研究論文

A Fast Algorithm for Enumerating Non-Bipartite Maximal Matchings
一般グラフの極大マッチングを列挙する高速アルゴリズム

Takeaki UNO
National Institute of Informatics
宇野 毅明
国立情報学研究所

ABSTRACT

For a graph $G=(V,E)$, a *stable set* in G is a vertex set such that no pair of vertices in the set are connected by an edge. Stable set enumeration problems have been studied because of their applications to optimization, computational geometry, etc. However, the problem of speeding up enumeration algorithms for stable sets is still open. In this paper, we consider the problem of enumerating all maximal matchings of a given non-bipartite graph $G=(V,E)$, which is a special case of the stable set enuemration problem, and propose an algorithm with a simple structure. By applying the stable set enumeration algorithms to this problem, the computation time is $\mathrm{O}(|V||E|^2N)$. Our algorithm runs in $\mathrm{O}(|E|+|V|+\Delta N)$ time, very fast compared with those algorithms. Here N denotes the number of maximal matchings in G, and Δ denotes the maximum degree of G.

要旨

グラフ $G=(V,E)$ の頂点集合で, 任意の 2 頂点間に枝の無いものを安定集合という. 集合の意味で極大な安定集合を列挙する問題は, 最適化, 計算幾何学などの多種の応用から研究されてきた. しかし, それを解くアルゴリズムに対して, 計算量を減少させる意味での高速化ができるかどうかは, まだ分かっていない. 本論文ではこの問題の特殊形である, 極大なマッチングを列挙する問題を考え, 高速化が行えることを証明する. 単純に一般グラフの極大マッチングを列挙する問題を考え, 簡単な構造を持つアルゴリズムを提案する. 極大安定集合を列挙するアルゴリズムを単純にこの問題に当てはめた場合, 計算量は $\mathrm{O}(|V||E|^2N)$ となるが, 提案するアルゴリズムの計算量は $\mathrm{O}(|E|+|V|+\Delta N)$ となり, 大幅な改善が行われた. ここで N は極大マッチングの数, Δ は G の頂点の最大次数である.

[Keywords]

enumeration, enumeration tree, listing, maximal matching

[キーワード]

列挙, 列挙木, 列挙アルゴリズム, 極大マッチング

1. Introduction

For a graph $G = (V, E)$, a *stable set* in G is a vertex set such that no pair of vertices in the set are connected by an edge. Stable set is a fundamental object in combinatorial problems, and optimization problems of stable sets have numerous applications. Although several kinds of stable set optimization problems (for an example, finding a maximum weight stable set) are quite hard, for that reason, many approaches and algorithms, such as branch and bound, cutting plane, heuristics, etc., have been proposed. Enumeration problems of stable sets have also been studied. Enumeration of stable sets also has applications, such as optimization problems of triangulation. Since enumeration of maximum stable sets in a non-bipartite graph is quite hard, enumeration of maximal stable sets has been studied. In 1977, S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa[8] proposed an algorithm for enumerating maximal stable sets. Its time complexity is $\mathrm{O}(|E||V|N)$ where N is the number of output (number of stable sets), and its space complexity is $\mathrm{O}(|V| + |E|)$. In practical terms, the algorithm is quite slow. One remarkable point of this field of study is that no improved algorithm has been proposed since 1977 while many linear time or constant time algorithms have been proposed for enumeration problem of other combinatorial objects, such as spanning trees, paths, and bipartite matchings. Here a linear time enumeration algorithm means an algorithm running in linear time of input size per output, and a constant time enumeration algorithm means an algorithm running in constant time per output. Speeding up stable set enumeration algorithm appears to be a difficult task.

On the other hand, for this reason, several closely related problems have been studied. In 1992, T. Kashiwabara , S. Masuda, K. Nakajima and T. Fujisawa studied the case of bipartite graphs and circular arc graphs, and proposed algorithms for enumerating maximum stable sets. The algorithms run in $\mathrm{O}(|E||V|^{1/2} + N)$ time and $\mathrm{O}(|E||V|^{3/2} + N)$ time, respectively.[6] The success of these algorithms is built on properties of the subject graphs.

In this paper, we consider another special case of the enumeration problem of stable sets, the enumeration of maximal matchings in a non-bipartite graph. Since matchings have good properties which are not satisfied by stable sets, we can obtain a significant improvement, i.e. the time complexity is reduced to $\mathrm{O}(|E| + |V| + \Delta N)$, where Δ denotes the maximum degree of G.

Again, we introduce our problem in detail. Let $G = (V, E)$ be a non-bipartite undirected graph with vertex set V and edge set $E = \{e_1, ..., e_m\}$. We denote the number of vertices by n. Let e_i have an index i. We assume that there are neither isolated vertices nor parallel edges. Δ denotes the degree of a vertex of the maximum degree in G. A *matching* M of the graph G is an edge set such that no two edges in M share their endpoints. For a matching M of $G = (V, E)$, let $\hat{E}(G, M)$ denote the edges of $E \setminus M$ adjacent to no edge of M. Note that $\hat{E}(G, M) = E$ if $M = \emptyset$. We call a matching which is contained in no other matching a *maximal matching*. M is a maximal matching of G if and only if $\hat{E}(G, M) = \emptyset$. This paper considers the problem of enumerating all maximal matchings in G.

A matching in a graph G is equivalent to a stable set in the line graph of G. The line graph of G is $(E, \tilde{E})$ such that $(e, e') \in E \times E$ is included in $\tilde{E}$ if and only if e and e' are adjacent in G. The problem of enumerating maximal matchings is reducible to the problem of enumerating stable sets of the line graph, hence we can enumerate maximal matchings by using the algorithm of Tsukiyama et al.[8] Since the line graph of G has m vertices and $\mathrm{O}(nm)$ edges, their algorithm takes $\mathrm{O}(nm^2N)$ time and $\mathrm{O}(nm)$ space where N denotes the number of maximal matchings in G. The computation time is probably too slow to be practical. In 1988, D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou[4] proposed another algorithm for enumerating maximal stable sets, however their algorithm has the same time complexity as Tsukiyama et al's.

Matchings have some “good” properties that stable sets do not have, hence many matching problems can be solved more easily than stable set problems. For example, we can find a maximum matching of a graph in polynomial time,[2, 3] but the maximum stable set problem is known to belong to the class of NP-hard problems. Therefore, there naturally seem to exist possibilities of making a fast algorithm for enumerating

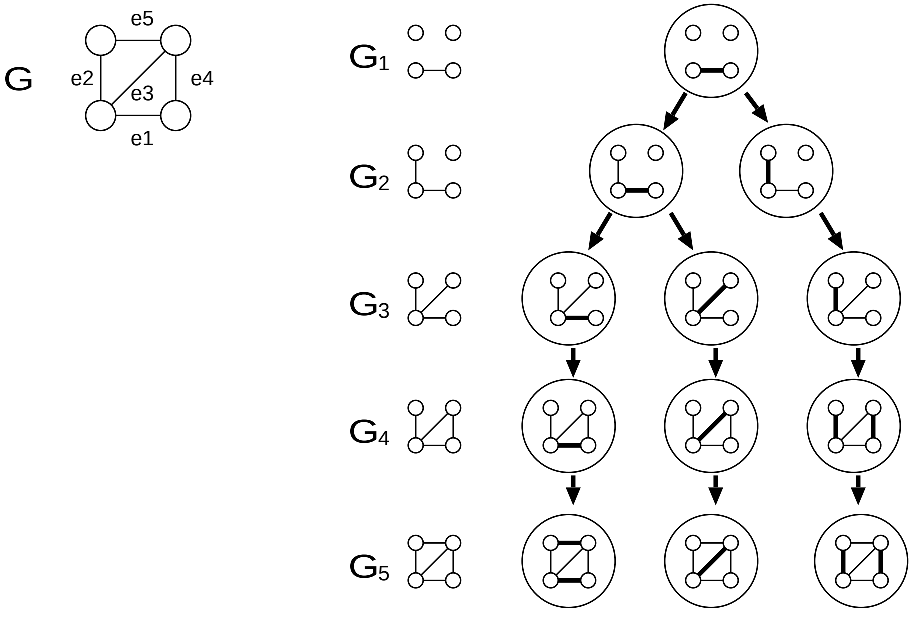

Figure 1: An instance of enumeration tree.

maximal matchings, if not for stable sets. In this paper, we use such "good" properties, and improve on the algorithm of Tsukiyama et al. by adapting it to maximal matchings.

Our improvements also are composed of two areas. The first is that we use several techniques to speed up iterations. In this way, we reduce the time complexity from $\mathrm{O}(nm^2N)$ to $\mathrm{O}(mN)$. The second is that we introduce a preprocessing of the input to the algorithm to decrease the number of iterations and amortized time complexity. By detailed analysis, we reduce the time complexity to $\mathrm{O}(\Delta N)$. Our improvements also result in optimal memory complexity.

In the following sections, we describe our enumeration algorithm and its method of analysis in detail. We describe the framework of our algorithm and the improvements in the first area in section 2. In section 3, we explain the improvements we made in the second area, and analyze the time complexity in detail so that we can show the reduction in time complexity.

2. Reverse Search Algorithm for Maximal Matchings

In this section, we describe the framework of our algorithm obtained by modifying the algorithm of Tsukiyama et al. We also show our techniques to reduce the computation time of an iteration.

For constructing enumeration algorithms, we have a scheme called *reverse search*.[1] The algorithm of Tsukiyama et al. can be considered as a type of reverse search, and our algorithm is thereby based on reverse search. Reverse search is a scheme for enumerating all elements of a set. It utilizes a parent-child relationship among elements of the set, which has to satisfy the following two conditions:

(1) any element except one element has its unique parent

(2) no element is a proper ancestor of itself.

The graph expression of this relationship, composed of vertices corresponding to its elements and edges connecting children to their parents, forms a tree under these conditions. The tree is called an *enumeration tree*. Reverse search traverses all vertices of the tree in a depth first search manner, and outputs all elements in the order in which they are visited. A feature of reverse search is that its memory complexity does not depend on the number of output.

Reverse search does not store the whole enumeration tree in the memory, but stores only the vertex of the tree that is currently being traversed. Reverse search finds the root vertex of the enumeration tree, then finds a child of the root. The search then moves to the child, and enumerates all descendants of the child, recursively. After visiting all descendants of the child, reverse search returns to the root vertex, and finds another child of the root vertex. If there is no other child, reverse search stops. Otherwise, reverse search enumerates all descendants of the child in the same way. The depth first traversal of the enumeration tree is thus achieved in this way. Therefore, a reverse search algorithm can be based on simply finding all children of the vertex currently being traversed. An important part of a reverse search algorithm is to construct a fast algorithm for this task.

Let us look at the operation of reverse search for maximal matchings arising from the method of Tsukiyama et al. Let $G_i = (V, E_i)$ where $E_i = \{e_1, ..., e_i\}$. A maximal matching M of a subgraph G_i is called an *i-maximal matching,* and is denoted by (M, i). Our parent-child relationship in the following is defined among all the i-maximal matchings. The 1-maximal matching, which is the unique maximal matching of G_1, has no parent in our relationship. The parent of an i-maximal matching $(M, i), i \neq 1$, denoted by $p(M, i)$, is defined by the $(i-1)$-maximal matching obtained by the following procedure.

```
Procedure  OBTAIN_PARENT ($(M, i)$)
(OP1) If $e_i \notin M$ then output $(M, i-1)$ ; stop
(OP2) $M' := M \setminus \{e_i\}$
(OP3) If $\hat{E}(G_{i-1}, M') = \emptyset$ then output $(M', i-1)$ ;
          stop
(OP4) $M' := M \cup \{$ the edge with the minimum
          index among $\hat{E}(G_{i-1}, M')\}$
(OP5) Go to (OP3)
```

From this algorithm, $p(M, i)$ is defined uniquely, and no i-maximal matching is its proper ancestor. Hence, we obtain an enumeration tree the vertices of which correspond to all the i-maximal matchings. The leaves of the tree correspond to all the maximal matchings in G (see Figure 1).

Next we explain how to find all children of an $(i-1)$-maximal matching $(M, i-1)$. The method is based on the following lemma. Let $E(M, i)$ be the set of edges of M that are adjacent to e_i.

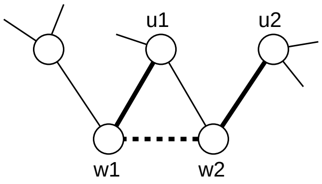

Figure 2: Generating a type-2 child: bold edges are edges of M. We obtain a matching M' by adding (w_1, w_2) to M and removing (w_1, u_1) and (w_2, u_2). If (w_1, u_1) has a larger index than e_1 or e_2, then M' is a type-2 child of M. If there is an edge (u_1, u_2), then M is not maximal.

Lemma 1 *(M', i) is a child of $(M, i-1)$ if and only if one of the following conditions hold.*

(a) $E(M, i) \neq \emptyset$, and $M' = M$

(b) $E(M, i) = \emptyset$, and $M' = M \cup \{e_i\}$

(c) $E(M, i) \neq \emptyset$, $p(M', i) = (M, i-1)$, and $M' = M \cup \{e_i\} \setminus E(M, i)$.

Proof: We first state the "if" part. In each case of (a), (b) and (c), M' is an i-maximal matching. If (a) holds, then e_i is adjacent to an edge of M'. Hence, $p(M', i) = (M, i-1)$. If (b) holds, then e_i is included in M'. Since $M' \setminus \{e_i\}$ is an $(i-1)$-maximal matching, $p(M', i) = (M, i-1)$. If (c) holds, then obviously $p(M', i) = (M, i-1)$.

We next state the "only if" part. Suppose that (M', i) is a child of $(M, i-1)$. If M' does not include e_i, then $M = M'$ and $E(M, i) \neq \emptyset$. Hence, (a) holds. If M' includes e_i and $M' \setminus \{e_i\}$ is an $(i-1)$-maximal matching, then $M = M \setminus \{e_i\}$, and $E(M, i) = \emptyset$. Hence, (b) holds. If M' includes e_i and $M' \setminus \{e_i\}$ is not an $(i-1)$-maximal matching, then $E(M, i) \neq \emptyset$, and $M' = M \cup \{e_i\} \setminus E(M, i)$. Hence, ($c$) holds.

Therefore, the lemma holds. ∎

We illustrate the case of (c) of the lemma in Figure 2. From the proof of the lemma, we can see that any i-maximal matching has a child satisfying (a) if $E(M, i) \neq \emptyset$, and a child satisfying (b) if $E(M, i) = \emptyset$. Moreover, any i-maximal matching has at most one child satisfying (c). We call a child satisfying (a) or (b) a *type-1 child*, and a child satisfying (c) *type-2 child*. From this, we can see that there are no fewer i-maximal matchings than there are $(i-1)$-maximal matchings.

A type-1 child (M', i) of $(M, i-1)$ is obtained from (M, i) in O(1) time by adding e_i if e_i is adjacent to no edge of M. There is not always a type 2 child. Hence, we have to check for the existence of a type-2 child. Checking for the existence in a simple way takes O(m) time. Hence, to speed this up, we introduce the following variables and state several lemmas. For an i-maximal matching (M, i) and a vertex v, let $A(v, M, i)$ be the set of edges $(v, u) \in E_i$ such that u is incident to no edge of M. If v is incident to an edge e_j of M, we define $l(v, M, i)$ by the number of edges e_l of $A(v, M, i)$ with $l < j$. Let w_1 and w_2 denote the endpoints of e_i. An instance of $A(v, M, i)$ and $l(v, M, i)$ is illustrated in Figure 3.

Lemma 2 *Suppose that* $|E(M, i-1)| = 1$, *and* $E(M, i-1) = \{(u_1, w_1)\}$. *Then,* $(M, i-1)$ *has a type-2 child if and only if the following conditions* (*1-a*) *and* (*1-b*) *hold.*

(*1-a*) $A(u_1, M, i-1) = \emptyset$.

(*1-b*) $l(w_1, M, i-1) = 0$, *and* (u_1, w_2) *has a larger index than* (u_1, w_1) *if* $(u_1, w_2) \in E_i$.

Proof: Suppose that (M', i) is a type-2 child of $(M, i-1)$. Then $M' = M \setminus \{(u_1, w_1)\} \cup \{e_i\}$, and u_1 is incident to no edge of M'. Hence, any vertex adjacent to u_1 is incident to an edge of M'. Since w_1 is incident to (u_1, w_1), (1-a) holds. From this, it follows that $\hat{E}(G_i, M' \setminus \{e_i\})$ is composed of edges in $A(w_1, M, i-1)$, and includes (u_1, w_2) if $(u_1, w_2) \in E_i$. Hence, (1-b) holds.

Let $M' = M \setminus \{(u_1, w_1)\} \cup \{e_i\}$. Suppose that (1-a) and (1-b) both hold. From (1-a), $\hat{E}(G_i, M \setminus \{(u_1, w_1)\})$ is composed of edges in $A(w_1, M, i-1) \cup \{(u_1, w_2)\}$. Hence, $\hat{E}(G_i, M') = \emptyset$, M' is an i-maximal matching, and $p(M', i)$ includes exactly one edge e' that is not included in M'. From (1-b), $e' = (u_1, w_1)$. Hence, $p(M', i) = (M, i-1)$. ▮

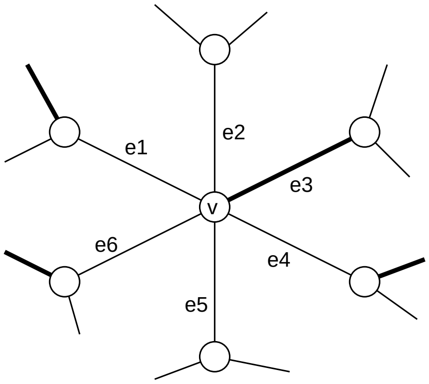

Figure 3: An instance of $A(v, M, i)$ and $l(v, M, i)$: The bold edges are edges of M. In this case, $A(v, M, i) = \{e_2, e_5\}$ and $l(v, M, i) = 1$.

Lemma 3 *Suppose that* $|E(M, i-1)| = 2$, *and* $E(M, i-1) = \{(u_1, w_1), (u_2, w_2)\}$ *where* (u_1, w_1) *has a smaller index than* (u_2, w_2). *Then,* $(M, i-1)$ *has a type-2 child if and only if the following conditions,* (*2-a*), (*2-b*), *and* (*2-c*), *hold.*

(*2-a*) $A(u_j, M, i-1) = \emptyset$ *for each* j, *and* $(u_1, u_2) \notin E_{i-1}$.

(*2-b*) $l(w_1, M, i-1) = 0$ *and any of* (w_2, u_1) *and* (w_1, u_2) *has a larger index than* (w_1, u_1).

(*2-c*) $l(w_2, M, i-1) = 0$.

Proof: This case is illustrated in Figure 2. Refer the figure for reading the proof. Suppose that (M', i) is a type-2 child of $(M, i-1)$. Then $M' = M \setminus \{(u_1, w_1), (u_2, w_2)\} \cup \{e_i\}$, and each u_j is adjacent to no edge of M'. Hence, E_i does not include (u_1, u_2). Since each w_j is incident to (u_j, w_j), $A(u_j, M, i-1) = \emptyset$. Hence, (2-a) holds. From this, it follows that $\hat{E}(G_i, M' \setminus \{e_i\})$ is composed of edges in $A(w_1, M, i-1) \cup A(w_2, M, i-1)$, and includes (u_1, w_2) and (u_2, w_1) if they exist in E_i. Hence, (2-b) holds. From (2-b), $\hat{E}(G_i, M' \setminus \{e_i\} \cup \{(u_1, w_1)\}) = A(w_2, M, i-1)$. Hence, (2-c) holds.

Let $M' = M \setminus \{(u_1, w_1), (u_2, w_2)\} \cup \{e_i\}$. Suppose that (2-a), (2-b) and (2-c) hold. From (2-a), $\hat{E}(G_i, M') = \emptyset$, and M' is an i-maximal matching. $p(M', i)$ is obtained by removing e_i and adding the minimum index edges among $\hat{E}(G_i, M' \setminus \{e_i\})$ repeatedly. Hence, from (2-b) and (2-c), $p(M', i) = (M, i-1)$. ▮

By using this, we obtain the following reverse search algorithm. In each iteration of the algorithm, A_v and l_v are equal to $A(v, M, i)$ and $l(v, M, i)$. Setting A_v and l_v to $A(v, \{e_1\}, 1)$ and $l(v, \{e_1\}, 1)$, respectively, and then executing ENUM_MAXIMAL_MATCHING ($\{e_1\}, 1$), we can enumerate all maximal matchings in G. To output maximal matchings, we use the *compact output method*[5, 7] in (EM1), (EM2) and (EM18). We describe the details later.

```
ALGORITHM ENUM_MAXIMAL_MATCHING $(M, i)$
(EM1) $O_i$ := edges in $M \setminus p(M, i)$ adjacent to $e_i$ and
          adjacent to no edge $e_j$ with $i < j$
(EM2) Output edges of $O_i$
(EM3) If $i = n$ then output "matching" ;
          go to (EM18)
(EM4) If $e_{i+1}$ is adjacent to an edge of $M$
          then $M' := M$ else $M' := M \cup \{e_{i+1}\}$
(EM5) Update each $A_v$ and $l_v$ to $A(v, M', i+1)$
          and $l(v, M', i+1)$
(EM6) Call ENUM_MAXIMAL_MATCHING $(M', i+1)$
(EM7) Update each $A_v$ and $l_v$ to $A(v, M, i)$
          and $l(v, M, i)$
(EM8) If $e_{i+1}$ is adjacent to no edge of $M$
          then go to (EM18)
(EM9) For each $(v, w_j) \in E(M, i+1)$
          if $A_v \neq \emptyset$ or $l_v > 0$ then go to (EM18)
(EM10) If $|E(M, i+1)| = 2$ then do
(EM11)     If $(u_1, u_2) \in E_i$ then go to (EM18)
(EM12)     For each $e_j \in \{(u_1, w_2), (u_2, w_1)\}$
               If $j <$ any index of any edge in $E(M, i+1)$
               then go to (EM18)
(EM13) End If
(EM14) $M' := M \cup \{e_{i+1}\} \setminus \{(u_1, w_1), ..., (u_k, w_k)\}$
(EM15) Update each $A_v$ and $l_v$ to $A(v, M', i+1)$
           and $l(v, M', i+1)$
(EM16) Call ENUM_MAXIMAL_MATCHING $(M', i+1)$
(EM17) Update each $A_v$ and $l_v$ to $A(v, M, i)$
           and $l(v, M, i)$
(EM18) Output "delete" and the edges of $O_i$
```

Here we explain the compact output method used in (EM1), (EM2) and (EM18). In (EM1), since edges of O_i are adjacent to no edge e_j with $i < j$, any edge of O_i is included in all j-maximal matchings with $j > i$ which are descendants of (M, i). Since edges of O_i are adjacent to e_i, any edge of O_i is not included in any O_j with $j < i$. Hence, in the case $i = n$, we have $M = \sum_{i=1}^{n} O_i$. Therefore, instead of outputting edges of M, we can output maximal matchings by outputting edges of O_i at the beginning of an iteration, and cancel it at the end of the iteration. This idea is called the compact output method. For the aim, (EM1) outputs edges of O_i, if $i = n$ then (EM3) outputs a message "matching" instead of outputting edges of M, and (EM18) cancels the output edges of O_i. To execute (EM1), we have to check at most two edges of M. Hence, it can be done in O(1) time. Therefore, the computation time for output is reduced to O(1) time per iteration.

(EM4) constructs the type-1 child, and (EM6) generates a recursive call with respect to the type-1 child. (EM8) through (EM13) check the existence of a type-2 child. If a type-2 child exists, then (EM14) generates the type-2 child, and (EM16) generates a recursive call with respect to it. (EM5), (EM7), (EM15) and (EM17) update A_v and l_v.

Lemma 4 *The time complexity of* ENUM_MAXIMAL_MATCHING *is* $O(mN)$ *and the space complexity of it is* $O(m + n)$.

Proof: The memory complexity is obviously O(m). As we saw, an iteration takes O(1) except for (EM5), (EM7), (EM15) and (EM17). We note that $E(M, i+1)$ can be obtained in O(1) time by putting a pointer from each vertex v to the edge of M that is incident to v, and maintaining these pointers as M changes in each iteration. Next, we explain the computation time required to update A_v and l_v. Without loss of generality, we explain this for the case of computing $A(v, M', i+1)$ and $l(v, M', i+1)$ from $A(v, M, i)$ and $l(v, M, i)$, where (M, i) is the parent of $(M', i+1)$.

Let $F \triangle F'$ denote the symmetric difference between two sets F and F'. For an edge set F, $V(F)$ denotes the vertices incident to an edge of F. Any vertex v satisfying $A(v, M, i) \neq A(v, M', i+1)$ or $l(v, M, i) \neq l(v, M', i+1)$ is adjacent to a vertex of $V(M \triangle M')$. Since $V(M \triangle M')$ includes at most four vertices, $|A(v, M, i) \triangle A(v, M', i+1)| \leq 4$. Hence,

$$\sum_{v | (v,u) \in E, u \in V(M \triangle M')} |A(v, M, i) \triangle A(v, M', i+1)|$$

$$\begin{aligned} &\leq \sum_{v \in V(M \triangle M')} 4d(v) \\ &= \mathrm{O}(\Delta), \end{aligned}$$

where $d(v)$ is the degree of v. Therefore, $A(v, M', i+1)$ for all vertices v can be obtained from $A(v, M, i)$ in $\mathrm{O}(\Delta)$ time. For any vertex $v \notin V(M \triangle M')$, we can also obtain $l(v, M', i+1)$ from $l(v, M, i)$ in $\mathrm{O}(1)$ time since no edge of $M \triangle M'$ is adjacent to v. For a vertex $v \in V(M \triangle M')$, we can obtain $l(v, M', i+1)$ in $\mathrm{O}(d(v))$ time. Hence, to obtain $l(v, M', i+1)$ for all vertices v, we take $\mathrm{O}(\sum_{v \in V(M \triangle M')} d(v))$ time. From these, (EM5), (EM7), (EM15) and (EM17) take $\mathrm{O}(\sum_{v \in V(M \triangle M')} d(v)) = \mathrm{O}(\Delta)$ time. If (EM15) and (EM17) are executed, then a type-2 child is generated. Since the number of type-2 children generated over all iterations is $N-1$, the total computation time required for (EM15) and (EM17) is $\mathrm{O}(\Delta N)$.

Let $\mathcal{C}$ be the set of (M, i) of the enumeration tree such that (M, i) is a type-2 child of $p(M, i)$. Consider a graph obtained from the enumeration tree by deleting edges $((M, i), p(M, i))$ for each $(M, i) \in \mathcal{C}$. The graph is composed of paths. We call each of these paths *type-1 child paths*, and use $\mathcal{P}$ to denote the set of all the type-1 child paths. An isolated vertex is also considered to be a type-1 child path. An example of generation of $\mathcal{P}$ is shown in Figure 4. Since any internal vertex has a type-1 child, one of the endpoints of any $P \in \mathcal{P}$ must be a leaf, hence $|\mathcal{P}| = N$. For a path P in the enumeration tree, let $T(P)$ be the total computation time except for (EM15) through (EM17) required by iterations corresponding to vertices in P.

Suppose that $P \in \mathcal{P}$ is composed of maximal matchings $(M_k, k), ..., (M_n, n)$. From the above,

$$\begin{aligned} T(P) &= \mathrm{O}(\sum_{i=k}^{n-1} \sum_{v \in V(M_i \triangle M_{i+1})} d(v)) \\ &= \mathrm{O}(2 \sum_{v \in V(M_n \setminus M_k)} d(v)) \\ &= \mathrm{O}(m) \end{aligned}$$

since any pair of M_i and M_{i+1} satisfies $M_i \subseteq M_{i+1}$. Therefore, the time complexity of this algorithm is $\mathrm{O}(mN)$. ■

In the next section, we reduce the time complexity without modifying the algorithm. We introduce a way of assigning indices to edges to decrease the number of iterations.

3. Reduce the Time Complexity

This section describes a further improvement of our algorithm. In the previous section, we bound the time complexity by $\mathrm{O}(mN)$ since any type-1 child path can have a length up to m. If the mean length of type-1 child paths is smaller than $\Theta(m)$, then we can reduce the time complexity. The lengths of type-1 child paths change with the indices of the edges changes. So, by finding a good ordering of edges, we may obtain a smaller time complexity. Consider an enumeration tree. If any type-1 child path P includes a number of vertices having type-2 children that is proportional to the length of P, then the computation time per type-1 child path can be reduced. Conversely, if several type-1 child paths have subpaths composed of matchings that have no type-2 child, then we may not be able to produce no 'good' analysis. Thus, we introduce an ordering of edges in consideration of these conditions.

Let us look at the following algorithm for generating desired indices of edges. The algorithm takes G as its input, then assigns indices by using a partition $B_1, ..., B_k$ of E which are generated in the computation of the algorithm.

```
Algorithm PUT_INDICES ($G = (V, E)$)
(PI1) $b, b'$ := edges adjacent to each other
(PI2) If no such pair exists then $i := 1$ ; $B_1 := E$ ;
          $K_0 := 0$
(PI3) Else $S := \{b, b'$, and all edges adjacent to $b$ or $b'\}$
(PI4)    $E := E \setminus S$ ; $i :=$ PUT_INDICES ($G$) ; $E := E \cup S$
(PI5)    $b_i := b$ ; $b'_i := b'$ ; $B_i := S$
(PI6) End if
(PI7) $K_i := K_{i-1} + |B_i|$
(PI8) Assign unique indices ranging from $K_{i-1} + 1$
          to $K_i$ to all edges of $B_i$
(PI9) Return $i$
```

Each K_i satisfies $K_i = \sum_{j=1}^{i-1} |B_j|$, hence the edges are assigned unique indices. The indices satisfy the property that the index of any edge $e \in B_i$ is smaller than that of any edge e' of B_j if $i < j$. For any i, we

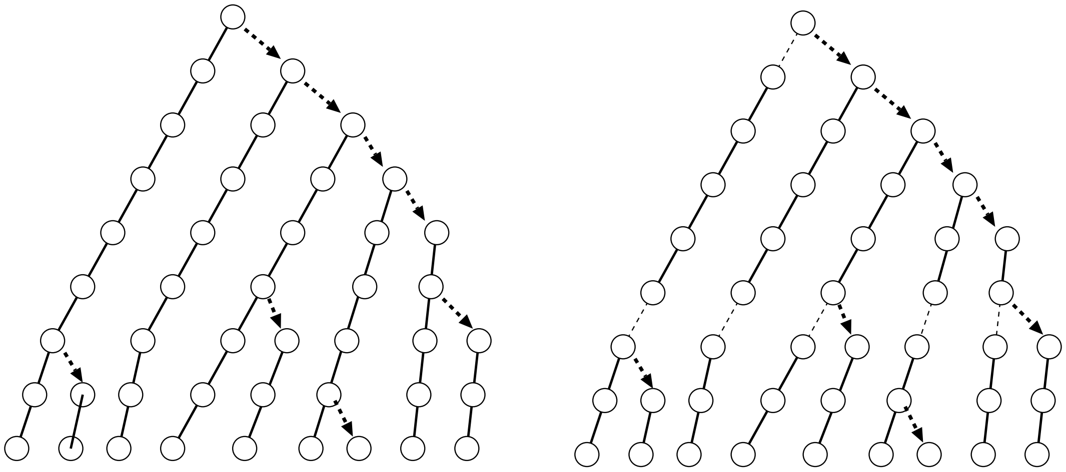

Figure 4: Partitioning the enumeration tree: the left side is partitioned into type-1 paths, and the right side is partitioned to paths of $\hat{\mathcal{P}}$.

have $|B_i| < 3\Delta$ and b_i and b'_i are adjacent to no edge of B_j, for any $j < i$. Since any edge of G is deleted only once by the algorithm, this algorithm takes $\mathrm{O}(m+n)$ time and $\mathrm{O}(m+n)$ memory.

Consider a partition of a type-1 child path obtained by removing edges $((M, K_j+1), p(M, K_j+1))$ for all possible $K'_j s$. Let $\hat{\mathcal{P}}$ be the set of all subpaths obtained by partitioning each type-1 child path. An example of the generation of $\hat{\mathcal{P}}$ is shown in Figure 4. For a path $P \in \hat{\mathcal{P}}$, let the head of P be the vertex of P which is an ancestor of all the other vertices of P. Since B_1 is a matching, any G_i that has $i \le K_1$ has only one maximal matching. Hence, only a path P_0 satisfies the property that the head (M, i) of P_0 satisfies $i < K_2$ among all paths in $\hat{\mathcal{P}}$. When the indices assigned by the algorithm are used, $\hat{\mathcal{P}}$ satisfies the following properties.

Property 1 *For any* $P \in \hat{\mathcal{P}}$, $T(P) = O(\Delta)$.

Proof: Suppose that P is composed of maximal matchings $\{(M_p, p), (M_{p+1}, p+1), ..., (M_q, q)\}$. If $P = P_0$, then the maximum degree of any $G_i, p \le i \le q$ is one, hence $T(P) = \mathrm{O}(p-q+1) = \mathrm{O}(\Delta)$. Consider the case $P \ne P_0$. Since all edges of B_i are adjacent to b_i or b'_i, $M_q \setminus M_p$ includes at most three edges. Hence, from the proof of Lemma 4, the condition can be seen to hold. ∎

Property 2 *For any vertex* $(M, K_i), 1 < K_i < n$, *at least two* K_{i+1}*-maximal matchings are descendants of* (M, K_i).

Proof: Since $K_i > 1$, both b_i and b'_i are defined. Since b_i and b'_i are incident to no edge of B_j for any $j < i$, there are two K_{i+1}-maximal matchings $(M'_1, K_{i+1}), M \cup \{b_i\} \subseteq M'_1$ and $(M'_2, K_{i+1}), M \cup \{b'_i\} \subseteq M'_2$. To obtain the parent of any (M', j), no edge with an index smaller than j is deleted. Hence, for each (M'_j, K_{i+1}), the ancestor $(\hat{M}, K_i)$ of (M'_j, K_{i+1}), which is a K_i-maximal matching, includes all edges of M. Since M is a K_i-maximal matching, $M = \hat{M}$. ∎

Property 3 $\hat{\mathcal{P}}$ *includes at most* $2N$ *paths.*

Proof: We describe a function $f : \hat{\mathcal{P}} \setminus \{P_0\} \to \mathcal{C}$ such that for any $c \in \mathcal{C}$, at most two paths $P \in \hat{\mathcal{P}} \setminus \{P_0\}$ satisfy $f(P) = c$. Note that $|\mathcal{C}| = N - 1$. For any $P \in \hat{\mathcal{P}} \setminus \{P_0\}$, if the head c of P is an element of $\mathcal{C}$, then we define $f(P) = c$. If not, from Property 2, at least one vertex of P has a type-2 child c'. Hence, we define $f(P) = c'$. From this, f is defined on all paths in $\hat{\mathcal{P}} \setminus \{P_0\}$, and at most two paths $P \in \hat{\mathcal{P}} \setminus \{P_0\}$ satisfy $f(P) = c$ for any $c \in \mathcal{C}$. Thus, we have $|\hat{\mathcal{P}}| \le 2N$. ∎

From these properties, we can bound the time complexity of the algorithm.

Theorem 1 *All maximal matchings of a non-bipartite graph* $G = (V, E)$ *can be enumerated in* $O(|E| + |V| + \Delta N)$ *time within* $O(|E| + |V|)$ *memory space where* N *is the number of maximal matchings in* G*, and* Δ *is the degree of the maximum degree vertex of* G.

Proof: From the above properties, we have

$$\begin{aligned}\sum_{P \in \mathcal{P}} T(P) &= \sum_{P \in \hat{\mathcal{P}}} \mathrm{O}(\Delta) \\ &= |\hat{\mathcal{P}}| \mathrm{O}(\Delta) \\ &= \mathrm{O}(\Delta N).\end{aligned}$$

Hence, the time complexity of the algorithm is $\mathrm{O}(\Delta N)$ and the space complexity is $\mathrm{O}(m + n)$. ▮

4. Conclusion

We have considered the problem of enumerating all maximal matchings of a given non-bipartite graph $G = (V, E)$. We have constructed a simple algorithm by improving the algorithm of Tsukiyama et al., and proved that the time complexity of the algorithm is bounded by $\mathrm{O}(\Delta N)$ by assigning indices to the edges in our way. The space complexity of the algorithm is $\mathrm{O}(|E|)$, the same as that of the algorithm of Tsukiyama et al. Here N denotes the number of maximal matchings in the graph, and Δ denotes the maximum degree of G. The second area where we have made improvements is not based on modification of the algorithm, which can be considered interesting from the viewpoint of algorithm engineering. However, the problem of decreasing the time complexity of stable set enumeration is still open. Further research may achieve solid results in this area.

Acknowledgement

We gratefully acknowledge Professor Masakazu Kojima of Tokyo Institute of Technology for giving a variety of comments. We also owe a special debt of gratitude Professor Akihisa Tamura of Kyoto University.

References

[1] Avis D. ; Fukuda K., "Reverse Search for Enumeration," *Discrete Appl. Math.* **65**, pp.21-46, 1996.

[2] Edmonds J., "Paths, Trees and Flowers," *Canadian J. Math.* **17**, pp.1-13, 1965.

[3] Hopcroft J. E. ; Karp R. M., "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *SIAM J. Comp.,* **2**, pp.225-231, 1973.

[4] Johnson D. S. ; Yannakakis M. ; Papadimitriou C. H., "On Generating All Maximal Independent Sets," *Info. Processing Lett.,* **27**, pp.119-123, 1988.

[5] Kapoor H. N. ; Ramesh H., "Algorithms for Generating All Spanning Trees of Undirected, Directed and Weighted Graphs," *Lec. Notes Comp. Sci.,* **519**, Springer-Verlag, pp.461-472, 1992.

[6] Kashiwabara T. ; Masuda S. ; Nakajima K. ; Fujisawa T., "Generation of Maximum Independent Sets of a Bipartite Graph and Maximum Cliques of a Circular-Arc Graph," *J. Algorithms,* **13**, pp.161-174, 1992.

[7] Shioura A. ; Tamura A. ; Uno T., "An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs, " *SIAM J. Comp.,* **26**, No. 3, pp.678-692, 1997.

[8] Tsukiyama S. ; Ide M. ; Ariyoshi H. ; Shirakawa I., "A New Algorithm for Generating All the Maximum Independent Sets," *SIAM J. Comp.,* **6**, pp.505-517, 1977.