectivity of an input graph. Such subgraphs ensure multiple routes between two vertices. We first present an algorithm that enumerates all the 2-edge-connected spanning subgraphs of a given plane graph with \( n \) vertices. The algorithm generates each 2-edge-connected spanning subgraph of the input graph in \( O(n^2) \) time. We next present an algorithm that enumerates all the \( k \)-edge-connected spanning subgraphs of a given general graph with \( m \) edges. The 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.

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

1. Introduction

Evacuation route planning in a road network requires at least one route from every point to a shelter. For example, a spanning tree of a network gives one evacuation route for each point. However, in time of disaster, it is easy to imagine that a lot of roads are broken. In the situation that we know only one route between the current place to a shelter, nobody can ensure that the route can be passed through in safety. Hence, we are required to ensure “multiple” evacuation routes to a shelter from every place. Moreover, to avoid traffic congestion at the time of evacuation, we need some evacuation routes. From these points of view, finding spanning graphs with high edge-connectivity is important, since such graphs ensure multiple routes between two points. In this paper, we focus on the problem of enumerating spanning subgraphs with high edge-connectivity.

Enumerating designated subgraphs is a fundamental and important problem. The subgraph enumeration is one of the strong and appealing strategies to discover valuable knowledge from enormous graph data in various research areas such as data mining, bioinformatics, and artificial intelligence. To discover valuable knowledge from practical graphs, enumeration algorithms for subgraphs with some properties are studied, such as paths [2], cycles [2], subtrees [16], spanning trees [12], [13], [15], cliques [5], [10], pseudo cliques [14], \( k \)-degenerate subgraphs [4], matchings [9], [15], connected induced subgraphs [1], [11], and so on.

In this paper, we focus on the problem of enumerating spanning subgraphs in a graph. Khachiyan et al. [8] studied the problem of enumerating all the minimal 2-vertex-connected spanning subgraphs. This enumeration problem is a natural extension of the well-known spanning tree enumeration problem. Boros et al. [3] proposed an algorithm that enumerates all the \( k \)-vertex-connected spanning subgraphs. Both papers focused on the vertex-connectivity of spanning subgraphs. It is well known that the vertex-connectivity is one of the most fundamental concepts in network reliability. On the other hands, edge-connectivity is also the well-known fundamental measure of network reliability. To the best of our knowledge, there is no result for the problem of enumerating \( k \)-edge-connected spanning subgraphs. In this paper, we focus on the problem of enumerating spanning subgraphs with high edge-connectivity.

We present the following two enumeration algorithms for spanning subgraphs using reverse search technique by Avis and Fukuda [1]. We first consider a spanning subgraph enumeration problem on plane graphs. A plane graph is a well-known model of road networks. We first present an algorithm that enumerates all the \( k \)-edge-connected spanning subgraphs of a given plane graph with \( n \) vertices. The algorithm generates each \( k \)-edge-connected spanning subgraph of the input graph in \( O(n^2) \) time. We next show that the algorithm for plane graphs can be extended to the one for general graphs. We present an algorithm that enumerates all the \( k \)-edge-connected spanning subgraphs of a given general graph with \( n \) vertices and \( m \) edges. The 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 [7], it can be observed that \( T = O(m + k^2 n \log (n/k)) \) holds.

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-edge and no self-loop. Throughout this paper, we suppose that graphs are simple unless otherwise noted. A graph $G$ is **$k$-edge-connected** if the removal of any $k - 1$ edges in $E(G)$ does not disconnect $G$. 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 $G$ are labeled such as $E = \{e_1, e_2, \ldots, e_m\}$. 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 divides the plane into connected regions called **faces**. The **contour** of a face is the list of edges on the boundary of the face. Let $e$ be an edge in a plane graph $G$.

From the definition of plane graphs, $e$ is shared by two faces. We denote the two faces by $f_1(e)$ and $f_2(e)$. The **dual graph** $D = (V(D), E(D))$ of a plane graph $G$ is a multi-graph where $V(D)$ is the set of the faces of $G$ and $E(D) = \{(f, g) \mid $ The contours of $f$ and $g$ share an edge$\}$ is a multi-set of edges. Note that, if two faces $f$ and $g$ share two or more edges, then the dual graph has multi-edges between $f$ and $g$.

### 3. Enumerating All Spanning Subgraphs

In this section, we first present an algorithm that enumerates all the 2-edge-connected spanning subgraphs of a given plane graph. We then present an algorithm that enumerates all the $k$-edge-connected spanning subgraphs of a given general graph.

Our algorithms are based on the reverse search technique by Avis and Fukuda [1]. In reverse search, we first define a rooted tree structure on a set of objects to be enumerated such that each vertex corresponds to an object and each edge corresponds to a parent-child relation between two objects. Then, we design an algorithm that recursively traverses the tree structure from its root object in a depth-first manner.

#### 3.1 2-Edge-Connected Spanning Subgraphs in Plane Graphs

Let $G = (V(G), E(G))$ be a plane graph. In this section, we give an algorithm that enumerates all the 2-edge-connected spanning subgraphs of $G$. If $G$ is not 2-edge-connected, then there is no 2-edge-connected spanning subgraph of $G$. Hence, without loss of generality, we suppose that $G$ is 2-edge-connected in this subsection.

We denote by $S_2(G)$ the set of 2-edge-connected spanning subgraphs of $G$. Note that $G$ itself is in $S_2(G)$. We first define a tree structure rooted at $G$ among $S_2(G)$. To define a tree structure, we define a **parent** for each 2-edge-connected spanning subgraph of $G$ except $G$. Let $H$ be a 2-edge-connected spanning subgraph in $S_2(G) \setminus \{G\}$. Let $sm(H)$ be the smallest edge in $E(G) \setminus E(H)$. Then, we define $par(H) := H + sm(H)$. From the definition, it is easy to observe that $par(H)$ is also a 2-edge-connected spanning subgraph of $G$ and $par(H)$ is defined uniquely. By repeatedly finding the parents starting from $H$, we obtain a sequence of 2-edge-connected spanning subgraphs. We call such a sequence the **appending sequence** of $H$. The sequence starts with $H$ and ends with $G$. We have the following lemma.

**Lemma 1:** Let $H \neq G$ be a 2-edge-connected spanning subgraph of a plane graph $G$. Then, the appending sequence of $H$ always ends up with $G$.

**Proof.** Let us define a potential function $\phi$ for a subgraph $H$ as $\phi(H) := |E(H)|$. Then, from the definition of the parent, $par(H)$ is 2-edge-connected spanning subgraph and $\phi(par(H)) = \phi(H) + 1$. Since the parent is defined for every 2-edge-connected spanning subgraph except $G$, we finally have $G$ in the appending sequence. Q.E.D.

Now, we are ready to define a tree structure. By merging all the appending sequences for all the subgraphs in $S_2(G)$, we have a rooted tree structure, called the **family tree**, such that (1) its root corresponds to $G$, (2) each vertex in the tree corresponds to a 2-edge-connected spanning subgraph of $G$, and (3) each edge in the tree corresponds to a parent-child relation. Figure 1 shows an example of the family tree.

![Fig. 1 The family tree of the input graph.](image-url)
It is easy to see that, if we can traverse the family tree, we can enumerate all the 2-edge-connected spanning subgraphs of $G$, since the family tree contains all the 2-edge-connected spanning subgraphs of $G$. To traverse it, we design an algorithm that enumerates all the children of a given subgraph in $S_2(G)$. By recursively applying the child-enumeration algorithm from the root, we can traverse the family tree in a depth-first manner.

Now, let us give a condition to be a child of a given 2-edge-connected spanning subgraph. Let $H$ be a 2-edge-connected spanning subgraph in $S_2(G)$. If an edge $e$ is removed from $H$, then we may obtain a child. However, if $H - e$ is not 2-edge-connected, then such $H - e$ is not a child. Similarly, if $e$ is not the smallest edge in $E(G) \setminus E(H - e)$, that is $e \neq \text{sm}(H - e)$, then such $H - e$ is not a child either. From the observations above, we have the following lemma.

**Lemma 2:** Let $H$ be a 2-edge-connected spanning subgraph in a plane graph $G$, and let $e$ be an edge of $H$. Then, $H - e$ is a child of $H$ if and only if $e < \text{sm}(H - e)$ holds and $H - e$ is 2-edge-connected.

From the lemma above, we have the algorithm shown in **Algorithm 1**. In the for-loop of the algorithm, we choose only the edges with $e = \text{sm}(H - e)$. For each such edge, we check 2-edge-connectivity of $H - e$. The algorithm does not check whether or not $H - e$ is a spanning subgraph explicitly, since $H - e$ is always a spanning subgraph. Note that, if $H$ is a 2-edge-connected spanning subgraph, $H - e$ is always a spanning subgraph for any edge $e$ in $H$.

Now, let us estimate the running time of **Algorithm 1**. We estimate the running time to be required for a vertex in a family tree. In the worst case, we check 2-edge-connectivity for each edge. For each edge $e$, 2-edge-connectivity of the graph $H - e$ can be checked in $O(n^2)$ time: A graph is 2-edge-connected if and only if the graph has no bridge. It is known that one can check whether or not a graph has no bridge in $O(m)$ time using depth first search [6, Problem 22-2]. Hence, we need $O(m^2)$ time in total. Since for a plane graph $m \leq 3n - 6$ holds, the following theorem is obtained.

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

Now, let us improve the running time of our algorithm. The bottleneck of **Algorithm 1** is the running time to check 2-edge-connectivity when an edge is removed. To check the connectivity more efficiently, we introduce an observation and a data structure.

**Lemma 3:** Let $H$ be a 2-edge-connected plane graph, and let $e$ be an edge of $H$. Then, $H - e$ is 2-edge-connected if and only if $f_1(e)$ and $f_2(e)$ share only the edge $e$ in $H$ (recall that $f_1(e)$ and $f_2(e)$ are the two faces sharing $e$).

**Proof.** ($\Rightarrow$) We assume for a contradiction that $f_1(e)$ and $f_2(e)$ share two or more edges in $H$. Let $e'$ be an edge shared by $f_1(e)$ and $f_2(e)$ except $e$. Then $e'$ is a bridge in $H - e$ (See Fig. 2), which is a contradiction. ($\Leftarrow$) The removal of $e$ combines $f_1(e)$ and $f_2(e)$ into a face. Let $f$ be the face in $H - e$ obtained by removing $e$. Then, any edge in $H - e$ is included in at least one cycle. Hence, $H - e$ is still 2-edge-connected.

**Q.E.D.**

We can use the lemma above to check the 2-edge-connectivity of $H - e$. Our algorithm first construct the dual graph of $G$ as an adjacency matrix representation, where each element for two faces $f$ and $g$ in the matrix stores the list and number of edges shared by $f$ and $g$, in $O(n^2)$ time. Besides, for each edge, we store the two faces sharing the edge. Then, the algorithm maintains the dual graph and incident face information for edges. Running time for maintenance is $O(n)$ for each child: Using the matrix of the dual graph, we can know the number of edges shared by any two faces. Hence, we can decide whether or not $H - e$ is 2-edge-connected. This check can be done in constant time. Note that, for each edge, its incident faces can be obtained using the data structure. To generate a child, an edge $e$ is removed. When $e$ is removed, $f_1(e)$ and $f_2(e)$ are merged into a face, say $f'$. The face $f'$ is adjacent to the faces adjacent to $f_1(e)$ or $f_2(e)$ except $f_1(e)$ and $f_2(e)$. For each edge incident to $f_1(e)$ or $f_2(e)$, we update incident face information such that both $f_1(e)$ and $f_2(e)$ are updated to $f'$. These can be done in $O(n)$ time. Therefore, we have the following theorem.

**Theorem 2:** Let $G$ be a plane graph with $n$ vertices. After $O(n^2)$-time preprocessing, one can enumerate every 2-edge-connected spanning subgraph of $G$ in $O(n)$ time for each.

**Proof.** Let us estimate the running time required for a subgraph $H \in S_2(G)$ 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 2-edge-connectivity.

**Algorithm 1: FIND-CHILDREN-2-CONNECTED(H)**

```plaintext
/* $G$ is an input plane graph and is stored in a global memory. $H$ is a 2-edge-connected spanning subgraph of $G$. */
1 Output $H$.
2 foreach $e \in E(H)$ with $e < \text{sm}(H)$ do
3 if $H - e$ is 2-edge-connected spanning subgraph of $G$ then
4    Find-Children-2-Connected($H - e$)
```

![Fig. 2](image)

Illustration for Lemma 3. (a) The two faces $f_1(e)$ and $f_2(e)$ share the two edges $e$ and $e'$. (b) After $e$ is removed, then $e'$ becomes to be a bridge.
of $H - e$. Recall that, from Lemma 3, it is sufficient to check the number of edges shared by the corresponding two faces in the dual graph of $H$. Hence, this check can be done in $O(1)$ time for each and $O(m)$ time in total. Since $m < 3n$ holds in a plane graph, the total running time for $H$ is $O(n)$ time.

Q.E.D.

3.2 $k$-Edge-Connected Spanning Subgraphs in General Graphs

The discussion in the previous subsection can be applied to general cases: we are given a general graph $G$ and are required to enumerate all the $k$-edge-connected spanning subgraphs of $G$. This section shows that Algorithm 1 can be applied to the problem with a slight modification.

Let $G$ be an input graph. We denote by $S_k(G)$ the set of all the $k$-edge-connected spanning subgraphs of $G$. If $G$ is not $k$-edge-connected, then $G$ has no $k$-edge-connected spanning subgraph. Thus, we assume that $G$ is $k$-edge-connected.

In a similar way, we define a family tree among the set of $k$-edge-connected spanning subgraphs of $G$, as follows. Let $H$ be a $k$-edge-connected spanning subgraph in $S_k(G) \setminus \{G\}$. Then, we define $par(H) := H + \text{sm}(H)$. By repeatedly finding the parents starting from $H$, we obtain a sequence of subgraphs in $S_k(G)$. By merging all the sequences for all the subgraphs in $S_k(G)$, we have a rooted tree structure, called the family tree, among $S_k(G)$ such that (1) its root corresponds to $G$, (2) each vertex in the tree corresponds to a spanning subgraph in $S_k(G)$, and (3) each edge in the tree corresponds to a parent-child relation.

Now, let us consider how to traverse the family tree. The algorithm is almost same as our first algorithm. The difference is how to check edge-connectivity when a child is generated. Let $H$ be a $k$-edge-connected spanning subgraph of $G$. Then, for an edge $e$ of $H$, $H - e$ is a child of $H$ if and only if (i) $e < \text{sm}(H)$ and (ii) $H - e$ is $k$-edge-connected. The condition (ii) can be checked using Gabow’s algorithm [7] in $O(T)$ time, where $T = O(m + k^2n \log(n/k))$. The condition (i) can be checked in $O(1)$ time by maintaining a sorted list of edges in $E(G) \setminus E(H)$. (It is sufficient to compare an edge $e$ with the smallest element in the list.) Besides, the list can be updated in $O(1)$ time. (If the conditions are satisfied, we update the sorted list by inserting an edge $e$ as the first element.) Hence, we can check whether or not $H - e$ is a child of $H$ in $O(T)$ time. To enumerate all the children of $H$, we check the edge-connectivity of $H - e$ for each edge $e$ in $E(G) \setminus E(H)$. Therefore, we have the following theorem.

Theorem 3: 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.

If $k = 1$, we check whether or not $H - e$ is connected for each edge $e$ in $E(G) \setminus E(H)$. This connectivity can be checked in $O(m^2)$ time using a depth-first search on $H - e$.

Corollary 1: Let $G$ be a graph. One can generate each connected spanning subgraph of $G$ in $O(m^2)$ time for each.

4. Conclusions

We have designed two algorithms for enumerating spanning subgraphs with edge-connectivity at least $k$. Our first algorithm enumerates all the $2$-edge-connected spanning subgraphs of a given plane graph with $n$ vertices in $O(n)$ time for each. The second algorithm enumerates all the $k$-edge-connected spanning subgraphs of a given general graph with $m$ edges in $O(mT)$ time for each, where $T$ is the running time to check the $k$-edge-connectivity of a graph.

Future works include improving the running time of our algorithms. Can we enumerate all the $2$-edge-connected spanning subgraphs of a given plane graph in constant time for each? Our first algorithm enumerates all the $2$-edge-connected spanning subgraphs of a “plane” graph. Can we efficiently enumerate all the $k$-edge-connected spanning subgraph of other graph classes? In this paper, we only consider the problem of enumerating “all” the $k$-edge-connected spanning subgraphs of a given graph. Can we enumerate only all the “minimal” $k$-edge-connected spanning subgraphs efficiently?

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers JP16K00002, JP17K00003, and JP18H04091.

References

Katsuhisa Yamanaka is an associate professor of Department of Systems Innovation Engineering, Faculty of Science and Engineering, Iwate University. He received B.E., M.E., and Dr. Eng. degrees from Gunma University in 2003, 2005 and 2007, respectively. His research interests include combinatorial algorithms and graph algorithms.

Yasuko Matsui received her B.E., M.E., and Dr. Eng. degrees from Tokyo University of Science, Japan, in 1991, 1994, and 2001, respectively. She was an Assistant Professor in the School of Science, Tokyo Metropolitan University from 1994 to 1997. She is currently a Professor in the Department of Mathematical Sciences, School of Science, Tokai University. Her research interests include graph algorithms and combinatorial optimization. She is a member of IEICE, IPSJ, and ORSJ.

Shin-ichi Nakano received his B.E. and M.E. degrees from Tohoku University, Sendai, Japan, in 1985 and 1987, respectively. In 1987 he joined Seiko Epson Corp. and in 1990 he joined Tohoku University. In 1992, he received Dr. Eng. degree from Tohoku University. Since 1999 he has been a faculty member of Department of Computer Science, Faculty of Engineering, Gunma University. His research interests are graph algorithms and graph theory. He is a member of IPSJ, JSIAM, and ACM.