

## A Fast Iterative Algorithm for Generating Set Partitions

An iterative algorithm for generating all partitions of the set  $\{1, \dots, n\}$  is presented. An empirical test shows that the new algorithm is faster than the previously fastest algorithm recently proposed by Er on some computers, though the former is slower than the latter on a computer where fast recursive call is provided based on an RISC architecture.

Received October 1988

### 1. Introduction

Let  $Z := \{1, \dots, n\}$ . A partition of the set  $Z$  consists of  $k$  classes  $\pi_1, \dots, \pi_k$  such that  $\pi_i \cap \pi_j = \emptyset$  if  $i \neq j$ ,  $\pi_1 \cup \dots \cup \pi_k = Z$ , and  $\pi_i \neq \emptyset$  for  $1 \leq i \leq k$ . In this paper we consider the problem of generating all partitions of the set  $Z$ .

Sequential algorithms for set partitioning are studied in the literature.<sup>2,3,4,6,7</sup> The fastest among them (as shown by Er)<sup>2</sup> is the recursive algorithm given by Er,<sup>2</sup> while the fastest previously known iterative algorithm is one given by Semba.<sup>6</sup>

The purpose of the paper is to derive an efficient algorithm for generating all partitions of the set  $Z$ , which is faster than both the Er<sup>2</sup> and Semba<sup>6</sup> algorithms, as shown by empirical results.

### 2. Algorithms

A codeword  $c_1 c_2 \dots c_n$  represents a partition of the set  $Z$  if and only if  $c_1 = 1$  and  $1 \leq c_r \leq \max(c_1, \dots, c_{r-1}) + 1$  for  $2 \leq r \leq n$ , where  $c_i = j$  if  $i$  is in  $\pi_j$ . A list of codewords and corresponding partitions for  $n = 4$  is as follows: 1111 = (1234), 1112 = (123)(4), 1121 = (124)(3), 1122 = (12)(34), 1123 = (12)(3)(4), 1211 = (134)(2), 1212 = (13)(24), 1213 = (13)(2)(4), 1221 = (14)(23), 1222 = (1)(234), 1223 = (1)(23)(4), 1231 = (14)(2)(3), 1232 = (1)(24)(3), 1233 = (1)(2)(34), 1234 = (1)(2)(3)(4).

We present an algorithm which is naturally derived from the above consideration of the codewords. In the program we use an array to store  $g_r := \max(c_1, \dots, c_r)$ .

```

program setpart1(n);
begin
  r := 0; c0 := 0; n1 := n - 1; g0 := 0;
  repeat
    while r < n1 do begin r := r + 1; cr := 1; gr := gr-1 end;
    for j := 1 to gn1 + 1 do begin cn := j;
      print out c1...cn; end;
    while cr > gr-1 do r := r - 1;
    cr := cr + 1;
    if cr > gr then gr := cr;
  until r = 1
end;

```

In the second WHILE a backtrack is made to find the largest  $r$  having an 'increasable'  $c_r$ , i.e.  $c_r \leq g_{r-1} + 1$ . Although the improvement in the execution times is significant compared with the program in Semba,<sup>6</sup> this improvement is mainly due to avoiding goto statements. We want to improve the above algorithm further. The new algorithm for generating set partitions goes as follows:

```

program setpart2(n);
begin
  r := 1; c1 := 1; j := 0; b0 := 1;
  n1 := n - 1;
  repeat
    while r < n1 do begin r := r + 1;
      cr := 1; j := j + 1; bj := r end;
    for j := 1 to n - j do begin cn := j;
      print out c1...cn; end;
    r := bj;
    cr := cr + 1;
    if cr > r - j then j := j - 1
  until r = 1
end;

```

In the presented iterative algorithm  $b_j$  is the position where current position  $r$  should backtrack after generating all codewords beginning with  $c_1 \dots c_{n-1}$ . An element of  $b$  is defined whenever  $g_r = g_{r-1}$ , which is recognized by either  $c_r = 1$  or  $c_r > r - j$  in the algorithm. It is easy to see that the relation  $r = g_{r-1} + j$  holds whenever  $j$  is defined (cf. Fig. 1). Thus the number of backtrack calls is equal to  $B_{n-1}$ , where  $B_j$  is the well-known Bell number giving the number of partitions of the set  $\{1, \dots, j\}$ . Each backtrack is done in constant time. This is the main improvement over setpart1, though we have to consume two statements:  $j := j + 1$  and  $b_j := r$  as many times as the backtrack is done (this is the

reason that both programs run in a comparable time). The backtrack by the array  $g_r$  is non-constant and, furthermore, the array  $b$  which we use always has less elements than the array  $g$ . In fact, at the printing step in our algorithm  $b$  has  $n - g_n$  elements, while  $g$  has  $n$  elements.

Compared to Er's algorithm,<sup>2</sup> ours treats the backtracking more efficiently in the sense that in the recursive algorithm,<sup>2</sup> backtrack again requires non-constant time (since it corresponds to the returning of the recursive calls).

It is of interest to note that the codewords generated by the algorithms of Er,<sup>2</sup> Semba<sup>6</sup> and the one described in this paper are always in lexicographic order.

### 3. Performance evaluation

In order to evaluate the performance of our newly proposed algorithms and to compare them with those of Er<sup>2</sup> and Semba,<sup>6</sup> all the algorithms have been implemented in Sun and VAX Pascals and compiled under the UNIX operating system on the Sun-4/280 and VAX 8800 computers (the optimising option is used). The actual CPU running times of the algorithms are summarised in Tables 1 and 2 (algorithms are run once without printing out

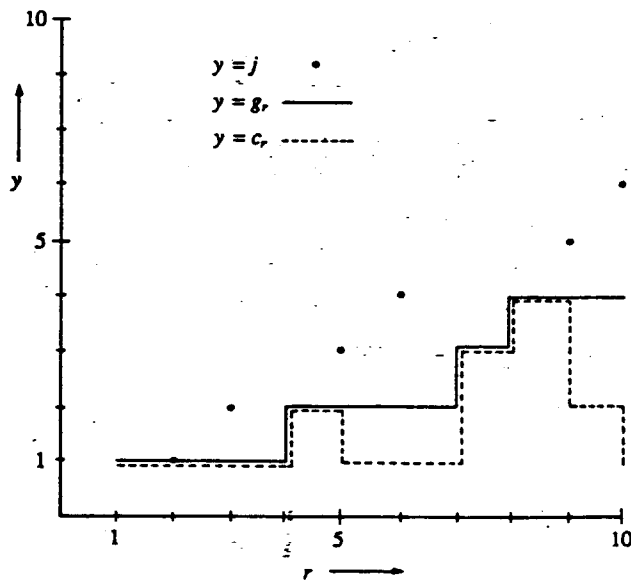


Figure 1.  $c = 1112113421$ .

Table 1. A comparison of the actual running times (measured in seconds of CPU time) of Er's, Semba's and our Setpart algorithms for generating all partitions of  $\{1, \dots, n\}$  on a VAX 8800 computer

$n$	Er's	%	Semba's	%	Setpart1	%	Setpart2	%
12	11.1	100	23.0	207	10.5	94.5	8.8	79.3
13	73.4	100	147.4	201	68.6	93.5	57.2	77.9
14	494.3	100	998.9	202	464.2	93.9	392.5	79.4
15	3489.9	100	7134.7	204	3271.7	93.7	2774.9	79.5
16	25820.2	100	52900.2	204	23839.7	92.3	20449.2	79.2

Table 2. A comparison of the actual running times (measured in seconds of CPU time) on a Sun-4/280 computer

$n$	Er's	%	Semba's	%	Setpart1	%	Setpart2	%
12	3.4	100	19.4	606	7.6	238	7.0	219
13	21.7	100	124.9	576	48.4	223	44.5	205
14	146.1	100	845.4	579	323.7	222	297.0	203
15	1030.4	100	6031.4	585	2285.1	222	2113.8	205
16	7577.1	100	45065.8	595	16825.8	222	15627.6	206

partitions). The results show clearly that both setpart1 and setpart2 are faster than Er's on the VAX computer (94 and 80%, respectively; a similar result is obtained on a Sun-3/180 computer). But they are much slower (more than twice) than Er's recursive one on the Sun-4 computer. The reason is that the recursive call and return consume more time than arithmetic operation on the VAX computer while on the Sun-4 computer, where an RISC (reduced instruction set computer) architecture is adopted, very fast call/return operations (comparable to register arithmetic operation) are provided for a small program like setpart by the aid of a good optimising compiler and fast registers (cf. Ref. 5). Actually, the recursive program compiled under the optimiser option is about four times faster than one under a no-optimizer option, while iterative programs are run twice as fast through compilation under the optimiser option (thus both programs run in a comparable time in a no-optimiser option). We note that this outstanding performance of Sun-4 could be rapidly reduced when the nesting of recursive calls goes deeper than a certain critical depth.<sup>9</sup>

#### 4. Concluding remarks

We have succeeded in deriving an efficient algorithm for generating set partitions. The algorithm is significantly faster than the previous fastest iterative algorithm; it is also faster than the previously reported fastest program, that of Er, on some computers. It is as simple as the programs given by Er<sup>2</sup> and Semba.<sup>6</sup> Though it runs much more slowly than Er's one on some computers (where recursive calls are optimised under an RISC architecture), the present algorithm has one more advantage over the recursive algorithm:<sup>2</sup> it enables an efficient adaptive and cost-optimal parallel algorithm to be devised, as described in another paper by the same authors.<sup>1</sup>

#### Acknowledgement

We appreciate the comments made by Dr Takio Kurita on the experimental data.

B. DJOKIĆ,<sup>1</sup> M. MIYAKAWA,<sup>2\*</sup>  
S. SEKIGUCHI,<sup>3</sup> I. SEMBA<sup>3</sup> and  
I. STOJMENOVIC<sup>4</sup>

<sup>1</sup>Department of Mathematics and Computer Science, University of Miami, P.O. Box 249085, Coral Gables, FL 33124, USA.

<sup>2</sup>Electrotechnical Laboratory, 1-1-4 Umezono Tsukuba 305, Japan.

<sup>3</sup>Ibaraki University, 2-1-1 Bunkyo, Mito-shi, Ibaraki 310, Japan.

<sup>4</sup>Department of Computer Science, University of Ottawa, 34 G. Glinksi, Ottawa, Canada K1N 6N5.

\* To whom correspondence should be addressed

#### References

1. B. Djokić, M. Miyakawa, I. Semba, S. Sekiguchi and I. Stojmenović, Parallel algorithms for generating subsets and set partitions to appear in *The Computer Journal*.
2. M. C. Er, A fast algorithm for generating set partitions. *The Computer Journal*, 31 (3), 283-284 (1988).
3. R. A. Kaye, A Gray code for set partitions. *Information Processing Letters*, 5, 171-173 (1976).
4. A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms*. Academic Press, New York (1978).
5. D. A. Patterson, C. H. Séquin, A VLSI RISC. *IEEE Computers*, 9, 8-21 (1982).
6. I. Semba, An efficient algorithm for generating all partitions of the set  $\{1, \dots, n\}$ . *Journal of Information Processing*, 7, 41-42 (1984).
7. M. B. Wells, *Elements of Combinatorial Computing*. Pergamon Press, Oxford (1971).
8. D. Wilson, The Sun 4/260 RISC-based technical workstation. *Unix Review*, July 1988, 91-98; Japanese translation: *Unix Magazine*, 10, 23-29 (1988).