AUSTRALASIAN JOURNAL OF COMBINATORICS
Volume **65(2)** (2016), Pages 152–169

A set partition number system

Jon T. Butler

Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121
U.S.A.
jon_butler@msn.com

Tsutomu Sasao[*]

Department of Computer Science
Meiji University
1-1-1 Higashi-Mita Tama-ku Kawasaki-shi, Kanagawa 214-8571
JAPAN
sasao@cs.meiji.ac.jp

Abstract

We propose a number system that is based on set partitions. This represents the third number system that is based on combinatorial objects, the other two being combinations and permutations. A number system based on set partitions is useful in the *hardware* enumeration of set partitions, which is significantly faster than *software* enumeration. Specifically, the restricted growth string $(b_0 b_1 \ldots b_{n-1})$ of a set partition π_I allows a unique index I to be associated with π_I, where I is a nonnegative integer in the number system that is represented as $I = b_0\omega_0 + \ldots + b_{n-2}\omega_{n-2} + b_{n-1}\omega_{n-1}$. Here, ω_i is specified by the set partition tree, a data structure derived from the generating tree of set partitions. We show another data structure, the set partition mesh, that is equivalent to the set partition tree. It also stores all set partitions but is much more compact. Indeed, it makes possible the design of hardware set partition generators for n-sets as large as $n = 32$, compared to the set partition tree, which limits the sets to size no greater than $n = 10$.

[*] Supported by a Grant-in-Aid for Scientific Research of the Japan Society for the Promotion of Science.

1 Introduction

A number system based on *combinations* (sometimes denoted as a combinadic [12]) is useful in the design of circuits that encode and decode data transferred on and off chip [1]. When all data is transferred using codewords with a fixed number of 1's and 0's (i.e., combinations), there is little variation in power consumption. Such circuits resist side-channel attacks that use changes in power consumption to determine the circuit's design. Number systems based on *permutations* are useful in the design of circuits that produce a permutation given an index of that permutation [2]. Such circuits are useful for data compression and for mapping shared memory to multiprocessors. Traditionally, the enumeration of combinations and permutations has been restricted to software[1] (e.g., [8, 16]). However, the advent of large logic resources in FPGAs (field-programmable gate arrays), SoCs (systems on chips), and reconfigurable computers has made it feasible to implement complex algorithms directly in hardware. This has made it possible to generate combinatorial objects by hardware at speeds unattainable by software (e.g., 60,000 times speedup in the generation of bent Boolean functions [18]).

Definition 1.1 In a **combinatorial number system** [8], a non-negative integer $I < \binom{n}{r}$ is represented as $c_r c_{r-1} \ldots c_1$, where

$$I = \binom{c_r}{r} + \binom{c_{r-1}}{r-1} + \ldots + \binom{c_1}{1}, \tag{1.1}$$

and $c_r > c_{r-1} > \ldots > c_1 \geq 0$.

Knuth [8] discusses this in detail and credits Pascal [14] for introducing the concept more than 125 years ago. A permutation number system (sometimes known as a factorial number system or a factoradic) uses factorials.

Definition 1.2 In a **factorial number system** [7], a non-negative integer $I < n!$ is represented as $s_{n-1}s_{n-2} \ldots s_2 s_1$, where

$$I = s_{n-1}(n-1)! + s_{n-2}(n-2)! + \ldots + s_1 1! + s_0 0!, \tag{1.2}$$

and $0 \leq s_i \leq i \leq n-1$.

It is natural to ask if there is a number system based on set partitions. In an interesting paper, Hankin and West [4] use partitions to solve optimization problems

[1]We use the term "software" to describe algorithms, code, etc. that is typically run on systems with a fixed architecture (e.g. a CPU) and whose computation is specified by a language, like C, Mathematica, MATLAB, FORTRAN, etc.. We use the term "hardware" to describe systems whose architecture is adapted to the computation (e.g. FPGAs and reconfigurable computers).

in scheduling, bioinformatics, and forensic science. With respect to scheduling, set partitions can be used to specify the ways tasks are allocated to processors. Each task requires some time to complete on one of n processors. The scheduling problem is the process of assigning tasks to processors so that all tasks are completed in the minimum time. In this case, one seeks a partition of the set of tasks into n blocks that corresponds to the shortest computation time. With respect to bioinformatics, research in computational molecular biology has shown the importance of partitions in understanding the role of genes in determining global characteristics of species. This problem requires the enumeration of partitions at a high rate of speed, since so many partitions must be considered. With respect to forensic science, the goal is to partition crimes according to perpetrators, so as to identify which perpetrators are the most likely to have committed which crimes.

While there are many papers on programs and algorithms for enumerating partitions [6, 8, 10, 13, 16, 17], we know of only two devoted to the hardware enumeration of set partitions, [3] and [9]. The latter considers two-part partitions only. The former enumerates all set partitions and is the *raison d'être* for this paper. Specifically, [3] shows how the results presented in this paper are used to design the hardware that enumerates set partitions.

2 Definitions

Definition 2.1 Let $S = \{0, 1, \ldots, n-1\}$ be an n-set. $\{S_0, S_1, \ldots, S_{m-1}\}$ is a **partition** of S if and only if 1) $S_i \subseteq S$, 2) $S_i \bigcap S_j = \emptyset$ if $i \neq j$, and 3) $\bigcup_{i=0}^{m-1} S_i = S$.

We can order the partitions. Table 1 shows a subset of the 52 partitions of 5-sets. Here, the order of the partitions is specified by the increasing lexicographical ordering of the restricted growth string [5, 19].

Definition 2.2 A **restricted growth string** of a partition on n-sets is a sequence $(b_0 b_1 \ldots b_{n-1})$ of integers, such that $b_0 = 0$ and for $i > 0$,

$$0 \leq b_i \leq \max\{b_0, b_1, \ldots, b_{i-1}\} + 1. \tag{2.1}$$

The descriptor "restricted" suggests that growth of elements in the restricted growth string is limited (to only 1 more than any previously specified element). Since a set partition is unchanged by a reordering of blocks, denote block 0 as the block in which $n-1$ is located. Then, $n-2$ is either in block 0, or a different block, block 1. Continue in this way until all elements are assigned a block. For example, the partitions $\{\{4, 3, 2, 1, 0\}\}$, $\{\{4, 2\}, \{3, 1\}, \{0\}\}$, and $\{\{4\}, \{3\}, \{2\}, \{1\}, \{0\}\}$ correspond to the restricted growth strings (*0 0 0 0 0*), (*0 1 0 1 2*), and (*0 1 2 3 4*), respectively. Table 1 shows the corresponding number representations of the numbers in the set partition number system. It is clear that there is a bijection between the set partitions of an n-set and the set of n-element restricted growth strings [19].

Table 1: The Index I, the I-th Set Partition π_I, the Restricted Growth String, and the Corresponding Number System Representation, for $n = 5$.

Index I	Partition π_I	Restricted Growth String	Number System Representation
0	$\{\{4,3,2,1,0\}\}$	*(0 0 0 0 0)*	$0+0\cdot 15+0\cdot 5+0\cdot 2+0\cdot 1=$ 0
1	$\{\{4,3,2,1\},\{0\}\}$	*(0 0 0 0 1)*	$0+0\cdot 15+0\cdot 5+0\cdot 2+1\cdot 1=$ 1
2	$\{\{4,3,2,0\},\{1\}\}$	*(0 0 0 1 0)*	$0+0\cdot 15+0\cdot 5+1\cdot 2+0\cdot 1=$ 2
3	$\{\{4,3,2\},\{1,0\}\}$	*(0 0 0 1 1)*	$0+0\cdot 15+0\cdot 5+1\cdot 2+1\cdot 1=$ 3
4	$\{\{4,3,2\},\{1\},\{0\}\}$	*(0 0 0 1 2)*	$0+0\cdot 15+0\cdot 5+1\cdot 2+2\cdot 1=$ 4
5	$\{\{4,3,1,0\},\{2\}\}$	*(0 0 1 0 0)*	$0+0\cdot 15+1\cdot 5+0\cdot 3+0\cdot 1=$ 5
6	$\{\{4,3,1\},\{2,0\}\}$	*(0 0 1 0 1)*	$0+0\cdot 15+1\cdot 5+0\cdot 3+1\cdot 1=$ 6
7	$\{\{4,3,1\},\{2\},\{0\}\}$	*(0 0 1 0 2)*	$0+0\cdot 15+1\cdot 5+0\cdot 3+2\cdot 1=$ 7
8	$\{\{4,3,0\},\{2,1\}\}$	*(0 0 1 1 0)*	$0+0\cdot 15+1\cdot 5+1\cdot 3+0\cdot 1=$ 8
9	$\{\{4,3\},\{2,1,0\}\}$	*(0 0 1 1 1)*	$0+0\cdot 15+1\cdot 5+1\cdot 3+1\cdot 1=$ 9
10	$\{\{4,3\},\{2,1\},\{0\}\}$	*(0 0 1 1 2)*	$0+0\cdot 15+1\cdot 5+1\cdot 3+2\cdot 1=$ 10
11	$\{\{4,3,0\},\{2\},\{1\}\}$	*(0 0 1 2 0)*	$0+0\cdot 15+1\cdot 5+2\cdot 3+0\cdot 1=$ 11
12	$\{\{4,3\},\{2,0\},\{1\}\}$	*(0 0 1 2 1)*	$0+0\cdot 15+1\cdot 5+2\cdot 3+1\cdot 1=$ 12
13	$\{\{4,3\},\{2\},\{1,0\}\}$	*(0 0 1 2 2)*	$0+0\cdot 15+1\cdot 5+2\cdot 3+2\cdot 1=$ 13
14	$\{\{4,3\},\{2\},\{1\},\{0\}\}$	*(0 0 1 2 3)*	$0+0\cdot 15+1\cdot 5+2\cdot 3+3\cdot 1=$ 14
15	$\{\{4,2,1,0\},\{3\}\}$	*(0 1 0 0 0)*	$0+1\cdot 15+0\cdot 10+0\cdot 3+0\cdot 1=$ 15
16	$\{\{4,2,1\},\{3,0\}\}$	*(0 1 0 0 1)*	$0+1\cdot 15+0\cdot 10+0\cdot 3+1\cdot 1=$ 16
17	$\{\{4,2,1\},\{3\},\{0\}\}$	*(0 1 0 0 2)*	$0+1\cdot 15+0\cdot 10+0\cdot 3+2\cdot 1=$ 17
18	$\{\{4,2,1\},\{3\},\{0\}\}$	*(0 1 0 1 0)*	$0+1\cdot 15+0\cdot 10+1\cdot 3+0\cdot 1=$ 18
19	$\{\{4,2,1\},\{3\},\{0\}\}$	*(0 1 0 1 1)*	$0+1\cdot 15+0\cdot 10+1\cdot 3+1\cdot 1=$ 19
20	$\{\{4,2,1\},\{3\},\{0\}\}$	*(0 1 0 1 2)*	$0+1\cdot 15+0\cdot 10+1\cdot 3+2\cdot 1=$ 20
$\cdots$	$\cdots$	$\cdots$	$\cdots$
49	$\{\{4\},\{3\},\{2,0\},\{1\}\}$	*(0 1 2 3 2)*	$0+1\cdot 15+2\cdot 10+3\cdot 4+2\cdot 1=$ 49
50	$\{\{4\},\{3\},\{2\},\{1,0\}\}$	*(0 1 2 3 3)*	$0+1\cdot 15+2\cdot 10+3\cdot 4+3\cdot 1=$ 50
51	$\{\{4\},\{3\},\{2\},\{1\},\{0\}\}$	*(0 1 2 3 4)*	$0+1\cdot 15+2\cdot 10+3\cdot 4+4\cdot 1=$ 51

3 Set Partition Tree

We show a logic circuit [3] that converts the index of a set partition into its restricted growth string using the set partition tree. The set partition tree is based on a popular enumerative approach, the generating tree. This allows the enumeration of set partitions at a rate that is much faster than software. That is, the indices can be produced by a fast counter circuit, while the index-to-set-partition circuit converts the index to a partition represented by its restricted growth string. When n is large, even this high speed circuit cannot exhaustively enumerate all set partitions in a reasonable time. For example, when n is 32, there are about 1.39×10^{26} set partitions, which would take roughly 4.4×10^{8} years to exhaustively enumerate at one per clock using a 100 MHz clock (a typical FPGA clock frequency). In this case, Monte Carlo simulations replace exhaustive enumeration. The hardware to produce Monte Carlo simulations is easily obtained by replacing the counter circuit with a hardware random number generator. In this application, hardware enumeration is still considerably faster than software enumeration, about 10 times faster [3].

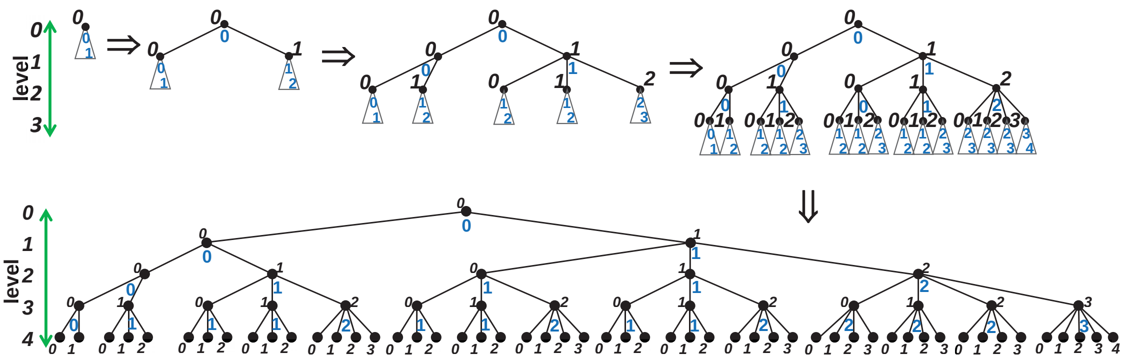


Figure 1: Applying the Iterative Algorithm for the Set Partition Tree on 5-Sets

3.1 Introduction to the Set Partition Tree

Fig. 1 shows the process of setting up a set partition tree. It starts with a single root node labeled by 0, representing the first element of all restricted growth strings. This is shown on the upper-left side of Fig. 1. The second element is represented as two successor nodes to the root node labeled 0 and 1 using the substitution shown on the left side of Fig. 2. This process continues, as the substitution rules shown in Fig. 2 are used to fill out the set partition tree shown in Fig. 1.

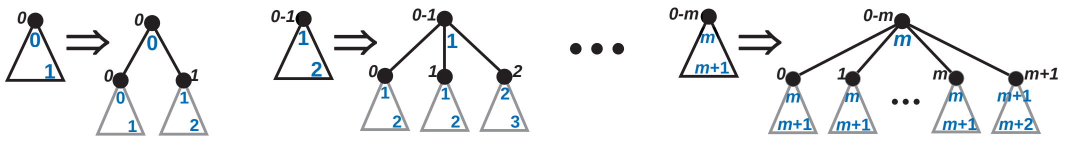


Figure 2: Substitution Rules for the Set Partition Tree

3.2 Formal Specification of the Set Partition Tree

We use the enumeration approach of generating trees of Example 2.79 in [11].

Definition 3.1 A **generating tree** is a labeled plane tree such that, if v_1 and v_2 are any two nodes with the same label L, then v_1 and v_2 have exactly the same number of children, L. To specify a generating tree, it is sufficient to specify:

1. **Root:** the label of the root, and
2. **Rule:** the labels of the children nodes as a function of the label of the parent.

Regarding generating trees for *set partitions*, from Example 2.79 in [11],

1. **Root:** [2]

2. **Rule:** $L \rightarrow [L^{L-1} \quad L{+}1]$,

where the children of L are 1) $L-1$ copies of L and 2) one copy of $L+1$. Iterating four times yields the generating tree shown in Table 2.

Table 2: The Generating Tree for Set Partitions

Level	Node Labels in the Generating Tree for Set Partitions
0	[2]
1	[2 3]
2	[2 3 3 3 4]
3	[2 3 3 3 4 3 3 4 3 3 4 4 4 4 5]
4	[23 334 334 334 4445 334 334 4445 334 334 4445 4445 4445 4445 55556]

The tree formed in this way has node labels that specify the number of children nodes. So, for example, the root node, labeled 2 at level 0, has two children, labeled 2 and 3 (i.e., $L \rightarrow [L^{L-1} \quad L{+}1]_{L=2} \;=\; [2 \;\; 3]$). Similarly, the node labeled 2 at level 1 has two children, while the node labeled 3 at level 1 has three children. Specifically, $[2 \;\; 3] \rightarrow [2 \;\; 3 \quad\; 3 \;\; 3 \;\; 4]$). These five nodes at level 2, in turn, have two, three, three, three and four children at level 3, as shown in Table 2.

To form the **set partition tree**, replace each node label with the node's *rank*, which is its position among the sibling nodes, with rank 0 being the leftmost sibling node, rank 1 being the next sibling node to the right, etc.. The rightmost sibling has rank $c-1$, where c is the number of children nodes of the parent. For the root node, the rank is 0.

Assign the weight ω_{ln} of each leaf node as an integer in order from left to right, as $0, 1, 2, \ldots, B_n - 1$ (green labels in Fig. 3 below). This creates the index I in ascending lexicographical order on the restricted growth strings.

Form the weights ω_e of the edges, as follows. For each edge e, derive two partial weights, $\omega_{\text{ln\_predecessor}}$ and $\omega_{\text{ln\_successor}}$. Set $\omega_{\text{ln\_predecessor}}$ to the weight of the leaf node η_1 that is obtained by traversing from the predecessor (upper) node of e to η_1 along edges with weight 0 (along the extreme left edge). Similarly, set $\omega_{\text{ln\_successor}}$ to the weight of the leaf node η_2 that is obtained by traversing from the successor (lower) node of e to η_2 along edges with weight 0 (along the extreme left edge). Set the weight ω_e of the current edge to $\omega_{\text{ln\_predecessor}} - \omega_{\text{ln\_successor}}$.

In the process of traversing any edge within the set partition tree, one progresses left-to-right across the leaf nodes. For example, consider the two edges incident to the leaf node labeled by 0 and 15. The edge labeled 15 signifies that the minimum weight of a leaf node that can be reached from the successor node η_a to the weight 15 edge is 15. This minimum is achieved only if all remaining edges traversed from η_a have weight 0. Similarly, the edge labeled 0 incident to the root node signifies that the minimum weight of a leaf node that can be reached by the successor node η_b to the weight 0 edge is 0. Again, this minimum is achieved only if all remaining edges from η_b traversed have weight 0.

The set partition tree is constructed in such a way that a complete traversal from the root node to a leaf node yields a total weight identical to the weight originally assigned to the leaf node above.

Example 3.2 Fig. 3 shows the set partition tree for $n = 5$. Here, the rank of each node is shown adjacent to the node, and the weight of each leaf node is shown just below the leaf node. The weight of each edge is shown adjacent to each edge. The level of the nodes is shown along the vertical line to the left of the set partition tree.

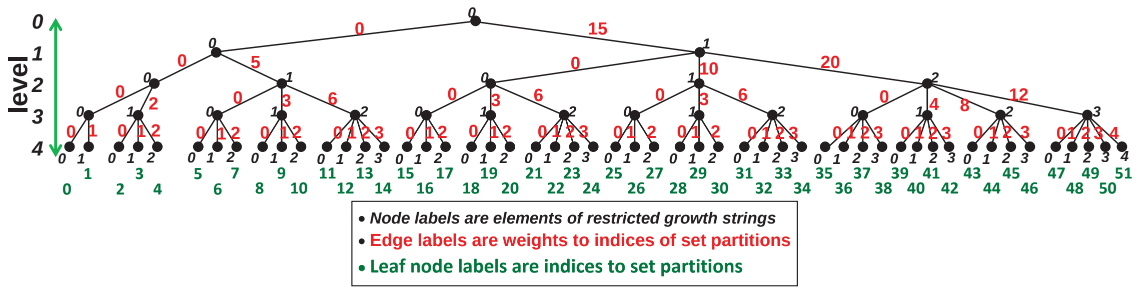

level
0
1
2
3
4
• Node labels are elements of restricted growth strings
• Edge labels are weights to indices of set partitions
• Leaf node labels are indices to set partitions

Figure 3: The Set Partition Tree for 5-Part Set Partitions

Traversing the leftmost path from the root node to the terminal node 0 yields the edge labels ($0\,0\,0\,0\,0$), which is the restricted growth string of partition $\{\{4, 3, 2, 1, 0\}\}$. The rightmost path from the root node to the terminal node 51 yields the edge labels ($0\ 1\ 2\ 3\ 4$), which is the restricted growth string of partition $\{\{4\}, \{3\}, \{2\}, \{1\}, \{0\}\}$.

The index associated with each terminal node can be obtained by summing the weights of edges along the path from the root node to a terminal node. In a traverse of the set partition tree, we have an index, and we use it to make decisions as to the path from the root node to a terminal node.

3.3 Traversal of a Set Partition Tree

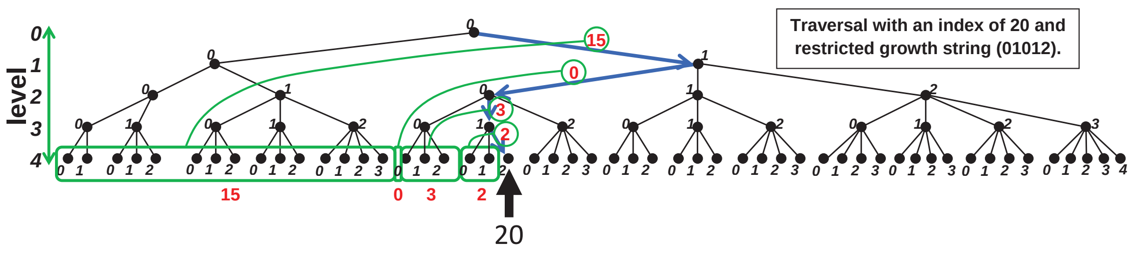
Algorithm 3.3 Traversal of the Set Partition Tree: Given an index I, such that $0 \leq I < B_n$, find the corresponding restricted growth string $(b_0 b_1 \ldots b_{n-1})$.

Set the level $\ell = 0$. Set the current node η_{cur} to the root node. Set b_0 to the label of the η_{cur}; i.e., set $b_0 = 0$. Iterate the following until $\ell = n$.

1. At the current node, η_{cur}, identify the outgoing edge e with the largest weight ω_e, such that $\omega_e \leq I$. Let η_{new} be the successor node to e.
2. Set $b_{\ell+1}$ to the label of η_{new}.
3. Set I to $I - \omega_e$.
4. Set η_{cur} to η_{new}.

5. Set ℓ to $\ell + 1$.

Example 3.4 Fig. 4 shows the traverse associated with index 20. Applying the traverse in Algorithm 3.3 yields the sequence of edge weights ω_e of 15, 0, 3, and 2 (which sum to 20), and the restricted growth string of (01012).


Traversal with an index of 20 and restricted growth string (01012).
level
15
0
3
2
20
Figure 4: Example of a Traversal of the Set Partition Tree.

4 Set Partition Mesh

4.1 Introduction to the Set Partition Mesh

Even for moderate n, the number of nodes in the set partition tree is large. For example, consider just the leaf nodes, which are counted by the Bell Numbers B_n. From [8], this number is $B_n = \Theta(n/\log n)^n$. The total number of nodes affects a MATLAB program that provides threshold values for circuits in the index-to-set-partition converter. To compute these thresholds, the data structure must be small enough so that it can be explored in a reasonable time. The very large number of nodes in the set partition tree strictly limits the size of the n-sets that can be explored. Indeed, it is not possible to design the index-to-set-partition converter circuit [3] using the set partition tree for $n > 10$ because the tree is too large. For $n = 11$, the design algorithm will take approximately 9.5 hours (based on the computation times for lower values of n on an Intel Core TM2 Duo P8400 processor running at 2.30 GHz using MATLAB). For $n = 12$, it is estimated to take 15.0 days. Paradoxically, there is enough logic to design an index-to-set-partition circuit for n up to 32. To achieve such a value, we must use a more compact representation.

4.2 Formal Specification of the Set Partition Mesh

An examination of the set partition tree in Fig. 3 reveals repetitive structures. For example, at level 2, there are three nodes that are root nodes of subtrees that are identical with respect to structure (black lines and circles), edge weights (red), and node weights (black). They differ only in the index labels (green).

Definition 4.1 The **set partition mesh** for n-sets is formed from the set partition tree as follows. Remove the weights from all leaf nodes. For each node η in the set partition tree, trace the sequence of nodes to the root node and assign the maximum label among those nodes to η (include η's label). Call this maximum label value M_η. Coalesce all nodes at level ℓ with the same M_η into one node labeled M_η. Assign to the coalesced edges a weight that is the smallest non-zero weight among all descendant edges of the coalesced node labeled M_η. This descendent (coalesced) edge is on the left. Remove the weight from the edge that is on the right.

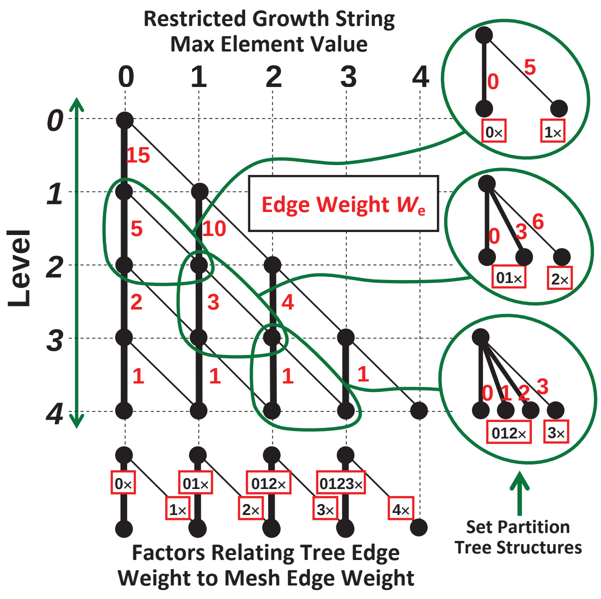


Figure 5: The Set Partition Mesh for 5-Sets.

Example 4.2 Fig. 5 shows the set partition mesh for $n = 5$. Here, the vertical edges formed by coalescing edges in the set partition tree of Fig. 3 are shown as thick lines (including vertical edges along the left hand side that result from the coalescing of one edge). The labeling along the top shows the maximum value of restricted growth string elements, while the labeling along the bottom shows factors f_e used to establish correspondence between the edge weights in the set partition mesh to edge weights in the set partition tree. So, the thick vertical lines correspond to edges in the set partition tree with labels $0\times$, $01\times$, $012\times$, and $0123\times$ depending on whether the maximum value in the restricted growth string is 0, 1, 2, or 3, respectively. The $\times$ notation represents the factors (f_e) associated with the individual edges of the set partition tree. For example, $0123\times$ means that the corresponding thick line in the set partition mesh represents four edges in the set partition tree, where one tree edge weight has 0 times the mesh edge weight, one tree edge weight has 1 times the mesh edge weight, one tree edge weight has 2 times the mesh edge weight, and one tree edge weight has 3 times the mesh edge weight. The inserts shown on the right of Fig. 5 illustrate how three parts of the set partition mesh (rounded triangles on the left) correspond to parts of the set partition tree (ellipses on the right). The thin (oblique) lines correspond to the first (leftmost) occurrence of a maximum restricted growth string element value. Such edges increase the maximum value by 1, as indicated by the fact that they go from a smaller maximum element value on the left to a larger maximum element value on the right. For example, $2\times$ next to an oblique line in the set partition mesh represents the first occurrence of a 2 as a restricted growth string causing the maximum element value to go from a 1 on the left to a 2 on the right. It is also an edge weight and represents an edge in the set partition tree whose edge weight is 2 times the edge weight shown in the set partition mesh.

Recall the three nodes at level 2 in Fig. 3 showing the set partition tree for $n = 5$ that are root nodes of subtrees that are identical with respect to structure (black lines and circles), edge weights (red), and node weights (black). These occur in the set partition mesh of Fig. 5 as a *single* node at level 2 incident to a thick line labeled by a red 3 and an unlabeled thin line. A green triangle with rounded corners surrounds it which is connected to green circle within which the set partition tree structure is expanded.

The leftmost path from the root node to a terminal node yields the node labels (*0 0 0 0 0*), which is the restricted growth string of partition $\{\{4,3,2,1,0\}\}$. The rightmost path from the root node to a terminal node yields the node labels (*0 1 2 3 4*), which is the restricted growth string of partition $\{\{4\},\{3\},\{2\},\{1\},\{0\}\}$.

In Section 6 we derive a recursion relation for the edge labels (red) in the set partition mesh.

4.3 Traversal of a Set Partition Mesh

Algorithm 4.3 Traversal of the Set Partition Mesh: Given an index I, such that $0 \leq I < B_n$, find the corresponding restricted growth string $(b_0 b_1 \dots b_{n-1})$ as follows.

Set the level $\ell = 0$. Set the current node η_{cur} to the root node. Set b_0 to the label of the η_{cur}; i.e., set $b_0 = 0$. Iterate the following until $\ell = n$.

1. At the current node, η_{cur}, identify the edge e such that the factor f_{e} from among the edges for which η_{cur} is a predecessor node, where f_{e} $(0 \leq f_{\text{e}} \leq \ell + 1)$ is the largest factor such that $I \geq \omega_{\text{e}} f_{\text{e}}$. Let η_{new} be the successor node to e.

2. Set b_ℓ to the label of η_{new}.

3. Set I to $I - \omega_{\text{e}} f_e$.

4. Set η_{cur} to η_{new}.

5. Set ℓ to $\ell + 1$.

Note that e is a specific edge in the set partition mesh. For example, considering Fig. 5, it is either among the edges represented by a thick line or it is the edge represented by a single thin line. The value of f_{e} distinguishes which edge it is.

Example 4.4 For example, considering Fig. 5, traverse the set partition mesh using index 20. Applying the traverse in Algorithm 4.3 yields the sequence of edge weights ω_{e} of 15, 0, 3, and 2 (which sum to 20), and the restricted growth string of (*0 1 0 1 2*).

5 Extracting the I-th Set Partition

We consider the extraction of the I-th set partition from the set partition tree and from the set partition mesh. First, consider the tree.

Theorem 5.1 *Using an index I, where $0 \leq I < B_n$, a traverse of the set partition tree for set partitions on n-sets, under Algorithm 3.3, yields the I-th set partition in lexicographical order of the restricted growth string.*

Proof: By its construction, the sequence of node labels resulting from any traverse from the root node to a leaf node is a restricted growth string, as described in (2.1). Further, all restricted growth strings occur as a unique path in the set partition tree, again by construction. In the set partition tree, smaller node labels are to the left and larger are to the right, this being imposed by the ascending left-to-right order of child node labels. Thus, the restricted growth strings are in ascending lexicographical order from left to right. Since the leaf node weights are also in ascending order from left-to-right, ranging from 0 to $B_n - 1$, the hypothesis follows. □

Next, consider the mesh.

Theorem 5.2 *Using an index I, where $0 \leq I < B_n$, a traverse of the set partition mesh under Algorithm 4.3, yields the I-th set partition in lexicographical order of the restricted growth string.*

Proof: A traversal of the set partition mesh according to Definition 4.3 corresponds to a traversal of the set partition tree according to Definition 3.3 and both yield the same restricted growth string, as follows. Initially, both traversals traverse the root node with a label of 0. Thus, b_0 is the same for both traversals. In the set partition tree, the successor edge to the root node has weight B_{n-1}, and this edge is traversed if and only if $i > B_{n-1}$, and it follows that $b_1 = 1$ if and only if $i > B_{n-1}$. In the set partition mesh, the left successor edge of the root node is labeled B_{n-1}, which means the right successor edge is traversed if and only if $i > B_{n-1}$.

Suppose that the tree and the mesh yield identical values b_0, b_1, …, $b_{\ell-1}$ in the restricted growth string for all indices i. In the set partition mesh, two nodes η_1

and η_2 at level ℓ are coalesced if the maximum label of the nodes along the paths from η_1 and η_2 to the root node are identical including the labels of η_1 and η_2. Let $b_{\max} = \max\{b_0, b_1, \ldots, b_{\ell-1}\}$ in the case of η_1 and η_2. The set partition tree at η_1 and η_2 are the same and have the same form as the set partition mesh at this point. □

6 The Set Partition Number System

In the traverse of a generating tree/mesh for set partitions, we are given an index I, and we derive the I-th restricted growth string. Because there is a bijection between set partitions and restricted growth strings, this process therefore derives a unique I-th set partition. At each step in the traverse, we use I to choose an element of the restricted growth string. Then, we modify I and repeat until I is 0. In effect, a traversal "dissects" I into a sequence of weights of the edges traversed. The first weight in the sequence is either 0 or B_{n-1} corresponding to the weights of the two edges whose predecessor is the *root* node, and correspond to the label on the successor node, $b_1 = 0$ or $b_1 = 1$, respectively. Therefore, either 0 or B_{n-1} of the leaf nodes along the bottom is extracted. The successor node is the predecessor node of the next edge weight, which extracts an adjacent segment of leaf nodes (including possibly 0 nodes). Further, another successor node to the next edge weight is generated, etc.. As the edge weights are extracted, so are the nodes along the leaf node side of the set partition tree until they meet at the leaf node corresponding to the index I.

One can invert this process and use the restricted growth string to find the index I. That is, as the traverse proceeds, it is the restricted growth string that specifies the choices made at each level. Here, the accumulation of the edge weights yields the value of I. The restricted growth string serves as the *digits* of a number system, while the edge weights serve as the *coefficients.* It is this viewpoint that we take to present the following result.

Theorem 6.1 *In a* **set partition number system***, a non-negative integer I, where $I < B_n$, is uniquely represented as $b_0b_1 \ldots b_{n-2}b_{n-1}$, where*

$$I = \sum_{i=1}^{n-1} \omega(i, M) b_i, \tag{6.1}$$

such that $(b_0b_1 \ldots b_{n-2}b_{n-1})$ is a restricted growth string, $M = \max\{b_0 = 0, b_1, \ldots, b_{i-1}\}$, and $\omega(i, M)$ is computed recursively, as follows:

$$\omega(i, 0) = B_i, \tag{6.2}$$

$$\omega(i,M) = \omega(i+1, M-1) - M\omega(i,M-1), \text{for } 0 \leq M < n, \tag{6.3}$$

where B_i is the i-th Bell number $(B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15, B_5 = 52, \ldots)$.

Proof: The proof is based on the observation that a traverse of the set partition tree guided by a restricted growth string yields a unique index I. The elements of the restricted growth string are digits of the number, I is the value of that number, and certain edge weights are the coefficients. It remains to show that the values of $\omega(i, M)$ given in (6.2) and (6.3) are the edge weights of the set partition mesh. Fig. 6 shows how the edge weights are specified.

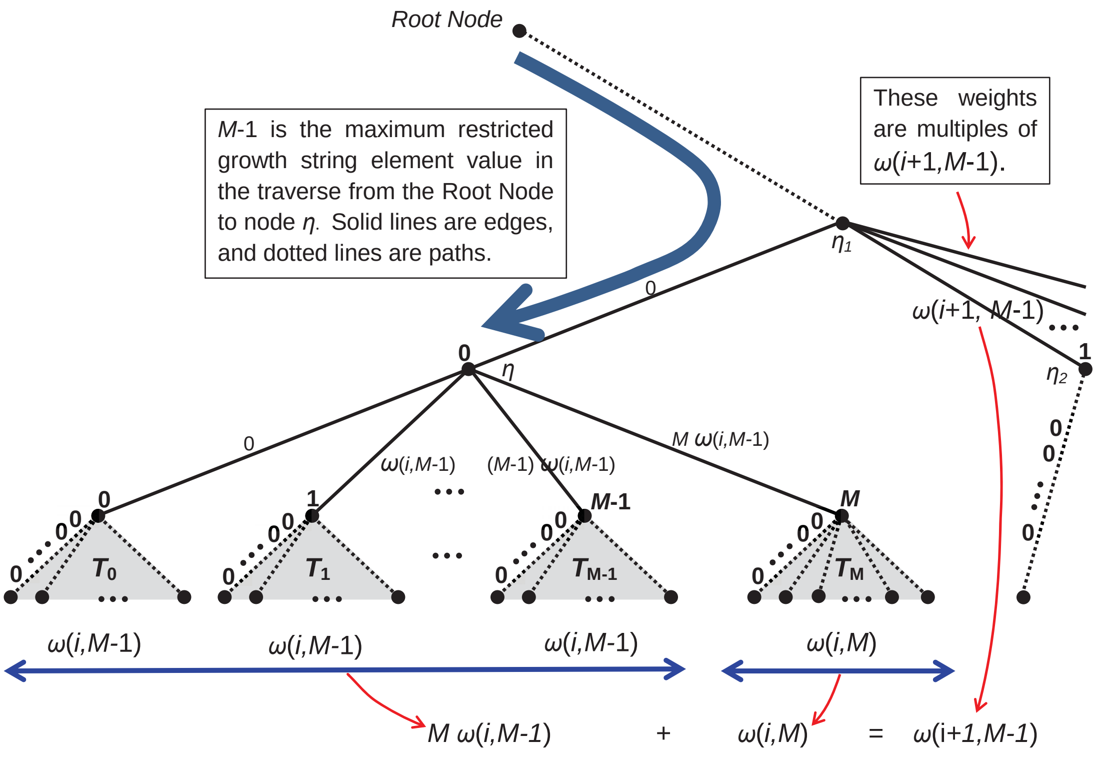

Root Node
M-1 is the maximum restricted growth string element value in the traverse from the Root Node to node η. Solid lines are edges, and dotted lines are paths.
These weights are multiples of ω(i+1,M-1).
η1
η2
η
ω(i+1, M-1)
0
1
M ω(i,M-1)
ω(i,M-1)
(M-1) ω(i,M-1)
M-1
M
T0
T1
TM-1
TM
ω(i,M-1)
ω(i,M)
M ω(i,M-1) + ω(i,M) = ω(i+1,M-1)

Figure 6: Calculation of Edge Weights in the Set Partition Tree.

Consider the edge in the upper right corner that is incident to both nodes η_1 and η_2. Let its edge weight be $\omega(i+1, M-1)$. Here, $i+1$ specifies a level in the set partition tree, and $M-1$ is the maximum value in the restricted growth string up through η_1. Recall that an edge weight represents the number of leaf nodes of a sub-tree of the set partition tree whose beginning and end are defined by projecting the predecessor node and the successor node to the leaf nodes by specifying remaining edge weights as 0. In Fig. 6, these projections are shown two paths of 0-edges beginning with nodes η_1 and η_2 going down to the leaf nodes. The number of leaf nodes is the number of leaf nodes across the subtrees T_0, T_1, …, T_{M-1}, and T_M. The first M subtrees, T_0, T_1, …, and T_{M-1} are identical; their structures depend on $M-1$, the maximum element in the restricted growth string specified so far. All have $\omega(i, M-1)$ leaf nodes. The rightmost subtree, T_M, has a different structure because its root node is M, increasing by 1 the maximum restricted growth string maximum element value as compared to all the previous M subtrees. T_M has $\omega(i, M)$ leaf nodes. Totaling the number of leaf nodes, yields $\omega(i+1, M-1) = M\omega(i, M-1) + \omega(i, M)$, from which (6.3) follows.

All edges on the extreme left side of the set partition tree have weight 0. The predecessor nodes to each of these edges each have two successor nodes. The right successor nodes are the root nodes of subtrees that themselves are complete set partition trees, all of which have a Bell number of leaf nodes. Therefore, traversing such nodes is equivalent to passing over a Bell number of leaf nodes. The weights of the right edges are given by (6.2).

Table 3(a) shows the values of $\omega(i, M)$ for $n = 5$. Blank values are unspecified by (6.2) and (6.3) and do not occur as edge weights. Here, the values specified by (6.2) are shown along the left column headed by $M = 0$.

Table 3: Set Partition Tree Coefficients for $n = 5$

	Restr. Gr. Str.→ Max.Value →				$(0\ 0\ 0\ 0\ 0)$ (0 0 0 0 0)				$(0\ 1\ 0\ 1\ 2)$ (0 1 1 1 2)				$(0\ 1\ 2\ 3\ 4)$ (0 1 2 3 4)			
$i\backslash M$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	15				**15**				**15**				**15**			
3	5	10			**5**	10			5	**10**			5	**10**		
2	2	3	4		**2**	3	4		2	**3**	4		2	3	**4**	
1	1	1	1	1	**1**	1	1	1	1	**1**	1	1	1	1	1	**1**
	$I\ =$				$0{\cdot}15+0{\cdot}5+0{\cdot}2+0{\cdot}1 = 0$				$1{\cdot}15+0{\cdot}10+1{\cdot}3+2{\cdot}1 = 20$				$1{\cdot}15+2{\cdot}10+3{\cdot}4+4{\cdot}1 = 51$			
	(a)				(b)				(c)				(d)			

□

Example 6.2 Consider the 5-set restricted growth string $(b_0b_1b_2b_3b_4) = (0\ 0\ 0\ 0\ 0)$. In this case, $\omega(4,0)\omega(3,0)\omega(2,0)\omega(1,0) = 15\ 5\ 2\ 1$, and $I = 0{+}0{\cdot}15{+}0{\cdot}5{+}0{\cdot}2{+}0{\cdot}1 = 0$. Table 3(b) shows how the coefficients are selected from all possible coefficients. In the case of $(b_0b_1b_2b_3b_4) = (0\ 1\ 0\ 1\ 2)$, $\omega(4,0)\omega(3,1)\omega(2,1)\omega(1,1) = 15\ 10\ 3\ 1$, and $I = 0 + 1 \cdot 15 + 0 \cdot 10 + 1 \cdot 3 + 2 \cdot 1 = 20$. Table 3(c) shows how the coefficients are selected from all possible coefficients. In the case of $(b_0b_1b_2b_3b_4) = (0\ 1\ 2\ 3\ 4)$, $\omega(4,0)\omega(3,1)\omega(2,2)\omega(1,3) = 15\ 10\ 4\ 1$, and $I = 0{+}1{\cdot}15{+}2{\cdot}10{+}3{\cdot}4{+}4{\cdot}1 = 51$. Table 3(d) shows how the coefficients are selected from all possible coefficients. This shows how the maximum value vector directly specifies how the coefficients are chosen from the set of all possible coefficients, as specified in (6.2) and (6.3).

We remark that the triangle of numbers in Table 3(a) is just the set of coefficients of the set partition mesh. This is similar to the Peirce Triangle [15]; indeed, the Peirce Triangle is described by (6.2) and (6.3) with the term $-M\omega(i,M{-}1)$ in (6.3) replaced by $-\omega(i,M{-}1)$ (viz. [8]).

7 The Number of Nodes in the Set Partition Tree/Mesh

An insight as to why the set partition mesh has so few nodes compared to the set partition tree comes from observing how the index is stored. In the set partition tree, there is one leaf node for every value of the index. In the set partition mesh, the index is stored as a quantity that is accumulated as the mesh is traversed. The number of leaf nodes is much smaller in comparison. Because the number of nodes is an important factor in determining whether or not we can design the hardware to enumerate set partitions, we compute the number of nodes in this section. Let $\mathcal{T}(n)$ be the number of nodes in the set partition tree on n elements.

Theorem 7.1 *The number of nodes $\mathcal{T}(n)$ in the set partition tree on n elements is*

$$\mathcal{T}(n) = \sum_{i=1}^{n} B_i, \tag{7.1}$$

where B_i is the i-th Bell number.

That is, at each step in the construction of the set partition tree for n-sets, a set partition tree for i-sets is formed, for $1 \leq i \leq n$. So, at each step in the construction, B_i leaf nodes are added to the total node count. For example, (7.1) yields $\mathcal{T}(5) = 75$, which can be verified in Figure 3.

For the set partition tree, the nodes form a triangle of size $n \times n \times n$. Counting along the diagonals, there are $\mathcal{M}(n) = 1 + 2 + \ldots + n$ nodes. Thus,

Theorem 7.2 *The number of nodes $\mathcal{M}(n)$ in a set partition mesh for n elements is*

$$\mathcal{M}(n) = \frac{n(n+1)}{2}. \tag{7.2}$$

For example, (7.2) yields $\mathcal{M}(5) = 15$, which can be verified in Figure 5.

Table 4 compares the number of nodes in the set partition tree with that in the set partition mesh for various values in the range $2 \leq n \leq 32$. Clearly, the set partition tree has many more nodes than the set partition mesh. Experimental data for the logic circuits described in [3] shows that, when the set partition tree is used to store data for the circuit realization, the circuit specification can accommodate up to only $n = 10$. However, when the set partition mesh was used, an n up to 32 can be accommodated.

8 Conclusions

This paper makes two contributions. First, it introduces the set partition number system. The digits are elements of the restricted growth string of a given set partition,

Table 4: Comparison of the Number of Nodes in the Set Partition Tree, $\mathcal{T}(n)$, with that in the Set Partition Mesh, $\mathcal{M}(n)$.

n	$\mathcal{T}(n)$ - No. of Nodes in Set Partition Tree	$\mathcal{M}(n)$ - No. of Nodes in Set Partition Mesh
2	3	3
3	8	6
4	23	10
5	75	15
6	278	21
7	$1,155$	28
8	$5,295$	36
9	$26,442$	45
10	$142,417$	55
11	$820,987$	66
12	$5,034,584$	78
13	$32,679,021$	91
14	$223,578,343$	105
15	$1,606,536,888$	120
16	1.2087×10^{10}	136
32	1.3928×10^{26}	528

and the number's value is a unique index to the set partition. This is an extension to what is known about number systems based on combinatorial objects. That is, only two other such number systems are known, one based on combinations and the other based on permutations.

Second, this paper introduces the set partition mesh, a data structure for storing set partitions. We show that the set partition mesh is much more compact than the set partition tree. It makes possible the design of hardware set partition generators for n-sets as large as $n = 32$, compared to the set partition tree, which limits the sets to a size no more than $n = 10$. Readers interested in the hardware to enumerate set partitions, are invited to read [3].

Acknowledgments

We appreciate the helpful comments of three referees.

References

[1] J. T. Butler and T. Sasao, Index to constant weight codeword converter, *Proc. 7th Int. Symp. on Applied Reconfigurable Computing*, Proc. Lec. Notes Comp. Sci. (LNCS 6576), Springer-Verlag Berlin Heidelberg, 2011, A. Koch et al. (Eds.), Belfast, N. Ireland, March 23-25, 2011, 193–204.

[2] J. T. Butler and T. Sasao, Hardware index to permutation converter, *19th Reconfigurable Architectures Workshop*, (RAW-2012), Proc. 26th IEEE Int. Parallel and Distributed Processing Symp., Shanghai, China, May 21-22, 2012, 424–429.

[3] J. T. Butler and T. Sasao, High-speed hardware partition generation, *ACM Trans. on Reconfigurable Technology and Systems*, Dec. 2014, 7 Issue 4, 1–17.

[4] R. K. S. Hankin and L. J. West, Set partitions in *R*, *J. Statist. Software*, 23, Code Snippet 2, Dec. 2007, `http://www.jstatsoft.org/`.

[5] G. Hutchinson, Partitioning algorithms for finite sets, *Comm. ACM*, 6, 1963, 613–614.

[6] S. Kawano and S. Nakano, Constant time generation of set partitions, *IEICE Trans. Fundamentals*, E88-A, No. 4, 930–934, April 2005.

[7] D. E. Knuth, Vol. 2: Seminumerical Algorithms, *The Art of Computer Programming*, (3rd ed.), Addison-Wesley, 209, 1997.

[8] D. E. Knuth, Vol. 4: Generating all combinations and permutations, *The Art of Computer Programming*, Fascicle 3, Addison-Wesley, 65, 2005.

[9] D. Lavenier and Y. Saouter, Computing Goldbach partitions using pseudo-random bit generator operators on an FPGA systolic array, *8th Int. Workshop, FPL'98*, Proc. Lec. Notes Comp. Sci. (LNCS 1482), R. W. Hartenstein and A. Keevallik (Eds.), Tallinn, Estonia, Aug.-Sept. 1998, 316–325.

[10] T. Mansour and G. Nassar, Gray codes, loopless algorithm and partitions, *J. Math. Model. Algorithms*, 7 (3) (2008), 291–310.

[11] T. Mansour, *Combinatorics of Set Partitions*, Chapman & Hall/CRC, CRC Press, Boca Raton, FL, 2012.

[12] J. McCaffrey, Generating the mth lexicographical element of a mathematical combination, `http://msdn.microsoft.com/en-us/library/aa289166%28VS.71%29.aspx`, July 2004.

[13] J. K. S. McKay, Algorithm 263, Partition Generator, *Comm. ACM* 8 No. 8 (1965), 493.

[14] E. Pascal, *Giornale di Matematiche* 25 (1887), 45–49.

[15] C. S. Peirce, On the algebra of logic, *Amer. J. Math.* 3 (1) (1880), 15–57.

[16] E. Reingold, J. Nivergelt and N. Deo, *Combinatorial Algorithms, Theory and Practice*, Prentice-Hall, 1977.

[17] I. Semba, An efficient algorithm for generating all partitions of the set $\{1,2, \dots, n\}$, *J. Inf. Proc.*, 7 (1) (1984), 41–42.

[18] J. L. Shafer, S. W. Schneider, J. T. Butler and P. Stanica, Enumeration of bent Boolean functions by reconfigurable computer, *The 18th Annual Int. IEEE Symposium on Field-Programmable Custom Computing Machines*, Charlotte, NC, May 2-4, 2010, 265–272.

[19] S. G. Williamson, Ranking algorithms for lists of partitions, *SIAM J. Comput.* 5 (1976), 602–617.

(Received 16 July 2015; revised 6 Apr 2016)