Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs

Takeaki UNO
Department of Systems Science, Tokyo Institute of Technology,
2-12-1 Oh-okayama, Meguro-ku, Tokyo 152, Japan. uno@is.titech.ac.jp

Abstract: For a bipartite graph $G = (V, E)$, (1) perfect, (2) maximum and (3) maximal matchings are matchings (1) such that all vertices are incident to some matching edges, (2) whose cardinalities are maximum among all matchings, (3) which are contained in no other matching. In this paper, we present three algorithms for enumerating these three types of matchings. Their time complexities are $O(|V|)$ per a matching.

Introduction

Let $G = (V1 \cup V_2, E)$ be an undirected bipartite graph with vertex sets V_1 and V_2 and edge set $E \subseteq V_1 \times V_2$. We denote the number of vertices and edges by n and m. A matching M of the graph G is an edge set such that no two edges of M share their endpoints. Edges of a matching are called *matching edges*. If a vertex is incident to a matching edge, we say that the vertex is *covered* by the matching, and otherwise *uncovered*. We call a matching with the maximum cardinality among all matchings in G a *maximum matching*, and a matching which is properly contained in no other matching a *maximal matching*. If all vertices of G are covered by a matching M, i.e. M covers all vertices, then we say that M is a *perfect* matching. In this paper, we consider the problems of enumerating all those matchings for a given bipartite graph.

In 1977, Tsukiyama et al. proposed an algorithm for enumerating all maximal stable sets of a general graph [4]. It can be applied for the problem of maximal matchings in bipartite graphs within $O(nm^2)$ time per a maximal matching. For perfect matchings in a bipartite graph, K. Fukuda and T. Matsui proposed an enumeration algorithm [1]. Their algorithm takes $O(n^{1/2}m + mN_p)$ time where N_p is the number of perfect matchings in the given graph. Our main result is to speed up this algorithm. We also obtain algorithms for maximum and maximal matchings by modifying the improved algorithm. They take only $O(n)$ time per a matching.

An Algorithm for Perfect Matchings

In this section, we describe an algorithm for enumerating perfect matchings of bipartite graphs. In our algorithm, we utilize a directed graph $D(G, M)$ defined for a graph G and a matching M. Its vertex set is V, and its arc set is given by orienting edges of M from V_1 to V_2, and the other edges of G in the opposite direction. For any directed cycle or path of $D(G, M)$, edges of M and the other edges appear alternatively in the corresponding cycle or path in G. We call these cycles and paths in G *alternating cycles* and *paths* (see Fig. 1). Especially, we call alternating paths in $D(G, M)$ "feasible" if

their end-vertices are not covered by matching edges outside

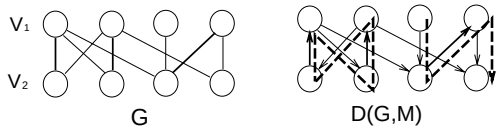

Fig. 1. The graph $D(G, M)$ and a cycle and a path corresponding alternating cycle and feasible path in G.

them. We can obtain a new matching different from M by exchanging matching edges with the other edges along an alternating cycle or a feasible path. On the other hand, the symmetric difference between M and a different matching is composed of some alternating cycles and paths, which correspond to cycles and feasible paths in $D(G, M)$.

A matching generated by an alternating cycle has cardinality equal to M. Since a perfect matching covers all vertices, the symmetric difference between two perfect matchings is composed only of alternating cycles. Hence there is another perfect matching iff there exists an alternating cycle.

The algorithm of [1] utilizes this property. Firstly it checks whether there is a perfect matching in the given graph or not by finding a maximum matching. We spend $O(n^{1/2}m)$ time to do this [2]. After finding the perfect matching M, the algorithm starts enumerating all perfect matchings taking as input the graph G and the matching M. Firstly, it checks whether there is another perfect matching or not. This is done by finding a cycle in $D(G, M)$ by depth-first search with $O(m + n)$ time [3]. If there is no other perfect matching, the algorithm stops. Otherwise, it constructs a perfect matching M' by using the cycle and outputs M'. Next it chooses an edge e included in both M and the cycle. That is, e is included in M but not in the generated perfect matching M'. The algorithm generates two subproblems, to enumerate all perfect matchings including e and not including e. For these subproblems, the algorithm constructs two subgraphs $G^+(e)$ and $G^-(e)$ of G. $G^+(e)$ is the graph obtained by deleting e and both endpoints and all edges adjacent to e. $G^-(e)$ is the graph obtained by deleting e from G. We have one-to-one corresponding between all perfect matchings in $G^+(e)$ and all perfect matchings in G including e, and between all perfect matchings in $G^-(e)$ and all perfect matchings in G not including e. The algorithm solves the subproblems by two recursive calls. One of them inputs a graph $G^+(e)$ and a perfect matching $M \setminus e$ of $G^+(e)$, and the other inputs $G^-(e)$ and M' (see Fig. 2).

Since all of these operations take $O(n)$ time, the bottle neck part of the algorithm on the time complexity is depth-first search with $O(m + n)$ time. Thus, the algorithm takes $O(m)$ time per a perfect matching. The memory complexity is bounded by $O(m)$. In each recursive call, we store the given graph. This requires $O(m)$ space, but by storing only deleted edges when a recursive call occurs, we can reduce the size of the required memory space. Since the number of these deleted edges does not exceed m, the accumulating storing space is bounded by $O(m)$.

Our algorithm is speeded up by adding some improvements to the algorithm. One of these improvements is to trim unnecessary edges which are included in no cycle before

generating a subproblem. Since it treats only arcs included in cycles of $D(G, M)$, we can delete arcs included in no cycle. The arcs included in no cycle of a directed

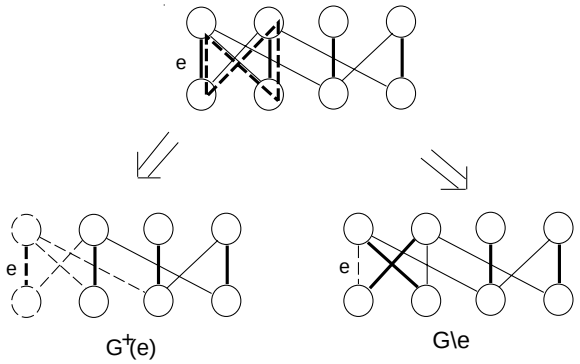

Fig. 2. A perfect matching M' is generated from M by exchanging edges along an alternating cycle. Two subproblems with the graph $G^+(e)$ and M, and $G^-(e)$ and M' are generated. Dotted lines are deleted edges from the graph G.

graph can be found in $O(m+n)$ time by strongly connected component decomposition. The second improvement is that we choose an edge satisfying some good properties to generate subproblems. The choosing criterion of the edge is the key to our algorithm. We show the details later. The whole algorithm may be described as follows.

ALGORITHM Enum_Perfect_Matchings (G)
Step 1: Find a perfect matching M of G and output M. If M is not found, stop.
Step 2: Trim unnecessary edges from G by a strongly connected component decomposition algorithm with $D(G, M)$
Step 3: Call Enum_Perfect_Matchings_Iter (G, M).

ALGORITHM Enum_Perfect_Matchings_Iter (G, M)
Step 1: If G has no edge, stop.
Step 2: Choose an edge e.
Step 3: Find a cycle containing e by a depth-first search algorithm.
Step 4: Find a perfect matching M' by exchanging edges along the cycle.Output M'
Step 5: Trim unnecessary edges from $G^+(e)$.
Step 6: Enumerate all perfect matchings including e by Enum_Perfect_Matchings_Iter with the obtained graph and M.
Step 7: Trim unnecessary edges from $G^-(e)$.
Step 8: Enumerate all perfect matchings not including e by Enum_Perfect_Matchings_Iter with the obtained graph and M'.

Since one iteration still spends $O(m)$ time, the algorithm seems to take $O(n^1/2m + mN_p)$ time. In the next section, we analyze the time complexity more carefully, and bound it by $O(n^1/2m + nN_p)$.

Properties for Bounding the Time Complexity

In this section, we bound the time complexity by proving some properties. To analyze the time complexity, we introduce the *enumeration tree* $\mathcal{T}$ expressing the movement of the algorithm. It is defined for the algorithm and an input, which is the given graph. Each vertex of $\mathcal{T}$ corresponds to a recursive call of the algorithm, and if a recursive call occurs in another one, an edge connects the corresponding vertices. The root of the tree corresponds to the start of the algorithm. The algorithm generates two subproblems if it outputs a perfect matching, thus all internal vertices of $\mathcal{T}$ have exactly two children and have a one-to-one correspondence to all perfect matchings.

Each recursive call corresponding to a vertex x of $\mathcal{T}$ inputs a graph. We denote the input graph by G_x. The time complexity of the algorithm is given by the sum of $|E(G_x)|+|V(G_x)|$ over all vertices of $\mathcal{T}$ where $E(G_x)$ and $V(G_x)$ are the edge set and the vertex set of the graph G_x. To bound it, we will show that if the input graph G has more than $4|V(G)|$ edges, then there is an edge e such that at least $\lceil |E(G)|/4 \rceil$ arcs are included in some cycle of $D(G^+(e), M)$. We will propose an algorithm running in $O(m+n)$ time for finding such edges later. We also show that if all arcs in $D(G, M)$ are included in some cycles, then G has at least $m-n$ perfect matchings. From these facts, for any internal vertex x of $\mathcal{T}$, the child of x corresponding to the recursive call with $G_x^+(e)$ has at least $(|E(G_x)| - |V(G_x)|)/4$ descendants. For each vertex x of $\mathcal{T}$ with $|E(G_x)| \geq 4|V(G_x)|$, we assign its $O(|E(G_x)|)$ computing time to the descendants of its child with $G^+(e)$ uniformly. Since the subproblem with $G_x^+(e)$ exactly less vertices than the original problem, a vertex is assigned $O(1)$ time by at most n ancestors. Thus the sum of assigned time for a vertex is $O(n)$ and the total time complexity is $O(nN_p)$.

We next show proofs of these statements. Through the section, we assume that G contains a perfect matching M and all arcs in $D(G, M)$ are included in some cycles.

Lemma 1. *G contains at least $m - n$ perfect matchings.*

Proof. Let H be a strongly connected component of $D(G, M)$, and T be a depth-first search tree of H. We assume that the search traverses an arc from its tail to head. Since H is strongly connected, T spans all vertices of H. We put indices to all vertices of H by the post-ordering of the depth-first search. In the ordering, a vertex has a larger index than any of its descendants, and for any *non-back arc* of H, the index of its tail is larger than of its head. A *non-back arc* is an arc not in T such that its head is not an ancestor of its tail. An arc not in T such that its head is an ancestor of its tail is called a *back arc*. Note that a back arc has a smaller index on its tail than head.

Let the index of an arc in T be 0, and the index of an arc not in T be the index of its tail. We will show that there are distinct cycles for each arc in $H \setminus T$. For each back arc e of T in H, we can generate a cycle by adding e to T so that e has the maximum index in the cycle. For each non-back arc e of T in H, let P be a directed path from its head to tail. As T is a depth-first search tree, P includes some ancestors of the tail of e (see Fig.3). To see the reason, we claim the following fact.

Claim. A directed path from a vertex v to a vertex u with larger index than v includes a common ancestor of u and v.

Proof. If it does not include, there exists a simple directed path P from v to u not including common ancestors. Let w be the ancestor of v which is the child of the

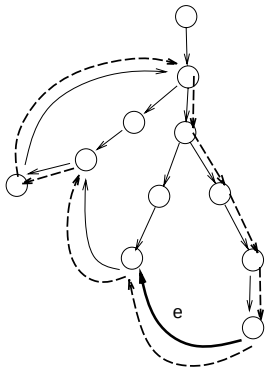

Fig. 3. Dotted lines compose the generated cycle for an arc $e \notin T$ by using T.

nearest common ancestor of v and u. By connecting the path from w to v on T and P, we obtain a path from w to u without any ancestor of w. This contradicts to the depth-firstness. □

From the claim, any path from the head of e to the tail of e includes at least one ancestor of the tail of e. Let v be the first vertex in P to appear among these ancestors. Note that v is always a common ancestor of the head and tail from the claim. We can make a cycle by merging the subpath P' of P from the head of e to v, the path in T from v to the tail of e and e. In this cycle, e is the maximum index arc. We prove this by contradiction. Let us assume that there is an arc $\hat{e}$ in P' with a larger index than e. Then we have some arcs in P' with a larger index on their heads than e and a smaller index on their tails than e. Let e' be the first arc in P' to appear among them. Because of the post-ordering, e' is a back arc. Since $\hat{e}$ is on the behind of e'. the head of e' is not v, and not an ancestor of the tail of e. From the depth-firstness, if the tail of e is not a descendant of an ancestor of the tail of e', the ancestor has a smaller index than the tail of e. Hence it contradicts that the head of e' is not v. Since all arcs in a cycle have distinct tails, the maximum index arc is unique.

Now we have distinct cycles for each arc not in T. Their number is $|E(H)|-|V(H)|$ for all strongly connected components H in $D(G, M)$. Therefore we have $|E(G)| - |V(G)| = m - n$ perfect matchings in G. □

Next, we show that if $m \geq 4n$, then there exists an edge e such that $G^+(e)$ has $\lceil m/4 \rceil$ edges after trimming unnecessary edges. In the rest of this section, we assume that G has at least $4n$ edges. Let $D'(G, M)$ be the graph obtained by contracting all arcs corresponding the edges of M in $D(G, M)$. $D'(G, M)$ has $m - (n/2)$ arcs and $n/2$ vertices. Each vertex of $D'(G, M)$ corresponds to each matching edge, and $D'(G^+(e), M \setminus e)$ is equal to the graph obtained by removing the vertex corresponding to e from $D'(G, M)$. To bound the time complexity, we will obtain a vertex such that $\lceil m/2 - n \rceil \geq \lceil m/4 \rceil$ arcs are included some cycles after removing the vertex. We show the existence of the arc corresponding to the vertex by proving the following lemma.

Lemma 2. *For a strongly connected directed graph $D'(G, M)$, there is a vertex such that the graph obtained by removing v includes at least one third of the arcs of the original graph.*

Proof. To obtain the vertex, we find a depth-first search tree T in $D'(G, M)$ with the root vertex r. Suppose that v denotes the vertex last visited by the search. Then v is a leaf of T. Indices are assigned to all vertices by post-ordering of the search. Let e be a non-back arc whose head u is not an ancestor of v. Since $D'(G, M)$ is strongly connected, there is a directed path P from u to v. From Claim , P must include a common ancestor w of v and u. By merging e, the subpath of P from u to w, and the path from w to the tail of e on T, we obtain a cycle including e. Since the subpath of P and the path on T do not include v, the cycle does not contain v. Therefore, any non-back arc whose head is not an ancestor of v is included in some cycles after removing v. Since any back arc not incident to v is included in some cycle after removing v, the arcs not included in a cycle after removing v are non-back arcs connecting two ancestors of v.

If there are at least $\lceil m/2-n \rceil$ arcs included in cycles after removing v, we obtain a vertex satisfying the condition. We suppose that at least $\lceil m/2+n \rceil$ arcs are included only in cycles containing v in $D'(G, M)$. They are composed of edges of T and non-back arcs connecting two ancestors of v. Let u_1 be the vertex with the largest index among ancestors of v which are heads of some arcs of these non-back arcs. Let P be a path from v to r, and u_2 be the ancestor of v with the largest index included in P except for r. We denote by u the parent of the vertex with the largest index u_1 and u_2. We denote non-back arcs by A which are not incident to u and connecting two ancestor of v. Since only the non-back arcs incident to u, of T and in A may be not included in the cycle, A includes at least $\lceil m/2-n \rceil$ arcs.

Let us consider how many non-back arcs are included in cycles in the graph obtained by removing u. If u is r, then u_2 is a child of r since there is no non-back arc from r to a child of r. By jointing the subpath of P from v to u_2 and the path from u_2 to v, we can obtain a cycle including v and u_2 which is a child of r. Thus we can obtain a cycle including an arc of A by adding it to the cycle. Therefore all arcs of A are included in some cycles in the graph. Otherwise, we have some paths from v to each ancestor of u via P and r in the graph, since u is not included in P. If u is the parent of u_1, then we have some paths from r to u_1 via an arc of A whose head is u_1. If u is the parent of u_2, we have a subpath of P from v to u_2. Therefore we have a path from v to each descendant of u. Since u is an ancestor of u_1, there is a path to v from the tail of any arc of A after removing u. Hence all arcs of A are included in some cycles in the graph. The number of arcs in A is at least $\lceil m/2 \rceil$. At most n arcs can be incident to u, therefore at least $\lceil m/2-n \rceil$ arcs are included some cycles after removing u from $D'(G, M)$. □

From the lemma, we can construct an algorithm for finding such a vertex. We describe the algorithm as follows.

Step 1: Find a depth-first search tree T in $D'(G, M)$ with the root vertex r.
Step 2: Apply a strongly connected component decomposition algorithm for the

graph obtained by removing the last visited vertex v. If $\lceil m/2 - n \rceil$ edges are included in some cycles, output v and stop.

Step 3: Find u_1 and u_2.

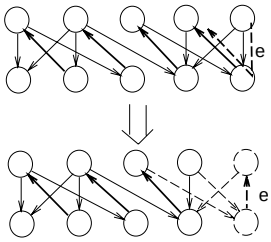

Fig. 4. Exchanging along a length 2 feasible path and generating graph $G^+(e)$. All edges changed their directions are eliminated and no edge of $G^+(e)$ is not changed its direction.

Step 4: Output the vertex nearest from r among parents of u_1 and u_2.

Since each step takes $O(m + n)$ time, the time complexity of the algorithm is $O(m + n)$. By utilizing the algorithm, we obtain the following theorem.

Theorem 1. *Perfect matchings in a bipartite graph can be enumerated in* $O(n^{1/2}m + nN_p)$ *time and* $O(m)$ *space.*

□

Enumerating Maximum Matchings

We have shown an algorithm for finding all perfect matchings in a bipartite graph. In this section, we add some modifications and construct an algorithm for enumerating all maximum matchings in a bipartite graph.

The framework of the algorithm is almost like that for perfect matchings. The algorithm inputs a graph G and a maximum matching M, and checks whether the other maximum matching exists or not. The symmetric difference between two maximum matchings is composed of some cycles and paths with even length. (Since they are maximum, no odd length alternating path is included.) Moreover, if an even length feasible path exists, the subpath of length 2 starting at an uncovered endpoint is also feasible (see Fig. 4). For any length 2 feasible path, one of its endpoints is incident to no matching edge. Conversely any length 2 path in $D(G, M)$ starting at uncovered vertex is feasible. Thus, length 2 paths exist if there are some uncovered vertices.

In our algorithm, we first try to find a cycle in $D(G, M)$ in the same manner as the previous section. If no cycle is found, then we try to find a feasible path. In this algorithm, we also trim unnecessary edges for finding cycles, but these trimmed edges may be included in some maximum matchings. Hence we delete unnecessary edges from only the graph $D(G, M)$. When a recursive call occurs, we remove an edge e, or e and edges adjacent to e from G and $D(G, M)$. (Now, $D(G, M)$ is treated as a variable. $D(G, M)$ is not always given by orienting edges of G.)

If we cannot find any cycle in $D(G, M)$, then we choose an uncovered vertex v, and find a feasible length 2 path P starting at v. It can done in $O(n)$ time. If there is

no such path in G, the algorithm terminates. Let e be the unique edge in $P \setminus M$. After finding the path, we construct a new matching M' by exchanging edges along P, and generate two subproblems with $G^-(e)$ and M, and $G^+(e)$ and M'. Since G includes no alternating cycle, $D(G^-(e), M)$ obviously also includes no cycle. And as the other edge in P is incident to e, no edge in $D(G^+(e), M' \setminus e)$ has the opposite direction from the corresponding edge in $D(G, M)$. Hence $D(G^+(e), M')$ contains no cycle. If there is no cycle in $D(G, M)$, we do not have to trim $D(G, M)$ and to try to find cycles in subproblems. The details of our algorithm are as follows.

ALGORITHM ENUM_MAXIMUM_MATCHINGS (G)

Step 1: Find a maximum matching M of G and output M.

Step 2: Trim unnecessary arcs from $D(G, M)$ by a strongly connected component decomposition algorithm.

Step 3: Call ENUM_MAXIMUM_MATCHINGS_ITER $(G, M, D(G, M))$.

ALGORITHM ENUM_MAXIMUM_MATCHINGS_ITER $(G, M, D(G, M))$

Step 1: If G has no edge, stop.

Step 2: If $D(G, M)$ contains no cycle, Go to **Step 8**.

Step 3: Choose an edge e as the same manner in ENUM_PERFECT_MATCHINGS_ITER.

Step 4: Find a cycle containing e by a depth-first-search algorithm.

Step 5: Exchange edges along the cycle and output the obtained maximum matching M'.

Step 6: Enumerate all maximum matchings including e by ENUM_MAXIMUM_MATCHINGS_ITER with $G^+(e)$, M and trimmed $D(G^+(e), M \setminus e)$.

Step 7: Enumerate all maximum matchings not including e by ENUM_MAXIMUM_MATCHINGS_ITER with $G^-(e)$, M' and trimmed $D(G^-(e), M')$. Stop.

Step 8: Find a feasible path with length 2 and generate a new maximum matching M'. Let e be the edge of the path not included in M.

Step 9: Call ENUM_MAXIMUM_MATCHINGS_ITER $(G^+(e), M', \emptyset)$.

Step 10: Call ENUM_MAXIMUM_MATCHINGS_ITER $(G^-(e), M, \emptyset)$.

The algorithm ENUM_MAXIMUM_MATCHINGS takes $O(n^{1/2}m)$ time. The algorithm ENUM_MAXIMUM_MATCHINGS_ITER takes $O(n)$ time except for trimming in Step 9 and 10. In Step 9 and 10 of each recursive call, we spend time proportional to the size of $D(G, M)$. But in the previous section, we showed that this time can be assigned to their descendants such that the total assigned time for each vertex of the enumeration tree does not exceed $O(n)$. Therefore the total time spent by the algorithm is $O(n)$ time per a maximum matching.

Theorem 2. *Maximum matchings in a bipartite graph can be enumerated in $O(mn^{1/2} + nN_m)$ time and $O(m)$ space, where N_m is the number of maximum matchings in G.* □

Enumerating Maximal Matchings

This last section describes an algorithm for maximal matchings. Those two algorithms in the previous sections are very similar. The algorithm in this section utilizes them, but the enumerating method is not so similar. In the previous problems, we can easily enumerate all perfect or maximum matchings including or not including a chosen edge e. In the problem of maximal matchings, we cannot enumerate such maximal matchings not including e so easily. Hence, we have to utilize some other schemes.

Let us consider two types of maximal matchings, on which covers a vertex v and

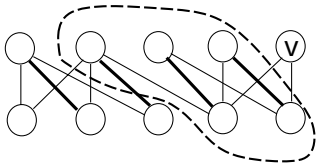

Fig. 5: A maximal matching and a vertex v. The subgraph circled by dotted line is $\hat{G}$.

the other which does not. All maximal matchings covering v include exactly one edge incident to v. All maximal matchings not covering v include no such edge, and from the maximality, all vertices adjacent to v are covered by these maximal matchings. Enumerating all maximal matching covering v is easy. For all edges e incident to v, we enumerate all maximal matchings in $G^+(e)$. For an edge e, a maximal matching in $G^+(e)$ and e forms a maximal matching in G, and conversely all maximal matchings including e contain maximal matchings in $G^+(e)$. Thus we can enumerate all maximal matchings covering v by enumerating all maximal matchings including edges incident to v. For matchings not covering v, let $\hat{G}$ be the subgraph composed of edges incident to vertices adjacent to v, except for edges incident to v (see Fig. 5). Any maximal matching not covering v includes a matching of $\hat{G}$ with the cardinality $d(v)$ where $d(v)$ is the degree of v, the number of vertices adjacent to v. Conversely, for any matching $\hat{M}$ of $\hat{G}$ with cardinality $d(v)$, some maximal matchings in G must include it. Thus, by finding all matchings having cardinality $d(v)$ and enumerating maximal matchings including them, we can enumerate all maximal matchings not including v. To enumerate maximal matchings including $\hat{M}$, we remove all edges and vertices adjacent and incident to edges of $\hat{M}$ from G and enumerate all maximal matchings in the obtained graph.

Now we can enumerate all maximal matchings by finding all maximal matchings covering v and not. We show the details of the algorithm.

ALGORITHM Enum_Maximal_Matchings (G)

Step 1: If all vertices of G have degrees 0 or 1,
output the unique maximal matching of G and stop.

Step 2: Choose a vertex v with degree at least 2.

Step 3: For each edge e in G incident to v, construct $G^+(e)$ and enumerate all maximal matchings including e by recursive calls.
After the recursive call, reconstruct G from $G^+(e)$.

Step 4: Find a maximum matching M in $\hat{G}$. If $|M| = d(v)$, then enumerate all maximum matchings in $\hat{G}$ by Enum_Maximum_Matchings_iter (M, $\hat{G}$).

Step 5: For each matching, enumerate all maximal matchings including it.

The algorithm spends $O(n)$ time in Step 1 and 2, and $O(n)$ time in Step 3 and 5 for a recursive call. We spend $O(d(v)m)$ time to find a maximum matching of $\hat{G}$ in Step 4. Therefore it seems to be $O(d(v)m)$ time algorithm for an output, but we will show that it terminates in shorter time. To bound the time complexity of the algorithm,

we introduce the enumeration tree of the algorithm. Each vertex corresponds to each recursive call, and each leaf of the tree corresponds to each maximal matching. Since we choose a vertex with degree at least 2 in Step 2, any internal vertex of the enumeration tree has at least two children. Therefore the number of internal vertices is less than N_l. where N_l is the number of maximal matchings. Now we show the following property.

Lemma 3. *G contains at least $\lceil (m-n)/2 \rceil$ maximal matchings.*

Proof. Let M be a maximum matching of G. For each edge e not in M, we generate a maximal matching by exchanging e and the matching edges adjacent to e. If the generated matching is not maximal, we add some edge so that it is maximal. The removed and added edges compose an alternating path or an alternating cycle. Since M is maximum, the length of the generated cycles and paths are at most 4. Thus, each matching is generated by at most two edges. Thus there are at least $\lceil (m-n)/2 \rceil$ maximal matchings. □

From the lemma, for each edge e incident to v, we have at least $\lceil (|G^+(e)|-n)/2 \rceil \geq \lceil (m-2n)/2 \rceil$ maximal matchings in $G^+(e)$. Thus we have at least $d(v)\lceil (m-2n)/2 \rceil$ maximal matchings in G. For a vertex x of the enumeration tree corresponding to a recursive call, $d(v_x)\lceil (|E(G_x)|-2|V(G_x)|)/2 \rceil \geq \frac{1}{2}d(v_x)|E(G_x)|$ if $|E(G_x)| \geq 4|V(G_x)|$ where v_x is the vertex chosen at Step 2 in the corresponding recursive call. In this case, we assign $O(d(v_x)|E(G_x)|)$ time to all descendants of it. Each descendant is assigned only $O(1)$ time. In the case that $|E(G_x)| < 4|V(G_x)|$, the algorithm takes $O(d(v_x)|V(G_x)|)$ time in Step 4, and we assign their time to all its children. Since the number of children is at least $d(v_x)$, each children is assigned at most $O(n)$ time by its parent. Since each vertex of the enumeration tree is assigned at most $O(n)$ time by its parent and $O(1)$ time by its at most n ancestors, the total time spent by Step 4 is bounded by $O(nN_l)$ time. We now obtain the following theorem.

Theorem 3. *Maximal matchings in a bipartite graph can be enumerated in $O(n)$ time per a maximal matching and $O(m)$ space.* □

Acknowledgment

We greatly thank Professor Akihisa Tamura of University of Electro-Communications for his kindly advice. We would like to express my sincere thanks to Professor Yoshiko Tamura Ikebe.

References

[1] K. Fukuda and T. Matsui, "Finding All the Perfect Matchings in Bipartite Graphs," Appl. Math. Lett. **7** 1 (1994) 15-18

[2] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," SIAM J. on Comp., Vol. 2: 225-231, 1973.

[3] R. E. Tarjan, "Depth-First Search and Linear Graph Algorithm," SIAM J. Comp. 1, 146-169, 1972.

[4] S.Tsukiyama, M.Ide, H.Ariyoshi and I.Shirakawa, "A New Algorithm for Generating All the Maximum Independent Sets," SIAM J. Comp.,Vol.6, No.3: 505-517,1977.