ISSN 1346-5597

National Institute of Informatics

NII Technical Report

Two General Methods to Reduce Delay and Change of Enumeration Algorithms

Takeaki Uno

NII-2003-004E
Apr.2003

URL: http://www.nii.ac.jp/ Tokyo 101-8430, JAPAN +81-3-4212-2000 FAX: +81-3-3556-1916

Two General Methods to Reduce Delay and Change of Enumeration Algorithms

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

Abstract: Enumeration and generation algorithms are for outputting all the elements of a given set just once for each. There have been numerous studies on developing algorithms for various problems. One goal of this research is to derive exact constant time algorithms, i.e., where the computation time between any two consecutive outputs is constant. Although are many amortized constant time algorithms, there are only few exact constant time algorithms in the literature. In this paper, we give simple ways to improve these algorithms so that they become exact constant time algorithms.

Keywords : enumeration, generation, algorithm, speeding up, delay, change

1 Introduction

Enumeration and generation pose several problems in the area of discrete algorithms. The problems are to output all the combinatorial objects, such as binary trees, permutations, strings, spanning trees, and matchings, and all these objects included in a given graph, strings, and set families. In other words, enumeration and generation are the problem of outputting all the elements in the given set just once for each.

Algorithms for solving these problems are called enumeration algorithms. As almost all enumeration algorithms in the literature are linear time for the number of outputs, basically, the speed of enumeration algorithms are measured by amortized computation time per output. Thus, many studies have been done on deriving *amortized constant time* algorithms whose amortized time complexities per output are constant. For example, constant time enumeration algorithms have been proposed for permutations [13], strings with some constraints [8, 9], binary trees [16], and spanning trees of a graph [5, 15].

In the other hand, *delay* and *change* are often considered as ways to measure the speed. Delay is the maximum computation time between two consecutive outputs, and change is the maximum size of symmetric difference between two consecutive outputs. An enumeration algorithm is said to be *polynomial delay* if its delay is polynomial in the input size, and *exact constant time* if its delay is constant. The amortized time complexity per output is often called *amortized delay.* An enumeration algorithm is said to be *constant change* if the change is constant. Usually, the number of outputs of an enumeration problem is exponential in the input size. Thus, an amortized constant time algorithm can take exponential time to output the next solution, while polynomial delay algorithms never do so. For solving several problems, we sometimes stop execution of enumeration algorithms if we obtain sufficiently many solutions. In these cases, polynomial delay algorithms have an advantage. Enumeration algorithms are often used as subroutines of another main algorithm, and the outputs are passed to the main algorithms for calculation. If the symmetric difference between two consecutive outputs is small, then the result of the previous calculation can be reused for the next calculation. It is an advantage to reduce the computation time of the main algorithm.

Even if we have an amortized constant time enumeration algorithm for a particular problem, it is not easy to obtain an exact constant time enumeration algorithm for it. Consider the problem of enumerating all n dimensional 0-1 vectors. These vectors can be considered as n bit numbers ranging from 0 to $2^n - 1$, hence it is very easy to construct an amortized constant time enumeration algorithm, which generates all n bit numbers in the increasing order. However, the change and the delay are not O(1) but O(n). We have to spend O(n) time to generate 100...0 following 011....1. Thus, the change is also O(n). An exact constant time algorithm requires sophisticated techniques.

Almost all enumeration algorithms in the literature are based on back tracking, binary partition, and reverse search. The recursive structures of these algorithms are basically tree shaped, and called *tree traversal*. Even if a tree traversal algorithm outputs a solution in each iteration, the delay and the change depend on the depth of the recursion tree. Hence, the scheme in the literatures for constructing constant delay and constant change algorithms use *path traversal* approaches: they traverse a path connecting all solutions to be output. To construct this type of algorithms is not easy, hence path traversal algorithms have been proposed for only primitive combinatorial objects, such as multi-set permutations, combinations of n elements chosen from r elements, and parenthesis strings [10, 6, 12, 13]. Therefore, there are numerous amortized constant time enumeration algorithms but only a few exact constant time enumeration algorithms in the literature. If we have a scheme for reducing delay and change of tree traversal algorithms such as back tracking, we may easily construct exact constant delay and constant change algorithms easily for various enumeration problems.

In this paper, we propose two methods to reduce the delay and the change of enumeration algorithms. These can be applied to tree traverse algorithms.

· **Alternative Output Method**
For a tree traversal enumeration algorithm, this method reduces its delay to the time complexity to generate a recursive call, and reduces its change to the size of the update of the current solution to generate a recursive call.

· **Output Queue Method**
For a tree traversal enumeration algorithm, this method reduces its delay to the amortized delay with the use of preprocessing time and extra memory.

To apply these methods, enumeration algorithms have to satisfy several conditions, however, the conditions do not lose the generality so much. Roughly speaking, if an amortized constant time enumeration algorithm does a constant number of operations for the current solution when it generates a recursive call, we can obtain an exact constant time algorithm by using these two methods. Applying these methods to amortized constant time algorithms, we can construct many exact constant time algorithms which have not been proposed in the literature. At the end of the paper, we will show some of those improved algorithms, which are for enumerating floor plans, permutations with forbidden places, spanning trees, triangulations, directed spanning trees, maximal independent vectors, and connected induced subgraphs.

2 Alternative Output Method

This section explains alternative output method. For an enumeration algorithm, we call the objects to be output *solutions*. For a tree traversal enumeration algorithm and an input, we define the *enumeration tree* by the rooted tree representing the recursive structure of the

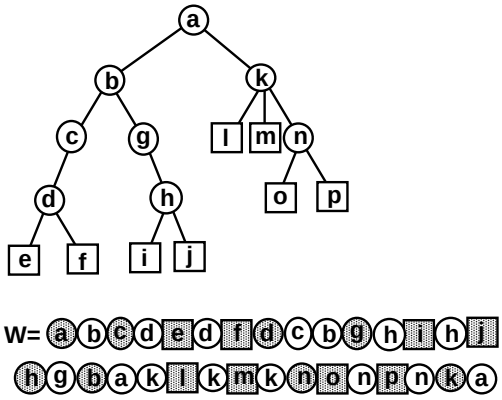

Figure 1: Illustration of W. Internal iterations are indicated by circles, and leaves are drawn by squares. Gray iterations are marked iterations.

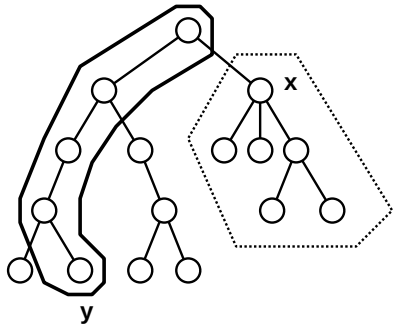

Figure 2: Consider computation time for ancestors of y, which are in the banana-shaped configuration. $T^*(P)$ is the maximum of this over all leaves. Iterations in the dotted polygon are the descendants of x. Consider (the computation time of descendants of x) / (the number of output by descendants of x). $\bar{T}(P)$ is the maximum among this over all iterations.

execution. We define the parent, children, descendants, and ancestors of an iteration of the enumeration tree in the usual way.

Suppose that an algorithm inputs problem P and enumerates all solutions. Let W be the sequence of iterations sorted in the order that the algorithm executed (see Fig. 2). Note that an iteration appears in W k times if it has $k-1$ children. The execution of the algorithm can be considered as a trace of W. During this trace of W, if a solution is output at an iteration of W, then we put a mark it there (see Figure 2). Note that an iteration may appear several times but just one of these is marked. Let $T(P)$ be the maximum computation time for passing through an iteration during a trace. In the other words, $T(P)$ is the maximum of
(1) computation time before the first recursive call,
(2) computation time from the end of a recursive call to the beginning of the next recursive call, and
(3) computation time after the last recursive call
over all iterations. Let $\bar{T}(P)$ be the maximum of
(computation time of descendants of an iteration) / (the number of outputs by the descendants)
over all iterations. The worst computation time for all the ancestors of a leaf is denoted by $T^*(P)$ (see Figure 2). We call an algorithm *internal output* if each iteration of the algorithm outputs a solution. For an internal output algorithm inputting problem P, let $C(P)$ be the

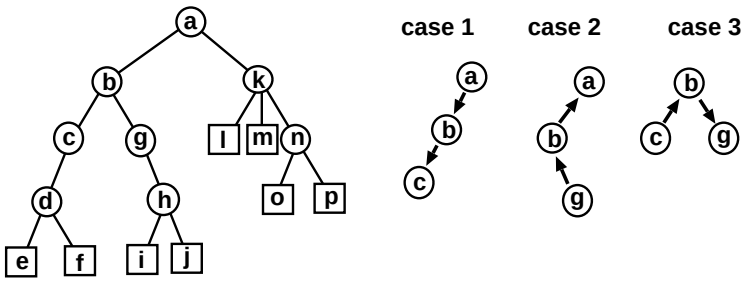

Figure 3: Three consecutive iterations are classified into 3 cases

maximum difference between the outputs of an iteration and its child.

Suppose that an internal output algorithm can be described as follows.

Internal Output Algorithm (P: problem)
(1) Find a solution x of P
(2) Output x
(3) Generate recursive calls for subproblems $P_1, ..., P_k$

Then, the algorithm modified by alternative output method is described as follows.

Modified Internal Output Algorithm (P: problem)
(1) Find a solution x of P
(2) **If** the depth of the iteration is odd **then** output x
(3) Generate recursive calls for subproblems $P_1, ..., P_k$
(4) **If** the depth of the iteration is even **then** output x

Theorem 1 *For an internal output algorithm, the delay and the change of the modified algorithm are $T(P)$ and $C(P)$, respectively.*

Proof : Let v_1, v_2 and v_3 be arbitrary three consecutive iterations of W. We will prove that at least one of them is always marked.

If one of them is a leaf, then the vertex is marked. Thus, let us consider where none of them is a leaf. Based on this assumption, we only have three cases (see Fig. 2).

(Case 1) v_1 is the parent of v_2, and v_2 is the parent of v_3
(Case 2) v_3 is the parent of v_2, and v_2 is the parent of v_1
(Case 3) v_2 is the parent of both v_2 and v_3

In Case 1, moving from v_1 to v_2 (resp., from v_2 to v_3) corresponds to the recursive call of v_2 by v_1 (resp., of v_3 by v_2). Thus, v_2 and v_3 correspond to the execution from the beginning of iterations to the first recursive calls. Since the depth of either v_2 or v_3 is odd, one of them is marked. Similarly, in Case 2, at least one of v_1 and v_2 is marked. In Case 3, v_1 corresponds to the execution after the last recursive call, and v_3 corresponds to the execution before the first recursive call. Hence, v_3 is marked if the depth of v_1 and v_3 is odd, and v_1 is marked otherwise. ∎

From the theorem, if an internal output algorithm satisfies that each iteration takes constant time for generating a recursive call, and the difference between outputs of an iteration

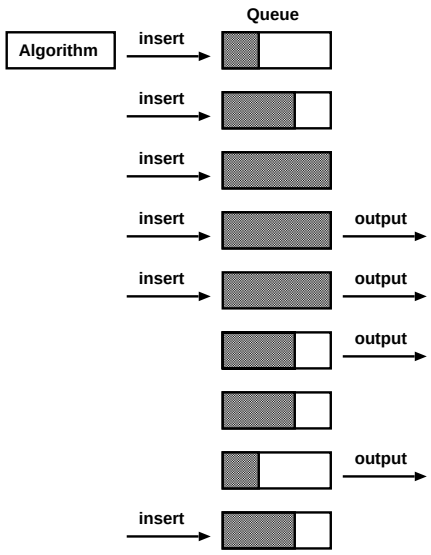

Figure 4: Illustration of output queue method. Algorithm inserts solutions into queue, and when the queue is full, output starts. A solution is output every 2 steps and when the solutions overflow the queue.

and its child is constant, then $C(P)$ and $T(P)$ are constant, and the algorithm modified by alternative output method is exact constant time and constant change.

An enumeration algorithm has to be internal output to apply alternative output method. This is not a hard condition, since many algorithms in the literature are internal output, or can be easily modified to be internal output. If an iteration may output several solutions, then partition the iteration such that each of these outputs just one solution. If only the leaves of the enumeration tree output no solution, then we can merge each leaf and its parent into one iteration. Many reverse search algorithms, such as those for enumerating triangulations, matchings, matroid bases, connected induced subgraphs, vertices of polytopes, floor plans, and directed spanning trees in a directed graph [1, 3, 7, 14], are internal output. Usually binary partition and back tracking algorithms are not internal output; however, if they can be described as follows, they will be internal output.

Binary Partition (P: problem, X: solution of P)
1. **If** P has no solution not equal to X **then return**
2. Find a solution $X' \neq X$ of P, and output X'
3. Divide P into P_1 and P_2 such that X is a solution of P_1 and X' is a solution of P_2
4. **Call Binary Partition** (P_1, X)
5. **Call Binary Partition** (P_2, X')

For example, enumeration algorithms for bipartite matchings, spanning trees, and maximal independent vectors satisfy the assumption [3, 15]. Moreover, fundarmental enumeration algorithms for trees, paths, connected subgraphs, matchings, permutations, and strings are internal output.

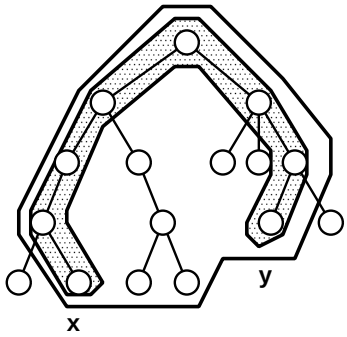

Figure 5: Iterations inside the outer polygon are S, and iterations inside the inner polygon are R

3 Output Queue Method

This section explains "output queue method." For a given enumeration algorithm, the method is described as follows.

Output Queue Method (P: problem)
(1) Allocate a queue Q of length $\lceil 2T^*(P)/\bar{T}(P) \rceil + 1$ for storing output solutions.
(2) During execution of the enumeration algorithm, insert output solutions to
the end of Q instead of outputting it directly.
If Q is full, then extract the top of Q and output it.
(3) After Q is full, extract and output the top of Q at every $\bar{T}(P)$ time.
(4) After the termination of the enumeration algorithm, output all the solutions in Q from the top.

This method does not increase the time complexity of the enumeration algorithm. The space complexity increases as much as the size of Q.

Theorem 2 *For an enumeration algorithm and a problem P, the delay of the algorithm modified by output queue method is $\bar{T}(P)$ with the use of $O(T^*(P) + \bar{T}(P))$ preprocessing time and $O(CT^*(P)/\bar{T}(P))$ extra memory space where C is the space required for storing a solution.* ∎

Proof : We first state the following lemma.

Lemma 1 *During the execution of an algorithm modified by output queue method, at least $(t' - t - 2T^*(P))/\bar{T}(P)$ solutions are inserted into Q from time t to time t'.*

Proof : Let x be the iteration executed at time t, and y be the iteration executed at time t'. Let S be the set of iterations executed from t to t', and R be the set of iterations included in the path connecting x and y in the enumeration tree. The computation time taken by the iterations of R is at most $2T^*(P)$. The iterations of $S \setminus R$ are composed of several groups each of which is the set of all descendants of a child of an iteration in R (see Figure 3). Since the computation time for iterations of $S \setminus R$ is at least $t' - t - 2T^*(P)$, the number of output iterations of $S \setminus R$ is at least $(t' - t - 2T^*(P))/\bar{T}(P)$. ∎

From this lemma, we derive the following lemmas.

Lemma 2 *The computation time from the beginning of the algorithm to the time when Q is first full is $O(T^*(P) + \bar{T}(P))$.*

Proof : Let t' be the time when the algorithm has output $\lceil 2T^*(P)/\bar{T}(P) \rceil + 1$ solutions. From lemma 2,

$$(t' - 0 - 2T^*(P))/\bar{T}(P) \leq \lceil 2T^*(P)/\bar{T}(P) \rceil + 1,$$

hence we have $t' \leq 4T^*(P) + 2\bar{T}(P)$. ∎

Lemma 3 *After once Q is full, Q will never be empty.*

Proof : For any time t' during which the algorithm is executed, let $t < t'$ be the time that Q was last full. From lemma 2, at least $(t' - t - 2T^*(P))/\bar{T}(P)$ solutions are output from t through t', hence Q has at least

$$\lceil 2T^*(P)/\bar{T}(P) \rceil + 1 - (t' - t)/\bar{T}(P) + (t' - t - 2T^*(P))/\bar{T}(P) \geq 1$$

solutions. ∎

From Lemma 2, the algorithm modified by output queue method takes $\mathrm{O}(T^*(P) + \bar{T}(P))$ time to output the first solution, and from Lemma 3, it outputs the following solutions in $\mathrm{O}(\bar{T}(P))$ per one solution. The space required for Q is $\mathrm{O}(CT^*(P)/\bar{T}(P))$. Therefore, the theorem holds. ∎

For any enumeration algorithm, $\bar{T}(P)$ is not greater than twice the delay. Moreover, it is usually in the order of amortized delay. This is because general amortized analysis of enumeration algorithms are done by passing the computation time of an iteration to its descendants. Hence, output queue method reduces the delay to the amortized delay for quite many enumeration algorithms.

4 Applications to Existing Enumeration Algorithms

In this section, we discuss several results obtained by applying our two methods to the enumeration algorithms in the literature.

4.1 Floor Plans

Problem: Enumerate all possible rectangular floor plans composed of at most n rectangular rooms.

An algorithm is proposed for this problem in [7]. The delay and change of the algorithm are $\mathrm{O}(n)$. The algorithm is based on family tree method, which outputs a solution at each iteration, thus the algorithm is internal output. Any iteration takes $\mathrm{O}(1)$ time to generate a recursive call, and pre/post-processing, hence $T(P) = \mathrm{O}(1)$ and $C(P) = \mathrm{O}(1)$. Thus, applying alternative output method to the algorithm, we obtain the following corollary.

Corollary 1 *There exists an exact constant time and constant change algorithm for enumerating all possible rectangular floor plans composed of at most n rectanglar rooms.* ∎

4.2 Permutations with Forbidden Places

Problem: For given sets of forbidden places for all indices, enumerate all permutations π any whose index is not placed at its constant size forbidden places.

This problem is an extension of the problem for which the existence of exact constant delay and constant change algorithm was open in [12]. A problem of enumerating permutations can be considered as a problem of enumerating perfect matchings of a bipartite graph. To solve the problem, an internal output algorithm is proposed in [3]. In an iteration, the algorithm locates an alternating cycle to find a solution and to generate two subproblems, and this takes $\mathrm{O}(|E|)$ time. However, if the size of forbidden places is bounded by a constant for any index, we can always find an alternating cycle with constant length, hence the time complexity of an iteration is reduced to a constant order. Thus, we have $T(P) = \mathrm{O}(1)$ and $C(P) = \mathrm{O}(1)$. We apply alternative output method to the algorithm and obtain the following corollary.

Corollary 2 *There exists an exact constant time and constant change algorithm for enumerating all permutations any whose index is not placed at its constant size forbidden places.* ∎

4.3 Spanning Trees

Problem: For a given undirected graph $G = (V, E)$, enumerate all spanning trees included in G.

For this problem, an algorithm based on binary partition is proposed [15]. Each iteration of the algorithm outputs a solution, and the difference between the output solutions of an iteration and its child is at most two edges, hence we have $C(P) = \mathrm{O}(1)$. Although an iteration takes $\mathrm{O}(|E|)$ time, the amortized delay is a constant. It is proved by amortized analysis with passing computation time of each iteration to its descendants. Thus, $\bar{T}(P) = \mathrm{O}(1)$ The depth of recursion is $\mathrm{O}(|E|)$, hence the delay and the change is $\mathrm{O}(|E|)$ and $\mathrm{O}(|V|)$ respectively, and $T^*(P) = \mathrm{O}(|E|^2)$. We apply alternative output method and output queue method to the algorithm, and obtain the following corollary.

Corollary 3 *There exists an exact constant time algorithm for enumerating all the spanning trees of a given undirected graph with $O(|E|^2)$ preprocessing time and $O(|E|^2)$ memory.* ∎

4.4 Triangulations of Points in Plane

Problem: For given n points in the plane, enumerate all triangulation of the points.

An internal output algorithm is proposed in [11] for this problem. It takes $\mathrm{O}(\log \log n)$ time to generate a recursive call and to pre/post process on an iteration, and the difference between outputs of an iteration and its child is constant. The depth of the recursion is $\mathrm{O}(n)$, hence the delay and the change are $\mathrm{O}(n)$. We apply alternative output method to the algorithm, and obtain the following corollary.

Corollary 4 *There exists a constant change and $O(\log \log n)$ delay algorithm for enumerating all triangulations of given n points.* ∎

4.5 Directed Spanning Trees

Problem: For a given directed graph $G = (V, E)$ and a vertex r, enumerate all the directed spanning trees of G rooted at r.

For the problem, a reverse search algorithm is proposed [15]. Each iteration of the algorithm outputs a solution, and the difference between output solutions of an iteration and its child is at most two edges. The time complexity of an iteration is $\mathrm{O}(|E|M(|V|))$, where $M(|V|)$ is the computation time to maintain the minimum spanning tree in a graph with $|V|$ vertices. Recently, it is proved that $M(|V|)$ is polylogarithmic in $|V|$ [4]. By passing the computation time of an iteration to its descendant, the amortized delay is bounded by $\mathrm{O}(M(|V|))$, thus $\bar{T}(P) = \mathrm{O}(M(|V|))$. Since the depth of the recursion is $\mathrm{O}(|E|)$, the delay, the change and $T^*(P)$ are $\mathrm{O}(|E|M(|V|))$, $\mathrm{O}(|V|)$, and $\mathrm{O}(|E|^2M(|V|))$, respectively. Hence, we can apply alternative output method and the output queue method to the algorithm, and obtain the following corollary.

Corollary 5 *There exists a constant change and $O(M(|V|))$ delay algorithm for enumerating all the directed spanning trees of a graph with $O(|E|^2M(|V|))$ preprocessing time and $O(|E|^2)$ memory.* ∎

4.6 Maximal Independent Vectors

Problem: For given n dimensional m vectors, enumerate all the maximal sets of independent vectors.

This problem can be solved by a reverse search algorithm [15]. Since the algorithm outputs a solution in each iteration, it is an internal output algorithm. As the time complexity of an iteration is $\mathrm{O}(nm^2)$ and the depth of recursion is $\mathrm{O}(m)$, $T(P) = \mathrm{O}(n^2m)$. The size of difference between outputs of an iteration and its child is constant. The amortized delay is bounded by $\mathrm{O}(n)$ by passing the computation time taken by each iteration to its descendants. We apply alternative output method and output queue method to the algorithm, and obtain the following corollary.

Corollary 6 *There exists a constant change and $O(n)$ delay algorithm for enumerating all the maximal sets of independent vectors for given n dimensional m vectors.* ∎

4.7 Connected Induced Subgraphs

Problem: For a given graph, enumerate all vertex subsets of the graph which induce connected subgraphs.

This problem can be solved by a reverse search algorithm [1]. As the algorithm outputs a solution in each iteration, it is an internal output algorithm. The size of difference between outputs of an iteration and its child is constant. We apply alternative output method to the algorithm and obtain the following corollary.

Corollary 7 *There exists a constant change algorithm for enumerating all vertex subsets of a given graph which induces connected subgraphs.* ∎

5 Concluding and Remarks

We proposed two methods for reducing the change and the delay of enumeration algorithms. The first, "alternative output method" reduces the delay to the time complexity to generate a recursive call and reduces the change to the maximum difference between outputs of an iteration and its child, if any iteration outputs a solution. For an algorithm whose amortized time complexity per output is $\bar{T}(P)$ for any descendants of iteration, the second method "output queue method" reduces the delay to $\bar{T}(P)$ with the use of $O(\bar{T}(P)+T^*(P))$ memory, where $T^*(P)$ is the worst case computation time taken by all ancestors of a leaf.

Using these methods, constant change and exact constant delay algorithms can be constructed for the problems of enumerating the following combinatorial objects: permutations with forbidden places, floor plans, spanning trees of a graph, directed spanning trees of a directed graph, triangulations of given points in plane, maximal independent vectors, and connected induced subgraphs of a graph. These methods will enable many enumeration algorithms to be improved in the future.

References

[1] D. Avis and K. Fukuda, "Reverse Search for Enumeration," *Discrete Appl. Math.* **65**, pp. 21-46 (1996).

[2] D. Avis and K. Fukuda, "A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra," *Discrete Comp. Geom.* **8**, pp. 295-313, (1992).

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

[4] J. Holm, K. de Lichtenberg and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *Proceedings of the thirtieth annual ACM symposium on theory of computing,* pp. 79-89 (1998).

[5] S. Kapoor and H. Ramesh, "Algorithms for Enumerating All Spanning Trees of Undirected and Weighted Graphs," *SIAM J. Comp.* **24**, pp. 247–265 (1995).

[6] A. Nijenhuis and H.S. Wilf, "Combinatorial Mathematics," *Academic Press* (1975).

[7] S. Nakano, "Enuemerating Floorplans with n Rooms," *Lec. Note in Comp. Sci.* **1350**, Springer-Verlag, pp. 107-115 (2001).

[8] F. Ruskey and J. Sawada, "Generating necklaces and strings with forbidden substrings," *SIAM J. Comp.* **29**, pp. 671-684 (1999).

[9] J. Sawada, "Generating Bracelets in Constant Amortized Time Generating," *SIAM J. Comp.* **31**, pp. 259-268 (2001).

[10] C. Savage, "A Survey of Combinatorial Gray Codes," *SIAM Review* **39,** pp. 605-629 (1997).

[11] Sergei Bespamyatnikh, "An efficient algorithm for enumeration of triangulations," *Computational Geometry: Theory and Applications,* **23**, No.3, pp. 271-279 (2002).

[12] T. Takaoka, "O(1) Time Algorithms for Combinatorial Generation by Tree Traversal," *Computer Journal* **42**, No. 5, pp. 400-408 (1999).

[13] T. Takaoka, "An O(1) Time Algorithm for Generating Multiset Permutations," *Lect. Note in Comp. Sci.* **1741** (1999).

[14] T. Uno, "An Algorithm for Enumerating All Directed Spanning Trees in a Directed Graph," *Lect. Notes in Comp. Sci.* **1178**, Springer-Verlag, Algorithms and Computation, pp. 166-173 (1996).

[15] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms and Its Application for Matroid Bases," *Lec. Note in Comp. Sci.* **1627**, Springer-Verlag, pp. 349-359 (1999).

[16] D. Zerling, "Generating Binary Trees by Rotations," *JACM* **32**, pp. 694-701 (1985).