

# Efficient Enumeration of Induced Subtrees in a K-Degenerate Graph

Kunihiro Wasa<sup>1</sup>, Hiroki Arimura<sup>1</sup>, and Takeaki Uno<sup>2</sup>

<sup>1</sup> Hokkaido University, Graduate School of Information Science and Technology,  
Japan, {wasa, arim}@ist.hokudai.ac.jp

<sup>2</sup> National Institute of Informatics, Japan, uno@nii.jp

**Abstract.** In this paper, we address the problem of enumerating all induced subtrees in an input  $k$ -degenerate graph, where an *induced subtree* is an acyclic and connected induced subgraph. A graph  $G = (V, E)$  is a  $k$ -degenerate graph if for any its induced subgraph has a vertex whose degree is less than or equal to  $k$ , and many real-world graphs have small degeneracies, or very close to small degeneracies. Although, the studies are on subgraphs enumeration, such as trees, paths, and matchings, but the problem addresses the subgraph enumeration, such as enumeration of subgraphs that are trees. Their induced subgraph versions have not been studied well. One of few example is for chordless paths and cycles. Our motivation is to reduce the time complexity close to  $O(1)$  for each solution. This type of optimal algorithms are proposed many subgraph classes such as trees, and spanning trees. Induced subtrees are fundamental object thus it should be studied deeply and there possibly exist some efficient algorithms. Our algorithm utilizes nice properties of  $k$ -degeneracy to state an effective amortized analysis. As a result, the time complexity is reduced to  $O(k)$  time per induced subtree. The problem is solved in constant time for each in planar graphs, as a corollary.

## 1 Introduction

*Subgraph enumeration problems* are enumeration problems that given a graph  $G$  and a graph class  $\mathcal{S}$ , output all subgraphs  $S$  of  $G$  satisfying  $S \in \mathcal{S}$  without duplicates. Subgraph enumeration problems are widely studied [1–3, 6–10]. Enumeration involves a huge number of solutions, thus enumeration algorithms are supposed to run in short time, with respect to the number of solutions  $N$ . For example, if an algorithm runs in  $O(Nf)$  time for small  $f$ , other than preprocessing, we can consider the algorithm is efficient. In this case, we say that the algorithm runs in  $O(f)$  time per solution, or  $O(f)$  time for each solution. Further, the maximum computation time between two consecutive outputs called *delay* is also considered as a more efficiency of enumeration algorithms. Note that delay will not be  $O(f)$  even if an algorithm runs in  $O(f)$  time per solution.

Enumeration algorithms are widely studied in these days. Especially, the data mining area has a large amount of studies on pattern mining problem. The algorithms have to deal with huge databases and a huge number of solutions,

thus there are great needs of the algorithm theory on efficient enumeration. As we show below, many recent studies focus on the development of small complexity algorithms. Compared to other algorithms, enumeration algorithms have some unique aspects. For example, by operating only on the differences between the solutions, one can develop algorithms that run in time shorter than the amount of exact output. Other than this, since the recursion is much more structured compared to optimization, we can develop a non-trivial amortized analysis. As a consequent, researches on the enumeration algorithms have great interests.

In what follows, we fix the input graph  $G = (V, E)$ , and let  $m = |E|$ ,  $n = |V|$ . In the 1970s, Tarjan and Read [8] studied a problem of enumerating spanning trees in the input graph. Their algorithm runs in  $O(m + n + mN)$  time. Shioura, Tamura, and Uno [6] improved the complexity to  $O(n + m + N)$  time. Tarjan [7] proposed an algorithm for enumerating all cycle in  $O((|V| + |E|)(|\mathcal{C}(G)| + 1))$  time, where  $\mathcal{C}(G)$  is all cycle in  $G$ . Birmelé *et al.* [2] improved the complexity to in  $O(m + \sum_{c \in \mathcal{C}(G)} |c|)$  total time. They also presented an enumeration algorithm for all st-paths in the input graph  $G$  in  $O(m + \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$  total time, where  $\mathcal{P}_{st}(G)$  is all st-paths in  $G$ . Ferreira *et al.* [3] proposed an enumeration algorithm that enumerating all subtree having exactly  $k$  edges in  $G$  in  $O(kN)$  time. Wasa *et al.* [10] presented an improved version of Ferreira *et al.*'s problem in constant time delay when the input is a tree. As we see, speed up of enumeration algorithms have been intensively studied in long history.

Compared to these studies, induced subgraph enumerations have not been studied well. Avis and Fukuda [1] considered the connected induced subgraph enumeration problem. Their algorithm is based on reverse search, and runs in  $O(mnN)$  time. Uno [9] proposed an enumeration algorithm for enumerating all chordless path connecting the given vertices  $s$  and  $t$  and all chordless cycle in  $O((m + n)N)$  time.

In this paper, we address the problem of enumerating all induced subtrees in the given graph, where an induced subtree is a connected induced subgraph that has no cycle. Assume that the set of vertices in an induced subtree is  $S$ . Then,  $V \setminus S$  is a feedback vertex set of  $G$ . Feedback vertices are also fundamental graph objects and their enumeration problem is equivalent to that of induced subtrees. If the input graph  $G$  is a tree, the connected induced subgraph of  $G$  is a subtree. Thus, Wasa *et al.*'s shows that the induced subtree enumeration problem can be solved in constant time delay when the input graph is a tree. Tree is a simple graph class, so we are motivated whether we can do better in more general graph classes with non-trivial algorithms.

As a main result of this paper, we propose an algorithm for the  $k$ -degenerate graph case. The algorithm runs in  $O(k)$  time per solution, after  $(|V| + |E|)$  preprocessing time. The algorithm starts from the empty subgraph, and adds a vertex recursively to enlarge the induced subtree. The vertex to be added has to be adjacent to the current induced subtree, and has not to make a cycle. By using the degeneracy, we efficiently maintain the addible vertices, and the time complexity is bounded by a sophisticated amortized analysis. Real world graphs usually have small degeneracies, or only few vertex removals result small

degeneracies, the algorithm is expected to be efficient in practice. Compared to other graph classes, this is a strong point of  $k$ -degenerate graphs. There have been not so many studies on the use of the degeneracy for enumeration algorithm, and thus our approach introduces one of new way of developing practically efficient and theoretically supported algorithms.

The rest of this paper is organized as follows: In Section 2, we give definitions in this paper and the definition of our problem. In Section 3, we propose a basic enumeration algorithm based on a binary partition method. In Section 4, we improve the algorithm by using a property of the degeneracy, and analyze its time complexity. Finally, we conclude this paper and give future works in Section 5.

## 2 Preliminaries

### 2.1 Graphs

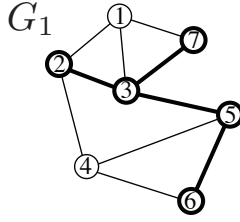
Let  $G = (V, E)$  be an *undirected graph*, where  $V$  is the set of *vertices* and  $E \subseteq V^2$  is the set of *edges*. In this paper, we assume that  $G$  is simple and finite. We denote by  $(u, v)$  the edge connecting  $u$  and  $v$ . For any vertices  $u, v$  of  $V$ , we say that  $u$  and  $v$  are *adjacent* to each other if  $(u, v) \in E$ . We denote by  $N_G(u)$  the set of all vertices adjacent to  $u$  in  $G$ . We define the *degree*  $d_G(u)$  of  $u$  in  $V$  as the number of vertices adjacent to  $u$ . In what follows, if it is clear from context, we omit the subscript  $G$ .

A *path* in  $G$  is a sequence of distinct vertices  $\pi(u, v) = (v_1 = u, \dots, v_j = v)$ , such that  $v_i$  and  $v_{i+1}$  are adjacent to each other for  $1 \leq i < j$ . If there is  $\pi(u, v)$  in  $G$ , we say that the path *connects*  $u$  and  $v$ . The *length* of path  $\pi(u, v)$  is the number of vertices in  $\pi(u, v)$  minus one. For any path  $\pi(u, v)$  of length larger than one,  $\pi(u, v)$  is called a *cycle* if  $u = v$ . We say that  $G$  is *connected* if there is a path connecting any pair of vertices in  $G$ .  $G$  is a *tree* if  $G$  has no cycle and is connected.

### 2.2 Induced subtrees

Let  $S$  be a subset of  $V$ . We denote by  $G[S] = (S, E[S])$  the graph *induced* by  $S$ , where  $E[S] = \{(u, v) \in E \mid u, v \in S\}$ . We call  $G[S]$  an *induced subgraph* of  $G$ . If no confusion, we regard  $S$  as  $G[S]$ .  $|S|$  is the size of  $S$ . We say that  $S$  is an *induced subtree* (see Fig. 1), if  $S$  is a tree. In the following, we state the problem of this paper.

*Problem (Induced subtree enumeration problem).* Enumerate all induced subtrees in  $G = (V, E)$ .



**Fig. 1.** An induced subtree  $S_1$  in  $G_1$ . In the figure, bolded vertices and edges represent vertices and edges in  $S_1$ .  $S_1$  consists of  $\{2, 3, 5, 6, 7\}$ .  $S_1$  is an induced subtree in  $G_1$  since  $S_1$  is connected and acyclic.

### 2.3 $K$ -degenerate graphs

A graph  $G$  is  $k$ -degenerate [4] if any its induced subgraph of  $G$  has a vertex whose degree is less than or equal to  $k$ . The *degeneracy* of  $G$  is defined as the smallest  $k$  satisfying the definition of  $k$ -degenerate graphs. Examples of graph classes with constant degeneracy include trees, grid graphs, outerplanar graphs, and planer graphs, thus degenerate graph is a large class of sparse graphs. These degeneracy are 1, 2, 2, and 5, respectively.

From the definition of  $k$ -degeneracy, we obtain a vertex sequence  $(u_1, \dots, u_{|V|})$  satisfying the condition

$$\forall 1 \leq i \leq |V|, |\{u_j \in N(u_i) \mid i < j \leq |V|\}| \leq k \cdots (\star).$$

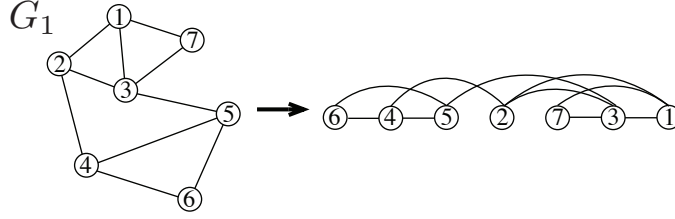
This condition  $(\star)$  implies that there exists an *ordering* among vertices of  $G$  such that for any vertex  $u$ , the number of vertices adjacent to  $u$  larger than it is at most  $k$ . Hereafter we assume that the vertices are indexed in this ordering. We say  $u < v$  ( $u > v$ , respectively) if the index of  $u$  is smaller than  $v$  ( $u$  is larger than  $v$ , respectively) with respect to this ordering. In Fig. 2, we show an example of the ordering satisfying  $(\star)$ . Matula and Beck [5] proposed an algorithm for obtaining the degeneracy of  $G$  and the ordering satisfying  $(\star)$ . By iteratively choosing the smallest degree vertex and removing it from  $G$ , their algorithm finds such an ordering in  $O(|V| + |E|)$  time.

## 3 Basic Binary Partition Algorithm

### 3.1 Candidate Sets and Forbidden Sets

Let  $S$  be an induced subtree of  $G$ . We define the *adjacency* of a vertex  $u \in V$  to  $S$  as  $\text{adj}(S, u) = |S \cap N(u)|$ , that is,  $\text{adj}(S, u)$  is the number of vertices of  $S$  adjacent to  $u$ .

**Lemma 1.** *Let  $S$  be any induced subtree in  $G$  and  $u$  be any vertex  $V \setminus S$ .  $S \cup \{u\}$  is an induced subtree if and only if  $\text{adj}(S, u) = 1$ .*



**Fig. 2.** An example of an ordering of  $G_1 = (V_1, E_1)$ . In the right graph, vertices are sorted by the ordering that satisfies  $(\star)$ .

*Proof.* If  $\text{adj}(S, u) > 1$ ,  $u$  is adjacent to two vertices  $v$  and  $w$  of  $S$ . Since  $S$  has a path  $\pi$  connecting  $v$  and  $w$ , the addition of  $u$  yields a cycle in  $S \cup \{u\}$ . If  $\text{adj}(S, u) = 0$ ,  $S \cup \{u\}$  is disconnected. If  $\text{adj}(S, u) = 1$ ,  $S \cup \{u\}$  is connected. Since the degree of  $u$  in  $G[S \cup \{u\}]$  is one,  $u$  is not included in a cycle. Thus,  $G[S \cup \{u\}]$  does not contain a cycle.  $\square$

In each iteration, we maintain the *forbidden set*  $X$  as the vertex set such that any vertex  $u$  in  $X$  satisfies either  $u$  belongs to  $S$ ,  $S \cup \{u\}$  includes a cycle, or  $u$  is forbidden to include in the solution by some ancestor iterations of the iteration. We also maintain the *candidate set*  $CAND$  as the set of vertices whose additions yield induced subtrees and are not included in  $X$ . We maintain  $CAND$  and  $X$  for efficient computation. From Lemma 1, they are disjoint, and for any vertex  $u$ , if  $\text{adj}(S, u) > 0$ ,  $u$  belongs to either  $CAND$  or  $X$ .

### 3.2 Basic Binary Partition

Our algorithm starts from the empty induced subtree  $S = \emptyset$ . In each iteration given an induced subtree  $S$ , we remove a vertex  $u$  from  $CAND$ , and partition the problem into two; enumeration of all induced subtrees including  $S \cup \{u\}$ , and those including  $S$  but not including  $u$ . We recursively do this partition until there is no vertex in  $CAND$ . The former can be solved by a recursive call with setting  $S$  to  $S \cup \{u\}$ . The latter is solved by a recursive call with setting  $X$  to  $X \cup \{u\}$ . In this way, we can enumerate all induced subtrees. We present the main routine ISE of our algorithm in Algorithm 1. We show how to update candidate sets and forbidden sets in the next two lemmas.

**Lemma 2.** *For an induced subtree  $S$  and a vertex  $u \in CAND$ , when we add  $u$  to  $S$  and remove  $u$  from  $CAND$ ,  $CAND$  changes to*

$$(CAND \setminus N(u)) \cup (N(u) \setminus (CAND \cup X)).$$

*Proof.* Any vertex in  $CAND$  other than  $N(u)$  remains in  $CAND$  after the addition of  $u$  to  $S$  since the adjacencies of the vertices do not change. If vertices in  $N(u) \cap (CAND \cup X)$  are added to  $S \cup \{u\}$ , they are in  $S$ , or they make cycles

---

**Algorithm 1** Main routine ISE: Enumerating all induced subtrees in  $G$ 


---

```

1: procedure ISE( $G = (V, E), S, CAND, X$ )
2:   if  $CAND = \emptyset$  then output  $S$ ; return;
3:   choose the smallest vertex  $u$  from  $CAND$  and remove  $u$  from  $CAND$ ;
4:   call ISE( $G, S, CAND, X \cup \{u\}$ );
5:   call ISE( $G, S \cup \{u\}, (CAND \setminus N(u)) \cup (N(u) \setminus CAND), X \cup \{u\} \cup (CAND \cap N(u))$ );

```

---

since they are adjacent to  $u$  and other vertices in  $S$ . The adjacency of any vertex in  $N(u) \setminus (CAND \cup X)$  is zero for  $S$ , and one for  $S \cup \{u\}$ . Any vertex  $v \notin S$  satisfying  $\text{adj}(S \cup \{u\}, v) = 1$  is either in  $N(u)$  or  $CAND$ . Thus, the statement holds.  $\square$

**Lemma 3.** *For an induced subtree  $S$  and a vertex  $u \in CAND$ , when we add  $u$  to  $S$  and remove  $u$  from  $CAND$ ,  $X$  changes to*

$$X \cup \{u\} \cup (CAND \cap N(u)).$$

*Proof.* Any vertex  $v \in X$  remains in  $X$  for  $S \cup \{u\}$ , since  $\text{adj}(S \cup \{u\}, v) \geq \text{adj}(S, v)$  always holds. From the definition of the forbidden set,  $u$  is in  $X$  for  $S \cup \{u\}$ . Further, any vertex  $v$  in  $CAND \cap N(u)$  makes cycles when they are added to  $S \cup \{u\}$ , since  $\text{adj}(S \cup \{u\}, v) \geq 2$  holds. By adding  $u$  to  $S$ , no other vertex is forbidden to be added, thus the statement holds.  $\square$

**Theorem 1.** *Algorithm ISE enumerates all induced subtrees in the input graph  $G = (V, E)$  without duplicates.*

## 4 Improved Binary Partition Algorithm

From Lemma 2 and Lemma 3, we can easily see that the computation time of updating the candidate set and the forbidden set is  $O(d_G(u))$  by checking all vertices adjacent to  $u$ . However, in this way, we must check some vertices again and again. Specifically, let us assume  $u, v$  are consecutively added to  $S$ , and  $w \notin S$  is adjacent to  $u, v$  and another vertex in  $S$ . When we add  $u$  to  $S$ , we check whether we can add  $w$  to the candidate set of  $S \cup \{u\}$ . After generating  $S \cup \{u\}$ , we check  $w$  again when we add  $v$  to  $S \cup \{u\}$ . In order to avoid this redundant checking, we improve the way of updating the candidate set and the forbidden set by using the following set.

**Definition 1.** *Suppose that  $u$  is a vertex of  $CAND$  for an induced subtree of  $G$ . We define a set  $\Gamma(u, X)$  as follows:*

$$\Gamma(u, X) = \{v \in N(u) \mid v \notin X, v < u\}.$$

**Lemma 4.** *Let  $S$  be an induced subtree of  $G$ ,  $u$  be the smallest in the candidate set  $CAND$  of  $S$ , and  $X$  be the forbidden set of  $S$ . Then, the following formula holds:*

$$N(u) \setminus (CAND \cup X) = (N'(u) \setminus (CAND \cup X)) \cup \Gamma(u, X),$$

where  $N'(u) = \{v \in N(u) \mid u < v\}$ .

*Proof.* Let  $Z$  be the set of vertices larger than  $u$ . Since  $u$  is the smallest vertex in  $CAND$ ,  $(N(u) \setminus (CAND \cup X)) \cap Z = (N'(u) \setminus (CAND \cup X))$ . From the definition of  $\Gamma(u, X)$  and  $u$  is the smallest in  $CAND$ ,  $(N(u) \setminus (CAND \cup X)) \cap (V \setminus Z) = N''(u) \setminus (CAND \cup X) = (N''(u) \setminus CAND) \cap (N''(u) \setminus X) = \Gamma(u, X)$ , where  $N''(u) = \{v \in N(u) \mid v < u\}$ . This concludes the lemma.  $\square$

In what follows, we use an adjacency lists for the sets  $CAND$ ,  $X$ , and  $\Gamma$ , so that a removal and the recover of the removed element can be done in  $O(1)$  time, and the merge of two sets can be done in linear time of their sizes.

**Lemma 5.** *When we add a vertex  $u$  to  $X$ , the update of  $\Gamma(v, X)$  for all vertices  $v$  is done in  $O(k)$  time.*

*Proof.* To update, it is suffice to remove  $u$  from  $\Gamma(v, X)$  from all  $v > u$ . Thus, it takes  $O(k)$  time.  $\square$

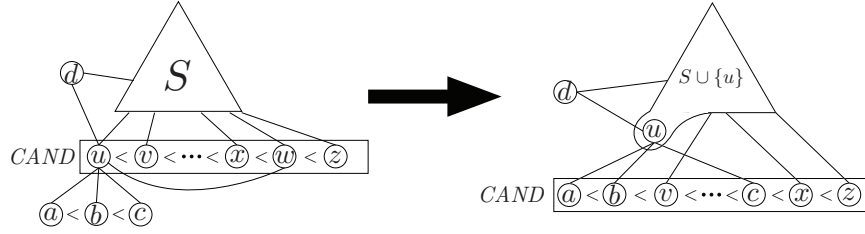
**Lemma 6.** *Let  $S$  be an induced subtree of  $G$ ,  $u$  be the smallest in the candidate set  $CAND$  of  $S$ , and  $X$  be the forbidden set of  $S$ . When we add  $u$  to  $S$  and remove  $u$  from  $CAND$ , the computation time of updating  $CAND$  and  $X$  are  $O(k + |\Gamma(u, X)|)$  and  $O(k)$  time, respectively.*

*Proof.* Since  $u$  is the smallest vertex in  $CAND$ ,  $|\Delta| \leq k$ , where  $\Delta = |CAND \cap N(u)|$ . Since vertices in  $N(u)$  are sorted by the ordering, the computation time of  $\Delta$  is  $O(k)$ . Thus, adding vertices in  $\Delta$  and  $u$  to  $X$  and removing  $\Delta$  from  $CAND$  are done in  $O(k)$  time. From Lemma 4, since  $|\{v \in N(u) \mid u < v\}| \leq k$ , the computation time of adding these vertex to  $CAND$  is  $O(k + |\Gamma(u, X)|)$ . Hence, the lemma holds.  $\square$

In Fig. 3, we show the changes of between the candidate set of  $S$  and that of  $S \cup \{u\}$  after adding  $u$  to  $S$ . We implement  $CAND$  and  $X$  by doubly linked lists. Thanks to the doubly linked list, the cost for a deletion and a recover of a vertex can be done in constant time.

**Theorem 2.** *Let  $G = (V, E)$  be the input graph and  $k$  is the degeneracy of  $G$ . Our algorithm enumerates all induced subtrees in  $G$  in  $O(k)$  time per solution after  $O(|V| + |E|)$  preprocessing time without duplicates using  $O(|V| + |E|)$  space.*

*Proof.* Since the update of  $CAND$  and  $X$  is correct, the correctness of the algorithm is obvious. (I) We discuss the time complexity of the preprocessing. First, our algorithm computes an ordering of vertices by Matula and Beck's algorithm [5] in  $O(|V| + |E|)$  time. Next, our algorithm sorts vertices belonging



**Fig. 3.** This figure shows the changes between candidate set  $CAND$  by the addition of  $u$  to  $S$ .  $S$  is an induced subtree and  $\{u, v, \dots, x, w, z\}$  is the candidate set of  $S$ . Let assume that  $a < b < u < c$  and  $d < u$ . Since  $d$  does not belongs to  $\Gamma(u, X)$ ,  $d$  is skipped checking.

to each adjacency list by using a bucket sort. Thus, the preprocessing time is  $O(|V| + |E|)$ .

(II) We consider an iteration inputting  $S$ ,  $X$ , and  $CAND$ , and assume that  $CAND'$  is the candidate set for  $S \cup \{u\}$ . Line 2 and line 3 run in  $O(1)$  time. From Lemma 5, line 4 needs  $O(k)$  time. From Lemma 6, since it is clear that  $|\Gamma(u, X)| < |CAND'|$ , our algorithm needs  $O(k + |CAND'|)$  time for computing  $CAND'$  and  $X$ . The update of  $\Gamma$ 's is done in  $O(k|CAND \cap N(u)|)$  time, from Lemma 5. We observe that for each vertex  $w$  such that  $v \in (N(u) \cap CAND)$  is removed from  $\Gamma(w, X)$ ,  $w$  is in  $CAND$  of  $S \cup \{v\}$ , that will be generated by a descendant of this iteration. We charge the cost of constant time to remove  $v$  from  $\Gamma(w, X)$  to the induced subtree  $S \cup \{v, w\}$ . Then, we can see that  $S \cup \{v, w\}$  is charged only from iterations inputting  $S$ , that divides the problem by  $u'$  such that  $(u', v) \in E$ , that is, the iteration generates  $S \cup \{u'\}$ . We consider the average amount of the charge over all induced subtrees of  $S \cup \{v, w\}$ ,  $v \in CAND$ , and  $w$  is in  $CAND$  of  $S \cup \{v\}$ . Since the number of pairs  $\{u, v\} \subseteq CAND$  is at most  $k|CAND|$ , we can see the average charge is  $O(k)$  for each  $S \cup \{v, w\}$ . Thus, in summary, we can see the update time for  $\Gamma$  in an iteration is bounded by  $O(k)$ , on average. Thus, an iteration takes  $O(k + k|CAND'|)$  time on average. We observe that the sum of  $|CAND'|$  over all iterations is no greater than the sum of  $|CAND|$  over all induced subtrees, since  $CAND'$  is the candidate set of  $S \cup \{u\}$  and forbidden set  $X \cup \{u\}$ , and  $S \cup \{u\}$  is generated only from  $S$ . Further, we can see that  $S \cup \{u\}$  is generated only from  $S$  this iteration. Hence, thus the sum of  $|CAND|$  over all induced subtrees is bounded by the number of induced subtrees. Therefore, the computation time for each iteration is bounded by  $O(k)$  on average.

In a binary partition algorithm, each iteration at the leaf of the recursion outputs a solution, and each non-leaf iteration generates exactly two recursive calls. Thus, the number of iterations (recursive calls) of a binary partition algorithm is at most  $2N$ . Hence, the computation time per induced subtree is  $O(k)$ . All all sets the algorithm maintains are of size  $O(|V| + |E|)$  in total.



We need a bit care to perform a recursive call. When a recursive call is made, we record the operations to prepare the parameters given to the recursive call on the memory. When the recursive call ends, we apply the inverse operations of the recorded operations to recover the variables such as *CAND* and *X*. In this way, we can recover the variables from the updated ones without increasing the time complexity. Since no vertex is added or deleted from the same variable twice, the accumulated space for the recorded operations is bounded by  $O(|V| + |E|)$ . From the above arguments, our algorithm runs in  $O(k)$  time per solution after  $O(|V| + |E|)$  preprocessing time using  $O(|V| + |E|)$  space.  $\square$

## 5 Conclusion

In this paper, we have presented an algorithm for enumerating all induced subtrees in  $k$ -degenerate graph. Our algorithm runs in  $O(k)$  time per solution after linear preprocessing time using linear space. From this result, we obtain the following corollary; if the input graph has a constant degeneracy, our algorithm is optimal with respect to the computation time per solution.  $K$ -degenerate graphs often appear in real-world data even when with much noise. Thus considering the applications, it is important to study on efficient computation on  $k$ -degeneracy. This result is one of the first steps for such studies, and researches on enumeration algorithms on  $k$ -degenerate graphs will be an important issue.

## Acknowledgement

This work was partially supported by MEXT Grant-in-Aid for Scientific Research (A) 24240021 and Grant-in-Aid for JSPS Fellows 25 · 1149.

## References

1. Avis, D. and Fukuda, K.: Reverse search for enumeration. *DAM*, 65:21–46, 1996.
2. Birmelé, E., Ferreira R. A., Grossi R., Marino A., Pisanti N., Rizzi R., and Sacomoto G.: Optimal Listing of Cycles and st-Paths in Undirected Graphs. In *Proc. SODA 2013*, 1884–1896, 2013.
3. Ferreira, R. , Grossi, R. , and Rizzi, R.: Output-sensitive listing of bounded-size trees in undirected graphs. In *Proc. ESA 2011*, LNCS 6942, 275–286, 2011.
4. Lick, D. R. and White, A. T:  $k$ -DEGENERATE GRAPHS. *Can. J. Math.*, XXII(5): 1082–1096, 1970.
5. Matula, D. W. and Beck, L. L.: Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3): 417–427, 1983.
6. Shioura, A., Tamura, A., and Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.
7. Tarjan, R. E.: Enumeration of the Elementary Circuits of a Directed Graph. *SIAM J. Comput.*, 2(3): 211–216, 1973.
8. Tarjan, R. E. and Read, R. C. .: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

9. Uno, T.: An output linear time algorithm for enumerating chordless cycles. *Technical Notes, 92nd SIGAL of IPSJ*, pages 47–53, 2003. In Japanese.
10. Wasa, K., Kaneta, Y., Uno, T., and Arimura, H.: Constant time enumeration of bounded-size subtrees in trees and its application. In *Proc. COCOON 2012*, LNCS 7434, 347–359, 2012.