An Efficient Algorithm for Enumerating Induced Subgraphs with Bounded Degeneracy

Kunihiro Wasa(✉) and Takeaki Uno

National Institute of Informatics, Tokyo, Japan
{wasa,uno}@nii.ac.jp

Abstract. We propose a polynomial delay and polynomial space algorithm for the enumeration of k-degenerate induced subgraphs in a given graph. A graph G is k-degenerate if each of its induced subgraphs has a vertex of degree at most k. The degeneracy is considered as an indicator of the sparseness of the graph. Real-world graphs such as road networks, social networks and internet networks often have small degeneracy. Compared to other kinds of graph classes, bounded degeneracy does not give many structural properties such as induced subgraph free, or minor free. From this, using bounded degeneracy to reduce the time complexity is often not trivial. In this paper, we investigate ways of handling the degeneracy and propose an efficient algorithm for the k-degenerate induced subgraph enumeration. The time complexity is $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$ time per solution with polynomial preprocessing time and the space complexity is linear in the input graph size, where Δ and k' are the maximum degree and the degeneracy of the input graph.

Keywords: Graph algorithms · Enumeration algorithms
Polynomial delay · k-degenerate graphs

1 Introduction

The *subgraph enumeration* is to output all the subgraphs of the given graph satisfying a certain structural condition such as being a tree and the density is no less than a threshold. It is one of the fundamental problems widely studied in the theoretical computer science for more than 40 years ([12] gives the overview of this area). The complexity analysis of subgraph enumeration algorithms has two main streams. One is of a usual style, that is evaluating the time complexity in the input size. Generally speaking, the number of solutions is exponential in the input size, thus the studies are done to reduce the constant factor c of the time complexity of $\mathcal{O}(c^n)$. This has an advantage that at the same time we can often obtain a combinatorial result of bounding the number of subgraphs. On the other hand, considering the practice, enumeration problems have less

This work was supported by JST CREST, Grant Number JPMJCR1401, Japan.

D. Kim et al. (Eds.): COCOA 2018, LNCS 11346, pp. 35–45, 2018.
https://doi.org/10.1007/978-3-030-04651-4_3

solutions, say polynomially many, thus evaluation only by the input size is too much overestimating. In such cases, output polynomial time is considered. The time complexity is evaluated by the input size n and the output size N that is the number of solutions. An algorithm is called an *output polynomial time algorithm* if it terminates in a time polynomial in the input size and the output size. Further, we say it runs in $poly(n)$ time for each solution if an algorithm runs in $\mathcal{O}(poly(n)N)$ time plus polynomial time preprocessing. Further, we say the algorithm runs in *polynomial delay* time if the computation time between any two consecutive output solutions is polynomial in the input size.

A graph G is said to be *k-degenerate* [10] if any of its induced subgraphs has at least one vertex of degree at most k. The *degeneracy* of G is the smallest k such that G is k-degenerate. Intuitively speaking, if a graph has a small degeneracy, then the graph is relatively sparse. It is said that real world graphs often have small degeneracies [6,8]. Particularly, some graph classes have constant degeneracies, e.g., trees, grid graphs, outer planar graphs, planar graphs, bounded treewidth graphs, and H-minor-free graphs for fixed H. Bounded-size degenerate graphs have received much attention. However, bounded degeneracy does not give many structural properties such as minor free, compared to other graph classes such as chordal graphs. Thus, despite its importance, there are not so many studies on algorithms that utilize degeneracy. In this paper, we address the problem of enumerating all induced k-degenerate subgraphs. We investigate efficient search strategies that have characterization of irredundant moves, and the way of checking the degeneracy in short time. These yield an efficient enumeration algorithm. There have been several studies on enumeration problems in a bounded degenerate graph [4,6,9], for cliques, dominating sets, induced trees, etc. To the best of our knowledge, there has been little research on the enumeration of k-degenerate subgraphs, or k-degenerate induced subgraphs. The case of $k = 1$ corresponds to the forest enumeration, thus there have been many studies, especially its connected version, that is, trees. Ferreira *et al.* proposed an enumeration algorithm for subtrees of size exactly h in an undirected graph [7]. Wasa *et al.* improved their result to optimal [14], that is, their algorithm runs in $\mathcal{O}(1)$ delay, when the input graphs are trees. For the induced version, the authors proposed an algorithm [13] that enumerates all the vertex subsets that induce a tree. Conte *et al.* proposed an enumeration algorithm for maximal k-degenerate induced subgraphs in a chordal graph that runs in polynomial delay and space [5]. On the other hand, Bauer *et al.* [2] studied the enumeration of all k-degenerate induced graphs having n vertices and m edges.

We propose an efficient algorithm for enumerating k-degenerate induced subgraphs in a given graph $G = (V, E)$. Our algorithm runs in $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$ time per solution and uses $\mathcal{O}(|V| + |E|)$ space, where Δ and k' are the maximum degree and the degeneracy of an input graph. Note that the algorithm outputs also disconnected subgraphs. In the enumeration of k-degenerate subgraphs, the time consuming part is the computation of the degeneracy of a newly generated subgraph. It takes $\mathcal{O}(|V| + |E|)$ time with straightforward ways. Further, if we follow the usual binary partition algorithm,

an iteration has to have up to $\mathcal{O}(|V|)$ trials for finding a vertex whose addition to the current solution yields a k-degenerate induced subgraph. Thus, an iteration has to spend $\mathcal{O}(|V|(|V|+|E|))$ time or more in this way. Instead of that, we developed a *reverse search* algorithm [1]. We define a *parent-child relationship* that defines a tree shaped traversal route spanning all k-degenerate induced subgraphs. By generating the child solutions of the current visiting solution recursively, we can traverse the tree in a depth-first manner whilst using polynomial time and polynomial space. We further developed a data structure that enables us to check the degeneracy in short time, that can be updated quickly along the movement on the traversal tree.

The organization of this paper is as follows: In Sect. 2, we give the basic notations and terminologies that are used in this paper. Section 3 shows the reverse search strategy, and Sect. 4 describes the data structure for checking the degeneracy. In Sect. 5, we conclude this paper.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$. Let $u, v \in V$ be two distinct vertices in G. Vertices u and v are mutually *adjacent* if $(u, v) \in E$. $N(u)$ denotes the set of vertices that are adjacent to u. We call $d(u) = |N(u)|$ the *degree* of u. Let $\Delta(G) = \max_{u \in V} d(u)$ and $\delta(G) = \min_{u \in V} d(u)$. For a vertex set S, let $d_S(u) = |N(u) \cap S|$ be the number of neighbors of u that are in S. For any edge $e = (u, v) \in E$, we say that u and v are the *endpoints* of e. G is *connected* if any pair of vertices u and v, there is a path between them. For any vertex subset S of V, the subgraph induced by S is the graph whose vertex set is S and edge set is the edges connecting two vertices in S, i.e., $\{(u, v) \in E \mid u, v \in S\}$. For conciseness, for a set S and an element v, we denote $S \cup \{v\}$ by $S \cup v$ and $S \setminus \{v\}$ by $S \setminus v$. In what follows, we assume that G is connected and G has no self loops and multi edges.

2.1 k-degenerate Graphs

A graph is *k-degenerate* [10] if any of its induced subgraphs has a vertex of degree at most k, and the *degeneracy* of G is the smallest k such that G is k-

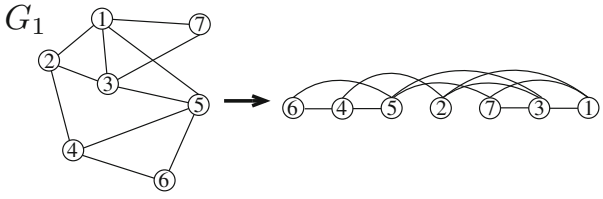

Fig. 1. G_1 is a 2-degenerate graph. A degeneracy ordering of G_1 is shown in the right part of the figure. In the figure, the leftmost vertex 6 is the smallest and the rightmost vertex 1 is the largest. For any vertex v in G_1, the number of larger adjacent vertices of v is at two.

degenerate. We call a vertex set $S \subseteq V$ a *k-degenerate vertex set* if $G[S]$ is an induced subgraph whose degeneracy is k. It is known that G is k-degenerate if and only if it admits a vertex ordering $>^*$, called a *degeneracy ordering*, such that for any vertex v in G, $|\{u \in N(v) \mid u >^* v\}| \leq k$ (See Fig. 1). That is, for each vertex, the number of larger neighbors is at most k. A degeneracy ordering of a graph can be obtained in linear time in the size of a graph [11], by recursively removing the smallest degree vertex from the graph. It is also known that graphs in some graph classes have a constant degeneracy. For example, the degeneracy of trees, grid graphs, outerplanar graphs [3], and planar graphs are respectively at most 1, 2, 2, and 5 [10]. In what follows, we assume the vertices of G are labeled according to some degeneracy ordering of G, and we write $>$ for $>^*$ if no confusion can arise.

We here describe the k-degenerate induced subgraphs enumeration problem.

Problem 1. Given a graph $G = (V, E)$ and positive integer k, enumerate all k-degenerate vertex subsets of G.

Note that those vertex sets may induce disconnected graphs, and are allowed to be solutions.

3 Reverse Search Algorithm

Our algorithm enumerates the solutions by traversing on a tree structure on the solution space, called *a family tree*, in a DFS manner. The basic idea of this strategy is proposed by Avis and Fukuda [1]. In this section, we give an algorithm for k-degenerate induced subgraphs. We first define the tree structure mentioned above. Let $R = (\emptyset, \emptyset)$ be the empty graph, that is a k-degenerate graph. We call R the *root*. The *parent vertex* $pv(S)$ of a k-degenerate vertex set S is defined as the smallest vertex v in S such that v is adjacent to at most k vertices of S. We define the parent of k-degenerate induced subgraphs of G as follows:

Definition 1 (The parent). Let G be a graph and k be a positive integer. We define the *parent* $P(S)$ of a non-empty k-degenerate vertex set $S \subseteq V$ as $S \setminus pv(S)$.

For any k-degenerate vertex set S', we say that S' is a *child* of S if $P(S') = S$. Let $\mathcal{C}_k(S)$ be the set of children of S. In what follows, we omit the subscript k of $\mathcal{C}_k$ if no confusion arises. A vertex u is called a *child generator* of S if $S \cup u$ is a child of S. It holds that $|P(S)| = |S| - 1$, thus by repeatedly applying the parent function, we can reach the root R from any k-degenerate vertex set S since every k-degenerate vertex set has a parent vertex $pv(S)$.

The *family tree* is a tree whose node set is all the k-degenerate vertex sets of G, and an edge connect two nodes if one is the parent of the other. From the above, we can see that the family tree contains no cycle and spans all k-degenerate vertex subsets, thereby is a tree rooted at R. The algorithm traverses

Algorithm 1. Reverse search algorithm

```
1 Procedure Main(G = (V, E), k)
2 |   R ← (∅, ∅);
3 |   Rec(G, R, k);
4 Subprocedure Rec(G, S, k)
5 |   Output S;
6 |   foreach S' ∈ C(S) do
7 |   |   Rec(G, S', k);
```

the tree by recursively moving to the children of the current visiting k-degenerate vertex subset. We can then see the correctness of the algorithm; the algorithm completely outputs the k-degenerate vertex sets without duplication. The algorithm is described in Algorithm 1.

4 Generating Children

The bottle neck part of Algorithm 1 is the computation of the children a current solution in line 6. A naïve way to generate all the children of a k-degenerate vertex set S is as follows: For every vertex $u \notin S$, check whether $S \cup u$ is k-degenerate or not, and $P(S \cup u)$ is S or not; if the answers of the both checking are yes, then add u to the set of child generators. This actually needs $\mathcal{O}(|V|(|V| + |E|))$ time. In this section, we propose an efficient method that avoids the above trial-and-error approach. We first consider the sufficient and necessary condition for a vertex u to be a child generator of S. Let k' be the degeneracy of G. In what follows, we assume $k < k'$. Otherwise, our problem can be solved in $\mathcal{O}(1)$ time per solution outputting all subgraphs in G. Let $N_>(u)$ be the set of vertices that are adjacent to u and larger than u. We assume that the graph is stored in the memory by the adjacency lists of its vertices, and the adjacency list is sorted in the degeneracy ordering. $N_>(u)$ is also stored in the memory and sorted in the degeneracy ordering.

4.1 Characterization of a Child Generator

We say that v is *black* if $d_S(v) > k$, v is *gray* if $d_S(v) = k$, and v is *white* otherwise. We also say that v is *non-black* if v is not black, that is, v is gray or white. Note that $pv(S)$ is the smallest non-black vertex in S. Let $sw_*(S)$ be the smallest white vertex in S, and $GS(S)$ be the set of the gray vertices in S that are smaller than $sw_*(S)$. Let $h_S(u)$ be the number of vertices in $GS(S)$ that are smaller than u, and $h'_S(u)$ be the number of vertices in $GS(S) \cap N(u)$ that are smaller than u. A vertex u is a child generator of S if and only if $u = pv(S \cup u)$ holds. This condition is characterized as follows.

Lemma 1. *A vertex $u \notin S$ is a child generator of a k-degenerate vertex subset S, i.e., $u = pv(S \cup u)$ if and only if u is non-black in $S \cup \{u\}$ and either one of the followings holds.*

(1) $u < pv(S)$
(2) $pv(S) < u < sw_*(S)$ *and* u *is connected to all vertices in* $\{v \in GS(S) \mid v < u\}$, *i.e.,* $h_S(u) = h'_S(u)$.

Proof. Suppose that S is a k-degenerate vertex subset and $u \notin S$ is a non-black vertex. Note that $S \cup u$ is k-degenerate. We consider the following three cases.

Case (A): Suppose that (1) holds. We observe that (i) any black vertex in S is also black in $S \cup u$, and (ii) $u < u'$ holds for any non-black vertex $u' \in S$ because $u < pv(S)$. It implies that $u = pv(S \cup \{u\})$.

Case (B): Suppose that (2) holds. Any non-black vertex u' of $S \cup u$ is either gray or white in S. If u' is white in S, then $u < u'$ holds from the condition of (2). If u' is gray in S, then u' is not connected to u. Thus, from the condition of (2), $u < u'$ holds. Therefore, $u < u'$ always holds, and $u = pv(S \cup \{u\})$.

Case (C): Both (1) and (2) do not hold implies that (i) $u > pv(S)$ and (ii) $u > sw_*(S)$ or u is not connected to a gray vertex $v \in GS(S)$ such that $v < u$. If $u > sw_*(S)$ holds, then $pv(S \cup \{u\}) \leq sw_*(S) < u$ holds since $sw_*(S)$ increases its degree by at most one in $S \cup u$, and thereby it is non-black. If u is not connected to a gray vertex $v \in S$ smaller than u, then $pv(S \cup \{u\}) \leq v < u$ holds since v is a non-black vertex of $S \cup \{u\}$. Hence, the statement holds. □

4.2 Data Structure for Finding Child Generators

Lemma 1 shows that we can find child generators by finding the non-black vertices satisfying the condition (1) or (2). Let $\mathcal{L}(S)$ (resp., $\mathcal{L}'(S)$) be the list of all the non-black vertices not in S (resp., in S) that are sorted by the degeneracy ordering. The non-black vertices satisfying (1) are efficiently found by tracing $\mathcal{L}(S)$ from the head. All vertices u in $\mathcal{L}(S)$ that satisfy $u < pv(S)$ are actually child generators of S. Hence, it takes $\mathcal{O}(1)$ time for each child generator. From above discussion, we can immediately obtain the following lema:

Lemma 2. *For any k-degenerate vertex subset S, the child generators smaller than $pv(S)$ are found in $\mathcal{O}(|\mathcal{C}(S)| + 1)$ time by using $\mathcal{L}(S)$.*

For finding those satisfying (2), we construct the list A composed of the first k vertices of $GS(S)$ by tracing the first k elements of $\mathcal{L}'(S)$. By using A, we compute $h_S(u)$ and $h'_S(u)$. We observe that any vertex larger than the $(k+1)$st vertex of $GS(S)$ cannot satisfy (2) since to satisfy (2) the vertex has to be adjacent to at least $k+1$ vertices of S that means the vertex is not a child generator. We then trace $N_>(v)$ for all vertices v in A. While tracing these larger neighbors, for all such neighbors u, we compute $h'_S(u)$. We first initialize $h'_S(u)$ to zero, and then increase the value by one while tracing. Since the total number of the neighbors is at most kk', this takes $\mathcal{O}(kk')$ time. For any $v \in A$, the computation of $h'_S(u)$ for all vertices u in $N_>(v)$ can be done in $\mathcal{O}(kk')$ time in total by tracing $N_>(v)$ and A, simultaneously. Thus, the computation of $h_S(u)$ and $h'_S(u)$ for all vertices u that are adjacent to at least one vertex of A is done in $\mathcal{O}(kk')$ time. Therefore, child generators satisfying (2) can be found in $\mathcal{O}(kk')$ time.

Lemma 3. *For any k-degenerate vertex subset S, the child generators larger than $pv(S)$ are found in $\mathcal{O}(kk')$ time by using $\mathcal{L}'(S)$.*

4.3 Efficiently Updating the Data Structure

The key to efficient computation in a recursive call, called an *iteration*, is the efficiency of the update process of the data structure described above. Suppose that u is a child generator of S and we are going to compute $\mathcal{L}(S \cup u)$ and $\mathcal{L}'(S \cup u)$ from $\mathcal{L}(S)$ and $\mathcal{L}'(S)$.

Lemma 4. *For any k-degenerate vertex subset S, $\mathcal{L}'(S \cup \{u\})$ is obtained from $\mathcal{L}'(S)$ in $\mathcal{O}(k)$ time, where $S \cup \{u\}$ is a child of S.*

Proof. Let denote by $S' = S \cup \{u\}$. To obtain $\mathcal{L}'(S')$, we compute $d_{S'}(v)$ from $d_S(v)$ for each v in $\mathcal{L}'(S)$ that is adjacent to u. Since $pv(S') = u$, all vertices in $\mathcal{L}'(S') \setminus \{u\}$ is larger than u. Hence, if a vertex v in $\mathcal{L}'(S)$ is still in $\mathcal{L}'(S')$, $u < v$ and $d_{S'}(v) \leq k$. Thus, we first remove all vertices v from $\mathcal{L}'(S)$ that satisfies $d_{S \cup u}(v) > k$. Since the number of such vertices v is at most k, this needs $\mathcal{O}(k)$ time. Finally, we insert u to the head of $\mathcal{L}'(S)$, and then we obtain $\mathcal{L}'(S \cup u)$. Hence, the statement holds. □

The computation of $\mathcal{L}(S \cup u)$ is done in the same way as the above, in $\mathcal{O}(\Delta)$ time.

Lemma 5. *For any k-degenerate vertex subset S, $\mathcal{L}(S \cup u)$ is obtained from $\mathcal{L}(S)$ in $\mathcal{O}(\Delta)$ time.*

From Lemmas 4 and 5, the computation of $\mathcal{L}'(S)$ and $\mathcal{L}(S)$ needs $\mathcal{O}(\Delta)$ time since $k < \Delta$. By recording the operations of these update, we can easily restore $\mathcal{L}'(S')$ and $\mathcal{L}(S')$ from $\mathcal{L}'(S)$ and $\mathcal{L}(S)$ in $\mathcal{O}(\Delta)$ time.

When Δ is large and k' is small, the following algorithm for updating $\mathcal{L}(S)$ is more efficient. The algorithm deals with the former and latter parts of $\mathcal{L}(S \cup u)$, where the former part is of vertices smaller than u and the latter part is of the other.

Lemma 6. *For any k-degenerate vertex subset S, the part of $\mathcal{L}(S \cup u)$ composed of vertices larger than u is obtained from $\mathcal{L}(S)$ in $\mathcal{O}(k')$ time.*

Proof. Removed vertices from the latter part are adjacent to u. Since G is k'-degenerate, the number of removed vertices is at most k'. Hence, by checking k' larger neighbors of u, the latter part is updated in $\mathcal{O}(k')$ time. □

For the update of the former part, we prepare another data structure called an *island*. An island of a vertex w is a maximal segment of $\mathcal{L}(S)$ composed only of gray vertices v such that $w \in N_>(v)$ and $v \notin S$. An island is stored in the memory by a doubly linked cyclic list of the segment in that the head and the tail of the list are linked (See Fig. 2).

Lemma 7. *For any k-degenerate vertex subset S, the accumulated size of the islands of all vertices in G is $\mathcal{O}(|V| + |E|)$.*

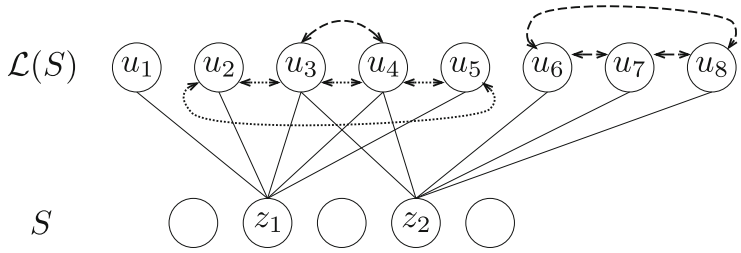

Fig. 2. Example of islands used in the update algorithm. Suppose that $|S \cap N(u_1)| < k$ and $|S \cap N(u_i)| = k$ for $i = 2, \ldots, 8$. Dotted arrows and dashed arrows imply the island of z_1 and z_2, respectively. The island of z_2 consists of two connected doubly linked lists.

Proof. Since each vertex u belongs to at most $|N(u)|$ islands, the sum of the members of islands of all vertices in G is at most $|E|$. Since the doubly cyclic linked list representing an island requires memory linear in the number of its members, the statement holds. □

We remove the vertices from $\mathcal{L}(S)$ that are not in $\mathcal{L}(S \cup u)$ by tracing $\mathcal{L}(S)$ from its head. When we encounter a vertex to be removed, that is, gray in S and adjacent to u, the vertex is the head of an island of u. We then go to its tail by using the cyclic link, and remove the island from the list by cutting off its head and tail from $\mathcal{L}(S)$. We can find the tail of a head in constant time by an array of at most k' pointers. That is, we can remove the island from the list in constant time. Note that the vertices in the island are all adjacent to u, and thus become black by $S \cup u$. In this way, we can update the former part of $\mathcal{L}(S)$ in time linear in the number of vertices of $\mathcal{L}(S \cup u)$ that are smaller than u. Since $u = pv(S \cup u)$, the computation time is $\mathcal{O}(|\mathcal{C}(S \cup u)| + 1)$. Hence, the following lema holds.

Lemma 8. *For any k-degenerate vertex subset S, the part of $\mathcal{L}(S \cup u)$ composed of vertices smaller than u is obtained from $\mathcal{L}(S)$ in $\mathcal{O}(|\mathcal{C}(S \cup u)| + 1)$ time.*

The computation of $d_{S \cup u}(v)$ for each vertex v of the former part of $\mathcal{L}(S \cup u)$ is done by tracing $N_>(v)$ to check whether u is adjacent to v or not. This is done in $\mathcal{O}(k')$ time per vertex, and thus computation for all the vertices in former part is done in $\mathcal{O}(k'(|\mathcal{C}(S \cup u)| + 1))$ time.

Lemma 9. *For any k-degenerate vertex subset S, $d_{S \cup u}(v)$ for all vertices v in $\mathcal{L}(S \cup u)$ are obtained from $N(v) \cap S$ in $\mathcal{O}(k + k'(|\mathcal{C}(S \cup u)| + 1))$ time.*

Lemma 10. *For any k-degenerate vertex subset S, all the islands in the part of $\mathcal{L}(S \cup u)$ composed of vertices smaller than u are constructed in $\mathcal{O}(k'(|\mathcal{C}(S \cup u)| + 1))$ time by using $\mathcal{L}(S \cup u)$.*

Proof. All the islands in the part of $\mathcal{L}(S \cup u)$ composed of vertices smaller than u are built from the scratch by tracing $\mathcal{L}(S \cup u)$. This is done by tracing all vertices in $N_>(v)$ for all vertices in $\mathcal{L}(S \cup u)$, thus is done in $\mathcal{O}(k'(|\mathcal{C}(S \cup u)| + 1))$ time. □

Algorithm 2. Island update algorithm

```
1  Procedure UpdateIsland(S, u)
2      foreach v ∈ N(u) ∩ L(S ∪ u) do
3          if d_{S∪u}(v) = k then
4              foreach Island I including the smaller neighbor of v on L(S ∪ u) do
5                  Put together I and v into a new island I' = I + v ;
6                  Link the head of I and v;
7              foreach Island J including the larger neighbor of v on L(S ∪ u) do
8                  Put together v and J into a new island J' = v + J ;
9                  Link v and the tail of I;
10             foreach Pair of islands I' and J' including v of the same vertex do
11                 Put together I' and J' into a new island I'' = I' + J';
12                 Link the head of I' and the tail of J';
13     return island(S ∪ u);
```

Lemma 11. *For any k-degenerate vertex subset S, all the islands in the part of $\mathcal{L}(S \cup u)$ composed of vertices larger than u are obtained from the islands of $\mathcal{L}(S)$ in $\mathcal{O}\left((k')^2\right)$ time.*

Proof. An island will change by the addition of u to S when a vertex of the island or a vertex neighboring to its head or its tail becomes gray, or becomes black. We observe that at most k' vertices larger than u become gray or black by adding u to S. By the change of a vertex v, at most $3k'$ islands, that include v or a vertex neighboring to v in $\mathcal{L}(S)$, will change. Concatenating two islands, splitting an island and appending a vertex to an island are all done in $\mathcal{O}(1)$ time, thus the update of the islands in the part of $\mathcal{L}(S \cup u)$ composed of vertices larger than u are obtained from the islands of $\mathcal{L}(S)$ in $\mathcal{O}\left((k')^2\right)$ time. □

Algorithm 2 shows the pseudo code for concatenating islands or appending a vertex to an island. We can also implement splitting an island in a similar way. We consider that the update of data structure is done in the iteration with respect to $S \cup u$, as an initialization. Then, an iteration of the algorithm takes $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\} + k'|\mathcal{C}(S \cup u)|\right)$ time.

Theorem 1. *The vertex subsets of a graph $G = (V, E)$ inducing k-degenerate graphs can be enumerated in $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$ time for each solution with $\mathcal{O}(|V| + |E|)$ space and $\mathcal{O}(|V| + |E|)$ preprocessing time, where Δ and k' denote the maximum degree and the degeneracy of G.*

Proof. An iteration of our algorithm takes $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\} + k'|\mathcal{C}(S)|\right)$ time. By assigning $\mathcal{O}(k')$ to each child, it will be $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$. Our algorithm outputs a solution in each iteration. It implies that the computation time is $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$ time for each solution. In the preprocessing phase, the algorithm needs to sort the vertices in the degeneracy ordering.

This takes $\mathcal{O}(|E| + |V|)$ time. We also sort the adjacency list of each vertex in $\mathcal{O}(|V| + |E|)$ time by using bucket sort. The sizes of $\mathcal{L}(S)$, $\mathcal{L}'(S)$ and the memory for remembering $d_S(\cdot)$ are all $\mathcal{O}(|V|)$. Since the total size of all islands is $\mathcal{O}(|E|)$, the statement holds. □

Corollary 1. *Let $G = (V, E)$ be a graph with constant degeneracy and k be a positive integer. Then, all k-degenerate induced subgraphs in G can be enumerated in constant time for each solution with $\mathcal{O}(|V| + |E|)$ space and $\mathcal{O}(|V| + |E|)$ preprocessing time.*

5 Conclusion

In this paper, we addressed the k-degenerate induced subgraph enumeration problem. As the main result, we proposed an efficient enumeration algorithm that runs in $\mathcal{O}\left(\min\left\{\Delta + kk', (k')^2\right\}\right)$ time per solution with polynomial preprocessing time and linear space. In this paper, we did not consider the connectivity since when we consider it, the parent-child relation in this paper does not work. Investigating other efficient enumeration strategies for connected k-degenerate subgraph is an interesting future research. The variant of the problem, in some graph classes, non-induced version and fixed size are also interesting.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. Discret. Appl. Math. **65**(1–3), 21–46 (1996). https://doi.org/10.1016/0166-218X(95)00026-N
2. Bauer, R., Krug, M., Wagner, D.: Enumerating and generating labeled k-degenerate Graphs. In: ANALCO 2010, pp. 90–98. Society for Industrial and Applied Mathematics (2010). https://doi.org/10.1137/1.9781611973006.12
3. Chartrand, G., Harary, F.: Planar permutation graphs. Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques **3**(4), 433–438 (1967)
4. Conte, A., Grossi, R., Marino, A., Versari, L.: Sublinear-space bounded-delay enumeration for massive network analytics: maximal cliques. In: ICALP 2016, pp. 148:1–148:15 (2016). https://doi.org/10.4230/LIPIcs.ICALP.2016.148
5. Conte, A., Kanté, M.M., Otachi, Y., Uno, T., Wasa, K.: Efficient enumeration of maximal k-degenerate subgraphs in a chordal graph. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 150–161. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_13
6. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 364–375. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20662-7_31
7. Ferreira, R., Grossi, R., Rizzi, R.: Output-sensitive listing of bounded-size trees in undirected graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23719-5_24
8. Goel, G., Gustedt, J.: Bounded arboricity to determine the local structure of sparse graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 159–167. Springer, Heidelberg (2006). https://doi.org/10.1007/11917496_15

9. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of minimal dominating sets and variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22953-4_26
10. Lick, D.R., White, A.T.: k-degenerate graphs. Can. J. Math. **22**(5), 1082–1096 (1970). https://doi.org/10.4153/CJM-1970-125-1
11. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. J. ACM **30**(3), 417–427 (1983). https://doi.org/10.1145/2402.322385
12. Wasa, K.: Enumeration of enumeration algorithms. CoRR abs/1605.05102 (2016), http://arxiv.org/abs/1605.05102
13. Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of induced subtrees in a K-degenerate graph. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 94–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_8
14. Wasa, K., Kaneta, Y., Uno, T., Arimura, H.: Constant time enumeration of bounded-size subtrees in trees and its application. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 347–359. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32241-9_30