Manuscript, Hokkaido University, 15 June 2013 (submitting).

Efficient Enumeration of Induced Subtrees in Undirected Graphs

Kunihiro Wasa[1], Takeaki Uno[2], and Hiroki Arimura[1]

[1] Graduate School of IST, Hokkaido University, N14 W9, Sapporo 060-0814, Japan
{wasa,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, 2-1-2, Hitotsubashi, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. In this paper, we study the enumeration problem for all induced subtrees in an input graph G. Although the subgraph version of the problem is known to be enumerable in output linear time, the induced version of the problem considered here has not been studied before and its complexity is not known so far. We first introduce the notion of the border of an induced subtree. Then, we present an enumeration algorithm for all induced subtrees in G in linear time per subtree, in which an induced subtree is obtained from another induced subtree by adding a node in the border. Finally, we present a faster algorithm that enumerates all induced subtrees in $O(d)$ per subtree and $O(n+m)$ space, where d is the maximum degree of G.

1 Introduction

In this paper, we discuss an *enumeration problem for all induced subtrees in an input graph G*. An induced subgraph S is a subgraph of a graph induced by nodes. S is an induced subtree if S is connected and acyclic. As our main result, we show that the problem can be solve in $O(d)$ delay, and thus, $O(sd)$ total time, and $O(n)$ space, where n is the total number of nodes in G, d is the maximum degree of G, and s is the number of solutions. Our proposed algorithm is the first algorithm for solving our problem in linear delay. We use reverse search technique presented by Avis and Fukuda [1] to construct our algorithm.

Related work: Below, we show related results on induced subgraphs enumeration. Table 1 shows the summary of the results. Enumeration problems for induced subgraphs with some properties are widely studied. Let G be an input graph with n nodes and m edges, and s be the total number of solutions of each problem. In what follows, we fix an input graph G. Johnson *et al.* [6] presented an enumeration for all maximal independent sets in an input graph in $O(sn^3)$ total time. Avis and Fukuda [1] presented an enumeration algorithm for all connected induced subgraphs within an input graph in $O(mns)$ total time. Uno [10] studied about a problem to enumerate all chordless cycles in an input graph. He also studied about a problem to enumerate all chordless paths in an input graph as a sub-problem of the version of a chordless cycle. A chordless cycle and chordless path are subcycle and subpath induced by nodes, respectively. Uno

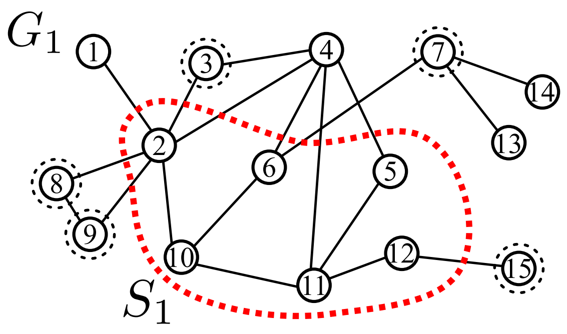


Fig. 1. An example of an input graph and an induced subtree.

presented enumeration algorithms to solve above both problems in $O(m+n)$ time per solution, and thus, in $O(s(m+n))$ total time. Makino and Uno [7] proposed an enumeration algorithm for all maximal cliques in G in $O(M(n))$ delay, where $M(n)$ denotes time need to multiply two $n \times n$ matrices, and the latest result is $M(n) = O(n^{2.3727}s)$ due to [13]. They also proposed $O(d^4)$ delay algorithm, where d is the maximum degree of G.

By contrast, there are many studies of enumeration problems of subgraphs that are induced by edges. Birmelé *et al.* [2] showed that all cycles and all st-paths in an input graph G can be enumerated in $O(m + \sum_{c \in \mathcal{C}(G)} |c|)$ total time and in $O(m + \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$ total time respectively, where $\mathcal{C}(G)$ is the set of all cycles in G and $\mathcal{P}_{st}(G)$ is the set of all st-paths in G. Shioura *et al.* [8] presented an optimal algorithm for enumerating all spanning trees in an input graph in $O(m+s)$ total time. Ferreira *et al.* [4] studied about a problem to enumerate all k-subtrees in an input graph and they showed that the problem can be solved in $O(sk)$ total time. A k-subtree is a connected and acyclic subgraph induced by $k-1$ edges. As a special case of Ferreira *et al.*'s problem, Wasa *et al.* [12] presented a constant delay enumeration algorithm for all k-subtrees in an input tree. However, there is no enumeration algorithm for all subtrees induced by nodes in spite of its fundamental.

Organizaition of this paper: In Sec. 2, we give besic definitions and our problem. In Sec. 3, we fisrt introduce a family tree, and in Sec. 4, then, we present a basic algorithm based on a family tree. In Sec. 5, we present a faster algorithm. Finally, in Sec. 6, we conclude.

2 Preliminaries

Let $G = (V, E)$ be a *finite undirected graph* with a *node set* $V = \{v_1, \ldots, v_n\}$ and an *edge set* $E = \{e_1, \ldots, e_m\} \subseteq V \times V$. Without loss of generality, we assume that G is a simple (without self-loops and parallel edges). We obtain a node set and an edge set of G from $V(G)$ and $E(G)$, respectively. We assume that u and v are arbitrary nodes in V. If $(u, v) \in E$ then u and v are *neighbourhoods* each other. We denote by $N(u)$ a *set of neighbourhoods* of u and denote by

Table 1. Summary of results on enumeration of subclasses of induced subgraphs in a graph. In the table, we denote by $n = |V|$, $m = |E|$, and s the numbers of vertices and edges of an input graph, and the number of all subgraphs as solutions, respectively. For simplicity, we omit additive factor $O(m + n)$ for preprocessing in total time.

Objects	Total Time	Space (words)	Result
connected induced subgraphs	$O(mns) = O(n^3 s)$	$O(m + n)$	AF'96 [1]
maximal independent sets (maximal cliques)	$O(n^3 s)$	exponential	JYP'88 *et al.* [6]
maximal cliques	$O(mns) = O(n^3 s)$	$O(m + n)$	Tukiyama'77 [9]
maximal cliques	$O(n^{2.3727} s)$	$O(n^2)$	MU'04 [7]
maximal cliques	$O(d^4 s)$	$O(m + n)$	MU'04 [7]
induced st-paths	$O((m + n)s)$	$O(m + n)$	Uno'03 [10]
induced cycles	$O((m + n)s)$	$O(m + n)$	Uno'03 [10]
induced trees	$O((m + n)s)$	$O(m + n)$	This (Thm. 2)
induced trees	$O(ds)$	$O(m + n)$	This (Thm. 3)

$d(u) = |N(u)|$ a *degree* of u. We call the subscript of each node $v \in V$ the *ID* of v. In what follows, we identify the node and the associated ID.

For any pair of nodes u and v in V, if there is a sequence $\pi = (v_1 = u, \ldots, v_k = v)$, where $1 \le i \le k-1$ and $(v_i, v_{i+1}) \in E$, then we say that there is a *path* from u to v. We call π a *cycle*, if $k > 2$ and $u = v$. G is a *acyclic graph* if there is no cycles in G. G is a *connected graph* if there is a path between every pair of nodes.

2.1 Induced subtrees

Let $G[S] = (S, E[S])$ be a *subgraph induced by* S, where $E[S] = \{(u, v) \in E \mid u, v \in S\}$. We say that $G[S]$ is a *induced subtree* if $G[S]$ is connected and acyclic. In the following, if it is clear from the context, we identify S with $G[S]$.

Example 1. We show an example of an input graph G_1 and an induced subtree S_1 included in G_1 in Fig. 1. Circles with an integer indicate nodes and segments connected nodes indicate edges. In what follows, we assume that an associated integer with a node is its ID. In this figure, a node set $S_1 = \{2, 5, 6, 10, 11, 12\}$ surrounded by a dashed circle is an induced subtree $G[S_1]$ since it is connected and acyclic. The edge set of S_1 is $\{(2, 10), (6, 10), (10, 11), (11, 5), (11, 12)\}$. Node set $R = \{2, 4, 5, 6, 10, 11\}$ is a connected induced subgraph of G_1, however R is not an induced subtree since $\{4, 5, 6, 10, 11\}$ makes a cycle.

Let S be an induced subtree included in an input graph G and let $|S|$ be the size of S. If $r \in S$ is the minimum ID in S, then we say r is the root of S and we denote by $\mathrm{root}(S)$ the root of S. In this paper, we assume that an induced subtree is a rooted tree. We call u is a *leaf of* S if $u \in S$, $u \neq \mathrm{root}(S)$, and u has only one neighbourhood in S.

Let $UP(u)$ be the unique path from u to $\mathrm{root}(S)$ on S. The *parent* of $u \neq \mathrm{root}(S)$ is its adjacent node on $UP(u)$ and the ancestors of u are the nodes on $UP(u)$. We say that u is a *child* of v if v is a parent of u and u is a *descendant* of v if v is a ancestor of u.

In addition to IDs, we assume that we number all nodes in S by the *DFS numbering*, which is the preorder numbering in the depth first search [3] on S. The ordering of visiting nodes by the DFS depends on the ordering of IDs. For any node u in S, we denote by $\mathrm{w}(u, S)$ the *weight of* u, where the weight of u is the number of the DFS-numbering. We denote by $\mathrm{wmax}(S)$ the maximum weight node in S. In the following, if it is clear from the context, we omit the argument of w. We denote by $\mathrm{cnt}(S, x) = |\{y \in S \mid (x, y) \in E\}|$ the *number of connections* of x with S. The number of connection of x indicates that the number of neighbourhoods of x within S.

2.2 Enumeration problem

We introduce terminology for enumeration algorithms according to Goldberg [5] and Uno [11]. An *enumeration algorithm* for an enumeration problem Π is an algorithm $\mathcal{A}$ that receives an *instance* I and outputs all *solutions* S in the answer set $S(I)$ into a write-only output stream O without duplicates. Let $n = ||I||$ and $m = |S(I)|$ be the input and the output size on I, respectively. We say that $\mathcal{A}$ is of *amortized constant time* if the total running time of $\mathcal{A}$ for computing all solutions on I is linear in m. For a polynomial $p(\cdot)$, $\mathcal{A}$ is of *constant delay* using preprocessing $p(n)$ if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a constant $c(n)$ after preprocessing in $p(n)$ time. As a computation model, we adopt the usual RAM [3]. We give the problem of this paper as follows:

Problem 1. Given an input graph G, then enumerate all induced subtrees included in G without duplications.

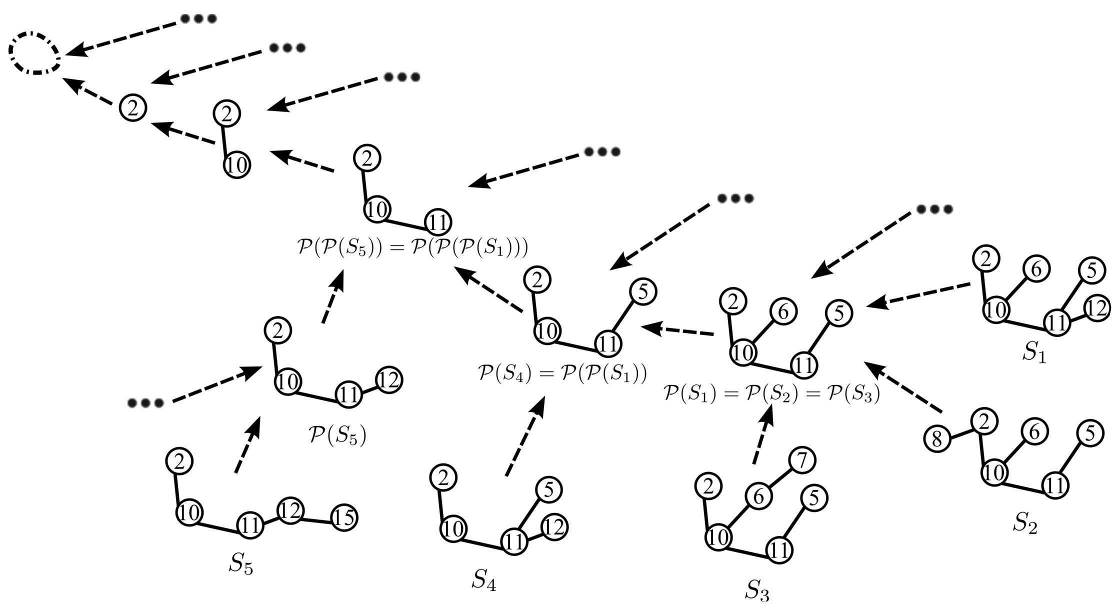
3 Family tree

In this section, we will give a definition of a family tree [1]. First, we define a parent of induced subtrees in an input graph $G = (V, E)$.

Definition 1. *Let S be an induced subtree with more than one nodes included in an input graph G. We define a parent $\mathcal{P}(S)$ of S as follows:*

$$\mathcal{P}(S) = S \setminus \{\mathrm{wmax}(S)\}$$

Example 2. In Fig. 2, the parent $\mathcal{P}(S_1)$ of S_1 has the node set $\{2, 5, 6, 10, 11\}$ and the parent $\mathcal{P}(\mathcal{P}(S_1))$ of $\mathcal{P}(S_1)$ has the node set $\{2, 5, 10, 11\}$.



$\mathcal{P}(\mathcal{P}(S_5)) = \mathcal{P}(\mathcal{P}(\mathcal{P}(S_1)))$
$\mathcal{P}(S_4) = \mathcal{P}(\mathcal{P}(S_1))$
$\mathcal{P}(S_1) = \mathcal{P}(S_2) = \mathcal{P}(S_3)$
$\mathcal{P}(S_5)$
S_1
S_2
S_3
S_4
S_5

Fig. 2. An example of the family tree $\mathcal{F}(G_1)$ of G_1.

Lemma 1. *Let S be an induced subtree in an input graph G. Then,* $\mathrm{wmax}(S)$ *is a leaf of S.*

Proof. We assume that $y = \mathrm{wmax}(S)$ is not a leaf of S. Then, there are some descendant nodes of y in S, and thus, the descendant nodes are clearly heavier than y. However, it is contradicts $y = \mathrm{wmax}(S)$. Thus, the lemma holds. $\square$

If we remove any leaf from a tree, the form of the tree obviously is still connected and acyclic. Thus, we immediately see the following lemma from Lemma 1.

Lemma 2. *Let S be an induced subtree in G and $|S| > 0$. Then, $\mathcal{P}(S)$ is a induced subtree in G.*

Now, we give a definition of the family tree based on the definition of the parent. The family tree plays a key role in our proposed algorithm.

Definition 2. *Let $\mathcal{IST}(G)$ be the all induced subtrees in an input graph G. We define $\mathcal{F}(G) = (\mathcal{IST}(G), E_{\mathcal{P}}(G))$ as a directed graph called the family tree of G, where the node set of $\mathcal{F}(G)$ is $\mathcal{IST}(G)$ and the edge set of $\mathcal{F}(G)$ is $E_{\mathcal{P}}(G) = \{(S, \mathcal{P}(S)) \mid S \in \mathcal{IST}(G) \setminus \{\emptyset\}\}$.*

Lemma 3. *The family tree $\mathcal{F}(G)$ of G forms a spanning tree over $\mathcal{IST}(G)$.*

Proof. Let S be an arbitrary induced subtree in G and $|S| > 0$. From Lemma 1 and 2, we see that an empty induced subtree is obtained by repeatedly applying $\mathcal{P}$ to S. Furthermore, S has the unique parent since the definition of the parent. Thus, we easily see that $\mathcal{F}(G)$ forms a spanning tree over $\mathcal{IST}(G)$. $\square$

From Lemma 3, the following corollary is obvious.

Corollary 1. *Let G be an input graph. Then, the root of $\mathcal{F}(G)$ is an empty induced subtree.*

Example 3. Fig. 2 shows a part of the family tree $\mathcal{F}(G_1)$ of G_1. Induced subtrees $S_1, \ldots, S_5$ in G_1 have some common ancestors each other. We construct $\mathcal{F}(G_1)$ by bundling sequences of ancestors obtained by repeatedly applying $\mathcal{P}$ to induced subtrees.

4 Basic algorithm

In this section, we give a basic algorithm solving our problem. The basic idea of the algorithm is backtracking on the family tree of an input tree G and enumerating all induced subtrees in G. First, we give the definition of the border of an induced subtree. Then, we define the children of an induced subtree using its border.

Definition 3. *Let S be an induced subtree in $G = (V, E)$. We denote by $\mathrm{B}(S)$ the border of S. Node $x \in V \setminus S$ is a member of $\mathrm{B}(S)$ if and only if x satisfies the following three conditions:*

1. *x is larger than or equals to the root of S.*
2. *x is the heaviest node of $S \cup \{x\}$.*
3. *$\mathrm{cnt}(S, x) = 1$.*

Example 4. In Fig. 1, nodes surrounded by the dotted line indicates the nodes in the border $\mathrm{B}(S_1)$ and $\mathrm{B}(S_1) = \{3, 7, 8, 9, 15\}$.

Lemma 4. *Let S and T be any two induced subtrees in an input graph G and S be the parent of T. Then, $\mathrm{wmax}(T)$ is a member of $\mathrm{B}(S)$.*

Proof. We assume that $x = \mathrm{wmax}(T)$. Thus, $T = S \cup \{x\}$. From Lemma 1, x is the heaviest leaf of $S \cup \{x\}$ and $\mathrm{cnt}(S, x) = 1$. Furthermore, x is obviously larger than or equals to $\mathrm{root}(S)$. Thus, from the definition of the border, x is a member of $\mathrm{B}(S)$. □

Definition 4. *Let S be an induced subtree in G. We define a set of all children of S as follows:*

$$\mathcal{C}(S) = \{S \cup \{x\} \mid x \in \mathrm{B}(S)\}.$$

We denote by $Ch(S, x) = S \cup \{x\}$ the child of S obtained by adding $x \in \mathrm{B}(S)$.

Lemma 5. *Let S and T be any two induced subtrees in G. If $S = \mathcal{P}(T)$, then $T = Ch(S, x)$ for some $x \in \mathrm{B}(S)$.*

Proof. From the definition of Ch, we easily see that $S \cup \{x\} = \mathcal{P}(T) \cup \{x\} = (T \setminus \{\mathrm{wmax}(T)\}) \cup \{x\}$. On the other hand, from Lemma 4, $\mathrm{wmax}(T) \in \mathrm{B}(S)$. Thus, if $x = \mathrm{wmax}(T)$, then $(T \setminus \{x\}) \cup \{x\} = T$. □

Algorithm 1 ISE algorithm

```
procedure MAIN($G = (V, E)$)     // an input graph $G$
    Number the nodes of $G$ from 1 to $|V|$ as ID;     // Initalization
    REC($\emptyset, V$);
end procedure

procedure REC($S, \mathrm{B}(S)$)
    Output $S$;
    for each $x = \mathrm{B}(S)$ do
        $T \leftarrow Ch(S, x)$;
        $\mathrm{B}(T) \leftarrow$ CALCBORDER($S$);
        REC($T, \mathrm{B}(T)$);
    end for
end procedure

procedure CALCBORDER($S$)
    $\mathrm{B}(S) \leftarrow \{x \in V \setminus S \mid \mathrm{root}(S) \leq x \wedge \mathrm{wmax}(S \cup \{x\}) = x \wedge \mathrm{cnt}(S, x) = 1\}$;
    return $\mathrm{B}(S)$;
end procedure
```

To avoid changing the root node after getting children, all nodes in the border has larger ID than the root node.

Lemma 6. *Let S and T be any two induced subtrees in G. If $T = Ch(S, x)$ for some $x \in \mathrm{B}(S)$, then $S = \mathcal{P}(T)$.*

Proof. From the definition of the border, x is the heaviest node in $T = S \cup \{x\}$. Thus, $\mathcal{P}(T) = \mathcal{P}(S \cup \{x\}) = S$. □

From Lemma 5 and 6, we can immediately see the following theorem.

Theorem 1. *Let S and T be any two induced subtrees in an input graph G. Then, $S = \mathcal{P}(T)$ if and only if $T = Ch(S, x)$ for some $x \in \mathrm{B}(S)$.*

We present a basic enumeration algorithm for all induced subtrees in an input graph G by backtracking on the family tree $\mathcal{F}(G)$ in Algorithm 1. From Theorem 1, we easily see the following theorem.

Theorem 2. *Algorithm 1 enumerates all induced subtrees in an input graph G in $O(m+n)$ time per subtree and $O(m+n)$ space, where m and n are the numbers of edges and vertices of G, respectively.*

Proof. The proof is by induction on the size of induced subtrees. Let k be the size of an induced subtree. If an induced subtree with size zero, that is, an empty set is obviously outputted. We assume that $k = \ell$ holds. Let S be an arbitrary induced subtree in G with k nodes. After outputting S, the algorithm computes a child T of S at line 8 and the border $\mathrm{B}(T)$ of T at line 9 based on the definition of the border. Then, by calling REC($T, \mathrm{B}(T)$), the algorithm outputs T at line 10.

From Theorem 1, the algorithm outputs all children of S by executing from line 7 to line 11 on each node of $\mathrm{B}(S)$. From the induction hypothesis, the algorithm outputs all induced subtrees with size $\ell+1$, and thus, the algorithm outputs all induced subtrees in G. The halt of the algorithm is immediately seen by $\mathcal{F}(G)$ forming a tree and the above discussion. $\square$

From the above theorem, all induced subtrees in a graph can be enumerated in $O((m+n)s)$ total time, where s is the number of induced subtrees. In the next section, we improve the running time of this algorithm when an input graph is *sparse* such that the maximum degree d of G is much smaller than m and n.

5 Faster algorithm

Algorithm 1 can enumerate all induced subtrees in an input graph $G = (V, E)$. However, it takes more than $O(m+n)$ time per induced subtree since the running time of CalcBorder, which computes the border from scratch, is at least $O(m+n)$ time. In this section, we present a faster algorithm that can enumerate all induced subtrees in $O(d)$ time per subtree, where d is the maximum degree in G. First, for the acceleration, we give a fast update method of the border.

Lemma 7. *Let S be a non-empty induced subtree in an input graph G and T be the child of S by adding x for any $x \in \mathrm{B}(S)$. Then,*

$$\mathrm{B}(T) = (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$$

holds, where $\Gamma = \{y \in \mathrm{N}(x) \setminus T \mid y > \mathrm{root}(T)\}$ and $\Delta = \{x\} \cup \{y \in \mathrm{B}(S) \mid \mathrm{wmax}(T \cup \{y\}) \neq y\} \cup \{y \in \mathrm{N}(x) \mid \mathrm{cnt}(T, y) > 1\}$.

Proof. $\mathrm{B}(T) \supseteq (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$: Let α be an arbitrary node in the node set $X = (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$. We will show that α is a member of $\mathrm{B}(T)$. (i) The root of S is smaller than any node in $\mathrm{B}(S)$ and the root of T is smaller than any node in Γ. Furthermore, $\mathrm{root}(S) = \mathrm{root}(T)$ since x is larger than any node in S. Thus, α is larger than $\mathrm{root}(T)$. (ii) X is the union of the subset of $\mathrm{B}(S)$ and the subset of $\mathrm{N}(x)$. First, we assume that $\alpha \notin \mathrm{N}(x)$. Then, α is a member of $\mathrm{B}(S)$, and thus, $\mathrm{cnt}(T, \alpha) = 1$. Next, we assume that $\alpha \in \mathrm{N}(x)$. Then, a node z satisfying $\mathrm{cnt}(T, z) \neq 1$ is not a member of X because of the assumption of Δ. Hence, $\mathrm{cnt}(T, \alpha) = 1$. (iii) We assume that $\alpha \notin \mathrm{N}(x)$. From the definition of Δ, X includes such node x as the maximum weight in $T \cup \{x\}$. Next, we assume that $\alpha \in \mathrm{N}(x)$. From $\mathrm{wmax}(T) = x$ and the definition of a weight, we can immediately see that α is the maximum weight in $T \cup \{x\}$. From (i)–(iii), α is a member of $\mathrm{B}(T)$. $\mathrm{B}(T) \subseteq (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$: The difference between T and S is whether x is included or not. Then, we easily see that $\mathrm{B}(T) \subseteq \mathrm{B}(S) \cup \mathrm{N}(x)$. Furthermore, $\mathrm{B}(T) \subseteq \mathrm{B}(S) \cup \Gamma$ since all nodes in $\mathrm{B}(T)$ are larger than $\mathrm{root}(T)$. From the definition of the border, $\mathrm{B}(T)$ obviously does not include any node in Δ. Thus, $\mathrm{B}(T) \subseteq (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$. From above argument, $\mathrm{B}(T) = (\mathrm{B}(S) \cup \Gamma) \setminus \Delta$ holds. $\square$

We give the definition of extended induced subtrees as follows:

Definition 5. *Let S be an induced subtree in an input graph G. The extended induced subtree of S is defined by $S^+ = (S \cup \mathrm{B}(S), E(G[S]) \cup E^+)$, where $E^+ = \{(x, y) \in E \mid x \in \mathrm{B}(S), y \in S\}$.*

Lemma 8. *Let S be an induced subtree in an input graph. The extended induced subtree S^+ of S is connected and acyclic, that is, tree.*

Proof. S^+ is connected since $\mathrm{cnt}(x, S) = 1$ for any $x \in \mathrm{B}(S)$. Furthermore, there are no edges between any pair of nodes in $\mathrm{B}(S)$ on S^+, thus S^+ is obviously acyclic. □

From Lemma 8, we can give an weight to each node in an extended induced subtree similar to an induced subtree. Now, we define the ordering of nodes in the border based on the weight.

Definition 6. *Let S be an induced subtree in an input graph. For any pair of nodes $u, v \in \mathrm{B}(S)$, if $\mathrm{w}(u, S^+) < \mathrm{w}(v, S^+)$ then we call u is smaller than v in the border of S and we denote by $u \prec_S v$ this relationship.*

Example 5. In Fig. 1, the ordering of the nodes in $\mathrm{B}(S_1)$ is $3 \prec_{S_1} 8 \prec_{S_1} 9 \prec_{S_1} 7 \prec_{S_1} 15$.

We immediately see the following lemma by observing Definition 6.

Lemma 9. *Let S be an induced subtree in an input graph $G = (V, E)$. For any pair of nodes u and v in $\mathrm{B}(S)$ satisfying $u \prec_S v$, if $(u, v) \notin E$, then $v \in \mathrm{B}(S \cup \{u\})$ and $u \notin \mathrm{B}(S \cup \{v\})$.*

We present the faster algorithm in Algorithm 2.

Theorem 3. *Algorithm 2 enumerates all induced subtrees in an input graph G in $O(d)$ time per induced subtree and $O(n + m)$ space, where d, m, and n are the maximum degree, and the numbers of edges and vertices of G, respectively.*

Proof. If the procedure correctly computes the border and all children of S, then we can easily see that the algorithm enumerates all induced subtrees in G in a similar manner to Theorem 2. Now, we show that the algorithm correctly updates the border. We assume that ModifiedRec in Algorithm 2 is called with S and $\mathrm{B}(S)$ for any induced subtree S in G. Adding nodes in Γ of S to $\mathrm{B}(S)$ corresponds to from line 30 to line 33 on Update. Next, we discuss deleting nodes in Δ of S from $\mathrm{B}(S)$. Let x be a member of $\mathrm{B}(S)$, Δ_1 be $\{x\} \cup \{y \in \mathrm{B}(S) \mid \mathrm{wmax}(T \cup \{y\}) \neq y\}$, and Δ_2 be $\{y \in \mathrm{N}(x) \mid \mathrm{cnt}(T, y) > 1\}$. That is, $\Delta = \Delta_1 \cup \Delta_2$. To delete nodes in Δ_1 from $\mathrm{B}(S)$ corresponds to at line 16 and runs in $O(1)$, since there is no need to delete nodes other than x from Lemma 9. Deleting nodes in Δ_2 from $\mathrm{B}(S)$ is computed by executing line 27 and 28 for each node in $\mathrm{B}(S)$. We easily see that Update runs in $O(d)$. Thus, the algorithm can update the border of a induced subtree in $O(d)$ time. In addition, the algorithm needs $O(n)$ time preprocessing at line 2. In order to get the border of S from the border of a child S at line 22, the algorithm uses the stack H that stores the history of operations and the algorithm restores the border on RESTORE based on H. H requires $O(n)$ space since the depth of $\mathcal{F}(G)$ is obviously at most n. Furthermore, each node in S has pointers for the border. Thus, Algorithm 2 runs in $O(d)$ time per induced subtree with $O(n+m)$ space. □

From the above theorem, all induced subtrees in a graph can be enumerated in $O((m+n)s)$ total time, where s is the number of induced subtrees. This improves on the previous $O((m+n)s)$-time algorithm by factor of $O((m+n)/d)$.

6 Conclusion

In this paper, we considered the enumeration problem of all induced subtrees in an input graph. As a main result, we presented an efficient enumeration algorithm runs in $O(d)$ time per induced subtree, where d is the maximum degree of an input graph. The remaining task is to discussing whether a constant delay enumeration algorithm exists.

References

1. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996. Sec. 3.4.
2. E. Birmelé, R. A. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi, and G. Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In *Proc. SODA'13*, pages 1884–1896, 2013.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.
4. R. Ferreira, R. Grossi, and R. Rizzi. Output-sensitive listing of bounded-size trees in undirected graphs. In *Proc. ESA'11*, LNCS 6942, pages 275–286, 2011.
5. L. A. Goldberg. Polynomial space polynomial delay algorithms for listing families of graphs. In *Proc. ACM STOC'93*, pages 218–225. ACM, 1993.
6. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
7. K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Proc. SWAT'04*, LNCS 3111, pages 260–272. Springer Berlin Heidelberg, 2004.
8. A. Shioura, A. Tamura, and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.
9. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977.
10. T. Uno. An output linear time algorithm for enumerating chordless cycles. *Technical Notes, 92nd SIGAL of IPSJ*, pages 47–53, 2003. In Japanese.
11. T. Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical Report NII-2003-004E, National Institute of Informatics, 2003.
12. K. Wasa, Y. Kaneta, T. Uno, and H. Arimura. Constant time enumeration of bounded-size subtrees in trees and its application. In *Proc. COCOON'12, LNCS 7434*, pages 347–359, 2012.
13. V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *Proc. STOC'12*, pages 887–898. ACM, 2012.

Algorithm 2 modified ISE algorithm

```
1: procedure ModifiedISE($G = (V, E)$)    // an input graph $G$
2:     Number the nodes of $G$ from 1 to $|V|$ as ID;
3:     Output $\emptyset$;
4:     for each node $x$ in $G$ do
5:         $S \leftarrow \{x\}$;
6:         $\mathrm{B}(S) \leftarrow$ an empty doubly linked list;
7:         $\mathrm{H} \leftarrow$ an empty stack;
8:         Update($S, \mathrm{B}(S), x, \mathrm{H}$);
9:         ModifiedRec($S, \mathrm{B}(S), \mathrm{H}$);
10:    end for
11: end procedure

12: procedure ModifiedRec($S, \mathrm{B}(S), \mathrm{H}$)
13:    Output $S$;
14:    while $\mathrm{B}(S)$ is not empty do
15:        $x \leftarrow$ the head of $\mathrm{B}(S)$;
16:        Remove the head of $\mathrm{B}(S)$;
17:        $S \leftarrow Ch(S, x)$;
18:        H.push(pivot, NIL);
19:        Update($S, \mathrm{B}(S), x, \mathrm{H}$);
20:        ModifiedRec($S, \mathrm{B}(S), \mathrm{H}$);
21:        Restore($\mathrm{B}(S), \mathrm{H}$);
22:        $S \leftarrow \mathcal{P}(S)$;
23:    end while
24: end procedure

25: procedure Update($S, \mathrm{B}(S), x, \mathrm{H}$)
26:    for each neighbourhood node $y$ of $x$ do    // increasing ordering of $y$'s ID
27:        if $\mathrm{cnt}(S, y) > 1$ then
28:            Remove $y$ from $\mathrm{B}(S)$;
29:            H.push(del, $y$);
30:        else if $y > \mathrm{root}(S)$ and $y \notin S$ then
31:            Add $y$ to $\mathrm{B}(S)$ on the head;
32:            H.push(add, $y$);
33:        end if
34:    end for
35: end procedure

36: procedure Restore($\mathrm{B}(S), \mathrm{H}$)
37:    while true do
38:        $(op, x) \leftarrow$ H.top();
39:        H.pop();
40:        if $op == pivot$ then
41:            break;
42:        else
43:            $\mathrm{B}(S).op(x)$;
44:        end if
45:    end while
46: end procedure
```