多次元分割の列挙

菊地洋右† 山中克久†† 中野眞一†††

† 津山工業高等専門学校 情報工学科 〒708-8509 岡山県津山市沼 624-1
†† 電気通信大学大学院 情報システム学研究科 〒182-8585 東京都調布市調布ケ丘 1-5-1
††† 群馬大学大学院 情報工学専攻 〒376-8515 群馬県桐生市天神町 1-5-1
E-mail: †kikuchi@tsuyama-ct.ac.jp, ††yamanaka@is.uec.ac.jp, †††nakano@cs.gunma-u.ac.jp

あらまし 本文では, 自然数 N の多次元分割を列挙するシンプルなアルゴリズムを提案する. この問題は, 整数分割や平面分割の列挙を含む, 組合せ論における基本的な問題の 1 つである. 本文では, 次元数 d が与えられたとき, d 次元分割を 1 つ当たり最悪でも $O(d)$ 時間で列挙するアルゴリズムを与える. 既存のアルゴリズムは非常に複雑で, 多くの “goto” 文を含んでいるが, 我々のアルゴリズムはシンプルかつ高速である. また, 提案したアルゴリズムを改良することにより, ちょうど d 次元の分割を 1 つ当たり $O(d)$ 時間で列挙できることを示す.

キーワード アルゴリズム, 列挙, 多次元分割, 家系木

A Simple Generation of Multi-dimensional Partitions

Yosuke KIKUCHI†, Katsuhisa YAMANAKA††, and Shin-ichi NAKANO†††

† Department of Electronics and Computer Engineering, Tsuyama National College of Technology, Numa 624-1, Tsuyama, Okayama, 708-8501, Japan
†† Graduate School of Information Systems, The University of Electro-Communications, Chofugaoka 1-5-1, Chofu, Tokyo, 182-8585, Japan
††† Graduate School of Computer Science, Gunma University, Tenjin-cho 1-5-1, Kiryu, Gunma, 376-8515, Japan
E-mail: †kikuchi@tsuyama-ct.ac.jp, ††yamanaka@is.uec.ac.jp, †††nakano@cs.gunma-u.ac.jp

Abstract This paper gives a simple algorithm to generate all multi-dimensional partitions of a positive integer N. The problem is one of the basic problems in combinatorics, and it includes generations of integer partitions and plane partitions. For a given integer d as dimension, our algorithm generates each partition of a given integer in $O(d)$ time for each without repetition. The known algorithm is complicated and includes many “goto” statements, while our algorithm is simple and efficient. Also, we propose an algorithm to generate all exactly d-dimensional partition in $O(d)$ time for each.

Key words algorithm, generation, multi-dimensional partition, family tree

1. Introduction

It is useful to have the complete list of objects for a particular class. One can use such a list to search for a counter-example to some conjecture, to find the best object among all candidates, or to experimentally measure an average performance of an algorithm over all possible inputs.

Many algorithms to generate all objects in a particular class, without repetition, are already known [6], [15], [16], [18], [22], [30], [33]. Many excellent textbooks have been published on the subject [11], [13], [14], [32].

In this paper we consider the following generation *problem*. For a positive integer N, a d-dimensional partition A_d of N is a kind of d-dimensional array of nonnegative integers [7], [8]. Let each element of the array be denoted by $A_d[i_1, \dots, i_d]$. An array is d-dimensional partition of N, if

$$\sum_{i_1,\dots,i_d} A_d[i_1,\dots,i_d] = N \qquad (1)$$

and

$$A_d[i_1,\dots,i_k,\dots,i_d] \geq A_d[i_1,\dots,i_k+1,\dots,i_d] \quad \text{for all } k. \qquad (2)$$

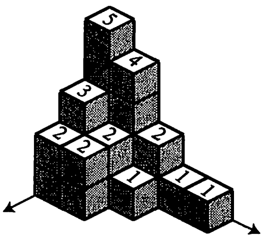

Figure 1 An illustration of a plane partition.

Let $S(N,d)$ be the set of all d-dimensional partitions of N. For example, $(A_4[1,1,1,1] = 2, A_4[1,2,1,1] = 1, A_4[2,1,1,1] = 1, A_4[2,2,1,1] = 1)$ is in $S(5,4)$. However, $(A_4[1,1,1,1] = 2, A_4[1,1,1,2] = 1, A_4[2,1,1,1] = 1, A_4[2,2,1,1] = 1)$ is not in $S(5,4)$, since $A_4[2,2,1,1] = 1$ and $A_4[1,2,1,1] = 0$ violate the decreasing condition (2). Note that we write a d-dimensional partition by omitting zero value elements.

If $d = 1$, the partition is called *integer partition*: For a positive integer N, a partition of N is a sequence $a_1a_2\dots a_m$ of nonnegative integers $a_1 \geq \cdots \geq a_m$ such that $N = a_1 + \cdots + a_m$. Let $S(N,1)$ be the set of all integer partitions of N. For instance, for $N = 5$ there are seven such partitions: 5, 41, 311, 2111, 11111, 32, 221 and $|S(5,1)| = 7$. Partitions above are rewritten by our notation: $(A_1[1] = 5)$, $(A_1[1] = 4, A_1[2] = 1)$, $(A_1[1] = 3, A_1[2] = 1, A_1[3] = 1)$, $(A_1[1] = 2, A_1[2] = 1, A_1[3] = 1, A_1[4] = 1)$, $(A_1[1] = 1, A_1[2] = 1, A_1[3] = 1, A_1[4] = 1, A_1[5] = 1)$, $(A_1[1] = 3, A_1[2] = 2)$, $(A_1[1] = 2, A_1[2] = 2, A_1[3] = 1)$. The asymptotic number of integer partitions of an integer N, $|S(N,1)|$, is given by the Hardy-Ramanujan-Rademacher asymptotic formula [2, p.70]

$$|S(N,1)| \sim \frac{1}{4N\sqrt{3}} e^{\pi\sqrt{\frac{2N}{3}}}.$$

If $d = 2$, the partition is called *plane partition* [3], [5], [17]. A plane partition is illustrated by stacking by cubes. For example, Fig. 1 represents a plane partition $(A_2[1,1] = 5, A_2[1,2] = 3, A_2[1,3] = 2, A_2[2,1] = 4, A_2[2,2] = 2, A_2[2,3] = 2, A_2[3,1] = 2, A_2[3,2] = 1, A_2[4,1] = 1, A_2[5,1] = 1)$. For $N = 5$, there are $|S(5,2)| = 24$ plane partitions: $(A_2[1,1] = 5)$ $(A_2[1,1] = 4,\ A_2[1,2] = 1)$, $(A_2[1,1] = 3,\ A_2[1,2] = 1,\ A_2[1,3] = 1)$, $(A_2[1,1] = 2,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[1,4] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[1,4] = 1, A_2[1,5] = 1)$, $(A_2[1,1] = 3,\ A_2[1,2] = 2)$, $(A_2[1,1] = 2,\ A_2[1,2] = 2,\ A_2[1,3] = 1)$, $(A_2[1,1] = 4,\ A_2[2,1] = 1)$, $(A_2[1,1] = 3,\ A_2[2,1] = 1,\ A_2[3,1] = 1)$, $(A_2[1,1] = 2,\ A_2[2,1] = 1,\ A_2[3,1] = 1,\ A_2[4,1] = 1)$, $(A_2[1,1] = 1,\ A_2[2,1] = 1,\ A_2[3,1] = 1,\ A_2[4,1] = 1,\ A_2[5,1] = 1)$, $(A_2[1,1] = 3,\ A_2[2,1] = 2)$, $(A_2[1,1] = 2,\ A_2[2,1] = 2,\ A_2[3,1] = 1)$, $(A_2[1,1] = 3,\ A_2[1,2] = 1,\ A_2[2,1] = 1)$, $(A_2[1,1] = 2,\ A_2[1,2] = 2,\ A_2[2,1] = 1)$, $(A_2[1,1] = 2,\ A_2[1,2] = 1,\ A_2[2,1] = 2)$, $(A_2[1,1] = 2,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[2,1] = 1)$, $(A_2[1,1] = 2,\ A_2[1,2] = 1,\ A_2[2,1] = 1,\ A_2[2,2] = 1)$, $(A_2[1,1] = 2,\ A_2[1,2] = 1,\ A_2[2,1] = 1,\ A_2[3,1] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[2,1] = 1,\ A_2[2,2] = 1,\ A_2[3,1] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[2,1] = 1,\ A_2[2,2] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[2,1] = 1,\ A_2[3,1] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[1,3] = 1,\ A_2[1,4] = 1,\ A_2[2,1] = 1)$, $(A_2[1,1] = 1,\ A_2[1,2] = 1,\ A_2[2,1] = 1,\ A_2[3,1] = 1,\ A_2[4,1] = 1)$. For plane partitions, there is MacMahon's generating function [17] and Bender and Knuth gave a simple proof of the generating function [5].

Multi-dimensional partitions are a natural extension of integer partitions and plane partitions. An asymptotic result for the number of multi-dimensional partitions is known. Bhatia *et. al.* [7] showed that asymptotically the number of d-dimensional partitions of an integer N goes as $\exp(CN^{d/(d+1)})$ where C is a positive finite constant independent of N but dependent on d.

Many algorithms to generate all integer partitions have been proposed. See [1], [8]~[10], [19], [20], [25]~[29], [31], [34]~[36]. The algorithm in [8] is for multi-dimensional partitions but complicated and includes many "goto" statements. On the other hand, we propose a simple and efficient algorithm to generate all multi-dimensional partitions.

Our main idea of the algorithm is as follows. We first define a rooted tree such that each vertex corresponds to a partition in $S(N,d)$, and each edge corresponds to a relation between two partitions. Then by traversing the tree all partitions in $S(N,d)$ are generated. This technique is called the reverse search and first proposed by Avis and Fukuda [4] and is applied to various generation problems [15], [21]~[24]. Note that our algorithm outputs each partition as the difference from the preceding one. With a similar technique we have already solved some generation problems for graphs [15], [21]~[24], set partitions [12] and integer partitions [34]. This paper extends the technique for d-dimensional partitions.

Above idea was applied to an algorithm for integer partitions in [34], then this paper extends their algorithm. Their algorithm generates all integer partitions for an integer in constant time for each in worst case. In other word, their algorithm outputs the partitions with *constant time delay* [36].

Our algorithm generates all d-dimensional partition of an integer in $O(d)$ time for each in worst case. For fixed dimension, it generates all d-dimensional partitions "in constant time" for each in worst case, including integer partitions and plane partitions.

In this paper we give a simple algorithm to generate all d-dimensional partitions of an integer N. Our algorithm employs addition and subtraction for non-negative integers. We assume that $n+1$ and $n-1$ can be calculated from given non-negative integer n in constant time.

If $N=1$, then generating all partitions is trivial. Hence we assume $N>1$ in this paper.

The rest of the paper is organized as follows. Section 2 shows a tree structure on the d-dimensional partitions of an integer. Section 3 presents our first algorithm. The algorithm generates each partition in $S(N,d)$ in $O(d)$ time on average. In Section 4 we improve the algorithm so that it generates each partition in $O(d)$ time in worst case. By slightly modifying the algorithm we give one more algorithm to generate all partitions of an integer with some additional property in Section 5. Finally Section 6 is a conclusion.

2. The Family Tree

In this section we define a tree structure among partitions in $S(N,d)$.

We define a d-dimensional partition $A_d \in S(N,d)$ of a positive integer N as follows. Let A_d be a d-dimensional integer array. Each element of A_d is denoted as $A_d[i_1,\dots,i_d]$. A_d is a *d-dimensional partition* of an integer N if $N=\sum_{i_1,\dots,i_d} A_d[i_1,\dots,i_d]$ and

$$A_d[i_1,\dots,i_k,\dots,i_d] \geqq A_d[i_1,\dots,i_k+1,\dots,i_d] \text{ for all } k.$$

The nonzero elements of A_d are called *the parts* of A_d. Throughout this paper, we denote A_d as a sequence of the parts such that each part appears in lexicographical order of its index. For convenience, we omit zero value elements. If the number of parts of A_d is 1, then $A_d=(A_d[1,\dots,1]=N)$ holds. We call it *the root partition* of $S(N,d)$.

Then we define *the parent partition* $P(A_d)$ of each partition A_d in $S(N,d)$ except for the root partition as follows. Suppose $A_d=(A_d[1,\dots,1],\dots,A_d[l_1,\dots,l_d])$ is a partition in $S(N,d)$, and A_d is not the root partition. Note that $A_d[1,\dots,1]$ and $A_d[l_1,\dots,l_d]$ are the first and the last part of A_d in lexicographical order, respectively. For some k and a part $A_d[i_1,\dots,i_k,\dots,i_d]$ of A_d, the part $A_d[i_1,\dots,i_k+1,\dots,i_d]$ is the *k-th lower neighbour part* of $A_d[i_1,\dots,i_k,\dots,i_d]$ and the part $A_d[i_1,\dots,i_k-1,\dots,i_d]$ is the *k-th upper neighbour part* of $A_d[i_1,\dots,i_k,\dots,i_d]$. We have the following two cases.

Case 1: $A_d[l_1,\dots,l_d]=1$.

We define $P(A_d)$ as the partition derived from A_d by removing $A_d[l_1,\dots,l_d]$ and adding one to $A_d[1,\dots,1]$. Note that the number of parts of $P(A_d)$ is one less than that of A_d.

Case 2: $A_d[l_1,\dots,l_d]>1$.

We define $P(A_d)$ as the partition derived from A_d by subtracting one from $A_d[l_1,\dots,l_d]$ and adding one to $A_d[1,\dots,1]$. Note that the number of parts of $P(A_d)$ is equal to that of A_d.

A d-dimensional partition A_d is called *a child partition* of $P(A_d)$. Note that A_d has a unique parent partition $P(A_d)$, and on the other hand $P(A_d)$ has at most $d+1$ child partitions, say d Case 1 children and a Case 2 child. We have the following lemma.

[Lemma 2.1] If $A_d \in S(N,d)$ and A_d is not the root partition, then $P(A_d) \in S(N,d)$.

By the lemma above, given a d-dimensional partition A_d in $S(N,d)$, where A_d is not the root partition, repeatedly finding the parent partition of the derived partition produces a unique sequence $A_d, P(A_d), P(P(A_d)),\dots$ of partitions in $S(N,d)$, which eventually ends with the root partition. By merging these sequences we have *the family tree* of $S(N,d)$, denoted by $T_{N,d}$, in which the vertices correspond to the partitions in $S(N,d)$, and each edge corresponds to the pair of each A_d and $P(A_d)$. For instance, $T_{5,2}$ is shown in Fig. 2.

3. Algorithm

This section gives an algorithm to construct $T_{N,d}$ and generate all partitions in $S(N,d)$.

If all child partitions of a given partition in $S(N,d)$ can be generated, then $T_{N,d}$ can be constructed in a recursive manner, and all partitions in $S(N,d)$ can be generated. How can we generate all child partitions of a given partition?

Let $A_d=(A_d[1,\dots,1],\dots,A_d[l_1,\dots,l_d])$ be a partition in $S(N,d)$. We are going to give a method to generate all child partitions of A_d.

We now need some definitions. Let $A_d[\pm]$ be a partition derived from A_d by subtracting one from $A_d[1,\dots,1]$ and adding one to $A_d[l_1,\dots,l_d]$. Intuitively $A_d[\pm]$ is a possible Case 2 child of A_d. Also, let $A_d[+i]$ be a partition derived from A_d by subtracting one from $A_d[1,\dots,1]$ and adding one to an element $A_d[m_1,\dots,m_d]$ such that $m_1=l_1,\dots,m_{i-1}=l_{i-1}, m_i=l_i+1, m_{i+1}=\cdots=m_d=1$. Intuitively $A_d[+i]$ is a possible Case 1 child of A_d. Note that $A_d[m_1,\dots,m_d]$ is the last part of $A_d[+i]$ in lexicographical order. Thus, $A_d[\pm]=(A_d[1,\dots,1]-1,\dots,A_d[l_1,\dots,l_d]+1)$ and $A_d[+i]=(A_d[1,\dots,1]-1,\dots,A_d[l_1,\dots,l_d], A_d[m_1,\dots,m_d])$.

If A_d is the root partition, then we can observe that it has exactly d Case 1 children. Otherwise we have the following two cases.

Case 1: $A_d[1,\dots,1]$ is equal to a lower neighbour of $A_d[1,\dots,1]$.

Let $A_d[k_1,\dots,k_d]$ be such a lower neighbour. In this case,

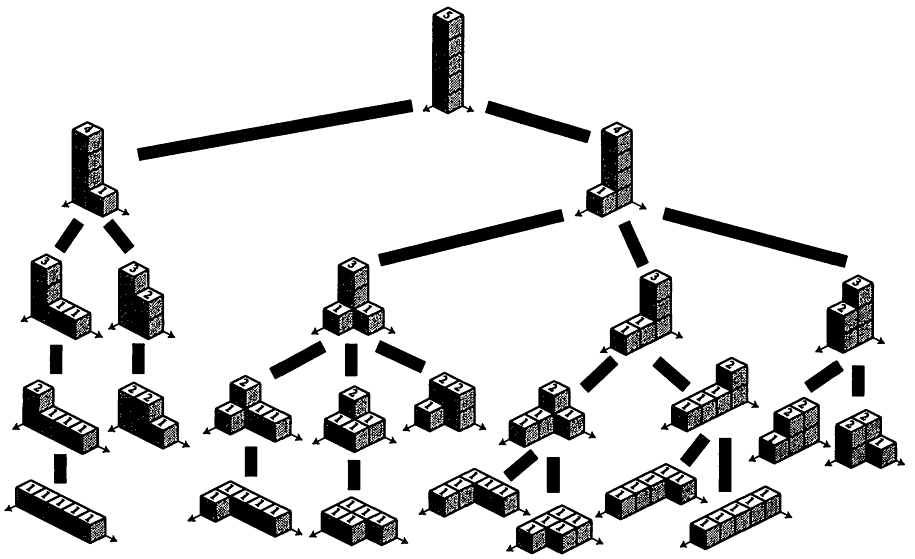

Figure 2 The family tree $T_{5,2}$.

$A_d[1,\dots,1]$ and $A_d[k_1,\dots,k_d]$ violate the decreasing condition (2) in $A_d[\pm]$ and $A_d[+i]$. Therefore A_d has no child partition.

Case 2: $A_d[1,\dots,1]$ is larger than its every lower neighbour.

For each $i = 1,\dots,d$, if every upper neighbour of $A_d[m_1,\dots,m_d]$ is nonzero integer, $A_d[+i]$ is a partition in $S(N,d)$, and $A_d[+i]$ is a Case 1 child of A_d. Note that $A_d[+1]$ is always a child partition of A_d.

Let's next consider $A_d[\pm]$. There are the following two cases.

Case 2(a): $A_d[l_1,\dots,l_d]$ is equal to an upper neighbour of $A_d[l_1,\dots,l_d]$.

Let $A_d[k_1,\dots,k_d]$ be the upper neighbour of $A_d[l_1,\dots,l_d]$ that is equal to $A_d[l_1,\dots,l_d]$. In this case, $A_d[l_1,\dots,l_d]$ and $A_d[k_1,\dots,k_d]$ violate the decreasing condition (2) in $A_d[\pm]$. Thus $A_d[\pm]$ is not partition, then A_d has no Case 2 child.

Case 2(b): $A_d[l_1,\dots,l_d]$ is smaller than its every upper neighbour.

If $A_d[l_1,\dots,l_d]$ is one of the lower neighbours of $A_d[1,\dots,1]$ and $A_d[1,\dots,1] - A_d[l_1,\dots,l_d] = 1$, then $A_d[1,\dots,1]$ and $A_d[l_1,\dots,l_d]$ violate the decreasing condition (2) in $A_d[\pm]$. Thus, $A_d[\pm]$ is not a partition in $S(N,d)$, and A_d has no Case 2 child. Otherwise, $A_d[\pm]$ is the Case 2 child of A_d.

For instance, for $A_4 = (A_4[1,1,1,1] = 5,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 1)$ in $S(20,4)$, $A_4[\pm] = (A_4[1,1,1,1] = 4,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 2)$ is the Case 2 child and $A_4[+1] = (A_4[1,1,1,1] = 4,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 1,\ A_4[3,1,1,1] = 1)$, $A_4[+2] = (A_4[1,1,1,1] = 4,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 1,\ A_4[2,2,1,1] = 1)$ are the Case 1 children of A_4. However, $A_4[+3] = (A_4[1,1,1,1] = 4,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 1,\ A_4[2,1,4,1] = 1)$ and $A_4[+4] = (A_4[1,1,1,1] = 4,\ A_4[1,1,2,1] = 4,\ A_4[1,1,3,1] = 2,\ A_4[1,2,1,1] = 3,\ A_4[2,1,1,1] = 3,\ A_4[2,1,2,1] = 2,\ A_4[2,1,3,1] = 1,\ A_4[2,1,3,2] = 1)$ are not the Case 1 child of A_4. They are the three child partitions of A_4.

Based on the case analysis above, we now give an algorithm to generate all partitions. Note that the array in the argument is passed by its pointer, so we use the array during the execution of the algorithm.

```
   Procedure find-all-children ($A_d = (A_d[1,\dots,1],\dots, A_d[l_1,\dots,l_d])$)
   {$A_d$ is the current partition.}
   begin
01   Output $A_d$
     {Output the difference from the preceding partition.}
02   if $A_d[1,\dots,1]$ is larger than every lower neighbour of
       $A_d[1,\dots,1]$ then
03   begin {$m_1,\dots,m_d$ are uniquely defined from $l_1,\dots,l_d$}
04     for each $i = 1,\dots,d$
05       if every upper neighbour of $A_d[m_1,\dots,m_d]$ is
```

```
            nonzero then
06              find-all-children($A_d[+i]$)
                {Cases 2(a) and 2(b)}
07          if $A_d[m_1, \ldots, m_d]$ is smaller than every upper neigh-
            bour of $A_d[m_1, \ldots, m_d]$ and ($A_d[m_1, \ldots, m_d]$ is not
            a lower neighbour of $A_d[1, \ldots, 1]$ or $A_d[m_1, \ldots, m_d] -$
            $A_d[1, \ldots, 1] > 1$) then
08              find-all-children($A_d[\pm]$)      {Case 2(b)}
09      end
    end
```

```
    Algorithm find-all-partitions($N$, $d$)
    begin
01      Output the root partition $R_d = (R_d[1, \ldots, 1] = N)$
02      for each $i = 1, \ldots, d$ find-all-children($R_d[+i]$)
    end
```

We have the following theorem.

[Theorem 3.1] The algorithm uses $O(N^d)$ space and runs in $O(d|S(N,d)|)$ time.

Proof. Since we traverse the family tree $T_{N,d}$ and output each partition at each corresponding vertex of $T_{N,d}$, we can generate all partitions in $S(N,d)$.

For the part $A_d[1, \ldots, 1]$ of A_d, we maintain the maximum lower neighbour of $A_d[1, \ldots, 1]$. With a help of the part we can check the condition in Line 02 in constant time.

In naive way, we take $O(d)$ time to check the condition in Line 05 for each $i = 1, \ldots, d$. Thus we take $O(d^2)$ time in total in Lines 04 and 05. However, with the following data structure, the condition in Line 05 can be check in constant time as follows. For each part $A_d[i_1, \ldots, i_d]$ of A_d, we maintain the number of nonzero upper neighbours, denoted by $u(A_d[i_1, \ldots, i_d])$. Note that $u(A_d[i_1, \ldots, i_d]) = d$ if and only if all upper neighbours of $A_d[i_1, \ldots, i_d]$ are nonzero. This number can be update in $O(d)$ time for each generation of a child partition. With a help of the above number, we can check the condition in Line 05 in constant time. Thus all Case 1 children for each partition can be generated in $O(d)$ time.

The running time of the other parts is bounded by $O(d)$ time except recursive calls. Thus all child partitions of A_d can be generated in $O(d)$ time. Note that the algorithm takes $O(d)$ time even if A_d has only one child partitions. Therefore the algorithm runs in $O(d|S(N,d)|)$ time. Note that the algorithm does not output each partition entirely, but the difference from the preceding partition. $\mathcal{Q.E.D.}$

Thus, the algorithm generates each partition in $O(d)$ time "on average." Note that our algorithm outputs only the difference between a partition and its preceding one. In the next section we improve the algorithm to generate each partition in $O(d)$ time "in worst case."

4. Modification

The algorithm in Section 3. generates all d-dimensional partitions in $S(N,d)$ in $O(d|S(N,d)|)$ time. Thus the algorithm generates each partition in $O(d)$ time "on average." However, after generating a partition corresponding to the last vertex in a large subtree of $T_{N,d}$, we have to merely return from the deep recursive call without outputting any partition. This takes much time. Therefore, each partition cannot be generated in $O(d)$ time in worst case.

However, a simple modification [24] improves the algorithm to generate each partition in $O(d)$ time. The algorithm is as follows.

```
    Procedure find-all-children2($A_d$, depth)
    { $A_d$ is the current partition, and depth is the depth of
    the recursive call.}
    begin
01      if depth is even then
02          Output $A_d$ {before outputting its child partitions.}
03      Generate child partitions by the method in Section 3,
        and recursively call find-all-children2 for each child
        partition.
04      if depth is odd then
05          Output $A_d$ {after outputting its child partitions.}
    end
```

Note that the above algorithm outputs only the difference between a partition and its predecessor.

One can observe that the algorithm generates all partitions so that each partition can be obtained from the preceding one by tracing at most three edges of $T_{N,d}$. Note that if A_d corresponds to a vertex v in $T_{N,d}$ with odd depth, then we may need to trace three edges to generate the next partition. Otherwise we need to trace at most two edges to generate the next partition.

[Theorem 4.1] The algorithm generates each d-dimensional partition in $O(d)$ time in worst case.

5. Exactly d-dimensional Partitions

For a given integer N, the following three partitions: $(A_d[1, \ldots, 1] = N)$, $(A_d[1, \ldots, 1] = N-1, A_d[2, 1, \ldots, 1] = 1)$ and $(A_d[1, \ldots, 1] = N-2, A_d[2, 1, \ldots, 1] = 1, A_d[3, 1, \ldots, 1] = 1)$ can be regarded as one-dimensional partitions, thus integer partitions. Similarly, the following two partitions $(A_d[1, \ldots, 1] = N-2, A_d[1, 1, \ldots, 1, 2] = 1, A_d[1, 1, \ldots, 2, 1] = 1)$ and $(A_d[1, \ldots, 1] = N-2, A_d[1, 2, 1, \ldots, 1] = 1, A_d[2, 1, \ldots, 1] = 1\}$ can be regarded as two-dimensional partitions. In other words, we can identify $(A_d[1,\ldots,1] = N-2, A_d[1,2,1,\ldots,1] = 1, A_d[2,1,\ldots,1] = 1)$ as $(A_2[1,1] = N-2, A_2[1,2] = 1, A_2[2,1] = 1,\}$, thus plane partition. We may consider that the algorithm in Section 3. generates "at most" d-dimensional all partitions for given N.

This section deals with "exactly" d-dimensional partitions. A d-dimensional partition A_d is *a exactly d-dimensional partition* if $A_d[1,\ldots,1,2] > 0$, $A_d[1,\ldots,1,2,1] > 0,\ldots,$ $A_d[2,1,\ldots,1] > 0$ hold. For instance, for $d = 3$ and $N = 7$, $(A_3[1,1,1] = 3, A_3[1,1,2] = 2, A_3[1,2,1] = 1, A_3[2,1,1] = 1)$ is a exactly 3-dimensional partition. However, $(A_3[1,1,1] = 5, A_3[2,1,1] = 1, A_3[3,1,1] = 1)$ and $(A_3[1,1,1] = 4, A_3[1,1,2] = 1, A_3[1,2,1] = 2)$ are not exactly 3-dimensional partitions.

This section gives an algorithm to generates exactly d-dimensional partitions. Let $S(N,=d)$ be the set of all exactly d-dimensional partitions of a positive integer N.

If $N = d+1$, then $S(N,=d)$ contains exactly one partition. In this case generating all partitions in $S(N,=d)$ is trivial. Otherwise, if $N < d+1$, there is no partition. Hence we assume $N > d+1$ in this section.

We set $R_d = (R_d[1,\ldots,1] = N-d, R_d[1,\ldots,1,2] = 1, R_d[1,\ldots,1,2,1] = 1,\ldots, R_d[1,2,1,\ldots,1] = 1, R_d[2,1,\ldots,1] = 1)$ as *the root partition* of $S(N,=d)$.

Then we define *the parent partition* $P(A_d)$ of each partition A_d in $S(N,=d)$ except for the root partition as follows. Let $A_d = (A_d[1,\ldots,1], \ldots, A_d[l_1,\ldots,l_d])$ be a partition in $S(N,=d)$, and assume that A_d is not the root partition. Let $A_d[k_1,\ldots,k_d]$ be the last part of A_d in lexicographical order and except the parts included in R_d. We can observe that A_d always has a part except the parts included in R_d if $|A_d| > d+1$ holds, where $|A_d|$ is the number of parts of A_d.

We have the following three cases.

Case 1: $|A_d| > d+1$ and $A_d[k_1,\ldots,k_d] = 1$.

We define $P(A_d)$ as a partition derived from A_d by removing $A_d[k_1,\ldots,k_d]$ and adding one to $A_d[1,\ldots,1]$. Note that the number of parts of $P(A_d)$ is one less than that of A_d.

Case 2: $|A_d| > d+1$ and $A_d[k_1,\ldots,k_d] > 1$.

We define $P(A)$ as a partition derived from A_d by subtracting one from $A_d[k_1\ldots,k_d]$ and adding one to $A_d[1,\ldots,1]$. Note that the number of parts of $P(A_d)$ is equal to that of A_d.

Case 3: $|A_d| = d+1$.

We denote $A_d[j_1,\ldots,j_d]$ as the part such that $A_d[j_1,\ldots,j_d] > 1$ holds and the last part in lexicographic order of A_d. Then we define $P(A_d)$ as a partition derived from A_d by subtracting one from $A_d[j_1,\ldots,j_d]$ and adding one to $A_d[1,\ldots,1]$. Note that the number of parts of $P(A_d)$ is equal to that of A_d.

Note that $P(A_d)$ is also in $S(N,=d)$.

Similar to Sections 2 and 3, by the definition of the parent partition, we can define the family tree of $S(N,=d)$, and also generate exactly d-dimensional partitions in $O(d)$ time for each.

[Corollary 5.1] One can generate each exactly d-dimensional partition in $O(d)$ time in worst case.

6. Conclusion

In this paper we have given simple algorithm to generate all d-dimensional partitions. We first define a family tree in which each vertex corresponds to each partition with the given property, then output each partition in $O(d)$ time by traversing the family tree. Also, we have given an algorithm to generate all exactly d-dimensional partitions in $O(d)$ time for each. Our algorithms can be simply implemented using a d-dimensional array.

References

[1] G. S. Akl and I. Stojmenović, *Parallel algorithms for generating integer partitions and compositions*, J. Combinatorial Math. and Combinatorial Computing, 13, 107–120, 1993.

[2] G. Andrews, *The theory of partitions*, Encyclopedia of Mathematics and its Applications, Vol. 2. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1976.

[3] G. E. Andrews and K. Eriksson, *Integer Partitions*, Cambridge University Press, 2004.

[4] D. Avis and K. Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 6, 21–46, 1996.

[5] E. A. Bender and D. E. Knuth, *Enumeration of plane partitions*, J. of Combinatoroal Theory (A), 13, 40–54, 1972.

[6] T. Beyer and S. M. Hedetniemi, *Constant time generation of rooted trees*, SIAM J. Comput., 9, 706–712, 1980.

[7] D. P. Bhatia, M. A. Prasad, and D. Arora, *Asymptotic results for the number of multidimensional partitions of an integer and directed compact lattice animals*, J. Phys. A: Math. Gen., 30, 2281–2285, 1997.

[8] P. Bratley and J. K. S. McKay, *Multi-dimensional partition generator*, Algorithm 313 Commun. ACM, 10, 666, 1967.

[9] T. I. Fenner and G. Loizou, *A binary tree representation and related algorithms for generating integer partitions*, The Computer Journal, 23, 332–337, 1980.

[10] T. I. Fenner and G. Loizou, *An analysis of two related loop-free algorithms for generating integer partitions*, Acta Inform., 16, 237–252, 1981.

[11] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*, Cambridge University Press, New York, 1993.

[12] S. Kawano and S. Nakano, *Constant time generation of set partitions*, IEICE Trans. Fundamentals, E88-A, 930–934, 2005.

[13] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms*, CRC Press, Boca Raton, 1998.

[14] D. E. Knuth, *The art of computer programmings*, 4, Fascicle 3, Addison-Wesley, 2005.

[15] Z. Li and S. Nakano, *Efficient generation of plane triangulations without repetitions*, Proc. ICALP2001, LNCS 2076, 433–443, 2001.

[16] G. Li and F. Ruskey, *The advantage of forward thinking in generating rooted and free trees*, Proc. 10th Annual ACM-

SIAM Symp. on Discrete Algorithms, 939–940, 1999.

[17] P. A. MacMahon, *Combinatory Analysis*, 2, Cambridge Univ. Press, New York, 1960.

[18] B. D. McKay, *Isomorph-free exhaustive generation*, J. Algorithms, 26, 306–324, 1998.

[19] J. K. S. McKay, *Partition generator*, Algorithm 263, Commun. ACM, 8, 493, 1965.

[20] J. K. S. McKay, *Partitions in natural order*, Algorithm 371, Commun. ACM, 13, 52, 1970.

[21] S. Nakano, *Enumerating floorplans with n rooms*, Proc. ISAAC2001, LNCS 2223, 104–115, 2001.

[22] S. Nakano, *Efficient generation of plane trees*, Inf. Process. Lett., 84, 167–172, 2002.

[23] S. Nakano, *Efficient generation of triconnected plane triangurations*, Comput. Geometry Theory and Applications, 27, 109–122, 2004.

[24] S. Nakano and T. Uno, *Constant time generation of trees with specified diameter*, Proc. WG2004, LNCS 3353, 33–45, 2004.

[25] T. V. Narayana, R. M. Mathsen, and J. Sarangi, *An algorithm for generating partitions and its applications*, J. Combinatorial Theory (A), 11, 54–61, 1971.

[26] A. Nijenhuis and H. S. Wilf, *Combinatorial algorithms*, Academic Press, NY, 1975.

[27] E. S. Page and L. B. Wilson, *An introduction to computational combinatorics*, Cambridge University Press, 1979.

[28] D. Rasmussen, C. D. Savage, and D. B. West, *Gray code enumeration of families of integer partition*, J. Combinatorial Theory (A), 70(2), 201–229, 1995.

[29] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial algorithms: theory and practice*, Prentice Hall, Englewood Cliffs, New Jersey, 1977.

[30] R. C. Read, *How to avoid isomorphism search when cataloguing combinatorial configurations*, Annals of Discrete Math., 2, 107–120, 1978.

[31] C. Savage, *Gray code sequences of partitions*, J. Algorithms, 10, 577–595, 1989.

[32] H. S. Wilf, *Combinatorial algorithms : An update*, SIAM, 1989.

[33] R. A. Wright, B. Richmond, A. Odlyzko, and B. D. McKay, *Constant time generation of free trees*, SIAM J. Comput., 15, 540–548, 1986.

[34] K. Yamanaka, S. Kawano, Y. Kikuchi, and S. Nakano, *Constant time generation of integer partitions*, IEICE Trans. Fundamentals, E90-A, 888–895, 2007.

[35] M. Zito, I. Pu, M. Amos, and A. Gibbons, *RNC algorithms for the uniform generation of combinatorial structures*, Proc. the 7th Annual ACM-SIAM Symposium on Discrete Algorithm, 429–437, 1996.

[36] A. Zoghbi and I. Stojmenović, *Fast algorithms for generating integer partitions*, International J. Computer Math., 70, 319–332, 1998.