On the Enumeration of Bipartite Minimum Edge Colorings

Yasuko MATSUI and Takeaki UNO

Abstract. For a bipartite graph $G = (V, E)$, an edge coloring of G is a coloring of the edges of G such that any two adjacent edges are colored in different colors. In this paper, we consider the problem of enumerating all edge colorings with the fewest number of colors. We propose a simple polynomial delay algorithm whose amortized time complexity is $\mathrm{O}(|V|)$ per output, whereas the previous fastest algorithm took $\mathrm{O}(|E| \log |V|)$ time per output. Although the delay of the algorithm is $\mathrm{O}(|E||V|)$, the delay of our algorithm can be reduced to $\mathrm{O}(|V|)$ by using a simple modification with a queue of polynomial size. We show an improvement to reduce the space complexity from $\mathrm{O}(|V||E|)$ to $\mathrm{O}(|E| + |V|)$. Furthermore, we obtain a lower bound $(|E| - |\hat{V}| + 1) \max\{2^{\Delta-3}, 2(|\hat{V}|/2 + 1)^{\Delta-3}/(\Delta - 1)\}/\Delta$ of the number of edge colorings included in G, where Δ is the maximum degree and $\hat{V}$ is the set of vertices of the maximum degree.

Keywords : enumeration, generation, listing, edge coloring, bipartite graph, algorithm, complexity, output polynomial.

1. Introduction

Enumeration problems and enumeration algorithms are quite fundamental in computer science. The subject has a long history, and many studies have been done [5, 7, 9, 14, 16, 18]. Enumeration has many applications in the other area of computer science, such as optimization, sampling, data mining, bioinformatics, and so on. For example, the basis of branch and bound algorithms is the enumeration, and many exact algorithms for NP-hard problems, which are actively studied in these 10 years, use enumeration algorithms. In data mining, the pattern mining algorithms, which finds all the patterns satisfying given constraints from a database, utilize the enumeration of candidate patterns[1]. Particularly, the recent increase of the power of computers supports the efficiency of enumeration approaches in practice.

A weak point of enumeration approach is that there are quite few generalized problems which contain many other enumeration problems as their special cases. For example, in optimization, linear programming is a generalized problem, and it includes many other problems as its special cases, such as maximum matchings, network flow problems, assignment problems. In contrast, only few enumeration problems can be efficiently solved by other enumeration algorithms. For example, if we enumerate all spanning trees in a given graph G by enumerating all subtrees of G, it possibly takes exponential time for each spanning tree, since the number of subtrees is often exponentially larger than the number of spanning trees. If we want to reduce an enumeration problem A to an enumeration problem B, we have to preserve the structures with respect to all the solutions. In contrast to it, in optimization problems, we have to preserve only structures with respect to optimal solutions. Intuitively, this is one of difficulties of the reduction on enumeration problems.

One approach to handle enumeration problems efficiently is to develop fundamental techniques commonly applicable to many enumeration problems. For achieving polynomial algorithms, there are several techniques, such as divide and conquer (binary partition), backtracking, and reverse search. Here an enumeration algorithm is polynomial time if the computation time is polynomial in the input size and the output size of the input problem. For reducing the order of the time complexity of polynomial time algorithms, such as using the sparsity, data structures, and amortized analysis of the time complexity. There are quite many kinds of enumeration problems, thus it is important to develop and summarize efficient techniques. One of the big tasks in the research of enumeration algorithms is to clarify what kind of structures of the problems help to reduce the time complexity, and what kind of techniques can be applicable to the structures. In the literatures, we can see many efficient but simple enumeration algorithms for fundamental graph objects such as paths and cycles[11], spanning trees[7, 16], independent sets and cliques[6], and matchings[4], and fundamental geometical objects such as vertices of polytopes[2], non-crossing spanning trees in plane[2], and floorplans[14].

In this paper, we consider the problem of enumerating all the minimum edge colorings of a given bipartite graph with multiple edges. Let $G = (V(= V_1 \cup V_2), E)$ be a bipartite graph with vertex set V and edge set E. An *edge coloring* of G is a coloring of all the edges of G such that no pair of adjacent edges is colored the same. An edge coloring with the minimum number of colors is called a *minimum edge coloring.* We simply denote a minimum edge coloring by an edge coloring if there is no confusion. We denote the maximum degree of G by $\Delta(G)$, and the set of vertices of maximum degree by $\hat{V}(G)$. If there is no confusion, we simply write them Δ and $\hat{V}$.

In 1916, König [8] proved that any minimum edge coloring of a bipartite graph G uses exactly Δ colors. Since no edges with the same color are adjacent, the set of edges with the same color forms a matching. Hence we can consider an edge coloring as a partition of E into Δ disjoint matchings. The enumeration problem of edge coloring considered here is to output all the ways of partitioning of E into disjoint Δ matchings.

An algorithm for solving the problem has been proposed by Matsui and Matsui [12, 13]. The amortized time complexity of the algorithm is $\mathrm{O}(|E| \log |V|)$ per edge coloring, and the space complexity is $\mathrm{O}(\Delta|E|)$. The current best time complexity algorithm for finding a minimum edge coloring in a given bipartite graph, which is proposed by Cole, et al., and Schrijver [3, 15], takes $\mathrm{O}(|E| \log |V|)$ time, hence naturally we may think that $\Theta(|E| \log |V|)$ is a kind of lower bound for the time complexity. However, the structure of the set of edge coloring seems to have an advantage. For any edge coloring, we can generate another edge coloring by exchanging several edges of two matchings in it, along an alternating cycle. This implies that when we traverse the set of edge colorings, we basically need the time to exchange the edges along a cycle, which seems to be very short in average. Thus, naturally there is a question "Is there an algorithm for enumerating all minimum edge colorings in short time for each".

In this paper, we give a positive answer to the question. We propose a simple algorithm running in $\mathrm{O}(|V|)$ time for each edge coloring. The space complexity is also reduced to $\mathrm{O}(|E| + |V|)$. In detail, the delay[6] of the algorithm is $\mathrm{O}(\Delta|E| \log |V|)$. The delay is the maximum computation time between two consecutive output. Actually, by the technique described in [17], we can reduce the delay to $\mathrm{O}(|V|)$ by using $\mathrm{O}(\Delta|E|)$ extra memory. We note that to output an edge coloring, our algorithm outputs the symmetric difference between an edge coloring and the next one instead of exact output. It reduces the computation time for output one edge coloring to $\mathrm{O}(|V|)$ on average.

The main technique to reduce the time complexity is on the analysis of the time complexity. Actually, our algorithm is obtained by adding slight and simple modifications to the previous algorithm. It uses neither complicated data structures nor sophisticated algorithms. The modifications of the algorithm avoid the worst cases which make the time complexity of the previous algorithm tight. It is interesting that such kind of simple modifications reduces the time complexity so much. This is also an advantage from both theoretical and application viewpoints. Furthermore, as a corollary of the amortized analysis, we give $(|E| - |\hat{V}| + 1) \max\{2^{\Delta-3}, 2(|\hat{V}|/2 + 1)^{\Delta-3}/(\Delta - 1)\}/\Delta$ as a lower bound of the number of edge colorings included in G.

The organization of this paper is as follows. We explain the framework of our enumeration algorithm in Section 2. In Section 3, we analyze the time complexity, and show a lower bound of the number of edge colorings included in a graph. Finally, we explain a way to reduce the space complexity in Section 4.

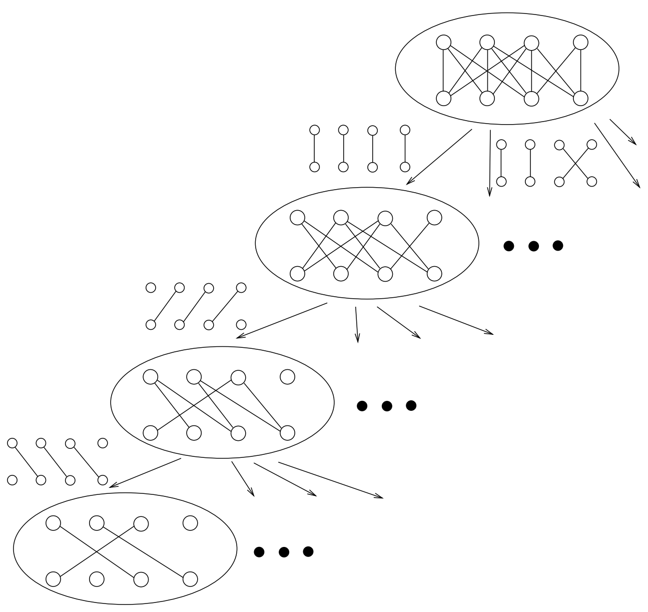

FIGURE 1. An example of enumeration of edge colorings. Each circle denotes a problem/subproblem, and arrows connect problems and their subproblems. Each arrow corresponds to a covering matching which is obtained from each subproblem.

2. Framework of Algorithm for Enumerating Edge Coloring

The basis of our enumeration algorithm is the same as that described in [12, 13]. We start the explanations with the definitions and properties. For a vertex set W, a matching covering all the vertices of W is called a *covering matching* for W. If no confusion can arise we will omit W. From König's theorem, any minimum edge coloring uses exactly Δ colors. This means that any vertex of $\hat{V}$ is incident to edges with all colors, and any matching which forms a minimum edge coloring is a covering matching for $\hat{V}$. Conversely, any covering matching M for $\hat{V}$ is included in at least one minimum edge coloring, since the removal of M from G is a graph with maximum degree $\Delta - 1$.

For an edge e, any edge coloring includes just one covering matching including e. Thus, edge colorings of G is partitioned into groups by the covering matchings including e. This observation immediately leads to the following algorithm.

```
ALGORITHM: BASIC_ALGORITHM ($G = (V, E), C$)
(BA1) If $G$ is a matching then output $C$
(BA2) $e :=$ an edge of $G$
(BA3) For each covering matching $M$ for $\hat{V}$ including $e$
          Call BASIC_ALGORITHM($(V, E \setminus M), C \cup \{M\}$)
```

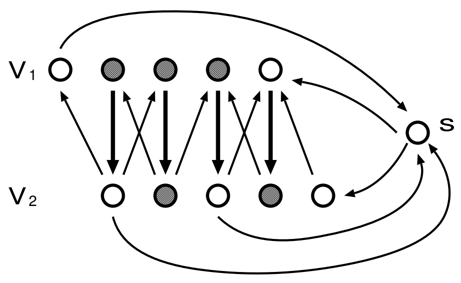

FIGURE 2. An instance of $D_1(G, M, W)$: Gray vertices are in W, and bold edges are in matching M.

Figure 1 shows an example of the execution of the algorithm. The enumeration of covering matchings in (BA3) can be done by using an algorithm proposed in [12, 13]. We explain their algorithm next. For conciseness, we modify their algorithm slightly.

For an edge $e = (u, v)$, let $G \setminus e = (V, E \setminus \{e\})$, and $G - \{u, v\}$ be the subgraph of G obtained by removing u, v, and all edges incident to either u or v. Here we consider the set of covering matchings for a vertex set W. Then, we can see that the set of the covering matchings not including e is equal to the set of covering matchings in $G \setminus e$. Similarly, we can see that the set of the covering matchings for $W \setminus \{u, v\}$ including e is equal to the set of matchings obtained by adding e to each covering matchings in $G - \{u, v\}$. Thus, this enumeration problem can be partitioned into two subproblems.

To partition the problem, we choose an edge e from the symmetric difference between two covering matchings M and M'. This ensures that both subproblems are non-empty, since either M or M' includes e. By augmenting the matching, we can find a covering matching which covers vertices in $\hat{V}$ We thus explain how to find a covering matching different from the given covering matching.

For the explanation, we define the following notation. Let Z be the set of isolated vertices in G. We recall that V_1 and V_2 are the partition of V so that G is a bipartite graph.

$\bar{U}_1$ is the set of vertices in $V_1 \setminus (Z \cup W)$ incident to an edge of M.
$\bar{U}_2$ is the set of vertices in $V_2 \setminus (Z \cup W)$ incident to an edge of M.
U_1 is the set of vertices in $V_1 \setminus (Z \cup W)$ incident to no edge of M.
U_2 is the set of vertices in $V_2 \setminus (Z \cup W)$ incident to no edge of M.

Let $D_1 = (G, M, W)$ be the directed graph obtained from G as follows:

1. remove all isolated vertices from G
2. orient the edges of M from V_1 to V_2, and the edges of $E \setminus M$ from V_2 to V_1
3. add a vertex s to V
4. add the arcs from s to each vertex of $\bar{U}_1 \cup U_2$, and

5. add the arcs from each vertex of $\bar{U}_2 \cup U_1$ to s.

For an example of $D_1(G, M, W)$, see Figure 2. For a directed cycle C of $D_1(G, M, W)$, we define $E(C)$ by the set of all edges corresponding to the arcs of C. From the rule of orienting edges, edges of M and edges not in M appear in $E(C)$ alternatively in any directed cycle. This is a common technique to find alternating paths and alternating cycles.

$D_1(G, M, W)$ is equivalent to the graph obtained by contracting the source and sink of the graph $G'(G, M)$ used in the **find-matching** algorithm of [13]. In [13], the following lemma is proved.

Lemma 1. [13] *The following two conditions hold.*
(1) For any directed cycle C of $D_1(G, M, W)$, $M \triangle E(C)$ is a covering matching for $\hat{V}$.
(2) If $D_1(G, M, W)$ has no directed cycle, then M is the unique covering matching of G. □

Since [13] is not a journal paper, we include a proof here.

Proof. For any two covering matchings M and M', the symmetric difference between them is composed of paths and cycles. Any cycle of those cycles forms a directed cycle in $D_1(G, M, W)$. Any path P of those paths forms a directed path P' of $D_1(G, M, W)$ whose end vertices are not in W. Moreover, since the end edges of P are not adjacent to the edges of M, one end vertex of P is in $\bar{U}_1 \cup U_2$ and the other is in $\bar{U}_2 \cup U_1$. Hence, by connecting s and the end vertices of P', we obtain a directed cycle.

Let C be a directed cycle of $D_1(G, M, W)$. If C does not include s, then $M \triangle E(C)$ is a covering matching, since the sets of vertices incident to the edges of matchings are the same for M and $M \triangle E(C)$. If C includes s, then $E(C)$ is a path connecting a vertex of $\bar{U}_1 \cup U_2$ and a vertex of $\bar{U}_2 \cup U_1$. Hence, $M \triangle E(C)$ is a matching. Since the end vertices of $E(C)$ are not in W, $M \triangle E(C)$ is a covering matching. □

A covering matching for $\hat{V}$ can be found in $O(|E| \log |V|)$ time by the algorithms of Cole et al., and Schrijver [3, 15]. Actually, the computation time can be also bounded by $O(|\hat{V}|^2 \Delta)$, by removing all the edges incident to no vertex of $\hat{V}$ in $O(|E|)$ time. Thus, the enumeration of covering matchings takes $O(|E| + |\hat{V}|^2 \Delta)$ time for the first covering matching, and $O(|E|)$ for each following covering matching.

Our algorithm is obtained by introducing three additional modifications to the basic algorithm. These modifications are the following. We note that a star is a graph such that all the edges are incident to a vertex.

(1) if the input graph is a star, then output the unique edge coloring directly
(2) choose an edge incident to not all the edges in (BA2)
(3) output by the difference from the previous output

The modified algorithm is described below. Note that the algorithm is composed of two procedures, the main part and the enumeration of covering matchings. The procedures are nested so that they call recursively each other. The algorithm memorize a graph with multiple edges by its underline graph with the multiplicity for each edge. Hence, the memory space never exceed $O(|V|^2)$. For outputting an edge coloring of a star, we output its underline graph and the multiplicities, and a message "each edge is a matching", instead of exact output. By this modification, the execution of (1) never take more than $O(|V|)$ time. In the next section we analyze the time complexity of the algorithm to bound it by $O(|V|)$ for each.

```
**ALGORITHM:** ENUM_EDGE_COLORING($G = (V, E)$:graph,
                $Col$:set of matchings to be edge colorings)
(1) **If** $\Delta(G) = 1$, **output** edge coloring $Col \cup \{E\}$
(2) **If** ($G$ is a star) or ($E$ is a matching) **then output** $Col$ and the unique edge
      coloring of $G$      // improvement (1)
(3) $(u, v) :=$ an edge not adjacent to all edges    // improvement (2)
(4) $M :=$ a covering matching for $\hat{V}(G)$ of $G$ including $e$
      // computed in $O(|\hat{V}|^2\Delta)$ time
(5) **Call** ENUM_COVERING_MATCHING ($G - \{u, v\}, \hat{V}(G) \setminus \{u, v\}, M \setminus \{(u, v)\}$, $G$, $Col$)
      // enumerate covering matchings

**ALGORITHM:** ENUM_COVERING_MATCHING($H$:graph, $W$:vertices to be covered,
            $M$:edge set to be a matching, $G$:original graph,
            $Col$:set of matching to be an edge coloring)
(6) **If** no directed cycle is in $D_1(H, M, W)$ **then**
      **Call** ENUM_EDGE_COLORING (($V, E \setminus M$, $Col \cup \{M\}$ )
      // generate recursive call when a new covering matching is found
(7) $C :=$ a directed cycle of $D_1(H, M, W)$ ; $M' := M \triangle E(C)$ ;
      $(u, v) :=$ an edge in $M \triangle M'$
(8) **If** $(u, v) \in M'$ **then swap** $M$ and $M'$
      // now $(u, v)$ is an edge in $M \setminus M'$
(9) **Call** ENUM_COVERING_MATCHING ($H - \{u, v\}), W \setminus \{u, v\}, M \setminus \{(u, v)\}, G, Col$)
      // enumerate covering matchings including $(u, v)$
(10) **Call** ENUM_COVERING_MATCHING ($H \setminus (u, v), W, M', G, Col$)
      // enumerate covering matchings not including $(u, v)$
```

3. Analysis of the Time Complexity

In this section, we now start with some definitions. We define an iteration of the enumeration algorithm of edge colorings by the set of operations to generate subproblems for enumerating edge colorings from an input graph, i.e. the union of the iteration of ENUM_EDGE_COLORING inputing graph G and all the iterations of

ENUM_COVERING_MATCHING which enumerate covering matchings of G. For each iteration x, we denote the input graph of x by $G_x = (V, E_x)$. Iterations generated by x are called the children of x. The depth of the recursion is up to Δ, and each iteration on the bottom of the recursion outputs an edge coloring.

Lemma 2. *For a graph G with $\Delta \geq 3$, a covering matching for $\hat{V}$ including an edge $e = (u, v)$ is unique if and only if all edges of G are adjacent to e.*

Proof. The 'if' part is obvious, thus we prove the 'only if' part by its contraposition. Let M be a covering matching for $\hat{V}$. We show that $D_1(G - \{u, v\}, M, \hat{V} \setminus \{u, v\})$ includes a directed cycle. From Lemma 1, this implies that G has a covering matching different from M. Since at least one edge is not adjacent to e, $G - \{u, v\}$ contains at least one edge. For any vertex w of $\hat{V} \setminus \{u, v\}$, w is incident to at most one of u and v. Hence, its degree in $G - \{u, v\}$ is no less than two, and w has at least one out-going arc in $D_1(G - \{u, v\}, M, \hat{V} \setminus \{u, v\})$. For any vertex w in $V \setminus \hat{V} \setminus \{u, v\}$, there is an arc (w, s) or (w, x) for some $x \neq u, v$ since (w, x) in M. Hence the out-degree of any non-isolated vertex of $D_1(G - \{u, v\}, M, \hat{V} \setminus \{u, v\})$ is at least one. Therefore, a depth-first search for the graph always reaches a vertex that has already been visited, and gives a directed cycle. □

If any edge of G_x is adjacent to all the other edges, then all the edges are incident to a vertex (note that G_x is bipartite), and G_x is a star. In this case the algorithm outputs the unique edge coloring in (1) and the iteration terminates, in $O(|V|)$ time. In the other case, the algorithm chooses an edge e so that at least one edge is not adjacent to e. From Lemma 2, at least two covering matchings includes e. Thus, we have the following corollary.

Corollary 1. *In algorithm* ENUM_EDGE_COLORING*, if an iteration has a child, the number of its children is at least two.* □

This corollary implies that the number of vertices in the enumeration tree is at most twice the number of leaves, which is the same as the number of edge colorings in G. To bound the number of iterations more, we give the following lemmas, where the first one is a standard result of graph theory.

Lemma 3. *For a directed graph $H = (V_H, A_H)$ in which each arc is included in a directed cycle, H contains at least $|A_H| - |V_H| + cc(H)$ directed cycles, where $cc(H)$ is the number of strongly connected components of H.*

Proof. If all the arcs are self-loops or $|V_H| = 1$, the statement holds. Assume that the statement holds if $|V_H| < k$, and we consider the case that $|V_H| = k$ and not all arcs of H are self-loops. Let C be a shortest directed cycle of H including no self-loop, and $|C|$ denote the number of arcs in C. Note that $|C| \geq 2$. Let $H' = (V_{H'}, A_{H'})$ be the graph obtained from H by removing all the arcs of C and contracting vertices of C into a vertex. Since $|V_{H'}| = |V_H| - |C| + 1 < k$, H' has at least

$$|A_{H'}| - |V_{H'}| + cc(H') = (|A_H| - |C|) - (|V_H| - |C| + 1) + cc(H)$$

directed cycles. Note that $cc(H) = cc(H')$. Since H has at least one more directed cycle (which is C) than H', H has at least

$$(|A_H| - |C|) - (|V_H| - |C| + 1) + cc(H) + 1 = |A_H| - |V_H| + cc(H)$$

directed cycles. □

Theorem 1. *Any graph G with $\Delta \geq 3$ and $|\hat{V}| \geq 3$ has at least $(|E| - |\hat{V}| + 1)\max\{2^{\Delta-3}, 2(|\hat{V}|/2+1)^{\Delta-3}/(\Delta-1)\}/\Delta$ edge colorings.*

Proof. Let Z be the set of isolated vertices in G, and M be a covering matching. In $D_1(G, M, \hat{V})$, any non-isolated vertex not in $\hat{V}$ is connected to s, hence its degree is at least 2, and the degree of s is $|V| - |Z| - |\hat{V}|$. Therefore, the number of arcs in $D_1(G, M, \hat{V})$ is at least $|E| + (|V| - |Z| - |\hat{V}|)$. For any edge e, an edge coloring of G includes both a covering matching which includes e and another covering matching which does not. This means that any arc of $D_1(G, M, \hat{V})$ is included in a directed cycle. From Lemma 3, $D_1(G, M, \hat{V})$ includes at least

$$|E| + (|V| - |Z| - |\hat{V}|) - (|V| + 1) + |Z| + 1 = |E| - |\hat{V}|$$

directed cycles. Thus, G includes at least $|E| - |\hat{V}| + 1$ covering matchings.

Let v be a vertex in $\hat{V}$ and F be the set of edges incident to v. Any covering matching includes an edge in F, hence there is an edge f in F such that f is included in at least $(|E| - |\hat{V}| + 1)/\Delta$ covering matchings. Thus, if $\Delta = 3$, G includes at least $(|E| - |\hat{V}| + 1)/3$ edge colorings.

We next consider the case $\Delta > 3$. Similar to the above, for any covering matching M, the graph $H = (V, E \setminus M)$ includes at least $(|E \setminus M| - |\hat{V}| + 1)/(\Delta - 1)$ covering matchings including an edge f, which is incident to a vertex in $\hat{V}(H)$. Since $|E| \geq \Delta|\hat{V}|/2)$, we have

$$(|E \setminus M| - |\hat{V}| + 1)/(\Delta - 1) \geq ((\Delta - 3)|\hat{V}(H)|/2 + 1)/(\Delta - 1).$$

From Lemma 3, if $|\hat{V}| \geq 3$, there is an edge f incident to a vertex in $\hat{V}(H)$ included in at least two covering matchings. Therefore, by induction, G has at least

$$\begin{aligned} &((|E| - |\hat{V}| + 1)/\Delta) \times \prod_{i=3}^{\Delta-1} \max\{2, ((i-1)|\hat{V}|/2 + 1)/i) \\ \geq\ & (|E| - |\hat{V}| + 1)\max\{2^{\Delta-3}, 2(|\hat{V}|/2+1)^{\Delta-3}/(\Delta-1)\}/\Delta \end{aligned}$$

edge colorings. □

In particular, we can directly obtain the following corollary from the fact that $|E| \geq \Delta|\hat{V}|/2$.

Corollary 2. *Any graph G with $\Delta = 3$ has at least $|\hat{V}|/6$ edge colorings.*

From these lemmas and corollary, we obtain the following theorem.

Theorem 2. *The algorithm* ENUM_EDGE_COLORING *enumerates all minimum edge colorings of a bipartite graph $G = (V, E)$ with multiple edges in $O(|V|N)$ time, where N is the number of minimum edge colorings of G.*

Proof. For any child y obtained at iteration x, we have $|E_y| \geq |E_x| - |V|/2$, since E_y is obtained by removing a matching from E_x. An iteration x takes $\mathrm{O}(|E_x| + |V| + |\hat{V}(G_x)|^2\Delta(G_x))$ time and $\mathrm{O}(|E_x| + |V|)$ time per child. By assigning a computation cost of $\mathrm{O}(|E_x| + |V|)$ to each child, we suppose that the computation time $T(x)$ of x is

$$T(x) = \begin{cases} c \times |V| & \text{if } \ \Delta(G_x) < 3 \ \text{ or } \ G_x \text{ is a star} \\ c \times (|\hat{V}(G_x)|^2\Delta(G_x) + |E_x| + |V|) & \text{otherwise} \end{cases} .$$

Here c is a constant, and $T(x)$ does not include the computation time for outputting the edge colorings. We have $|\hat{V}(G_x)| \leq |\hat{V}(G_y)|$, since any vertex of $\hat{V}(G_x)$ is the maximum degree in G_y. Hence, if $\Delta(G_x) > 3$ and G_x is not a star,

$$\begin{aligned} T(x) &= c \times (|\hat{V}(G_x)|^2\Delta(G_x) + |E_x| + |V|) \\ &\leq c \times (|\hat{V}(G_y)|^2(\Delta(G_y) + 1) + |E_y| + 1.5|V|). \end{aligned}$$

Hence, from Corollary 1, for any $4 \leq k \leq \Delta(G)$,

$$\begin{aligned} & \sum_{x|\Delta(G_x)=k, G_x \text{ is not a star}} T(x) \\ \leq & \sum_{y|\Delta(G_y)=k-1} c \times (|\hat{V}(G_y)|^2(\Delta(G_y)+1) + |E_y| + 1.5|V|)/2 \\ \leq & \sum_{y|\Delta(G_y)=k-1} 3T(y)/4. \end{aligned}$$

Therefore, from Lemma 2,

$$\begin{aligned} \sum_{x|\Delta(G_x)\geq 3} T(x) &\leq \sum_{x|\Delta(G_x)=3} 4T(x) \\ &\leq \sum_{x|\Delta(G_x)=3} 48c(|\hat{V}(G_x)||V|) \\ &\leq 48cN|V| \end{aligned}$$

Next we consider the computation time for outputting the obtained edge colorings. Consider the recursive structure of our algorithm as a recursion tree. Our algorithm outputs the difference from the edge coloring output just before. From the edge coloring which is outputted just before, the size of the difference is at most $|V|/2$ times the number of edges in the recursion tree traced by the algorithm. Hence, the sum of these numbers of edges in the output is at most $|V|/2 \times 2(\#$ of iterations). Since the number of iterations is less than $2N$, the computation time for output is $\mathrm{O}(|V|N)$ per output. We note that the algorithm outputs the unique edge coloring in a star in $\mathrm{O}(|V|)$ time. $\square$

4. Reducing Delay

The delay is the maximum computation time between two consecutive outputs. An enumeration algorithm is said to be polynomial delay if its delay is polynomial of the input size[6]. If an enumeration algorithm is polynomial delay, it is output polynomial time, but an output polynomial time algorithm is not always polynomial delay. Thus, polynomial delay is a stronger result than output polynomial. We recall that an enumeration algorithm is output polynomial if the computation time is bounded by a polynomial of the input and output size. In this section, we carefully analyze the delay of our algorithm.

First, we show that the delay of covering matchings enumeration can be bounded by $\mathrm{O}(|E|)$. In [10, 17], they claimed that if an enumeration algorithm outputs a solution in each iteration, then the delay can be 3 times the maximum computation time of an iteration. Algorithm ENUM_COVERING_MATCHING finds a new matching in each iteration, thus we can modify the algorithm to satisfy the condition. The idea in [10, 17] is to modify the algorithm so that the algorithm outputs a solution before executing recursive calls at odd levels of the recursion, and after the recursive calls at even levels. Then, we can see that at least one iteration of any consecutive three iterations must output a solution. Thus, delay is $\mathrm{O}(|E|)$ without increasing neither time nor space complexity.

Next, we reduce the delay of the main algorithm. For an iteration I of an enumeration algorithm, let $Out(I)$ be the set of solutions output by the iterations which are descendants of I. Suppose that the delay of an enumeration algorithm A is D, and satisfies that for any iteration I, the amortized computation time taken by all descendants of I is T per solution in $Out(I)$. In [17], we can see that under these conditions, the delay can be reduced to $\mathrm{O}(T)$ with using $\mathrm{O}(D/T \times S)$ memory where S is the maximum size of output. The main idea is that we make a buffer and insert each solution into the buffer when the algorithm outputs it. The solutions in the buffer is extracted and output one by one with keeping the intermediate computation time equal to $6T$ unless the buffer is overflow. We can prove that after the buffer is once full, it never be empty. At the beginning of the enumeration, we do not extract solutions from the buffer until the first overflow of the buffer. Then, the delay is $6T = \mathrm{O}(T)$.

On our algorithm, $T = \mathrm{O}(|V|)$ and $S = |V|$. Since each iteration of both procedures takes $\mathrm{O}(|E|)$ time, and the depth of the recursion is $\mathrm{O}(\Delta)$, the delay is $\mathrm{O}(\Delta|E|)$. According to the above, we can reduce the delay to $\mathrm{O}(|V|)$ by using $\mathrm{O}(\Delta|E|)$ memory.

Theorem 3. *Minimum edge colorings in a bipartite graph $G = (V, E)$ can be enumerated in $O(|V|)$ delay by using $O(\Delta|E|)$ memory, where Δ is the maximum degree in G.*

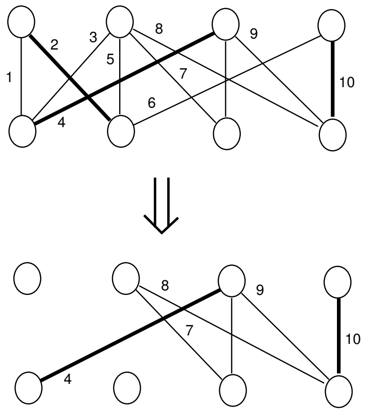

FIGURE 3. An example of G and $G(I \cup M, l)$. In the figure, the edges of $I \cup M$ are shown by the bold lines, and $l = 6$.

5. Reducing Space Complexity

In this section, we describe a way for reducing the space complexity. Since our enumeration algorithm is composed of two nested enumeration algorithms, which are for edge colorings and for covering matchings, the total depth of two recursions can be up to $\Theta(\Delta|E|)$. Note that here the iterations are different from the definition in Section 3. Here we consider that an iteration is the computation time in a recursive call except for the computation done in the further recursive calls generated in it.

Each recursive call requires $\Omega(1)$ memory, hence the total required memory is up to $\Theta(\Delta|E|)$. When ENUM_COVERING_MATCHING generates a subroutine call of ENUM_COVERING_MATCHING, $O(|E| + |V|)$ memory is required to store H in each level, hence the accumulated memory for storing H is up to $O(\Delta(|E| + |V|))$. These two parts are the bottle neck of the space complexity. Note that the total accumulated memory to store M and I is $O(|E|)$, since each time the algorithm executes, stored matchings $M \cup I$ compose a subset of an edge coloring. Note also that $\hat{V}$ can be computed from $\hat{V}$ and I in $O(|E| + |V|)$ time.

To improve these memory-consuming parts, we add the following two modifications to ENUM_COVERING_MATCHING. The first one is to use a loop instead of generating a recursive call with respect to $H \setminus e$. By this modification, a recursive call always adds an edge to I or a matching to C, hence the depth of the recursion is at most $|E| + |V|$.

The second modification is to use the minimum possible index edge e to partition the problem, in ENUM_COVERING_MATCHING. For a graph G, a matching M, and an index j, let $G(M, j) = (V, X)$ where

$$X = \{e_i \in E | i \geq j \text{ and } (e_i \text{ is not adjacent to any edge} e_h \in M, h < j\}$$

For an example of $G(M, j)$, see Figure 3.

Suppose that an iteration of Enum_Covering_Matching inputs a graph H and a covering matching M. Let e_l be the minimum index edge of H. If $e_l \in E(C)$ for a directed cycle C of $D_1(H, M, \hat{V})$, the condition holds for both subproblems generated by e_l. If $e_l \notin E(C)$ for any directed cycle C of $D_1(H, M, \hat{V})$, the condition holds for H and $G(I \cup M, l)$. Therefore, by induction, at any iteration of Enum_Covering_Matching inputting $H, \hat{V}, M$, and I such that the minimum index edge of H is e_l, the set of covering matchings in H and that of $G(I \cup M, l)$ are equal. From this, we can see that a graph equivalent to H can be constructed from $G, I \cup M, l$ in $O(|E| + |V|)$ time.

By using these two modifications, the space complexity is reduced to $O(|E| + |V|)$. We now describe the algorithm with these two modifications.

```
**ALGORITHM:** Enum_Covering_Matching2 $(H, M, \hat{V})$
(1) If $D_1(H, M, \hat{V})$ includes no directed cycle then output $M$ ; return
(2) $e_l(= (u_l, v_l))$ := the minimum index edge of $H$ included in some directed cycle
(3) $H := G(M, l)$    // remove edges less than $l$ and not included in $M$
(4) $C$ := a directed cycle including $e_l$ ; $M' := M \triangle E(C)$
(5) If $e_l \in M$ then swap $M$ and $M'$
(6) Call Enum_Covering_Matching2 $(H - \{u_l, v_l\}, M' \cup \{(u_l, v_l)\}, \hat{V} \setminus \{u_l, v_l\})$
            // $H$ will be changed by the execution of the recursive call
(7) $H := G(M, l)$
(8) Remove $e_l$ from $H$ ; go to (1)
```

Step (1) outputs a covering matching if the covering matching of H is unique. An execution of (2) through (8) corresponds to an internal iteration which has some children in the enumeration tree. e_l is the edge to be used partitioning the problem, step (6) generates a recursive call for enumerating covering matchings including e_l, and step (8) corresponds to the recursive call for enumerating covering matchings not including e_l. Setting H to $G(M, l)$ in (3) is equivalent to removing the edges whose indices are less than l and not included in M. Since the recursive call in (6) changes H, H is not preserved after the termination of the recursive call. However, G is preserved in the execution[1], we can reconstruct H by setting H to $G(M, l)$. This is the key to save the memory for storing H during the execution of the recursive call. Step (8) removes e_l from H and go to the beginning. It corresponds to the recursive call for enumerating covering matchings not including e_l.

By using this algorithm, we obtain the following theorem.

Theorem 4. *The algorithm* Enum_Edge_Coloring *with* Enum_Covering_Matching2 *enumerates all minimum edge colorings of a bipartite graph $G = (V, E)$ with multiple edges in $O(|V|N)$ time and $O(|E| + |V|)$ space, where N is the number of minimum edge colorings of G.* □

[1]In exact, G is changed by recursively calling Enum_Edge_Coloring, however it is reconstructed after the termination by adding the covering matching which is removed before the execution

6. Conclusion

We proposed an algorithm for enumerating all minimum edge colorings in a bipartite graph $G = (V, E)$ without using any sophisticated data structure or any sophisticated algorithm. The amortized time complexity of the algorithm is $\mathrm{O}(|V|)$ per output. It improves the previous algorithm by a factor of $|E| \log |V| / |V|$. We also reduced the space complexity of the algorithm from $\mathrm{O}(|E|\Delta)$ to $\mathrm{O}(|E| + |V|)$. Although the delay of the algorithm is $\mathrm{O}(\Delta|E|)$, we can reduce it to $\mathrm{O}(|V|)$ by using a queue with $\mathrm{O}(\Delta|E|)$ memory. We further give a lower bound $(|E| - |\hat{V}| + 1) \max\{2^{\Delta-3}, 2(|\hat{V}|/2 + 1)^{\Delta-3}/(\Delta - 1)\}/\Delta$ of the number of edge colorings included in G.

Acknowledgments

We would like to thank Professor Akihisa Tamura of Kyoto University and Professor Yoshiko T. Ikebe of Tokyo University of Science for their advice, and Professor David Avis of McGill University for his careful reading of our paper. This research is supported by Grant-in-Aid for scientific research from the ministry of education, science, sports and culture of Japan.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast Discovery of Association Rules," *Advances in Knowledge Discovery and Data Mining,* MIT Press (1996), pp. 307–328, .

[2] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics,* **65** (1996), pp. 21–46.

[3] R. Cole, K. Ost and S. Schirra, "Edge-Coloring Bipartite Multigraphs in $\mathrm{O}(E \log D)$ Time," *Combinatorica* **21** (2001), pp. 5-12.

[4] K. Fukuda and T. Matsui, "Finding All the Perfect Matchings in Bipartite Graphs," *Applied Mathematics Letters* **7** (1994), pp. 15-18.

[5] L. A. Goldberg, "Efficient Algorithms for Listing Combinatorial Structures," *Cambridge University Press*, New York, (1993).

[6] D. S. Johnson, M. Yanakakis and C. H. Papadimitriou, "On Generating All Maximal Independent Sets," *Information Processing Letters*, **27** (1998), pp. 119–123.

[7] S. Kapoor, and H. Ramesh, "An Algorithm for Enumerating All Spanning Trees of a Directed Graph," *Algorithmica* **27** (2000), pp. 120-130.

[8] D. König, "Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlhere," *Math. Ann.* **77** (1916), pp. 453-465.

[9] D. L. Kreher and D. R. Stinson, "Combinatorial Algorithms," *CRC Press*, Boca Raton, (1998).

[10] S. Nakano, T. Uno, "Constant Time Generation of Trees with Specified Diameter," *Lecture Notes in Computer Science* **3353**, Springer-Verlag, (2004) pp. 33-45.

[11] R. C. Read and R. E. Tarjan, "Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees," *Networks* **5**, 237-252 (1975).

[12] Y. Yoshida, Matsui and T. Matsui, "Finding All the Edge Colorings in Bipartite Graphs," *T.IEE. Japan 114-C* **4** (1994), pp. 444-449 (in Japanese).

[13] Y. Matsui, and T. Matsui, "Enumeration Algorithm for the Edge Coloring Problem on Bipartite Graphs," *Lecture Notes in Computer Science* **1120**, Springer-Verlag, (1996), pp. 18-26.

[14] S. Nakano, "Enumerating Floorplans with n Rooms," *Lecture Notes in Computer Science* **1350**, Springer-Verlag, (2001), pp. 107-115.

[15] A. Schrijver, "Bipartite Edge-Colouring in $O(\Delta m)$ Time," *SIAM Journal on Computing* **28** (1999), pp. 841–846.

[16] A. Shioura, A. Tamura, and T. Uno, "An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs, " *SIAM Journal on Computing* **26** (1997), pp. 678-692.

[17] Takeaki Uno "Two General Methods to Reduce Delay and Change of Enumeration Algorithms," *NII Technical Report*, (2004), http://research.nii.ac.jp/TechReports/index.html

[18] H. S. Wilf, "Combinatorial Algorithms : An Update", SIAM, (1989).

Yasuko MATSUI
Department of Mathematical Sciences, Faculty of Science, Tokai University, 1117, Kitakaname, Hiratsuka-shi, Kanagawa, Japan `yasuko@ss.u-tokai.ac.jp`

Takeaki UNO
National Institute or Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, JAPAN.
e-mail: `uno@nii.jp`