A New Approach for Speeding Up Enumeration Algorithms

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 relies on data structures deeply, but utilizes some analysis of its computation time. The enumeration algorithms for directed spanning trees, matroid bases, and some bipartite matching problems are speeded up by this approach. For a given graph $G = (V, E)$, the time complexity of the algorithm for directed spanning tree is $\mathrm{O}(\log^2 |V|)$ per a directed spanning tree. For a given matroid $\mathcal{M}$, the algorithm for matroid bases runs in $\mathrm{O}(T/n)$ time per a base. Here n denotes the rank of $\mathcal{M}$, and T denotes the computation time to obtain elementary circuits. Enumeration algorithms for matching problems spend $\mathrm{O}(|V|)$ time per a matching.

1 Introduction

For many graph and geometry objects, enumeration algorithms have been developed. Speeding up is one of important and interesting parts of the studies on enumeration algorithms. There are many algorithms and their improvements, especially for spanning trees and paths [1, 2, 3, 4], although neither generalized technique nor framework is proposed for these improvements. Almost all fast enumeration algorithms are improved by speeding up their iterations with some data structures. Hence if we can not speed up iterations, we may obtain no fast algorithm.

In this paper, we propose a new approach "trimming and balancing" for speeding up enumeration algorithms. Our approach is not based on data structures, hence we can apply it to many enumeration algorithms which have not been improved in the existing studies. Our approach adds two new phases to an enumeration algorithm, which are called the trimming phase and the balancing phase. By adding these, an iteration of a modified algorithm may take much computation time than original one, but the total time complexity of the modified algorithm is often considerably smaller than the original. Some our algorithms with the worst case time complexities of an iteration larger than the original one often attain a better upper bound of the time complexity than the original. The time complexity of a trimming and balancing algorithm is not so easy to analyze. We use a technique to analyze time complexities. In the next section, we show the framework of the approach and the technique of our analysis. We also show a trimming and balancing algorithm for the enumeration problem of directed spanning trees.

2 Framework of Trimming and Balancing

At the beginning of this section, we explain the idea of the trimming phase and balancing phase. To explain it, we use a binary partition algorithm for the enumeration problem of directed spanning trees whose root is a specified vertex r. The algorithm is proposed by H.N.Gabow and E.W.Myers [1] in 1978. A directed spanning tree (denoted by DST) is a spanning tree of a digraph satisfying that no its arc shares its head with the other. The algorithm inputs a digraph, and chooses an not "unnecessary" arc a^* (which is called a partitioning arc). An unnecessary arc is an arc included in all DSTs or no DST. The algorithm divides the problem into two subproblems of enumerating all DSTs including a^*, and all those not including a^*. These subproblems can be solved recursively with the graph obtained by removing all the arcs sharing their heads with a^*, and that obtained by removing a^*. If there is only one DST, then all the arcs are unnecessary. Hence the algorithm stops. The algorithm is known to take O($|A|$) time per an iteration, and per an outputted DST. Here we show the details of the algorithm.

```
ALGORITHM: ENUM_DST ($G = (V, A)$ )
Step 1: Find a partitioning arc $a^*$. If there are only unnecessary arcs,
        then output the unique DST and stop.
Step 2: Remove all arcs sharing their heads with $a^*$, and call ENUM_DST($G$).
Step 3: Remove $a^*$ from $G$, and call ENUM_DST ($G$) recursively.
```

The trimming phase removes unnecessary parts from the input. In this algorithm, unnecessary arcs can be removed or contracted, since removals and contractions of these arcs effect no change to the original problem. These removals and contractions reduce the size of the input, thus the reduced input includes many outputs for its size. For example, as we show in the later section, a digraph $G = (V, A)$ including no unnecessary arc ("unnecessary" is characterized in the just above) contains $\Omega(|A|)$ DSTs. By the trimming phase, the computation time of an iteration will be not so large for the number of outputs, hence the total computation time per an output will be small.

The trimming phase decreases the computation time but does not always decrease the computation time of the worst case. Suppose that the algorithm inputs a digraph shown in the Figure 1. As we can see, no arc is unnecessary. Now we suppose that the algorithm choose the arc a as a partitioning arc. Since only one DST includes a, one of the subproblem terminates immediately. To construct the other subproblem, a is removed. By the trimming algorithm, the arc sharing its head with a is contracted. The shape of the obtained graph is same as the original. If all the subproblems occurring in the algorithm choose

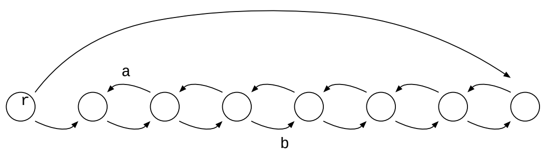

Fig. 1. A digraph for enumeration problem of DSTs

the end-arc as a partitioning arc like the above, the total computation time will be $\mathrm{O}(|A|+(|A|-2)+(|A|-4)+...+4)$. One of the subproblems of them outputs a DST, hence the total computation time is $\mathrm{O}(|A|)$ per a DST.

Why the worst case running time does not decrease? The answer is that the choosing rule of partitioning arcs is bad. To reduce the time complexity, we have to make a good rule. The balancing phase is added for this reason. It chooses a partitioning arc such that both generated subproblems output not so few DSTs. The subproblems have not so many arcs for the number of outputs after the trimming phase, since we have a lower bound of the number of DSTs for its input size. Thus, if the input includes few DSTs like the above example, then the inputs of both subproblems will be small. Moreover, the depth of the recursion is small in this case, hence the total time complexity will be reduced. As we show in the later section, there is an arc such that both generated subproblems have at least $|A|/4$ arcs after the trimming phase. In the figure 1, the arc b is such an arc. The both generated subproblems by b have about $|A|/2$ arcs after the trimming phase. The shapes of the obtained graphs are also like the figured graph. Hence, by choosing partitioning arcs similarly, the depth of the recursion is $\mathrm{O}(\log |A|)$. The total time complexity is also reduced.

The trimming and balancing approach decreases the time complexities of enumeration algorithms, but it is not easy to estimate a good upper bound of those time complexities. There are some difficult algorithms to analyze their time complexities. In this paper, we also show a technique for analyzing the time complexities of enumeration algorithms. We explain it in the following.

Before explaining our analysis, we introduce a virtual tree called an *enumeration tree* of an enumeration algorithm, which expresses the structure of the recursive calls. 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 satisfy for any input.

To analyze the time complexity of an enumeration algorithm, we consider the following distribution rule on the enumeration tree. Let $\mathcal{T}$ be an enumeration tree, and $D(x)$ be the number of descendants of a vertex x of $\mathcal{T}$. Suppose that $T(x)$ is an upper bound of the time complexity of an iteration x. We also suppose that $\hat{T}$ is an upper bound of $\max_{x\in\mathcal{T}}\{T(x)/D(x)\}$. Our analysis uses a parameter T^*. The distribution is done from a vertex to its children in the top-down manner. For a vertex x of the enumeration tree, let $T_p(x)$ be the computation time distributed by the parent of x to x. We distribute $T_p(x)+T(x)-T^*$ of the computation time to its children such that each child receives the computation time proportional to the number of its descendants. We distribute the computation time of each child recursively.

By this distribution rule, some vertices may receive much computation time. Thus we define excess vertices for a specified positive constant $\alpha > 1$, and stop the distribution on the excess vertices. A vertex x is called *excess* if $T_p(x) + T(x) > \alpha\hat{T}D(x)$. The children of an excess vertex receive no computation time from their parent. The distribution rule is also applied to the descendants of excess vertices. By this new rule, $T_p(x)$ is bounded by $\alpha\hat{T}D(x)$ for any vertex x, since the computation time distributed from a parent to its child is proportional to the number of descendants of the child.

After the distribution, no vertex except excess vertices has more than $O(\hat{T}+T^*)$ on it. Next, we distribute the computation time on each excess vertex x to all its descendants uniformly. Since the excess time $T_p(x)+T(x)-T^*$ 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. By using X^* we obtain an upper bound $O(T^*+\hat{T}X^*)$ of the time complexity per an iteration. From these facts, we obtain the following theorem.

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

Our analysis requires $\hat{T}$ and X^*. To obtain a good upper bound of the time complexity, 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 $D(x)$. In the enumeration tree, it is hard to identify excess vertices, although we can obtain an efficient upper bound X^*. 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. *At least one vertex w of $P_{yx}\setminus y$ satisfies the condition that $T(w)$ is larger than the sum of $\frac{\alpha}{\alpha+1}T(u)$ over all children u of w.*

Proof. If $P_{yx}\setminus y$ includes no such vertex, then all vertices w of $P_{yx}\setminus y$ satisfy the condition that $T_p(w)\le\alpha\hat{T}D(w)$. It contradicts to the assumption of the statement. We prove it by induction. Any child of y satisfies the condition since y is an excess vertex. Suppose that a vertex w of $P_{yx}\setminus y$ holds $T_p(w')\le\alpha T(w')$, where w' is the parent of w. Then $T_p(w)\le(\alpha{+}1)T(w')D(w)/D(w')$. From the assumption, $T(w')$ is not greater than the sum of $\frac{\alpha}{\alpha+1}\hat{T}D(u)$ over all children u of w'.

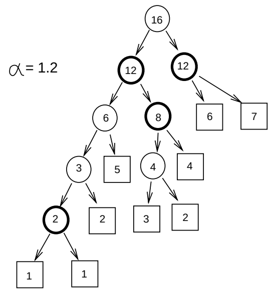

Fig. 2. The enumeration tree and computation time on each vertex: The vertices satisfying the condition of Lemma 1 are drawn by emphasized circles, and all leaves are drawn by rectangles. In this tree, we can set $\hat{T}$ to 7, and X^* to 2.

Since the sum of $D(u)$ is not greater than $D(w')$, we have $T(w') \leq \frac{\alpha}{\alpha+1}\hat{T}D(w')$. Therefore we have $T_p(w) \leq \frac{\alpha(\alpha+1)}{\alpha+1}\hat{T}D(w')D(w)/D(w') = \alpha\hat{T}D(w)$. $\square$

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 1 *If* $\hat{T} = \max_{x\in\mathcal{T}}\{T(x)/\bar{D}(x)\}$, *a vertex* w *of* $P_{yx} \setminus y$ *satisfies that* $\bar{D}(w)$ *is larger than the sum of* $\frac{\alpha}{\alpha+1}\bar{D}(u)$ *over all children* u *of* w. $\square$

These conditions can be easily checked, and are often sufficient to analyze. In the following sections, we describe one of them, that for DSTs. To see the algorithms for perfect matchings, refer [5].

3 Enumerating Directed Spanning Trees

In this section, we consider an enumeration algorithm for DSTs. Our algorithm is obtained by adding a trimming phase and a balancing phase to the algorithm explained in the previous section. We describe these algorithms in the following sections.

3.1 A Trimming Algorithm

Our trimming algorithm removes unnecessary parts of input. Firstly, the algorithm removes all multiple arcs since at most one of multiple arcs can be included in a DST. Next we see characterizations for unnecessary arcs which are included in all DSTs or no DST.

Property 1 *For an arc* (u, v), *there is a DST including it if and only if there is a simple dipath from* r *to* u *not including* v. $\square$

Property 2 *For an arc* (u, v), *there is a DST not including it if and only if there is a simple dipath from* r *to* v *not including* (u, v). $\square$

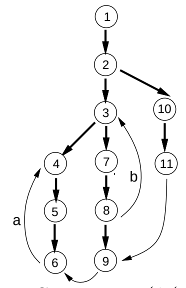

Fig. 3. Back arc a is included in a DST since $i(h(6,6)) = 2$. Back arc b is included in no DST since $i(h(7,7)) = 3$ and $i(h(8,8)) = 7$.

From Property 2, if all arcs not satisfying the former condition are removed, then an arc is not included in a DST if and only if it shares its head with the other arc. Let T be a depth-first search tree of G with the root r. For a non-back arc a of T, the dipath from r to the tail of a in T does not include the head of a. Thus the former condition holds for any non-back arc. In the following, we show a way for checking the condition for back arcs.

Let us put indices $i(v)$ to all vertices v in the order of visiting of the depth-first search. We denote the unique path in T from a vertex u to its descendant v by P_{uv}. For an index i, vertices v and u, we call a dipath from u to v an *i-bypass* if all its internal vertices have indices larger than or equal to i. For a vertex v and an index $i \leq i(v)$, let $h(v,i)$ be the minimum index vertex among vertices satisfying that there are some i-bypasses from the vertices to v. Since any dipath to v from a vertex with an index smaller than v includes a common ancestor of them, $h(v,i)$ is an ancestor of v. Note that an ancestor has an index smaller than any its descendant. By using these notations, we state the following lemmas (see Figure 3).

Lemma 3. *For a back arc (u,v), there is a dipath from r to u not including v if and only if $i(h(w,i(w))) < i(v)$ holds for a vertex w in $P_{vu} \setminus v$.*

Proof. The "if" part of the lemma is obviously. Suppose that there is a simple dipath P from r to u not including v. Then P includes some vertices of $P_{vu} \setminus v$. Let w be the first vertex of them to appear in the path. The subpath of P from r to w includes some ancestors of v. Let w' be the last vertex of them to appear in the subpath. Note that no vertex of $P_{w'w}$ is an internal vertex of the subpath from w' to w of P. Moreover, the subpath includes no vertex u with $i(u) < i(v)$ since any path from u to v includes some common ancestors of u and v, since T is a depth-first search tree. It contradicts the choosing way of w'. □

From this lemma, we can identify unnecessary arcs by using h. Our algorithm firstly obtains $h(v, i(v))$ for the vertex v with $i(v) = n$. In each iteration, we decrease i one by one, and obtain $h(v,i)$ from $h(v,i+1)$ for all v with $i(v) \geq i$. The updating method of h is based on the following properties.

Lemma 4. *Suppose that a vertex u has the index $i-1$.*
(1) $h(u,i-1)$ is the minimum index vertex among all v and $h(v,i)$ satisfying that there are arcs (v,u).
(2) If $h(v,i) \neq h(v,i-1)$ holds for a vertex v, then v is a descendant of u and holds $h(v,i-1) = h(u,i-1)$.

Proof. (1) Let P be an $i-1$-bypass from $h(u,i-1)$ to u. If P is not an arc from $h(u,i-1)$ to u, the vertex v next to u satisfies $h(v,i) = h(u,i-1)$. Thus the condition holds. (2) If $h(v,i-1) \neq h(v,i)$, u is included in any $i-1$-bypass from $h(v,i-1)$ to v. Hence $h(v,i-1) = h(u,i-1)$. Since (a) a dipath from u to a vertex with an index larger than $i-1$ includes a common ancestor of them, and (b) any ancestor of u has an index smaller than $i-1$, v is a descendant of u. Therefore $h(v,i-1) \neq h(v,i)$ holds only for descendants of u. □

From the lemma, we can see that $i(h(v,j)) \leq i(h(v,i))$ for any $j < i$.

Lemma 5. *Let u be a vertex satisfying that $i(u) > i$, and v be a descendant of u. If we have $h(u,i) = h(v,i)$, then $h(u,j) = h(v,j)$ holds for any $j < i$.*

Proof. Since u is an ancestor of v, $i(h(u,j)) \geq i(h(v,j))$ holds for any j. Suppose that an index j satisfies $i(h(u,j)) > i(h(v,j))$. We assume that j is the maximum index among indices satisfying the condition. From this assumption, there is a j-bypass from $h(v,j)$ to v not including u. Note that the bypass includes the vertex v' whose index is j. Since T is a depth-first search tree, any dipath from v' to v includes some common ancestors of v and v'. These ancestors are also ancestors of u, thus we can obtain a j-bypass from $h(v,j)$ to u by merging the j-bypass from $h(v,j)$ to v and $P_{v'u}$. It contradicts the assumption. □

From the lemma, if we have $h(u,i) = h(v,i)$ for u and its child v, the equation holds for all $j < i$. Hence, in the graph obtained by contracting u and v, h is preserved. Thus we contract them if such a vertex and a child exist. For a vertex u, if a descendant v of u satisfies that $i(h(v,i(u)+1)) > i(h(v,i(u)))$, the child v' of u included in the dipath from u to v satisfies the condition $i(h(v',i(u)+1)) > i(h(u,i(u)))$. Hence v' and u satisfy that $i(h(v',i(u))) = i(h(u,i(u)))$. Therefore, in each iteration with the index i, we find the child v' of the vertex u with $i(u) = i$ which maximizes $i(h(v',i))$ among all children of u. If v' satisfies $i(h(v',i)) < i(h(u,i))$, all descendants v of u satisfies $i(h(v,i)) < i(h(u,i))$. Otherwise, we have that $h(v',i) = h(u,i)$, hence we contract u and v'. We do this operation until there is no child v satisfying $i(h(v,i)) = i(h(u,i))$. After contracting all these vertices, no descendant v of u satisfies that $h(u,i) \leq h(v,i+1)$. Therefore no vertex v satisfies $h(v,i) \neq h(v,i+1)$ in the contracted graph. Hence we can update all h by this contracting operation. Since using some binary trees, the total time complexity of the trimming algorithm is $O(|A| \log |V|)$.

3.2 A Balancing Algorithm

For an arc a, let G' be the graph obtained by deleting all arcs sharing their heads with a except for a. Under the condition that G is a trimmed graph, the

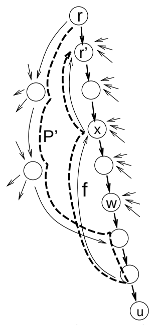

Fig. 4. Vertices r, r', u, x and w. The paths are P_{ru} and P'.

following lemma holds. Suppose that a dipath from a vertex to an arc is a simple dipath from the vertex to the head of the arc which includes the arc.

Lemma 6. *Let P be a dipath from r to a. If an arc e is included in all DSTs or no DST of G', then the head of e is on P.*

Proof. We state the lemma by proving its contraposition. For an arc e whose head is not on P, there exists a dipath Q in G from r to e, because of Property 1. If Q includes no arc of $G \setminus G'$, then it is also included in G'. Thus e is included in some DSTs of G'. Otherwise, we consider the subgraph given by $Q \cup P \cap G'$. The subgraph includes a simple dipath from r to e, since the head of e is not on P. Thus, only arcs which have their heads on P may be included in no DST of G'.

Since G is trimmed, an arc e whose head is not on P shares its head with some other arcs. The arcs are also included in some DSTs of G'. Hence they are also not included in a DST of G'. Therefore any arc included in all DSTs of G' or no DST of G' has its head on P. □

Let a' be an arc sharing its head with a, and P' be a dipath from r to a'.

Lemma 7. *If an arc e is included in all DSTs or no DST of $G \setminus a$, then its head is on P'. In the case that at least two arcs share their heads with a, for any arc e of the arcs, there are both DSTs including e and not including e.*

Proof. $G \setminus a$ includes the graph obtained by deleting all arcs sharing their heads with a' except for a'. Thus, for any arc whose head is not on P', there are both DSTs including the arc and those not including the arc in $G \setminus a$ from Lemma 6. Therefore the first assertion is proved.

For any arc of $G \setminus a$ sharing its head with a', a dipath from r to it does not include a. Hence, from Property 1, the arc is included in some DSTs of G'. Except for the case that only two arcs a and a' have their heads on v, there are some DSTs not including the arc in $G \setminus a$. □

By using these lemmas, we select a partitioning arc as follows. Let the weight of a dipath be the number of arcs whose heads are on the path. The weight of a dipath P is denoted by $w(P)$. In the balancing phase, we will find an arc a^* satisfying the conditions that (a) the weight of a dipath from r to a^* is at most $3|A|/4$, and (b) for an arc a' sharing its head with a^*, the weight of a dipath from r to the tail of a' is at most $|A|/2$. If an arc a^* satisfies these conditions, $G \setminus a^*$ includes at most $|A|/2 + 2$ arcs which are included in all DSTs or no DST. Under the condition that $|A| \geq 8$, $|A|/2 + 2 \leq 3|A|/4$. The graph obtained by removing all arcs sharing whose heads with a^* also includes at most $3|A|/4$ such arcs. We consider the method for finding such an arc a^*.

Let T be a DST of G. To select an arc, we consider the following three cases. (1) If there is a non-back arc (u, v) of T such that $w(P_{ru}) \leq |A|/2$ and $w(P_{rv}) \leq |A|/2$, then the arc of T whose head is v satisfies the above conditions. (2) In the other case, let u be the vertex maximizing $w(P_{ru})$ among all vertices. We denote the vertex next to r in P_{ru} by r'. Let P' be a simple dipath from r to r' not including the arc (r, r'). P' always exists since G is a trimmed graph. Let w be the vertex minimizing $w(P_{rw})$ among all vertices v of P_{ru} satisfying that $w(P_{rv}) > |A|/2$. We suppose that x denotes the first vertex to appear in P' among vertices v of P_{ru} with $w(P_{rv}) \leq |A|/2$. Since any vertex except r is the head of at least two arcs, at least $2|V| - 4$ arcs of G have their heads not on w. On the other hand, at most $|V| - 1$ arcs have their heads on w, thus the number of arcs whose heads are w does not exceed $|A|/2$. Therefore the weight of r' is at most $|A|/2$, and x always exists in P'. Let f be the arc of P' whose head is x. We show an example of these vertices and arcs in Figure 3. If the subpath P'' of P' from r to f satisfies $w(P'') \leq |A|/2$, then the arc of T sharing its head with f satisfies the conditions (a) and (b).

(3) If f does not satisfies these conditions, $w(P'') > |A|/2$. From the definition of x, the vertices included in both P'' and P_{rw} are at most r and w. Hence P'' includes w since more than $|A|/2$ arcs have their heads on P_{rw}. Suppose that P_1 denotes the path with the smaller weight among P_{uw} and the subpath of P'' from r to w. We also denote the other path by P_2. Let d denote the number of arcs whose heads are w. Since $P_1 \cap P_2 = \{r, w\}$, we have $w(P_1) \leq (|A| - d)/2 + d = |A|/2 + d/2 \leq 3|A|/4$, and $w(P_2 \setminus w) \leq |A|/2$.

From the above observations, there is an arc satisfying the above two conditions in any trimmed digraph. To find such an arc, what we have to do is only to find a DST and some dipaths. They can be done in $\mathrm{O}(|A| + |V|)$ time.

By utilizing these algorithms, we obtain a trimming and balancing algorithm. The algorithm takes $\mathrm{O}(|A| \log |V|)$ time for an iteration in the worst case. In the following subsection, we bound the time complexity more tightly by using the distribution of computation time.

3.3 Bounding the Total Time Complexity

To estimate an amortized time complexity of our algorithm, we firstly estimate a lower bound of the number of descendants of a vertex of the enumeration tree by the following lemma. Let G_x denote the graph which an iteration x inputs.

Lemma 8. *If any arc of G is included in a DST and not included in the other DST, G includes at least $|A|/2$ distinct DSTs.*

Proof. Let T be a DST of G. From Property 1, for any non-tree arc a of T, there is a simple dipath P from r to a. Hence, for any non-tree arc, we can construct its own DST T' by adding P to T and deleting some arcs. Since there are at least $|A|/2$ non-tree arcs in G, G includes at least $|A|/2$ DSTs. □

From the lemma, we obtain a lower bound $\bar{D}(x) = |E(G_x)|/2$ of $D(x)$. Here $D(x)$ is the number of descendants of a vertex x of the enumeration tree, and $E(G_x)$ denotes the arc set of G_x. The computation time on a vertex x is $\mathrm{O}(|E(G_x)| \log |V|)$ time, hence we set $\hat{T} = \mathrm{O}(\log |V|)$, and $T^* = \mathrm{O}(\log |V|)$.

From Corollary 1, we can bound the number of excess vertices by the number of vertices satisfying that $\bar{D}(x) > \frac{\alpha}{\alpha+1}(\bar{D}(x_1) + \bar{D}(x_2))$ where x_1 and x_2 are the children of x. If the condition holds, we have that $|E(G_x)|/2 > \frac{\alpha}{\alpha+1}(|E(G_{x_1})|/2 + |E(G_{x_2})|/2)$, thus $|E(G_{x_i})| \le \frac{\alpha+1}{\alpha}|E(G_x)| - |E(G_x)|/4$ for each child x_i. By setting $\alpha = 8$, $|E(G_{x_i})| \le (7/8)|E(G_x)|$. Therefore any path of the enumeration tree from the root to a leaf has at most $\log_{8/7} |A|$ excess vertices. We obtain an upper bound $X^* = \log_{8/7} |A|$. From the Theorem 1, we have the following theorem.

Theorem 9. *All DSTs in a digraph can be enumerated in $O(|A| \log |V| + |V| + N \log^2 |V|)$ time and $O(|A| + |V|)$ space where N is the number of DSTs.* □

Acknowledgments

We greatly thank to Associate Professor Akihisa Tamura of University of Electro-Communications and all the members of the research group "myogadani-club" for their kindly advise.

References

1. H.N.Gabow and E.W.Myers, "Finding All Spanning Trees of Directed and Undirected graphs," SIAM J.Comp.**7**, pp.280-287 (1978).
2. H.N.Kapoor and H.Ramesh,"Algorithms for Generating All Spanning Trees of Undirected, Directed and Weighted Graphs," LNCS.**519**, Springer-Verlag, pp.461-472 (1992).
3. 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).
4. T.Uno, "An Algorithm for Enumerating All Directed Spanning Trees in a Directed graph," LNCS **1178**, Springer-Verlag, pp.166-173 (1996).
5. T.Uno, "Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs," LNCS **1350**, Springer-Verlag, pp.92-101 (1997).