International Journal of Computer Mathematics

Distribution of distinguishable objects to bins: generating all distributions
Muhammad Abdullah Adnan a; Md. Saidur Rahman a

a Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Bangladesh

Online Publication Date: 01 July 2007
To cite this Article: Adnan, Muhammad Abdullah and Rahman, Md. Saidur (2007) 'Distribution of distinguishable objects to bins: generating all distributions', International Journal of Computer Mathematics, 84:7, 953 - 965

To link to this article: DOI: 10.1080/00207160701294392

URL: http://dx.doi.org/10.1080/00207160701294392

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: http://www.informaworld.com/terms-and-conditions-of-access.pdf

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.
Distribution of distinguishable objects to bins: generating all distributions

MUHAMMAD ABDULLAH ADNAN* and MD. SAIDUR RAHMAN†
Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh

(Received 04 October 2006; revised version received 08 January 2007; accepted 07 February 2007)

In this paper we give an algorithm to generate all distributions of distinguishable objects to bins without repetition. Our algorithm generates each distribution in constant time. To the best of our knowledge, our algorithm is the first algorithm which generates each solution in $O(1)$ time in the ordinary sense. As a byproduct of our algorithm, we obtain a new algorithm to enumerate all multiset partitions when the number of partitions is fixed and the partitions are numbered. In this case, the algorithm generates each multiset partitions in constant time (in the ordinary sense). Finally, we extend the algorithm to the case when the bins have priorities associated with them. Overall space complexity of the algorithm is $O(mk \log n)$, where there are $m$ bins and the objects fall into $k$ different classes. In a companion paper, the generation of all distributions of identical objects to bins is also considered.

Keywords: Combinatorial objects; Algorithm; Generating problems; Multiset; Set partitions

AMS Subject Classifications: O5A15; O5A17; O5A18

1. Introduction

A well known counting problem in combinatorics is counting the number of ways objects can be distributed among bins [1–3]. The paradigm problem is counting the number of ways of distributing fruits to children. For example, Kathy, Peter and Susan are three children. We have four fruits to distribute among them without cutting the fruits into parts. In how many ways can the children receive fruits? The fruits or the objects that we want to distribute may be identical or of different kinds. Based on this criteria, the problem can be subdivided into two parts – the identical case and the non-identical case. Since the latter is more general, in this paper we focus on the non-identical case and we call such objects ‘distinguishable objects’. Let there be $m$ bins and $n$ distinguishable objects where the objects fall into $k$ different classes. Objects within a class are identical to each other, but are distinguishable from those of other classes. Let, $n_j$ represent the number of objects in the $j$th class where $1 \leq j \leq k$. The paradigm problem is to distribute the different types of fruit to the children. Suppose we have three apples, two pears and one banana to distribute to Kathy, Peter and Susan. Then $m = 3$, which is the number of

*Corresponding author. Email: adnan@cse.buet.ac.bd
†Email: saidurrahman@cse.buet.ac.bd
children. There are \( k = 3 \) groups, with \( n_1 = 3, n_2 = 2, \) and \( n_3 = 1. \) Since there are 6 objects in total, so \( n = 6. \)

Now the question is: Can we count the number of solutions? For identical objects, the number of distributions for \( m \) bins and \( n \) identical objects is \((n + m - 1)! / (n!(m - 1)!)[1–3].\) We use this formula to solve the counting problem for distinguishable objects as follows. We first distribute the fruits of class 1 to all the bins. The number of such distributions with \( n_1 \) objects and \( m \) bins is \((n_1 + m - 1)! / (n_1!(m - 1)!).\) Then we distribute the objects of the second class and so on up to the \( k \)th class. Thus the total number of distributions will be the product of all these solutions, as in the following expression:

\[
\frac{(n_1 + m - 1)!}{n_1!(m - 1)!} \frac{(n_2 + m - 1)!}{n_2!(m - 1)!} \cdots \frac{(n_k + m - 1)!}{n_k!(m - 1)!}.
\]

Let \( D(n, m, k) \) represent the set of all distributions of \( n \) objects to \( m \) bins where the objects fall into \( k \) different classes and each bin gets zero or more objects. For the previous example, we have \( D(6, 3, 3) \) representing all distributions where the number of distribution is \((5!/3!2!) \cdot (4!/2!2!) \cdot (3!/1!2!)) = 180. \) Thus we count the number of distributions. However, in this paper we are not interested in counting the number of distributions, rather we are interested in generating all distributions. Generating all distributions has practical applications in channel allocation in computer networks, CPU scheduling, memory management, etc. [1, 4, 5]. In these days of automation, machines may also require to distribute objects among candidates optimally.

Generating the complete list of all solutions of such combinatorial problems has many useful applications. For example, one can use such a list to search for a counter-example to some conjecture, to find the best solution among all solutions or to test and analyse an algorithm for its correctness or computational complexity. Early work in combinatorics focused on counting; because generating all objects requires a huge amount of computation. With the aid of fast computers it has now become feasible to list the objects in combinatorial classes. However, in order to generate the entire list of objects from a class of moderate size, extremely efficient algorithms are required even with the fastest computers. For the reason mentioned above, recently many researchers have concentrated their attention on developing efficient algorithms to generate all objects of a particular class without repetition [6]. Examples of such exhaustive generation of combinatorial objects include generating all integer partitions and set partitions, enumerating all binary trees, generating permutations and combinations, enumerating spanning trees, etc. [1, 6–15]. Generally, generating algorithms produce huge outputs, and the outputs dominate the running time of the generating algorithms. Therefore, many generating algorithms output solutions in an order such that each solution differs from the preceding one by a very small amount; they output each solution as the ‘difference’ from the preceding one. Such orderings of solutions are known as Gray codes [1, 9, 13].

Klingsberg [16] gave an average constant time algorithm for sequential listing of the composition of an integer \( n \) into \( k \) parts. Using an efficient tree traversal technique, Adnan and Rahman [1] improved the time complexity to constant time (in the ordinary sense) and proposed an efficient algorithm to generate all distributions of objects to bins when the objects are identical. They used an efficient generation method based on the family tree structure of the distributions. However, in this paper we are interested in generating all distributions of distinguishable objects. This problem is more difficult than that of the identical case since the solution space is larger. If we apply the algorithm for identical objects to distinguishable objects, there will be omission of distributions. Hence the algorithm for generating identical objects is not applicable to distinguishable objects.

The problem of generating all distribution of distinguishable objects can be viewed as generating multiset partitions when the partitions are ‘fixed’, ‘numbered’ and ‘ordered’. That
Distribution of distinguishable objects to bins

955

Figure 1. The family tree $T_{3,3,2}$.

means the number of partitions is fixed, the partitions are numbered and the assigned numbers to bins are not altered. Kawano and Nakano [17] gave an algorithm to generate multiset partitioning but the algorithm does not give solutions in constant time in the ordinary sense. Using the algorithm [9] for set partitioning, their algorithm [17] constructs a family tree for each type of element. Then they combine the solutions of each family tree to output each multiset partition. Since there are $k$ family trees for $k$ types of elements in the set, their algorithm takes $O(k)$ time to generate each solution. Their method is not applicable here since the partitions are fixed, ordered and numbered. Also we need to construct only one family tree for all solutions. Hence there is no need for recombination and our algorithm generates each solution in constant time for fixed, numbered and ordered partitions.

In this paper we propose an algorithm to generate all distributions of $n$ distinguishable objects to $m$ bins where the objects fall into $k$ different classes. Here, the number of bins is fixed and the bins are numbered and ordered. The algorithm generates each distribution in constant time without repetition. The main feature of this algorithm is that we define a tree structure, that is parent–child relationships, among the distributions in $D(n, m, k)$ (see figure 1). In such a ‘tree of distributions’, each node corresponds to a distribution of objects to bins and each node is generated from its parent in constant time. In the algorithm, we construct the tree structure among the distributions in such a way that the parent–child relationship is unique, and hence there is no chance of producing duplicate distributions. Once such a parent–child relationship is established, one can generate all the distributions in $D(n, m, k)$ by traversing the tree using the relationship. However, the problem of ordinary traversal is that after generating a distribution corresponding to the last vertex at the largest level in the tree, we have to return from the deep recursive call without outputting any sequence and hence ordinary traversal generates each distribution in constant time ‘on average’. To generate each distribution in $O(1)$ time (in the ordinary sense), we define two additional types of relationships: (i) relationship between left sibling and right sibling; and (ii) leaf-ancestor relationship. Thus our algorithm reduces many non-generation steps, and outputs each distribution in constant time in the ordinary sense (not in the average sense). Our algorithm, generates a new distribution from an existing one by making a constant number of changes and outputs each distribution as the difference from the preceding one. Thus we can regard the derived sequence of outputs as a combinatorial Gray code [3, 9, 13] for distributions. Our algorithm also generates the distributions in place, which means that the space complexity is linear. To the best of our knowledge, our algorithm is the first algorithm to generate all distributions in constant time per distribution in the ordinary sense. We also extend our algorithm to the case when the bins have priorities associated with them. In this case, the bins are numbered in the order of priority. The sequence of generations maintains an order so that the successive generations maintain priority.
The rest of the paper is organized as follows. Section 2 gives some definitions. In section 3, we define a tree structure among distributions in $D(n, m, k)$. In section 4, we present the algorithm for generating all distributions of distinguishable objects to bins. Finally section 5 gives conclusions.

2. Preliminaries

In this section we define some terms used in the 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 family tree is a rooted tree with a parent–child relationship. The vertices of a rooted tree have levels associated with them. The root has the lowest level, i.e. 0. The level for any other nodes is one more than its parent except the 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.

For a positive integer $n$ and $k < n$, set partition is the set of all partitions of $\{1, 2, \ldots, n\}$ into $k$ non-empty subsets. For instance, for $n = 4$ and $k = 2$ there are seven such partitions:

$$\{1, 2, 3\} \cup \{4\}, \{1, 2, 4\} \cup \{3\}, \{1, 3, 4\} \cup \{2\}, \{2, 3, 4\} \cup \{1\}, \{1, 2\} \cup \{3, 4\}, \{1, 3\} \cup \{2, 4\}, \{1, 4\} \cup \{2, 3\}.$$

A simple set is a set of elements where all the elements are identical. A multiset is a set of elements where all the elements are not identical. The elements of a multiset fall into different classes where the elements in the same class are identical but are distinguishable from those of other classes. For example, $\{1, 1, 2, 3, 1, 3, 2, 2\}$ is an example of multiset.

For positive integer $k$, let $a$ be a sequence of positive integers $t_1, t_2, \ldots, t_k$ where $t_j \geq 0$ for $1 \leq j \leq k$. We call $a$ a zero sequence if $t_1 = t_2 = \cdots = t_k = 0$. That means all the integers in a zero sequence are 0. We call $a$ a nonzero sequence if there exists an index $j$, $1 \leq j \leq k$, such that $t_j \neq 0$. That means a sequence is nonzero if at least one of the integers in the sequence is nonzero. Let $b$ be another sequence of positive integers $u_1, u_2, \ldots, u_k$ for $1 \leq j \leq k$. By addition of two sequences $a + b$ we mean the addition of corresponding elements $t_j + u_j$ where $1 \leq j \leq k$. Similarly, by subtraction of two sequences $a - b$ we mean the subtraction of corresponding elements $t_j - u_j$ where $1 \leq j \leq k$, and by equality of two sequences $a = b$ we mean the equality of corresponding elements $t_j = u_j$ where $1 \leq j \leq k$.

A listing of combinatorial objects is said to be in gray code order if each successive combinatorial objects in the listing differs by a constant amount. For example, the swapping of elements, or the flipping of a bit. In this paper, we establish such an ordering of all distributions of objects to bins so that each distribution can be generated by making a constant number of changes to the preceding distribution in the order.

For positive integers $n$, $m$ and $k$, let $A \in D(n, m, k)$ be a distribution of $n$ objects to $m$ bins where the objects fall into $k$ classes. Let, $n_j$ represent the number of objects in the $j$th class where $1 \leq j \leq k$. Clearly, $n_1 + n_2 + \cdots + n_k = n$ since every object must be in a class. The bins are ordered and numbered as $B_1, B_2, \ldots, B_m$. Each bin contains objects of different classes. We order the different types of objects in a bin so that we can keep track of objects of different classes. Let $a_i$ be a sequence of positive integers $(t_{i1}, t_{i2}, \ldots, t_{ik})$ where
Distribution of distinguishable objects to bins

Figure 2. Representation of the distribution of 3 objects to 2 bins where the objects fall into 2 classes, and 2 objects are from class 1 and 1 object from class 2.

t\_{ij}\) represents the number of objects of \(j\)th type in \(i\)th bin \(B_i\), for \(1 \leq i \leq m\), \(1 \leq j \leq k\). Then we can represent each \(A \in D(n, m, k)\) by a unique sequence \((a_1, a_2, \ldots, a_m)\). We call each \(a_i\) an inner sequence of \(A\). The sequence for \(A\) is unique for each distribution because the bins are ordered and numbered and also the objects of different types are ordered. For example, the sequence \(((0, 0), (2, 1))\) indicates that there are 2 bins because there are two sequences of integers and 3 objects, which is the sum of all the integers in the sequence and there are 2 classes of objects where 2 objects are from class 1 and 1 object from class 2. Also the second bin contains 3 objects, i.e. 2 objects from class 1 and 1 object from class 2 and the first bin is empty (see figure 2).

For each such sequence of sequences in \(D(n, m, k)\), we have the following equations:

\[
\sum_{i=1}^{m} \sum_{j=1}^{k} t_{ij} = n, \quad (1)
\]

\[
\sum_{i=1}^{m} t_{ij} = n_j, \quad \text{for } 1 \leq j \leq k, \text{ and} \quad (2)
\]

\[
\sum_{j=1}^{k} n_j = n. \quad (3)
\]

Equation 1 denotes that the sum of all the integers in the sequence of sequences is equal to the total number of objects. This holds because the number of objects are fixed and every object is distributed somewhere in some bin. In equation 2, we describe that the same kind of every object are present in the sequence. Since every object must be in a class, equation 3 holds.

In the following sections we describe an algorithm to generate all distributions of distinguishable objects to bins. For that purpose we define a unique parent–child relationship among the distributions in \(D(n, m, k)\) so that the relationship among the distributions can be represented by a tree with a suitable distribution as the root. Figure 1 shows such a tree of distributions where each distribution in the tree is in \(D(3, 3, 2)\). Once such a parent–child relationship is established, we can generate all the distributions in \(D(n, m, k)\) using the relationship. We do not need to build or store the entire tree of distributions at once, rather we generate each distribution in the order it appears in the tree structure.

In section 3 we define a tree structure among distributions in \(D(n, m, k)\) and in section 4 we present our algorithm, which generates each solution in \(O(1)\) time in the ordinary sense.

3. The family tree of distributions

In this section we define a tree structure \(T_{n,m,k}\) among the distributions in \(D(n, m, k)\).

For positive integers \(n\), \(m\) and \(k\), let, \(A \in D(n, m, k)\) be a distribution of \(n\) objects to \(B_1, B_2, \ldots, B_m\) bins where the objects fall into \(k\) classes. Let, \(n_j\) represent the number of objects in the \(j\)th class where \(1 \leq j \leq k\). From equation (3), \(n_1 + n_2 + \cdots + n_k = n\). For
each \( A \in D(n, m, k) \), we define a unique sequence of sequences of positive integers \((a_1, a_2, \ldots, a_m)\), where \( a_i \) represents a sequence of integers \((t_{1i}, t_{2i}, \ldots, t_{ki})\) where \( t_{ij} \) represents the number of objects of the \( j \)th type in the \( i \)th bin \( B_i \), for \( 1 \leq i \leq m, 1 \leq j \leq k \).

Now we define the family tree \( T_{n,m,k} \) as follows. Each node in \( T_{n,m,k} \) represents a distribution in \( D(n, m, k) \). If there are \( m \) bins then there are \( m \) levels in \( T_{n,m,k} \). A node is in level \( i \), \( 0 \leq i < m \) in \( T_{n,m,k} \) if \( t_{ij} = 0 \) for \( 1 \leq j \leq k \), \( 1 \leq l < (m - i) \) and \( a_{m-i} \) is nonzero sequence. So, a node at level \( m - 1 \) has no leftmost inner zero sequence before the leftmost inner nonzero sequence. As the level increases, the number of leftmost inner zero sequences decreases and vice versa. Since the family tree 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 \( T_{n,m,k} \) if \( t_{ij} = 0 \) for \( 1 \leq j \leq k \), \( 1 \leq l < m \) and \( t_{mj} \neq 0 \) for \( 1 \leq j \leq k \).

We also have from equation 1 that \( \sum_{i=1}^{m} \sum_{j=1}^{k} t_{ij} = n \). Substituting the values for \( t_{ij} \) for \( 1 \leq j \leq k \), \( 1 \leq l < m \) we find that \( \sum_{j=1}^{k} t_{mj} = n \). By using equation (2) and equation (3), we get \( t_{mj} = n_j \) where \( 1 \leq j \leq k \). Thus we can say that there can be exactly one such node which is our root. So, the sequence for root is \(((0, \ldots, 0), (0, \ldots, 0), \ldots, (0, \ldots, 0), (a_1, n_2, \ldots, n_k))\). In other words, we can say that the number of leftmost inner zero sequences before any inner nonzero sequence in the root is greater than any other sequence for any distribution in \( D(n, m, k) \).

To construct \( T_{n,m,k} \), we define two types of relationships: (a) parent–child relationship; and (b) child–parent relationship among the distributions in \( D(n, m, k) \), which are discussed in the following sections.

(a) Child-parent relationship

It is convenient to consider the child-parent relationship before the parent–child relationship.

Let, \( A \in D(n, m, k) \) be a sequence of sequences \((a_1, a_2, \ldots, a_m)\) which is not a root sequence, where \( a_i \) represents a sequence of integers \( t_{ij} \) for \( 1 \leq j \leq k \), \( 1 \leq l \leq m \). The sequence \( A \) corresponds to a node of level \( i \), \( 1 \leq i < m \). So, we have \( t_{ij} = 0 \) for \( 1 \leq j \leq k \), \( 1 \leq l < (m - i) \) and \( a_{m-i} \) is a nonzero sequence. Let \( A' \in D(n, m, k) \) be another sequence of sequences \((p_1, p_2, \ldots, p_{m-i}, p_{m-i+1}, \ldots, p_m)\), \( 1 \leq i < m \) such that \( p_1 = p_2 = \cdots = p_{m-i} \) are zero sequences and \( p_{m-i+1} = a_{m-i} + a_{m-i+1} \) and \( p_1 = a_l \) for \( m - i + 1 \leq l \leq m \). Then \( A' \) is at level \( i - 1 \) of \( T_{n,m,k} \). We call the sequence \( A' \) as the parent sequence of \( A \). Thus for each consecutive level we only deal with two sequences \( a_{m-i} \) and \( a_{m-i+1} \) and the rest of the sequences remains unchanged. The number of leftmost inner zero sequence increases in the parent sequence by applying the child–parent relationship. For example, the solution \(((1, 1), (1, 0), (1, 0), (0, 0), (0, 0))\), for \( n = 3, m = 2, k = 2 \) and \( n_1 = 2, n_2 = 1 \), is a node of level 1 because \( a_1 \) is a nonzero sequence. It has a unique parent \(((0, 0), (2, 1))\) as shown in figure 3.

(b) Parent-child relationship

The parent–child relationship is just the reverse of child–parent relationship.

Let, \( A \in D(n, m, k) \) be a sequence \((a_1, a_2, \ldots, a_m)\), where \( a_l = (t_{1l}, t_{2l}, \ldots, t_{kl}) \), \( 1 \leq l \leq m \). The sequence \( A \) corresponds to a node of level \( i \), \( 0 \leq i < m \). So, we have \( t_{ij} = 0 \) for \( 1 \leq j \leq k \), \( 1 \leq l < (m - i) \) and \( a_{m-i} \) is a nonzero sequence. Let \( A' \in D(n, m, k) \) be another sequence of sequences \((c_1, c_2, \ldots, c_{m-i-1}, c_{m-i}, \ldots, c_m)\), \( 0 \leq i < m \) such that \( c_1, c_2, \ldots, c_{m-i-2} \) are zero sequences, \( c_{m-i-1} + c_{m-i} = a_{m-i} \), \( c_{m-i-1} \) is a nonzero sequence and \( c_l = a_l \) for \( m - i + 1 \leq l \leq m \). Then \( A' \) is at level \( i + 1 \) of \( T_{n,m,k} \). We call the sequence \( A' \) a child sequence of \( A \). Like

![Figure 3](image-url)

The sequence \(((0, 0), (2, 1))\) has five children.
the child–parent relationship, here we also deal with only two inner sequences $a_{m-i-1}$ and $a_{m-i}$ and the rest of the sequences remains unchanged. Hence from the child–parent relationship, one can observe that the number of children of $A$ is equal to $(\prod_{j=1}^{k}(t_{m-i}+1)) - 1$. The number of leftmost zero sequence decreases in the child sequence by applying the parent–child relationship. For example, the solution $((0, 0), (2, 1))$, for $n = 3$, $m = 2$, $k = 2$ and $n_1 = 2$, $n_2 = 1$, is a node of level 0 because $a_1$ is a zero sequence, $a_2$ is not a zero sequence. Here, $(\prod_{j=1}^{k}(t_{m-i}+1)) - 1 = (2 + 1)(1 + 1) - 1 = 5$ so it has five children and they are $((1, 0), (1, 1)), ((2, 0), (0, 1)), ((0, 1), (2, 0)), ((1, 1), (1, 0))$ and $(2, 1), (0, 0))$ as shown in figure 3.

From the above definitions we can construct the family tree $T_{n,m,k}$. We take the sequence $A_r = a_1, a_2, \ldots, a_m$ as root where $a_1, a_2, \ldots, a_{m-1}$ are zero sequences and $a_m = (n_1, n_2, \ldots, n_k)$ as we mentioned before. The family tree $T_{n,m,k}$ for the distributions in $D(n, m, k)$ is shown in figure 1. Based on the above parent–child relationship, the following lemma proves that every distribution in $D(n, m, k)$ is present in $T_{n,m,k}$.

**Lemma 3.1** For any distribution $A \in D(n, m, k)$, there is a unique sequence of distributions that transforms $A$ into the root $A_r$ of $T_{n,m,k}$.

**Proof** Let, $A \in D(n, m, k)$ be a sequence, where $A$ is not the root sequence. We determine the level of $A$ in the family tree $T_{n,m,k}$. Then by applying child–parent relationship, we find the parent sequence $P(A)$ of $A$. Now if $P(A)$ is the root sequence, then we stop. Otherwise, we apply the same procedure to $P(A)$ and find its parent $P(P(A))$. By continuously applying this process of finding the parent sequence of the derived sequence, we have the unique sequence $A, P(A), P(P(A)), \ldots$ of sequences in $D(n, m, k)$ which eventually ends with the root sequence $A_r$ of $T_{n,m,k}$. We observe that $P(A)$ has at least one more zero than $A$ in its sequence. Thus $A, P(A), P(P(A)), \ldots$ never leads to a cycle and the level of the derived sequence decreases ending up with the level of the root sequence $A_r$. 

Lemma 3.1 ensures that there can be no omission of distributions in the family tree $T_{n,m,k}$. Since there is a unique sequence of operations that transforms a distribution $A \in D(n, m, k)$ into the root $A_r$ of $T_{n,m,k}$, by reversing the operations we can generate that particular distribution, starting from the root. We now have to make sure that the family tree $T_{n,m,k}$ represents distributions without repetition. Based on the parent–child and child–parent relationships, we have the following lemma for this property of $T_{n,m,k}$.

**Lemma 3.2** The family tree $T_{n,m,k}$ represents distributions in $D(n, m, k)$ without repetition.

**Proof** Given a sequence $A \in D(n, m, k)$, we define the children of $A$ in such a way that no other sequence in $D(n, m, k)$ can generate the same child. Let $A, B \in D(n, m, k)$ be two different sequences at level $i$ of $T_{n,m,k}$. For a contradiction, assume that $A$ and $B$ generate the same child $C$. Then $C$ is a sequence of level $i + 1$ of $T_{n,m,k}$. The sequences for $A, B$ and $C$ are $a_j, b_j$ and $c_j$ for $1 \leq j \leq m$. Clearly, $a_i = b_i$ for $1 \leq l \leq m - i - 1$ and the parent–child relationship yields $a_i = b_i = c_i$ for $m - i + 1 \leq l \leq m$. Therefore $a_l = b_l$ for $l \neq m - i$ and $1 \leq l \leq m$. But we have $a_1 + a_2 + \cdots + a_m = b_1 + b_2 + \cdots + b_m$ by equation 1. Then $a_l$ must be equal to $b_l$, for $1 \leq l \leq m$. This implies that $A$ and $B$ are the same sequence, a contradiction. Hence every sequence has a single and unique parent.

4. Generating distributions

In this section, we give an algorithm to construct $T_{n,m,k}$ and generate all distributions.
M. A. Adnan and M. S. Rahman

One can use the parent–child relationships described in the previous section to construct the family tree $T_{n,m,k}$ and hence generate all the distributions in $D(n, m, k)$ by traversing the tree using the relationships. If we can generate all child sequences of a given sequence in $D(n, m, k)$, then in a recursive manner we can construct $T_{n,m,k}$ and generate all sequences in $D(n, m, k)$. We have the root sequence $A_r = ((0, \ldots, 0), (0, \ldots, 0), \ldots, (0, \ldots, 0), (n_1, n_2, \ldots, n_k))$. We get the child sequence $A_c$ by using the parent to child relation discussed above.

**Procedure Find-All-Child-Distributions**

\(A = ((t_{11}, t_{12}, \ldots, t_{1k}), (t_{21}, t_{22}, \ldots, t_{2k}), \ldots, (t_{m1}, t_{m2}, \ldots, t_{mk})), i)\)

\{ $A$ is the current sequence, $i$ indicates the current level, $A_c$ is the child sequence \}

begin

Output $A$ \{Output the difference from the previous distribution\}

for $i_k = 0$ to $t_{(m-i)k}$

for $i_{k-1} = 0$ to $t_{(m-i)(k-1)}$

\ldots

for $i_1 = 1$ to $t_{(m-i)1}$

\begin{verbatim}
Find-All-Child-Distributions( $A_c = ((t_{11}, t_{12}, \ldots, t_{1k}), (t_{21}, t_{22}, \ldots, t_{2k}), \ldots, (t_{(m-i-2)1}, t_{(m-i-2)2}, \ldots, t_{(m-i-2)k}), (i_1, i_2, \ldots, i_k), (t_{(m-i+1)} - i_1, t_{(m-i+2)} - i_2, \ldots, t_{(m-i)k} - i_k), \ldots, (t_{m1}, t_{m2}, \ldots, t_{mk})), i + 1))$
\end{verbatim}

end;

Algorithm Find-All-Distributions($n, m$)

begin

Find-All-Child-Distributions($A_r = ((0, \ldots, 0), (0, \ldots, 0), \ldots, (0, \ldots, 0), (n_1, n_2, \ldots, n_k)), 0)$;

end.

Lemma 3.1 and 3.2 ensure that Algorithm Find-All-Distributions generates all distributions without repetition. We now have the following lemma.

**Lemma 4.1** The algorithm Find-All-Distributions uses $O(mk \log n)$ space and runs in $O(|D(n, m, k)|)$ time.

**Proof** We traverse the family tree $T_{n,m,k}$ and output each sequence at each corresponding vertex of $T_{n,m,k}$ when we visit the vertex for the first time. Hence, the algorithm takes $O(|D(n, m, k)|)$ time, i.e. constant time on average for each output. Our algorithm outputs each distribution as the difference from the previous one. The data structure that we use to represent the distribution is a sequence of sequences of integers where each integer represents the number of objects in a particular class in a particular bin. The value of each integer in the sequence can be at most $n$. Hence we need $\log n$ bits to represent the number of objects of each type in each bin. Thus, the memory requirement is $O(mk \log n)$ bits, where $k$ is the number of types of objects and $m$ is the number of bins and $n$ is the total number of objects.

The algorithm Find-All-Distributions generates all sequences in $D(n, m, k)$ in $O(|D(n, m, k)|)$ time. Thus the algorithm generates each sequence in $O(1)$ time ‘on average’. However, after generating a sequence corresponding to the last vertex in the largest level in a large subtree of $T_{n,m,k}$, we have to return from the deep recursive call without outputting any sequence and hence we cannot generate each sequence in $O(1)$ time (in the ordinary sense). To generate each distribution in $O(1)$ time (in the ordinary sense), we introduce two additional types of relationships:
(i) relationship between left sibling and right sibling; and
(ii) leaf–ancestor relationship.

(i) Relationship between left sibling and right sibling

Let, \( A \in D(n, m, k) \) be a sequence of sequences \((a_1, a_2, \ldots, a_m)\) which is not a root sequence, where \( a_j \) represents a sequence of integers \( t_{ij} \) for \( 1 \leq j \leq k, 1 \leq l \leq m \). The sequence \( A \) corresponds to a node of level \( i \), \( 0 \leq i < m \). So, we have \( t_{ij} = 0 \) for \( 1 \leq j \leq k, 1 \leq l \leq (m - i) \) and \( a_{m-l} \) is a nonzero sequence. We say the right sibling sequence, \( A_s \in D(n, m, k) \) of this node \( A \) exists if \( a_{m-l+1} \) is a nonzero sequence at level \( i \). Then we call the sequence \( A \) left sibling of \( A_s \).

We define the sequence for \( A_s \) as \( s_1, s_2, \ldots, s_{m-i}, s_{m-i+1}, \ldots, s_m, 1 \leq i < m \) where \( s_1, s_2, \ldots, s_{m-i-1} \) are zero sequences and \( s_j = a_j \) for \( m - i + 2 \leq j \leq m \) and to find \( s_{m-i}, s_{m-i+1} \) we apply the child–parent relationship and then the parent–child relationship. Thus, we observe that \( A_s \) is a node of level \( i \), \( 1 \leq i < m \) and so \( s_1, s_2, \ldots, s_{m-i-1} \) are zero sequences and \( s_{m-i} \) is nonzero sequence for \( 1 \leq i < m \). For example, the solution \( ((0, 0), (1, 0), (1, 1))\), for \( n = 3, m = 3, k = 2 \) and \( n_1 = 2, n_2 = 1 \), is a node of level 1 because \( a_2 \) is the first nonzero sequence. It has a unique right sibling \( ((0, 0), (2, 0), (0, 1)) \) as shown in figure 4.

(ii) Leaf-ancestor relationship

To avoid returning from a deep recursive call without outputting any sequence, we define a leaf–ancestor relationship. After generating the sequence \( A_l \) of the last vertex in the largest level, i.e. rightmost leaf, we do not return to the parent. Instead, we return to the nearest ancestor \( A_a \) that has right sibling. By rightmost leaf we mean that leaf which has no right sibling. Thus this leaf–ancestor relation saves many nongeneration steps. Another reason for defining the leaf–ancestor relationship is that the nearest ancestor can be generated from the leaf sequence by just a simple swap operation between two inner sequences in the sequence. This is possible due to the data structure that we use for this case (as described in the following subsection). For the swap operation we just swap pointers to sequences. The other inner sequences remain unchanged.

Let, \( A_l \in D(n, m, k) \) be a sequence of sequences \((a_1, a_2, \ldots, a_m)\) of a leaf, where \( a_p \) represents a sequence of integers \( t_{pj} \) for \( 1 \leq j \leq k, 1 \leq p \leq m \). The sequence \( A_l \) corresponds to a node of level \( m - 1 \). So, we have that \( a_{m-l} \) is a nonzero sequence. We say that the ancestor sequence \( A_a \in D(n, m, k) \) of this node \( A_l \) exists if \( a_2 \) is a zero sequence, that is it has no right sibling. We define a unique ancestor sequence of \( A_l \) at level \( m - 1 - q \) where \( a_2, a_3, \ldots, a_{q+1} \) are zero sequences and \( a_{q+2} \) is a nonzero sequence. This means we want to skip the long sequence of the inner zero sequence in the sequence for \( A_l \). The nearest ancestor sequence is determined by the number of zero sequences in this sequence. We denote the number of inner zero sequences as \( q \). This \( q \) will determine the level and sequence of the nearest ancestor \( A_a \) that has a sibling.

We define the sequence for \( A_a \) as \( s_1, s_2, \ldots, s_q, s_{q+1}, \ldots, s_m \), where \( s_1, s_2, \ldots, s_q \) are zero sequences and \( s_{q+1} = a_1 \) and \( s_j = a_j \) for \( q + 1 < j \leq m \). In other words, we just swap the sequences \( a_1 \) and \( a_{q+1} \) in the sequence and the rest of the inner sequences remain unchanged.
For example, in figure 4 the solution \( ((2,0), (0,0), (0,1)), \) for \( n = 3, m = 3, k = 2 \) and \( n_1 = 2, n_2 = 1, \) is a node of level 2 because \( a_1 \) is a nonzero sequence. It has a unique ancestor \( ((0,0), (2,0), (0,1)) \), which is obtained by swapping the first and second sequences. We have the following lemma on uniqueness of the nearest ancestor \( A_a \) of \( A_l \).

**Lemma 4.2** Let \( A_l \) be a leaf sequence of \( T_{n,m,k} \) having no right sibling. Then \( A_l \) has a unique ancestor sequence \( A_a \) in \( T_{n,m,k} \). Furthermore, either \( A_a \) has a right sibling in \( T_{n,m,k} \) or \( A_a \) is the root \( A_r \) of \( T_{n,m,k} \).

**Proof** Let the sequence for \( A_l \in D(n, m, k) \) be \( (a_1, a_2, \ldots, a_m) \) and it corresponds to a node of level \( m - 1 \) of \( T_{n,m,k} \). Note that \( a_1 \) is a nonzero sequence and \( a_2 \) is a zero sequence. We get the sequence for \( A_a \) by swapping \( a_1 \) and \( a_{q+1} \) where \( q \) is the number of consecutive inner zero sequences after \( a_1 \). Clearly \( A_a \) is an ancestor of \( A_l \). Note that \( A_a \) is at level \( m - 1 - q \). By lemma 3.2, the parent–child relation is unique. Hence by repeatedly applying the child–parent relation on \( A_l \), we will reach a unique ancestor at level \( m - 1 - q \). For \( q = m - 1 \), one can observe that we get the root sequence \( A_r \) by swapping \( a_1 \) and \( a_m \). For \( 1 \leq q < m - 1 \), we get the unique ancestor sequence \( A_a \) that has a right sibling. \( \blacksquare \)

Lemma 4.2 ensures that \( A_l \) has a unique ancestor \( A_a \). As we see later \( A_a \) plays an important role in our algorithm. Now we present the algorithm to generate all distributions in \( D(n, m, k) \).

We use three relations in this algorithm; they are the parent–child relation, the relation between left sibling and right sibling, and the leaf–ancestor relation. By applying the parent–child relation, we go from the root down the family tree \( T_{n,m,k} \) until we reach the leaf at level \( m - 1 \). Then we apply the relationship between left sibling and right sibling to traverse horizontally until we reach a node that has no right sibling. Then by applying the leaf–ancestor relation, we return to the nearest ancestor that has a right sibling. Then we again apply the relation between left sibling and right sibling. The sequence of applying relationships and generating distributions continues until we reach the root. This algorithm thus reduces the number of nongeneration steps and generates each sequence in \( O(1) \) time (in the ordinary sense).

**Procedure Find-All-Child-Distributions2**

\[ A = ((t_{11}, t_{12}, \ldots, t_{1k}), (t_{21}, t_{22}, \ldots, t_{2k}), \ldots, (t_{m1}, t_{m2}, \ldots, t_{mk})), i) \]

\{ \( A \) is the current sequence \}

**begin**

Output \( A \) \{Output the difference from the previous distribution\}

if \( A \) has child **then**

begin

Generate the first child \( A_c \) of \( A \);

Find-All-Child-Distributions2(\( A_c \), \( i + 1 \));

end

**else**

if \( A \) has right sibling **then**

begin

Generate right sibling \( A_s \);

Find-All-Child-Distributions2(\( A_s \), \( i \));

end

else

begin

Generate the ancestor \( A_a \) of \( A \) at level \( i - q \) such that either \( A_a \) has right sibling or \( A_a \) is root;

end

end

procedure
if $A_a$ is the root at level 0
then done
else
begin
Generate right sibling $A_{as}$ of $A_a$;
Find-All-Child-Distributions2($A_{as}, i - q$);
end
end;

Algorithm Find-All-Distributions2($n, m$)
begin
Find-All-Child-Distributions2($A_r = ((0, \ldots, 0), (0, \ldots, 0), \ldots, (0, \ldots, 0), n_1, n_2, \ldots, n_k), 0$);
end.

Tree traversal according to the proposed efficient algorithm is depicted in figure 4. Now we describe the data structure that we use to represent a distribution in $D(n, m, k)$ that will help us to generate each distribution in constant time. Note that we may need to return to ancestor $A_a$ if the current node is a leaf $A_l$ and for a leaf sequence $A_l$ we have $a_1$ is a nonzero sequence. $A_a$ is obtained from $A_l$ by swapping $a_1$ and $a_{q+1}$ where $q$ is the number of consecutive zero sequences after $a_1$. Now, to find $q$ we have to search the sequence $A_l$ from $a_1$ to $a_{q+1}$ such that $a_2, a_3, \ldots, a_{q+1}$ are inner zero sequences and $a_1, a_{q+2}$ are inner nonzero sequences. We reduce the search complexity by keeping extra information as shown in figure 5. The information consists of the number of subsequences of consecutive inner zero sequence and the number of inner zero sequence in each subsequence after $a_{m-i}$, where $i$ is the current level. For this we keep a stack of size $m/2$. The top of the stack determines the current $q$. Initially the stack is empty. As soon as we find a zero sequence, when moving from parent to child or left sibling to right sibling, we push a 1 on the stack. We increment the top of the stack for consecutive zero sequence. We perform a pop operation when we apply the leaf–ancestor relation. The stack operations are shown in figure 5. One can observe that there can be at most $m/2$ subsequences of consecutive inner zero sequence in a sequence of size $m$. Therefore, in worst case we need a stack of size $m/2$.

The operations that we use to generate distributions are addition, subtraction, increment, decrement and swap. The indices of the two operands for these operations are known. So, we might think of keeping an array of integers. Since for distinguishable objects we deal with a sequence of sequences, we may want to use an array of an array of integers, which means a two-dimensional array of integers. But note that to apply the leaf–ancestor relationship we need to swap an entire sequence of integers. By keeping a 2D-array of integers it will take $O(k)$ time to swap such an array of integers. This is not efficient. To do the swap operation

![Figure 5. Efficient traversal of $T_{4,3}$ keeping extra information.](image-url)
in constant time, we use the special data structure shown in figure 6. We keep an array of pointers for each bin pointing to an array of integers. The array of integers represents the inner sequence that is the sequence of different types of objects in a particular bin. The structure may be viewed as an array of objects where each object is an array of integers in the object-oriented sense. Thus by swapping the pointers we will be able to swap an entire array in $O(1)$ time.

Now we examine the memory requirement of the data structure that we use. There are a sequence of pointers of $O(m)$ size where $m$ is the number of bins. Each pointer points to an array of integers of size $O(k)$ where $k$ is the number of types of objects. The value of each integer in the array can be at most $n$. Hence we need $\lg n$ bits to represent the number of objects of each type in each bin. Thus, the memory requirement is $O(mk \lg n)$ bits. The other operations addition, subtraction, increment and decrement operations remain constant time and are not hampered by the modification of the data structure since the array structure is also kept intact.

In summary, for a sequence we need $O(mk \lg n)$ space and additional $m/2$ space is required for stack manipulation. Hence the algorithm takes $O(mk \lg n)$ space. One can observe that the algorithm generates all sequences such that each sequence in $T_{n,m,k}$ can be obtained from the preceding one by at most two operations. Note that if $A$ corresponds to a vertex $v$ of $T_{n,m,k}$, which has no child and no sibling, we need two steps, one for tracing its ancestor and the other for tracing the ancestor’s right sibling. Otherwise, we need only one step to generate the next sequence. Thus the algorithm generates each sequence in $O(1)$ time per sequence. Note that each sequence is similar to the preceding one, since it can be obtained by at most two operations (see figure 7). Thus, we can regard the derived sequence of the sequences as a combinatorial Gray code [3, 9, 13] for distributions. Thus we have the following theorem.

**Theorem 4.3** The algorithm **Find-All-Distributions** uses $O(mk \lg n)$ space and generates each distribution in $D(n, m, k)$ in constant time (in the ordinary sense).

5. Conclusions

In this paper we give a simple elegant algorithm to generate all distributions in $D(n, m, k)$. The algorithm generates each distribution in constant time with linear space complexity. We
Distribution of distinguishable objects to bins

also present an efficient tree traversal algorithm that generates each solution in \(O(1)\) time. Note that each sequence is similar to the preceding one, since it can be obtained by at most two operations. Thus, we can regard the derived sequence of the sequences as a combinatorial Gray code \([3, 9, 13]\) for distributions. Our algorithm can also be extended to the case when the bins have priorities associated with them. In this case, the bins are numbered in the order of priority. The sequence of generations will maintain an order such that the bin with highest priority gets the highest number of objects first and then the priorities of the bins are decreased one by one. Thus the sequence of generations maintains an order so that the generations maintain priority.

The main feature of our algorithms is that they are constant time solutions, which is a very important requirement for generation problems.

Acknowledgements

We wish to thank the two anonymous referees for their valuable comments and suggestions for improving the presentation of the paper and for correcting the space complexity and figure 7.

References