A New Approach for Speeding Up Enumeration Algorithms and Its Application for Matroid Bases

Takeaki UNO
Dept. Industrial Engineering and Management, Tokyo Institute of Technology,
2-12-1 Oh-okayama, Meguro-ku, Tokyo 152, Japan. uno@me.titech.ac.jp

Abstract: We propose a new approach for speeding up enumeration algorithms. The approach does not rely on data structures deeply, instead utilizes analysis of computation time. It speeds enumeration algorithms for directed spanning trees, matroid bases, and some bipartite matching problems. We show one of these improved algorithms: one for enumerating matroid bases. For a given matroid $\mathcal{M}$ with m elements and rank n, an existing algorithm runs in $O(T)$ time per base. We improved the time complexity to $O(T/n)$, or $O(T/m(m-n))$.

1 Introduction

Enumeration algorithms for many graph and geometry objects have been developed, and one of the most important and interesting fields of related to these algorithms devoted to making them faster. Although there are numerous algorithms and numerous ways to improve enumeration algorithms, especially for spanning trees and paths, no general technique or framework for their improvements has been proposed. Most enumeration algorithms have been taking advantage of data structures that speed their iterations. So if we can not speed the iterations, we cannot obtain a fast algorithm in the usual way.

In this paper, we describe a new approach, "trimming and balancing" to making enumeration algorithms faster. It is not based on the use of data structures and it speeds enumeration algorithms without speeding iterations. It can thus be used to improve many enumeration algorithms that have not been improved in the usual way. By applying our approach, we can reduce the time complexity of the enumeration algorithm for perfect matchings from $\mathrm{O}(m)$ per matching to $\mathrm{O}(n)$, that of the enumeration algorithm for maximum matchings from $\mathrm{O}(m)$ per matching to $\mathrm{O}(n)$, and that of the enumeration algorithm for directed spanning trees from $\mathrm{O}(n)$ per tree to $\mathrm{O}(\log n)$. Here m and n denote the number of edges and vertices of the given graph. In this paper, we also report an enumeration algorithm for matroid bases that has improved by using the trimming and balancing approach. The problem of enumerating matroid bases includes some other enumeration problems, spanning trees and maximal sets of independent columns of a matrix. There are some studies on enumeration algorithms for these problems, especially spanning trees [2, 1]. Although several algorithms achieve optimal time and space complexities, they require the use of data structures that are not simple. Our Trimming and Balancing algorithms for these problems attain smaller, or at least not larger time complexities without using complicated data structures.

Our approach adds two new phases to an enumeration algorithm, which we call a trimming phase and a balancing phase. An iteration of a modified algorithm may take much more computation time than that of the original algorithm,

but the total time complexity of the modified algorithm is often considerably smaller than that of the original algorithm. Some our algorithms take much more computation time in an iteration than the original algorithm, but it attains smaller time complexities. In the next section, we describe the framework of trimming and balancing. A trimming and balancing algorithm surely terminate in short time, but to prove it is not so easy by using usual analysis of time complexity. Hence we also describe a technique for analyzing time complexities in the next section.

2 Framework of Trimming and Balancing

To explain the concept of trimming and balancing, we use here a simple binary partition algorithm for enumerating of all the bases in the given matroid $\mathcal{M}$. The structure of the algorithm is quite simple. For a given matroid $\mathcal{M} = (E, \mathcal{I})$, we denote the cardinality of its grand set by m and we denote its rank by n. For a base B of $\mathcal{M}$, we denote the elementary circuit of $e \notin B$ by $Cir(B, e)$. We also denote the elementary cut set of an element $e \in B$ by $Cut(B, e)$. For an element e of $\mathcal{M}$, we denote the matroid obtained by contracting e by $\mathcal{M}/e$, and the matroid obtained by deleting e by $\mathcal{M} \setminus e$. To obtain elementary circuits and elementary cuts, the simple algorithm uses an *independent set oracle algorithm*, or an *elementary circuit oracle algorithm*. The former checks whether the given set is independent or not, and the latter outputs the elementary circuit included in the union of the given base and element. We denote the time complexity of these oracle algorithms by $T_{ind}(m, n)$ and $T_{cir}(m, n)$. To contract and delete an element of the matroid, we use matroid subroutines whose time complexity are denoted by $T_{cnt}(m, n)$ and $T_{del}(m, n)$. Assume that the contraction and the deletion of k elements simultaneously also take only $T_{cnt}(m, n)$ and $T_{del}(m, n)$ time.

By using these algorithms, we can obtain a simple binary partition algorithm. For a given matroid $\mathcal{M} = (E, \mathcal{I})$ and its base B, we find a partitioning element $e \in B$ and an element $f \notin B$ such that $B \setminus e \cup f$ forms a base. If there is another base in $\mathcal{M}$, such a pair of elements always exists. We spend $\mathrm{O}((m-n)T_{cir}(m, n))$ or $\mathrm{O}((m-n)nT_{ind}(m, n))$ time to find them. By using e, we divide the problem into two subproblems: enumerating all bases including e and enumerating all those not including e. All bases including e are bases of $\mathcal{M}/e$. All bases not including e are also bases of $\mathcal{M} \setminus e$. Hence they can be enumerated by recursive calls: one inputs $\mathcal{M}/e$, and the other inputs $\mathcal{M} \setminus e$. Thus we obtain an enumeration algorithm simply by using algorithms for constructing these two matroids, and it runs in $\mathrm{O}((m-n)\min\{T_{cir}(m, n), nT_{ind}(m, n)\} + T_{cnt}(m, n) + T_{del}(m, n))$ time. We denote the computation time by $T(m, n)$.

Our trimming and balancing approach reduces the time complexity of enumeration algorithms like this one by adding a trimming phase and a balancing phase. The trimming phase removes unnecessary parts from the input. In the above algorithm considered here, for example, unnecessary parts are elements included in all bases, or included in no base. These elements can be contracted or removed, hence we can transform the problem into a smaller one. The new problem includes many outputs for its size rather than the original problem. We show in a later section that a matroid $\mathcal{M}$ including no unnecessary element contains $\Omega(n(m-n))$ bases. By the trimming phase, the computation time of an

iteration will be not so large for the number of outputs, so the total computation time per output will be small.

The trimming phase does not always decrease the total computation time. Suppose that the algorithm input is a matroid composed of one circuit $\{e_1, ..., e_m\}$. As we can see, no element is unnecessary. Now we suppose that the algorithm chooses the element e_1 as a partitioning element. Since only one base does not include e_1, one of the subproblems terminates immediately. To construct the other subproblem, e_1 is contracted and we obtain a matroid almost identical to the original matroid: one composed of only one circuit with $m-1$ elements. Hence the total computation time will be $\mathrm{O}(T(m, m-1)+T(m-1, m-2)+...+T(2, 1))$. Under the condition that $T(m, n)$ is polynomial, the sum is $O(mT(m, n))$, which is $\mathrm{O}(T(m, n))$ per base.

Why does the worst case running time not decrease? The answer is that the way of partitioning the problem is not good. We therefore reduce the time complexity by adding a balancing phase. It partitions the original problem into "good subproblems" such as subproblems that are not small after trimming by, for example, choosing a good element for partitioning. If we add a balancing phase, the structure of the recursion will be balanced. Hence, if the number of outputs is small for the input size, then the both of the subproblems generated will be small, and the total time complexity will also be small.

The problem of enumerating bases of a circuit can be partitioned into one subproblem which is to enumerate bases including elements $\{e_1, ..., e_{\lfloor m/2 \rfloor}\}$, and another subproblem which is to enumerate bases including $\{e_{\lfloor m/2 \rfloor+1}, ..., e_m\}$. All the bases include $m-1$ elements of $\{e_1, ..., e_m\}$, the problem is surely partitioned. By using this partitioning rule in this case, we can partition the problem into two subproblems of almost equal sizes, thus the total computation time will be reduced to $\mathrm{O}((\log m)T(m, m-1))$ or smaller. In fact, this way of trimming can not attain our result. In later section, we describe the other way of trimming which is utilized in our algorithm for enumerating matroid bases.

Next we show our technique for analyzing the time complexity of enumeration algorithms. Before explaining our analysis, we introduce a virtual tree which is called an *enumeration tree*. Enumeration tree is defined for an enumeration algorithm and an input, and captures the structure of the recursive calls occurring in the algorithm. For a given enumeration algorithm and its input, let $\mathcal{V}$ be a vertex set whose elements correspond to all recursive calls occurring in the algorithm. We consider an edge set $\mathcal{E}$ on $\mathcal{V} \times \mathcal{V}$ such that each whose edge connects two vertices if and only if a recursive call corresponding to one of the vertices occurs in the other. Since the structure of a recursive algorithm contains no circulation, the graph $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ forms a tree. This tree is called an enumeration tree of the algorithm. The root vertex of the tree corresponds to the start of the algorithm. To analyze enumeration algorithms, we show some properties of enumeration trees which are satisfied for any input.

To analyze the time complexity of an enumeration algorithm, we consider the following distribution rule on the enumeration tree. Let $D(x)$ be the number of descendants of a vertex x of $\mathcal{T}$, and $T(x)$ be an upper bound of the computation time on x. Suppose that $\hat{T}$ is an upper bound of $\max_{x \in \mathcal{T}}\{T(x)/D(x)\}$. Our analysis uses an arbitrary constant number α. Using these upper bounds and this constant, we distribute the computation time. The distribution is done

from a parent vertex to its children in the top-down manner. Let $T_p(x)$ be the computation time distributed to x by the parent of x. x has computation time $T_p(x)+T(x)$. We store $\alpha\hat{T}$ of $T_p(x)+T(x)$ on x, and distribute $T_p(x)+T(x)-\alpha\hat{T}$ to the children of x. We have two distribution rules type 1 and 2 as the following. In each case, we distribute the computation time of each child recursively. Type 1 is that each child y receives computation time proportional to the number of the descendants of y. The computation time distributed to y is $(T_p(x)+T(x)-\alpha\hat{T})D(y)/(D(x)-1)$. Type 2 is that each child y receives computation time proportional to the computation time of y. If a child y receives more than $\alpha\hat{T}D(y)$ then we distribute as the same way of type 1. The computation time distributed to y is $(T_p(x)+T(x)-\alpha\hat{T})T(y)/(D(x)-1)$.

By this distribution rule, some vertices may receive much computation time. Thus we define excess vertices as those satisfying $T_p(x)+T(x) > \alpha\hat{T}D(x)$ and we stop the distribution on the excess vertices. The children of an excess vertex receive no computation time from their parent. Note that the above distribution rule is also applied to the descendants of excess vertices. By this new rule, $T_p(x)$ for any vertex x is bounded by $\alpha\hat{T}D(x)$ since the computation time distributed from a parent to its child is proportional to the number of descendants of the child.

After this distribution, no vertex except excess vertices has more than $\mathrm{O}(\hat{T})$ on it. We then distribute the computation time on each excess vertex to all its descendants uniformly. Since the computation time on any excess vertex x is bounded by $(\alpha+1)\hat{T}D(x)$, each descendant receives at most $(\alpha+1)\hat{T}$ time from an excess ancestor. Let X^* be an upper bound of the maximum number of the excess vertices on a path from the root to a leaf. Using X^*, we have an upper bound $\mathrm{O}(\hat{T}X^*)$ of the time complexity per iteration. From these, we obtain the following theorem.

Theorem 1. *An enumeration algorithm runs in $O(\hat{T}X^*)$ time per iteration.* □

Our analysis requires $\hat{T}$ and X^*. To obtain a good upper bound of the computation time, we have to set X^* and $\hat{T}$ to sufficiently good values. As a candidate of $\hat{T}$, we can utilize $\max_{x\in\mathcal{T}}\{T(x)/\bar{D}(x)\}$ where $\bar{D}(x)$ is a lower bound of $D(x)$. Although it is hard to identify excess vertices in the enumeration tree, we can obtain an efficient upper bound. Let x and y be excess vertices such that y is an ancestor of x and no other excess vertex is in the path P_{yx} from y to x in the enumeration tree. Note that P_{yx} has at least one internal vertex.

Lemma 2. *If we use type 2 rule on all vertices, a vertex w of $P_{yx}\setminus y$ satisfies the condition that $T(w) > \frac{\alpha-1}{\alpha}\sum_{u\in C(w)}T(u)$ where $C(w)$ is the set of children of w.*

Proof. We show that all vertices w of $P_{yx}\setminus y$ satisfy the condition that $T_p(w) < (\alpha-1)\hat{T}(w)$ under the assumption that $P_{yx}\setminus y$ includes no such vertex. Any child of y satisfies the condition since y is an excess vertex. Suppose that a vertex w of $P_{yx}\setminus y$ satisfies the condition that $T_p(w') \le (\alpha-1)T(w')$, where w' is the parent of w. From the assumption, we have

$$\begin{aligned} T_p(w) &= ((T_p(w')+T(w')-\alpha\hat{T})T(w)) \;/\; (\textstyle\sum_{u\in C(w')}T(u)) \\ &\le (\alpha T(w')T(w)) \;/\; (\textstyle\sum_{u\in C(w')}T(u)) \\ &\le (\alpha T(w)\tfrac{\alpha-1}{\alpha}\textstyle\sum_{u\in C(w)}T(u)) \;/\; (\textstyle\sum_{u\in C(w')}T(u)) \qquad = \quad (\alpha-1)T(w) \end{aligned}$$

Since $T(w) \leq \hat{T}D(w)$, we have $T_p(w) \leq (\alpha-1)\hat{T}D(w)$. This implies that $T_p(x)+T(x) \leq \alpha\hat{T}D(x)$, and contradicts that x is an excess vertex. □

From this lemma, we can obtain X^* by estimating an upper bound of the number of vertices satisfying this condition in any path from the root to a leaf. Similarly, we can obtain the following corollary.

Corollary 3. *If we use type 1 rule on all vertices and* $\hat{T} = \max_{x\in\mathcal{T}}\{T(x)/\bar{D}(x)\}$, *a vertex* w *of* $P_{yx} \setminus y$ *satisfies that* $\bar{D}(w) > \sum_{u\in C(w)} \frac{\alpha}{\alpha+1}\bar{D}(u)$. □

These conditions can be easily checked, and are often sufficient to analyze. In the next section, we describe a trimming and balancing algorithm for matroid bases. To see trimming and balancing algorithms for directed spanning trees or perfect matchings, refer [3, 4].

3 Trimming and Balancing Algorithms for Matroid Bases

In this section, we describe algorithms for trimming phase and balancing phase of our enumeration algorithm for matroid bases. Our trimming algorithm is quite simple. For a given matroid, if an element of the grand set forms a circuit, then it is included in no base. We call the element a *loop*. In the matroid obtained by deleting all loops, the set of bases is preserved. If the elementary cut of an element is composed of only the element, it is included in all bases. These elements can be contracted. The trimming algorithm contracts or deletes these unnecessary elements. These conditions for all elements can be checked by finding the elementary circuits for all elements outside a base. This can be done in $\mathrm{O}((m-n)\min\{nT_{ind}(m,n), T_{cir}(m,n)\})$.

We next consider the balancing algorithm. By partitioning the given matroid, we obtain two matroids $\mathcal{M}/e$ and $\mathcal{M} \setminus e$. If one is a sufficiently smaller than the other after trimming, the enumeration tree of the algorithm may be biased. Hence we use a balancing algorithm to avoid it. To obtain our balancing algorithm, we use the following properties of $\mathcal{M}/e$ and $\mathcal{M}\setminus e$ for a base B of $\mathcal{M}$.

Property 1 *Suppose that* $e \in B$ *and* B' *is the base of* $\mathcal{M}/e$ *obtained by* $B \setminus e$. *The following conditions hold for* $\mathcal{M}/e$ *and* B'. *(1) For any element* $f \notin B$, *the difference between* $Cir(B,f)$ *and* $Cir(B',f)$ *is either* e *or nothing. (2) For any element* $f \in B \setminus e$, $Cut(B,f) = Cut(B',f)$. □

Property 2 *The following conditions hold for* $e \notin B$. *(1) For any element* $f \notin B$, *the elementary circuits of* f *on* B *in* $\mathcal{M}$ *and of* f *on* B *in* $\mathcal{M}\setminus e$ *are equal. (2) For any element* $f \in B$, *the difference between the elementary cuts* $Cut(B,f)$ *in* $\mathcal{M}$ *and* $Cut(B,f)$ *in* $\mathcal{M} \setminus e$ *is either nothing or* e. □

Using these properties, we construct the balancing algorithm. First we examine the case in which some elements of E are loops in $\mathcal{M}/e$. In this case, we partition the original problem in a way that is based on the following properties.

Property 3 *For two elements* e *and* e' *of* E *such that* $\{e,e'\}$ *is a circuit,* B *is a base including* e *if and only if* $B \setminus e \cup e'$ *is a base.* □

Since $\mathcal{M}$ is trimmed, any loop $\{e'\}$ of $\mathcal{M}/e$ composes a circuit of $\mathcal{M}$ with e. Thus we can see that all bases of $\mathcal{M}$ including a loop e' of $\mathcal{M}/e$ can be

enumerated by enumerating all bases including e. All bases including e' are constructed from the bases including e simply by replacing e with e'.

Suppose that $e_2, ..., e_k$ are loops in $\mathcal{M}/e_1$. We then divide the problem into the subproblems of enumerating bases including each e_i, and of enumerating bases not including any e_i. Since any matroid $\mathcal{M}/e_i$ can be constructed from $\mathcal{M}/e_1$ simply by renaming e_1 to e_i, the time complexity of an iteration does not increase.

Next we examine the case in which some elements of $\mathcal{M} \setminus e$ have elementary cuts consisting of only those elements. The partitioning method we use in this case is based on the following lemma.

Lemma 4. *For a trimmed matroid $\mathcal{M}$, let $e_1, ..., e_k$ be elements satisfying the condition that any base not including e_1 always includes $e_2, ..., e_k$. All bases include at least $k-1$ of these elements.*

Proof. Suppose that there is a base not including e_i and e_j. From the assumption, e_i and e_j are not e_1. Since $\mathcal{M}$ is trimmed, by removing e_1 and adding an element to the base, we obtain a base not including e_1 and e_j, or not including e_1 and e_i. This contradicts the assumption. □

Lemma 5. *Let $e_1, ..., e_k$ be elements satisfying the condition that any base not including e_1 always includes $e_2, ..., e_k$. For any base not including e_1, there is a base obtained by exchanging e_1 and e_i. Conversely, for any i and any base including e_1, there is a base obtained by exchanging e_1 and e_i.*

Proof. To prove this lemma, we show that any circuit includes all e_i, or includes no e_i. Suppose that a circuit C includes e_i and does not include e_j. By removing e_i from C, we can obtain an independent set. Let B be a base including the independent set. From Lemma 4, B includes e_j. Since the matroid is trimmed, there is a base not including e_j. Hence there is an element e' whose elementary circuit includes e_j. Since $Cir(B, e_i)$ is C, e' is not e_i. By exchanging e' and e_j, we obtain a base including neither e_i nor e_j. This contradicts Lemma 4.

We now have that any circuit includes all e_i or no e_i. Hence for any base not including e_1, we have a base obtained by exchanging e_1 and e_i. For any i and any base including e_1, we also have a base obtained by exchanging e_1 and e_i. □

Let us consider that $e_2, ..., e_k$ are included in any base of $\mathcal{M} \setminus e_1$. In this case, we partition the problem into the subproblems of enumerating all bases not including each e_i and of enumerating all bases including all e_i. Since any matroid $\mathcal{M} \setminus e_i$ can be constructed from $\mathcal{M} \setminus e_i$ by renaming e_1 to e_i, each additional subproblem is constructed in O(1) time. We now describe the details of the algorithm as follows. Assume that $\mathcal{M}$ is trimmed.

ALGORITHM: T&B_for_Matroid_Bases ($\mathcal{M}$)
Step 1: Choose an element $e \in B$.
Step 2: Find a base B' not including e.
Step 3: Generate and trim $\mathcal{M}/e$ and $\mathcal{M} \setminus e$.
Step 4: If one of the subproblems loses some elements, partition the problem again by using the balancing algorithm.
Step 5: Solve the subproblems by recursive calls.

This algorithm takes $O(T(m,n))$ time on an iteration. we bound the total time complexity by a quite small order with the use of our analysis in the next section.

4 Bounding the Amortized Time Complexity

To analyze the time complexity, we use the enumeration tree $\mathcal{T}$ of our algorithm. For a vertex x of $\mathcal{T}$, we denote the matroid by $\mathcal{M}_x$. which a recursive call corresponding to a vertex x inputs. Since x corresponds to an iteration, leaves correspond to bases one-to-one, and any internal vertex has at least two children. Hence the number of iterations does not exceed twice the number of bases. We denote the size of the grand set of $\mathcal{M}_x$ by m_x and the rank by n_x.

We first establish a lower bound $\bar{D}(x)$. It is given by the lower bound on the number of bases in a trimmed matroid.

Lemma 6. *A trimmed matroid $\mathcal{M}$ includes at least $(m-n)n$ bases.*

Proof. We prove the lemma by induction. Let B be a base of $\mathcal{M}$. For each element e in B, there is an element not in B whose elementary circuit includes e. By exchanging e and the element, we obtain another base of $\mathcal{M}$. The generated bases are distinct, hence we have at least n other bases in the matroid. For each element e' not in B, we have an elementary circuit including at least one element of B. By exchanging e' and an element of the circuit, we can obtain $m-n$ distinct bases. Hence we have at least $m-n$ bases in the matroid. Therefore the condition of the lemma holds if $n = 1$ or $m - n = 1$.

Assume that the condition of the lemma holds if $m - n < k$ and $n \leq k'$ hold, or $m - n \leq k$ and $n < k'$ hold. Under this assumption, we prove that the condition of the lemma holds for the matroid $\mathcal{M}$ with $m - n = k > 1$ and $n = k' > 1$. Let e be an element of the grand set of $\mathcal{M}$. If the matroids $\mathcal{M}/e$ and $\mathcal{M} \setminus e$ lose no element by the trimming algorithm, $\mathcal{M}$ includes at least $(m-n-1)n + (m-n)(n-1) = 2(m-n)n - m$ bases. Since $m - n$ and $n > 1$, we have $2(m-n)n - m > (m-n)n$.

Let us consider the other cases. If $\mathcal{M}/e$ loses k $(1 \leq k \leq m-n)$ elements by the trimming algorithm, then the balancing algorithm partitions the bases of $\mathcal{M}$ to bases of at least $k+1$ trimmed matroids with $m-k-1$ elements and the rank $n-1$. Since these matroids have at least $(m-k-1-(n-1))(n-1)$ bases from the assumption, $\mathcal{M}$ has at least $(k+1)(m-k-n)(n-1) \geq (m-n)n$ bases.

In the case that $\mathcal{M} \setminus e$ loses k $(1 \leq k \leq n)$ elements, the bases of $\mathcal{M}$ are partitioned to bases of $k+1$ matroids with $m-k-1$ elements and the rank $n-k$. Since all these matroids have at least $(m-k-1-(n-k))(n-k)$ bases by the assumption, $\mathcal{M}$ has at least $(k+1)(m-n-1)(n-k) \geq (m-n)n$ bases. □

Next we set X^* to an upper bound on excess vertices. We set α to 30. From Corollary 3, for X^*, we can use an upper bound of the number of vertices x in a path from the root to a leaf satisfying the condition (a) that $\bar{D}(x) < \frac{\alpha}{\alpha+1} \sum_{u \in C(x)} \bar{D}(u)$. We show which vertices of $\mathcal{T}$ satisfy the condition. We first consider the case that $m_x - n_x = 1$ or $n_x = 1$. In this case, the balancing algorithm generates $m_x - n_x$ or n_x subproblems with only one element in the grand set. This element may be an excess vertex, although the number of this type of excess vertices in the path from the root to a leaf is at most 1.

Next we consider the case that $m_x - n_x, n_x > 1$. If both generated subproblems of x lose no element by the trimming algorithm, the number of bases included in $\mathcal{M}/e$ or $\mathcal{M} \setminus e$ is at least $(m_x - n_x - 1)n_x + (m_x - n_x)(n_x - 1) = 2(m_x - n_x)n_x - m_x$. For a fixed m_x, $(m_x - n_x)n_x$ takes its minimum value $2m_x - 2$ when $n_x = 2$ and $n_x = m_x - 2$. Since $2m_x - 2 \geq m_x/30$ for any $m_x > 2$, we have $(m_x - n_x)n_x \leq \frac{29}{30}(2(m_x - n_x)n_x - m_x)$. Hence x does not satisfy the condition (a). Suppose that x is an excess vertex. Then a subproblem of x loses k elements by the trimming algorithm. Thus we have $\frac{30}{29}(m_x - n_x)n_x > (k+1)(m_x - k - n_x)(n_x - 1)$ or $\frac{30}{29}(m_x - n_x)n_x > (k+1)(m_x - n_x - 1)(n_x - k)$ from the proof of lemma 6. Hence we have $\frac{1.04n_x}{n_x - 1} > \frac{(k+1)(m_x - k - n_x)}{(m_x - n_x)}$, or $\frac{1.04(m_x - n_x)}{m_x - n_x - 1} > \frac{(k+1)(n_x - k)}{n_x}$.

The former condition holds only for $k > m_x - n_x - 4$, and the latter condition holds only for $k > n_x - 4$. Thus any child y of x satisfies $m - n \leq 4$ or $n \leq 4$. The rank and the cardinality of the grand set decrease strictly in the children, hence there are at most 8 the above excess vertices in a path from the root to a leaf. Therefore we can set X^* to 9.

Since $T(x) = \mathrm{O}(T(m_x, n_x))$, we set $\hat{T}$ to $\max_x\{\mathrm{O}(T(m_x, n_x)/\bar{D}(x))\} = \mathrm{O}(T(m,n)/(m_x - n_x)n_x)$. We obtain the following theorem.

Theorem 7. *We can enumerate all matroid bases in $O(T(m,n))$ preprocessing time and $O(T(m,n)/n(m-n)\})$ time per base, if the time complexities of oracle algorithms are depend on the input size of contracted and deleted matroids.* □

5 On Graphic Matroids

In the following sections, we show some applications of our algorithm for some enumeration problems of matroid bases. Here we see the case of graphic matroids. The grand set of a graphic matroid $\mathcal{M}$ is given by the edge set E of an undirected graph $G = (V, E)$, and its independent set is given by a forest of the graph. If the graph is connected, any base is a spanning tree. Hence the problem is that of enumerating all spanning trees of given undirected graphs. There are numerous studies for this problem, and numerous algorithms have been proposed and improved. The time complexities of the enumeration algorithms have been reduced from $\mathrm{O}(|E| + |V|)$ to $\mathrm{O}(|V|)$ and to $\mathrm{O}(1)$ per spanning tree. And their space complexities have been reduced from $\mathrm{O}(|V||E|)$ to $\mathrm{O}(|E| + |V|)$. The methods by which these algorithms were improved relied on some advanced data structures, but our algorithm in this subsection requires neither complicated algorithm nor advanced data structure.

Since any spanning tree of G has $|V| - 1$ edges, we have that the number of elements m is $|E|$ and the rank n is $|V| - 1$. A base of a graphic matroid can be found by a graph search algorithm in $\mathrm{O}(m + n)$ time. The contraction and the deletion of the matroid are also easy. $\mathcal{M}/e$ is given by the graph obtained by contracting the edge e, and $\mathcal{M} \setminus e$ is given by the graph obtained by deleting e. Since any independent set of the matroid includes no cycle, a circuit of the matroid is a cycle in G. Thus an elementary circuit oracle algorithm with $\mathrm{O}(n)$ running time can be constructed.

We thus have $T_{cnt}(m,n) = \mathrm{O}(n)$, $T_{del}(m,n) = \mathrm{O}(m-n)$, and $T_{cir}(m,n) = \mathrm{O}(n)$. The time complexity of an iteration of the trimming and balancing algorithm using these oracle algorithms is bounded by $\mathrm{O}((m-n)n)$. Since all loops are detected in $\mathrm{O}(m)$ time and all elements with elementary cuts composed of

only themselves can be found by the 2-connected component decomposition in $O(m+n)$ time, the time complexity of the trimming algorithm can be reduced to $O(m+n)$. Therefore the trimming and balancing algorithm takes $O(m+n)$ time for preprocessing and $O(\frac{O((m-n)n)}{(m-n)n}) = O(1)$ time per spanning tree. It requires only $O(m+n)$ memory. These complexities are equal to those of the optimal algorithm reported by Shioura et al.[2]. Here we have the following theorem.

Theorem 8. *All bases of a graphic matroid can be enumerated by a trimming and balancing algorithm in $O(m+n+N)$ time and $O(m+n)$ memory where N denotes the number of bases.* □

6 On Linear Matroids

For an m-by-n matrix M with $m > n$, the grand set of the linear matroid is given by the set of column vectors, and an independent set is given by a set of independent column vectors. The independent set family is given by the correction of all the independent sets. If the given matrix is not degenerate, the bases of the linear matroid are all submatrices composed of n independent column vectors. For a given non-singular m-by-n matrix, we consider the problem of enumerating all these n-by-n non-singular submatrices, which are composed of n independent column vectors. For this problem, Y. Matsui [1] proposed an enumeration algorithm with a time complexity of $O(n^2)$ per base.

A base of the linear matroid can be found easily. By some LDL or LU decomposition algorithms, we can obtain a non-singular submatrix B in at most $O(n^2m)$ time. Our algorithm for linear matroids utilizes an elementary circuit oracle algorithm. To obtain a fast elementary circuit oracle algorithm, we consider a linear matroid equivalent to $\mathcal{M}$. The matroid is given by a matrix $U(M,B)$ generated from M as follows. All ith column vectors of B are replaced by the unit vector e_i whose ith element is 1 and other elements are 0. A column vector $x \notin B$ is replaced by the vector y satisfying $By = x$. Since a set of column vectors of M is independent if and only if the set of column vectors corresponding to them is independent in $U(M,B)$, the linear matroid given by $U(M,B)$ is equivalent to $\mathcal{M}$. We enumerate bases of the matroid instead of the bases of $\mathcal{M}$.

As column vectors of B are unit vectors, the elementary circuit of a column vector x can be found easily. The ith column vector of B is in $Cir(B,x)$ if and only if the ith element of x is not 0. In each iteration, we exchange ith column vector of B and the other column vector x and thereby obtain the other base B'. x is generally not a unit vector, thus we have to obtain $U(M,B')$. Since only x is not a unit vector in B', for any column vector z of M, we can obtain a vector y satisfying $B'y = z$ in $O(n^2)$ time. Thus we take $O(n^2(m-n))$ time to obtain $U(M,B')$.

We next show contract and delete operation of an element. The deleting operation is simple, since it involves only the deletion of a column vector. And because in each iteration we delete a column vector outside the base, we do not need to obtain $U(M,B')$. To contract the matroid by the ith column vector x_i, we fix x_i in B and consider only the bases including x_i. Since x_i is included in any base, we can restrict the rest of the column vectors to the linear subspace orthogonal to x_i. Hence we consider the matrix obtained by deleting ith element from all column vectors of M. As the linear matroid given by the matrix is

equivalent to $\mathcal{M}/x_i$, we enumerate the bases of the matroid instead of the bases of $\mathcal{M}/x_i$.

Because each of the above operations takes at most $\mathrm{O}(n^2(m-n))$ time, for an iteration x inputting an m_x-by-n_x matrix, the time complexity is $\mathrm{O}(n_x^2(m_x-n_x))$ Since $n^2(m-n)/n(m-n) = n$, we obtain the following theorem.

Theorem 9. *All bases of the linear matroid given by an m-by-n matrix are enumerated in $O(n^2m)$ preprocessing time and $O(n)$ time per base.* □

7 On Matching Matroids

Matching matroids are given by bipartite graphs. For a bipartite graph $G = (V_1 \cup V_2, E)$, the grand set of a matching matroid is given by V_1, and an independent set of the matroid is given by a set of vertices of V_1 covered by a matching of G. A base B of a matching matroid is obtained by finding a maximum cardinality matching M of G. For a vertex $v \in V_1 \setminus B$, $u \in B$ is in $Cir(B, v)$ if and only if there is an alternating path from v to u in G. An alternating path is a path of G satisfying that edges in M and edges not in M appear in the path alternatively. By exchanging edges of M and the other edges along the alternating path from v to u, we obtain the other maximum matching M' which covers v and does not cover u. To obtain the elementary circuit of v, we find the vertices of the base to which there are some alternating paths from v. We can find them in $\mathrm{O}(m+n)$ time by using a graph search algorithm [3].

To delete a vertex from a matching matroid is easy. It is done by deleting the vertex from G. The contracted matroid is not obtained by reforming G. Instead, we simply mark the vertex to contract, and never output marked vertices when we find elementary circuit. By this modification of the elementary circuit oracle algorithm, we can obtain the contracted matching matroid.

An iteration x takes $\mathrm{O}((|E_x|-|V_x|)(|E|+|V|))$ time where E_x and V_x denote the edge set and vertex set of the input graph of x. Note that E is the edge set of the graph of the original problem. From our analysis, we know that the time complexity of this enumeration algorithm is $\frac{\mathrm{O}((|E_x|-|V_x|)(|E|+|V|))}{|V_x|(|E_x|-|V_x|)} = \mathrm{O}(|E|+|V|)$ per iteration. Therefore we have the following theorem.

Theorem 10. *All bases of a matching matroid given by a bipartite graph $G = (V, E)$ can be enumerated in $O(|E| + |V|)$ time per base.* □

References

1. Y. Matsui, "An Algorithm for Generating All the Bases of Equality Systems ," Research Report, Dept. of Math. Engineering and Info. Physics, University of Tokyo, Tokyo (to appear).
2. A.Shioura, A.Tamura and T.Uno, "An Optimal Algorithm for Scanning All Spanning Trees of Undirected graphs, " SIAM J.Comp.**26**, pp.678-692 (1997).
3. T.Uno, "Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs," LNCS **1350**, Springer-Verlag, pp.92-101 (1997).
4. T.Uno, "A New Approach for Speeding Up Enumeration Algorithms" LNCS **1533**, Springer-Verlag, pp.287-296 (1998)