Efficient Generation of Evolutionary Trees

MUHAMMAD ABDULLAH ADNAN[1]
MD. SAIDUR RAHMAN[2]

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh
[1]adnan@cse.buet.ac.bd
[2]saidurrahman@cse.buet.ac.bd

Abstract. For the purposes of phylogenetic analysis, it is assumed that the phylogenetic pattern of evolutionary history can be represented as a branching diagram like a tree, with the terminal branches (or leaves) linking the species being analyzed and the internal branches linking hypothesized ancestral species. To a mathematician, such a tree is simply a cycle-free connected graph, but to a biologist it represents a series of hypotheses about evolutionary events. In this paper we are concerned with generating all such probable evolutionary trees that will guide biologists to research in all biological subdisciplines. We give an algorithm to generate all evolutionary trees having n ordered species without repetition. We also find out an efficient representation of such evolutionary trees such that each tree is generated in constant time on average.

Keywords: Bioinformatics, Evolutionary Trees, Graphs, Algorithm, Generating Problems.

(Received June 26, 2006 / Accepted January 03, 2007)

1 Introduction

In *bioinformatics*, we frequently need to establish evolutionary relationship between different types of species [4, 6]. Biologists often represent this relationship in the form of binary trees. Such complete binary trees having different types of species in its leaves are known as *evolutionary trees* (see Figure 1). In a rooted evolutionary tree, the root corresponds to the most ancient ancestor in the tree. Leaves of evolutionary trees correspond to the existing species while internal vertices correspond to hypothetical ancestral species.

Evolutionary trees are used to predict predecessors of existing species, to comment about future generations, DNA sequence matching, etc. Prediction of ancestors can be easy if all possible trees are generated. Moreover, it is useful to have the complete list of evolutionary trees having different types of species. One can use such a list to search for a counter-example to some conjecture, to find best solution among all solutions or to experimentally measure an average performance of an algorithm over all possible input evolutionary trees. Many algorithms to generate a given class of graphs without repetition are already known [1, 2, 3, 5, 7, 8, 9, 10, 11, 12].

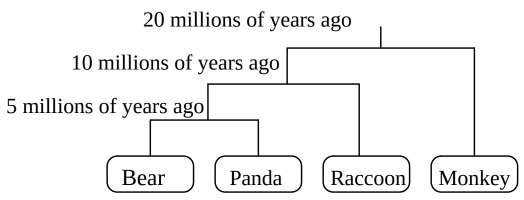

Figure 1: The evolutionary tree having four species.

In this paper we first consider the problem of generating all possible evolutionary trees. The main challenges in finding algorithms for enumerating all evolutionary trees are as follows. Firstly, the number of such trees is exponential in general and hence listing all of them requires huge time and computational power. Secondly, generating algorithms produce huge outputs and the outputs dominate the running time. For this reason, reducing the amount of output is essential. Thirdly, checking for any repetitions must be very efficient. Storing the entire list of solutions generated so far will not be efficient, since checking each new solution with the entire list to prevent repetition would require huge amount of memory and overall time complexity would be very high. So, if we can compress the outputs, then it considerably improves the efficiency of the algorithm. Therefore, many generating algorithms output objects in an order such that each object differs from the preceding one by a very small amount, and output each object as the "difference" from the preceding one.

Generating evolutionary trees is more like generating complete binary rooted trees with 'fixed' and 'labeled' leaves. That means there is a fixed number of leaves and the leaves are labeled. There are some existing algorithms for generating rooted trees with n vertices [2, 5, 7, 8, 9]. But these algorithms do not guarantee that there will be fixed and labeled leaves. If we generate all binary trees with n leaves with existing algorithms then we have to label each tree and permutate labels to generate all trees. Since the siblings are not ordered, permutating the labels lead to repetition. Thus modifying existing algorithms we cannot generate all evolutionary trees.

In this paper we first give an efficient algorithm to generate all evolutionary trees with fixed and ordered number of leaves. The order of the species is based on evolutionary relationship and phylogenetic structure. For instance, Bear is more related to Panda than Monkey and Raccoon is more related to Panda than Bear. Thus a species is more related to its preceding and following species in the sequence of species than other species in the sequence. The order of labels maintains this property. This property implies that each species in the sequence shares a common ancestor either with the preceding species or with the following species. We apply the above restriction on the order of leaves with two goals in mind. First, the solution space is reduced so that more probable solutions are available for the biologists to predict quickly and easily. Second, each such probable evolutionary tree must be generated in constant time. We also find out a suitable representation of such trees. We represent a labeled and ordered complete binary tree with n leaves by a sequence of $(n-2)$ numbers. Our algorithm generates all such trees without repetition.

Furthermore the algorithm for generating labeled and ordered trees is simple and generates each tree in constant time on average without repetition. Our algorithm generates a new tree from an existing one by making a constant number of changes and outputs each tree as the difference from the preceding one. The main feature of our algorithm is that we define a tree structure, that is parent-child relationships, among those trees (see Figure 2). In such a "tree of evolutionary trees", each node corresponds to an evolutionary tree and each node is generated from its parent in constant time. In our algorithm, we construct the tree structure among the evolutionary trees in such a way that the parent-child relation is unique, and hence there is no chance of producing duplicate evolutionary trees. Our algorithm also generates the trees *in place*, that means, the space complexity is only $O(n)$.

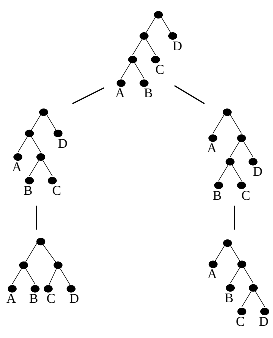

Figure 2: The Family Tree F_4.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 depicts the representation of evolutionary trees. Section 4 shows a tree structure among evolutionary trees. In Section 5 we present our algorithm which generates each solution in $O(1)$ time on average. Finally, section 6 is a conclusion.

2 Preliminaries

In this section we define some terms used in this paper.

In mathematics and computer science, a *tree* is a connected graph without cycles. A *rooted tree* is a tree with one vertex r chosen as root. A *leaf* in a tree is a vertex of degree 1. Each vertex in a tree is either an internal vertex or a leaf. A *complete binary tree* is a rooted tree with each internal node having exactly two descendants.

A *family tree* is a rooted tree with parent-child relationship. The vertices of a rooted tree have *level*s associated with them. The root has the lowest level i.e. 0. The level for any other node is one more than its parent except root. Vertices with the same parent v are called *siblings*. The siblings may be ordered as $c_1, c_2, \ldots, c_l$ where l is the number of children of v. If the siblings are ordered then c_{i-1} is the *left sibling* of c_i for $1 < i \leq l$ and c_{i+1} is the *right sibling* of c_i for $1 \leq i < l$. The *ancestors* of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex and including the root itself. The *descendants* of a vertex v are those vertices that have v as an ancestor. A leaf in a family tree has no children.

An *evolutionary tree* is a graphical representation of the evolutionary relationship among three or more species. In a rooted evolutionary tree, the root corresponds to the most ancient ancestor in the tree and the path from the root to a leaf in the rooted tree is called an *evolutionary path*. Leaves of evolutionary trees correspond to the existing species while internal vertices correspond to hypothetical ancestral species.

In this paper, we represent evolutionary tree in terms of complete binary trees. Each existing species of evolutionary tree is a leaf in the complete binary tree (see Figure 3). We give labels to each leaf. The label identifies the existing species. For example, labels A, B, C and D represent Bear, Panda, Raccoon and Monkey. The labels are fixed and ordered. The order of the species is based on evolutionary relationship and phylogenetic structure. For instance, Bear is more related to Panda than Monkey and Raccoon is more related to Panda than Bear. So, a species is more related to its preceding and following species in the sequence of species than other species in the sequence. The order of labels maintains this property. This property implies that each species in the sequence shares a common ancestor either with the preceding species or with the following species. Our complete binary tree will maintain this property and we will generate all such trees with exactly n leaves.

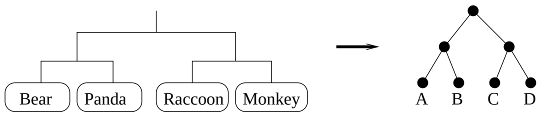

Figure 3: Representation of evolutionary tree in terms of complete binary tree.

3 Representation of Evolutionary Trees

In this section we give an efficient representation of a labeled and ordered evolutionary tree in $T(n)$. We represent such trees with n species with a sequence of $(n-2)$ numbers.

Let $T(n)$ be the set of all evolutionary trees with n labeled and ordered leaves. Now, we find out a representation of each evolutionary tree $t \in T(n)$. Our idea here is to represent a tree with a sequence of numbers. For this, we find out an intermediate representation of each tree $t \in T(n)$. A complete binary tree with n labeled leaves can be represented with a string of valid parenthesization of n labels $l_1, l_2, \ldots, l_n$. Figure 4 shows the representation of complete binary tree having 5 leaves. Thus the number of such trees corresponds directly to *Catalan number*. Thus the total number of complete binary trees with n fixed and labeled leaves is given by $\frac{{}^{2(n-1)}C_{(n-1)}}{n}$.

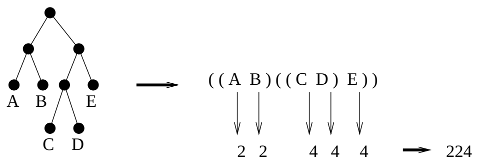

Figure 4: Representation of an evolutionary tree having five species.

We now count the number of opening parenthesis '(' before each label l_i, $1 \leq i \leq (n-2)$ in the string of valid parenthesis of each intermediate representation. This gives us a sequence of $(n-2)$ numbers a_1, a_2, $\ldots$, a_{n-2} where a_i represents the number of '(' before label l_i, for $1 \leq i \leq (n-2)$. Since the labels are fixed and ordered, we do not need to count for l_{n-1} and l_n and so we omit these two numbers in the sequence. For example, the sequence 244 represents a evolutionary tree with 5 leaves which corresponds to the string of valid parenthesis $((l_1((l_2l_3)l_4))l_5)$. One can observe that for each sequence $a_1 \leq a_2 \leq \cdots \leq a_{n-2}$ and $1 \leq a_i \leq (n-1)$ for $1 \leq i \leq (n-2)$. Thus, we say that a sequence of $(n-2)$ numbers uniquely represents a evolutionary tree with labeled and ordered leaves as shown in Figure 4.

Let $S(n)$ denote the set of all such sequence. Each sequence $s \in S(n)$ uniquely identifies a tree $t \in T(n)$. We have the following lemma.

Lemma 3.1 *A sequence $s \in S(n)$ of $(n-2)$ numbers uniquely represents an evolutionary tree $t \in T(n)$.*

Proof. In an evolutionary tree $t \in T(n)$ the labeled leaves $l_1, l_2, \ldots, l_n$ are ordered. A leaf l_i, $1 < i < n$ can only be paired with either with l_{i-1} or l_{i+1} in the sequence of labels. We take any two labels, l_i and l_j, $1 < i \leq n-2$ and $j \in \{i-1, i+1\}$. If l_i and l_j are paired, the count of the '(' is same for both of them. This implies that $s_i = s_j$. If l_i and l_j are not paired, their count of the '(' is different which implies $s_i \neq s_j$.

For any two trees $t_1 \in T(n)$ and $t_2 \in T(n)$, $t_1 \neq t_2$, we will find at least two labels l_i and l_j which are paired in one and not paired in another. Thus, their count is different i.e. $s_i \neq s_j$. Hence the sequence $s \in S(n)$ of $(n-2)$ numbers represents exactly one evolutionary tree $t \in T(n)$. $\mathcal{Q.E.D.}$

4 The Family Tree

In this section we define a tree structure F_n among evolutionary trees in $T(n)$.

For positive integer n, let $t \in T(n)$ be an evolutionary tree with n leaves having $l_1, l_2, \ldots, l_n$ labels. For each $t \in T(n)$, we get unique sequence $s \in S(n)$ of $(n-2)$ numbers $a_1, a_2, \ldots, a_{n-2}$ where a_i represents the number of '(' before label l_i, for $1 \leq i \leq (n-2)$. Also, for each sequence $a_1 \leq a_2 \leq \cdots \leq a_{n-2}$ and $1 \leq a_i \leq (n-1)$ for $1 \leq i \leq (n-1)$.

Now we define the family tree F_n as follows. Each node of F_n represents an evolutionary tree. If there are n species then there are $(n-1)$ levels in F_n. A node is in level i in F_n if $a_1 \leq a_2 \leq \ldots \leq a_i < (n-1)$ and $a_{i+1} = \ldots = a_{n-2} = (n-1)$ for $1 < i \leq (n-1)$. For example, the sequence 224 is at level 2. As the level increases the number of rightmost $(n-1)$ decreases and vice versa. Thus a node at level $n-2$ has no rightmost $(n-1)$ number i.e. $a_{n-2} < (n-1)$. Since F_n is a rooted tree we need a root and the root is a node at level 0. One can observe that a node is at level 0 in F_n if $a_1, a_2, \ldots, a_{n-2} = (n-1)$ and there can be exactly one such node. We thus take the sequence $(n-1, n-1, \ldots, n-1)$ as the root of F_n. Clearly, the number of rightmost $(n-1)$ in root is greater than that of any other sequence for any evolutionary tree in $T(n)$.

To construct F_n, we define two types of relations among the evolutionary trees in $T(n)$:

(a) Parent-child relationship and

(b) Child-parent relationship.

We define the parent-child relationship among the evolutionary trees in $T(n)$ with two goals in mind. First, the difference between an evolutionary tree s and its child $C(s)$ should be minimum, so that $C(s)$ can be generated from s by minimum effort. Second, every evolutionary tree in $T(n)$ must have a parent except the root and only one parent in F_n. We achieve the first goal by ensuring that the child $C(s)$ of an evolutionary tree s can be found by simple subtraction. That means s can also be generated from its child $C(s)$ by simple addition. The second goal, that is the uniqueness of the parent-child relationship is illustrated in the following subsections.

4.1 Parent-Child Relationship

Let $t \in T(n)$ be an evolutionary tree with n ordered leaves having $l_1, l_2, \ldots, l_n$ labels and $s \in S(n)$ be the sequence of numbers $a_1, a_2, \ldots, a_{n-2}$ corresponding to t. s corresponds to a node of level i, $0 \leq i \leq (n-2)$ of F_n. Thus we have $a_1 \leq a_2 \leq \cdots \leq a_i < (n-1)$ and $a_{i+1} = \cdots = a_{n-2} = (n-1)$ for $1 < i \leq (n-2)$. The number of children it has is equal to $(a_{i+1} - a_i)$. The sequence of the children are defined in such a way that to generate a child from its parent we have to deal with only one integer in the sequence and the rest of the integers remain unchanged. The integer is determined by the level of parent sequence in F_n. The operation we apply is only subtraction and assignment. The number of rightmost $(n-1)$ decreases in the child sequence by applying parent-child relationship.

Let $C_j(s) \in S(n)$ be the sequence of jth child, $1 \leq j \leq (a_{i+1} - a_i)$ of s. Note that s is in level i of F_n and $C_j(s)$ will be in level $i+1$ of F_n. We define the sequence for $C_j(s)$ as $c_1, c_2, \ldots, c_{n-2}$ where $c_k = a_k$ for $k \neq j$ and $c_j = (a_{i+1} - j)$. Thus, we observe that C_j is a node of level $i+1$, $0 \leq i < n-2$ of F_n and so $c_1 \leq c_2 \leq \cdots \leq c_{i+1} < (n-1)$ and $c_{i+2} = \cdots = c_{n-2} = (n-1)$ for $0 \leq i < (n-2)$. Thus for each consecutive level we only deal with the integer a_{i+1} and the rest of the integers remain unchanged. For example, 244 for $n = 5$ is a node of level 1 because $a_1 < 4$ and $a_2 = a_3 = 4$. Here, $a_2 - a_1 = 2$ so it has two children and the two children are shown in Figure 6.

4.2 Child-Parent Relationship

The child-parent relation is just the reverse of parent-child relation. Let $t \in T(n)$ be an evolutionary tree with n ordered leaves having $l_1, l_2, \ldots, l_n$ labels and $s \in S(n)$ be the sequence of numbers $a_1, a_2, \ldots, a_{n-2}$ corresponding to t. s corresponds to a node of level i, $0 \leq i \leq (n-2)$ of F_n. Thus we have $a_1 \leq a_2 \leq \ldots \leq a_i < (n-1)$ and $a_{i+1} = \ldots = a_{n-2} = (n-1)$ for $1 < i \leq (n-1)$. We define a unique parent sequence of s at level $i-1$. Like the parent-child relationship here we also deal with only one integer in the sequence. The operations we apply here is only addition and assignment. The number of rightmost $n-1$ increases in the parent sequence by applying child-parent relationship.

Let $P(s) \in S(n)$ be the parent sequence of s. We define the sequence for $P(s)$ as $p_1, p_2, \ldots, p_{n-2}$ where $p_j = a_j$ for $j \neq (i-1)$ and $p_{i-1} = (n-1)$. Thus, we observe that $P(s)$ is a node of level $i-1$, $1 \leq i < (n-2)$ of F_n and so $p_1 \leq p_2 \leq \cdots \leq p_{i-1} < (n-1)$ and $p_i = \cdots = p_{n-2} = (n-1)$ for $1 \leq i \leq (n-2)$. For example, 224 for $n = 5$ is a node of level 2 because $a_1 \leq a_2 \leq 4$ and $a_3 = 4$. It has a unique parent 244 as shown in Figure 6.

4.3 The Family Tree

From the above definitions we can construct F_n. We take the sequence $s_r = a_1, a_2, \ldots, a_{n-2}$ as root where $a_1, a_2, \ldots, a_{n-2} = n-1$ as we mentioned before. The family tree F_n for the evolutionary trees in $T(n)$ is shown in Figure 5 and Figure 6 shows the representation of family tree F_n.

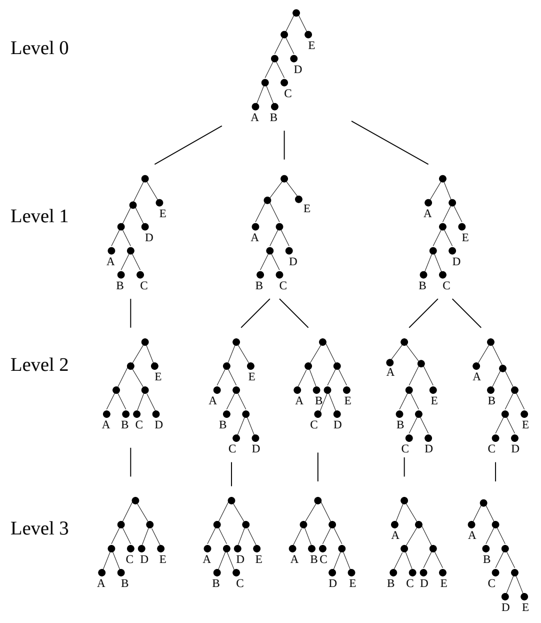

Figure 5: Illustration of Family Tree F_5.

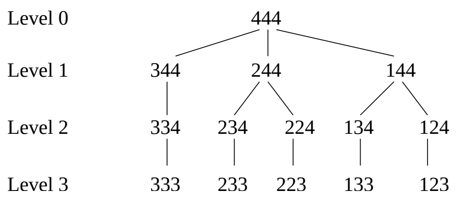

Figure 6: Representation of Family Tree F_5.

Based on the above parent-child relationship, the following lemma proves that every evolutionary tree in $T(n)$ is present in F_n.

Lemma 4.1 *For any evolutionary tree $t \in T(n)$, there is a unique sequence of evolutionary trees that transforms t into the root t_r of F_n.*

Proof. Let $s \in S(n)$ be a sequence, where s is not the root sequence, representing an evolutionary tree $t \in T(n)$. By applying child-parent relationship, we find the parent sequence $P(s)$ of the sequence s. Now if P(s) is the root sequence, then we stop. Otherwise, we apply the same procedure to $P(s)$ and find its parent $P(P(s))$. By continuously applying this process of finding the parent sequence of the derived sequence, we have the unique sequence $s, P(s), P(P(s)), \ldots$ of sequences in $S(n)$ which eventually ends with the root sequence s_r of $T_{n,m}$. We observe that $P(s)$ has at least one $(n-1)$ number more than s in its sequence. Thus $s, P(s), P(P(s)), \ldots$ never lead to a cycle and the level of the derived sequence decreases which ends up with the level of root sequence s_r. $\mathcal{Q.E.D.}$

Lemma 4.1 ensures that there can be no omission of evolutionary trees in the family tree F_n. Since there is a unique sequence of operations that transforms an evolutionary tree $t \in T(n)$ into the root t_r of F_n, by reversing the operations we can generate that particular evolutionary tree, starting from root. Now we have to make sure that F_n represents evolutionary trees without repetition. Based on the parent-child and child-parent relationships, the following lemma proves this property of F_n.

Lemma 4.2 *The family tree F_n represents evolutionary trees in $T(n)$ without repetition.*

Proof. Given a sequence $s \in S(n)$, representing a $t \in T(n)$, the children of s are defined in such a way that no other sequence in $S(n)$ can generate same child. For contradiction let two sequences $A, B \in S(n)$ are at level i of F_n and generate same child C. Thus C is a sequence of level $i+1$ of F_n. The sequences for A, B and C are a_j, b_j and c_j for $1 \leq j \leq n-2$. Clearly, $a_k = b_k = n-1$ for $i+1 \leq k \leq n-2$ and we will get at least one j such that $a_j \neq b_j$ for $1 \leq j \leq i$. According to parent-child relationship, to generate C from its parent A or B, only one integer a_{i+1} or b_{i+1} is changed in the sequence. Thus child of A, $C(A)$ and child of B, $C(B)$ are different since $a_{i+1} = b_{i+1}$ and $a_j \neq b_j$ for $1 \leq j < i+1$. Thus A and B does not generate same child C. By contradiction, every sequence has a single and unique parent.

$\mathcal{Q.E.D.}$

5 Algorithm

In this section, we give an algorithm to construct the family tree F_n and generate all trees.

If we can generate all child sequences of a given sequence in $S(n)$, then in a recursive manner we can construct F_n and generate all sequence in $S(n)$. We have the root sequence $s_r = (n-1)\ldots(n-1)$. We get the child sequence s_c by using the parent to child relation discussed above.

```
Procedure Find-All-Child-Trees($s = a_1 a_2 \ldots a_{n-1}$, $i$)
{ $s$ is the current sequence, $i$ indicates the current level and $s_c$ is the child sequence }
begin
  Output $s$ {Output the difference from the previous evolutionary tree};
   for $j = 1$ to $(a_{i+1} - a_i)$
     Find-All-Child-Trees( $s_c = a_1 a_2 \ldots (a_{i+1} - j) \ldots a_{n-2}), i+1$);
end;
Algorithm Find-All-Evolutionary-Trees($n$)
begin
  Find-All-Child-Trees( $s_r = (n-1)\ldots(n-1), 0$ );
end.
```

The following theorem describes the performance of the algorithm **Find-All- Evolutionary-Trees**.

Theorem 5.1 *The algorithm* **Find-All-Evolutionary-Trees** *uses $O(n)$ space and runs in $O(|T(n)|)$ time.*

Proof. In our algorithm we only use the simple addition or subtraction operation to generate a new evolutionary tree from an old one. Thus each evolutionary tree is generated in constant time without computational overhead. Since we traverse the family tree F_n

and output each sequence at each corresponding vertex of F_n we can generate all the evolutionary trees in $T(n)$ without repetition. By applying parent to child relation we can generate every child in $O(1)$ time. Then by using child to parent relation we go back to the parent sequence. Hence, the algorithm takes $O(|T(n)|)$ time i.e. constant time on average for each output.

Our algorithm outputs each evolutionary tree as the difference from the previous one. The data structure that we use to represent the evolutionary trees is a sequence of $n-2$ integers. Therefore, the memory requirement is $O(n)$, where n is the number of species.

$\mathcal{Q.E.D.}$

6 Conclusion

In this paper, we find out an efficient representation of an evolutionary tree having ordered species. We also give an algorithm to generate all evolutionary trees having n ordered species. The algorithm is simple, generates each tree in constant time on average, and clarifies a simple relation among the trees that is a family tree of the trees.

References

[1] M. A. Adnan and M. S. Rahman, *Distribution of objects to bins: generating all distributions*, Proc. of International Conference on Computer and Information Technology (ICCIT'06), 2006 (to appear).

[2] M. Belbaraka and I. Stojmenovic, *On generating B-trees with constant average delay and in lexicographic order*, Information Processing Letters, 49, pp. 27-32, 1994.

[3] T. I. Fenner and G. Loizou, *A binary tree representation and related algorithms for generating integer partitions*, The Computer Journal, 23, pp. 332-337, 1979.

[4] N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, The MIT Press, Cambridge, Massachusetts, London, England, 2004.

[5] S. Kawano and S. Nakano, *Constant time generation of set partition*, IEICE Trans. Fundamentals, E88-A, 4, pp. 930-934, 2005.

[6] D. E. Krane and Michael L. Raymer, *Fundamental Concepts of BioInformatics*, Pearson Education, San Francisco, 2003.

[7] S. Nakano and T.Uno, *Efficient generation of rooted trees*, NII Tech. Report, NII-2003-005E, July 2003.

[8] S. Nakano and T. Uno, *Constant time generation of trees with specified diameter*, Proc. of WG 2004, LNCS 3353, pp. 33-45, 2004.

[9] S. Nakano and T. Uno, *Generating colored trees*, Proc. of WG 2005, LNCS 3787, pp. 249-260, 2005.

[10] C. Savage, *A survey of combinatorial gray codes*, SIAM Review, 39, pp. 605-629, 1997.

[11] K. Yamanaka and S. Nakano, *Generating all realizers*, IEICE Trans. Inf. and Syst., J87-DI, 12, pp. 1043-1050, 2004.

[12] A. Zoghbi and I. Stojmenovic, *Fast algorithm for generating integer partitions*, Intern. J. Computer Math, 70, pp. 319-332, 1998.