Speeding Up Enumeration Algorithms with Amortized Analysis

Takeaki UNO
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo 101-8430, Japan. uno@nii.jp

1 Introduction

In this paper, we characterize a class of fast enumeration algorithms and propose a method for constructing algorithms belonging to the class. The characterization is obtained from a new method of amortized analysis. We propose fast algorithms for enumerating matroid bases, directed spanning trees, and bipartite perfect matchings. These are our results in recent four years.

Studies on speeding up enumeration algorithms have a long history. Many output polynomial algorithms (running in polynomial time of input size and output size) and small delay (maximum computation time between two consecutive outputs) algorithms have been proposed. However, there have not been so many studies on reducing time complexity per output. Reducing the time complexity is an important topic in the research of algorithms. Thus, we focus on this topic here.

A typical approach to reduce time complexity is to reduce the time complexity of iterations by using algorithmic techniques, such as binary search and data structures. This approach succeeded for several enumeration algorithms, however, it was not efficient for many others. One of the reason is that usually the structure of an enumeration algorithm is quite simple, and there are no more parts that can be improved. Conversely, the amortized analysis approach is an efficient approach to improve enumeration algorithms. In [1, 2, 3], several such algorithms were proposed for enumerating spanning trees, directed spanning trees, and k-best spanning trees. The time complexities of these algorithms are quite small. The desired results could not be obtained by reducing the time complexity of iterations.

Naturally, it is considered that improvements based on amortized analysis are efficient for enumeration. Thus, we researched a new method for amortized analysis. Then, we developed a method for improving enumeration algorithms with the use of the analyzing method. We explain them in the following.

2 Amortized Analysis and Class of Fast Enumeration Algorithms

For algorithm A and an input, we denote the computation time of an iteration x by $T(x)$ and the child iterations of x by $Chd(x)$. Let $\hat{T}$ be the maximum computation time among the iterations on the bottom level of the recursion. Let T^r be the maximum computation time among all iterations.

To level the computation time of the iterations, we pass the computation time of an iteration to its child iterations in the top-down manner so that each child iteration receives computation time proportional to the computation time of itself. After this, iterations on the bottom level may receive huge computation time from their ancestors, however, if the following conditions hold, it never occurs.

Theorem 1 *(a) The amortized time complexity of A is $O(\hat{T})$ per iteration if there exists a constant $\alpha > 1$ such that $\alpha T(x) \leq \sum_{y \in Chd(x)} T(y)$ for any iteration x of any input.*

Theorem 2 *(b) The amortized time complexity of A is $O(\hat{T} \log T^r)$ per iteration if there exists a constant $\alpha > 0$ such that there exists a partition C_1 and C_2 of $Chd(x)$ such that $\alpha T(x) \leq \sum_{y \in C_i} T(y)$ for any iteration x of any input.*

From the theorems, we can characterize that algorithms having small $\hat{T}$ and satisfying an condition (a) or (b) are fast.

3 Method for Constructing Fast Algorithms and Examples

Next we explain our method for improving enumeration algorithms. The improvement of an algorithm is realized by adding two new phases.

The first phase is called trimming phase. This is to remove or contract unnecessary parts of the input of each child iteration such that the input size will not be large for the number of descendants of the child iteration. For example, if the input size is a polynomial of the number of descendants, then $\hat{T}$ is constant. Conversely, if the input size is large for the number of descendants, then $\hat{T}$ is large, or the algorithm satisfies neither condition (a) nor (b).

The second phase is called balancing phase. This is performed to obtain a good balance of the input sizes of child iterations so that condition (a) or (b) is satisfied. This is done by choosing a good variable to fix, extra processes for special cases, etc.

By using the method, we developed the following fast algorithms. They are based on binary partition, that is, to choose an element e, partition the solutions into those including e and those not including e, and enumerate them recursively. In the following, "basic trimming" means to contract all the element included in all solutions, and delete all the elements included no solution. For an iteration x, we denote its inputs by adding subscript x, such as G_x, and the number of descendants of x by D_x.

Matroid base (proposed in [5])
input: a matroid $\mathcal{M} = (E, \mathcal{I})$ (rank is denoted by n)
output: all the bases of $\mathcal{M}$
(I_x, C_x : computation time of independent oracle and elementary circuit oracle, respectively)
Trimming phase: basic trimming $\Rightarrow \quad D_x = \Omega(n_x|E_x|)$
Balancing phase: If the input size is small enough, then enumerate bases directly.
$\Rightarrow$ Condition (a) is satisfied.
Time complexity: iteration = $O(|E_x| \min\{n_x I_x, C_x\})$, $\hat{T} = O(\min\{I_x, C_x/n_x\})$, total $O(\hat{T})$ per output

Directed spanning tree (proposed in [4])
input: a directed graph $G = (V, E)$ and a vertex r
output: all the directed spanning tees of G rooted at r
Trimming phase: basic trimming $\Rightarrow \quad D_x = \Omega(|E_x|)$
Balancing phase: If the input size of child iteration is small, re-partition the problem.
$\Rightarrow$ Condition (b) is satisfied.
Time complexity: iteration = $O(|E_x| \log |V|)$, $\hat{T} = O(\log |V|)$, total $O(\log^2 |V|)$ per output

Bipartite perfect matching (proposed in [6])
input: a bipartite graph $G = (V, E)$
output: all the perfect matchings of G
Trimming phase: basic trimming and contraction of consecutive degree 2 vertices. $\Rightarrow D_x = \Omega(|E_x|)$
Balancing phase: If the input size of child iteration is small, re-partition the problem.
$\Rightarrow$ Condition (b) is satisfied.
Time complexity: iteration = $O(|E_x|)$, $\hat{T} = O(1)$, total $O(\log |V|)$ per output

The time complexities of the existing algorithm for the problems are $O(\min\{n|E|I, |E|C\})$, $O(|V|^{1/2})$, and $O(|V|)$, respectively. This is evidence of the efficiency of our improving method.

References

[1] D. Eppstein, "Finding the k Smallest Spanning Trees," SWAT '90 Lec. Notes Comp. Sci. **447**, Springer Verlag, pp.38-47 (1990).
[2] S. Kapoor and H. Ramesh, "Algorithms for Enumerating All Spanning Trees of Undirected and Weighted Graphs," *SIAM J. Comp.* **24**, pp. 247–265 (1995).
[3] A. Shioura and A. Tamura, "Efficiently Scanning All Spanning Trees of an Undirected Graph," J. Operation Research Society Japan (1993).
[4] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms," Lect. Note in Comp. Sci. 1533, Springer-Verlag, pp. 287-296, 1998
[5] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms and Its Application for Matroid Bases," Lect. Note in Comp. Sci 1627, Springer-Verlag, pp. 349-359, 1999
[6] T. Uno, "A Fast Algorithm for Enumerating Bipartite Perfect Matchings," Lect. Note in Comp. Sci. 2223, Springer-Verlag, pp. 367-379, 2001