A Fast Iterative Algorithm for Generating Set Partitions

An iterative algorithm for generating all partitions of the set \{1, \ldots, n\} is presented. An empirical test shows that the new algorithm is faster than the previously fastest algorithm recently proposed by Er\textsuperscript{2} 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 \(Z = \{1, \ldots, n\}\). A partition of the set \(Z\) consists of \(k\) classes \(\pi_1, \ldots, \pi_k\) such that \(\pi_i \cap \pi_j = \emptyset\) if \(i \neq j\), \(\pi_i \cup \pi_j = Z\), 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\textsuperscript{3,4,5,6,7}. The fastest among them (as shown by Er\textsuperscript{2}) is the recursive algorithm given by Er\textsuperscript{2} while the fastest previously known iterative algorithm is one given by Semb\textsuperscript{6}.

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\textsuperscript{2} and Semb\textsuperscript{6} algorithms, as shown by empirical results.

2. Algorithms

A codeword \(c_1, c_2, \ldots, c_n\) represents a partition of the set \(\{1, \ldots, n\}\) if and only if \(c_1 = 1\) and \(1 \leq c_i \leq c_{i-1}\), \(i = 2, \ldots, n\), \(c_i = j\) if \(i\) is in \(\pi_j\). A list of codewords and corresponding partitions for \(n = 4\) is as follows:

\[
\begin{align*}
1111 &= \{1, 2, 3, 4\}, \\
1112 &= \{1, 2, 3\}, \\
1121 &= \{1, 2\}, \\
1211 &= \{1, 3\}, \\
1221 &= \{1\}, \\
2112 &= \{2, 3\}, \\
2211 &= \{2\}.
\end{align*}
\]

We present an algorithm which is naturally derived from the above consideration of the codewords. In the program we use an array to store \(g_n = \max(c_1, \ldots, c_n)\).

program setpart1(n):

begin
    \(r := 0\), \(c_1 := 0\), \(n! := n - 1\), \(g_n := 0\);
    repeat
        \(r := r + 1\), \(c_r := g_r\);
        for \(j := 1\) to \(g_1 + 1\) do begin \(c_j := j\); print out \(c_1, \ldots, c_n\); end;
        while \(c_r > g_{r-1}\) do begin \(r := r - 1\); \(c_r := c_r - 1\); end;
        if \(c_r > \max(c_1, \ldots, c_n)\) then \(g_r := c_r\); end;

end;

In the second WHILE a backtrack is made to find the largest \(r\) having an ‘increasable’ \(c_r\), i.e., \(c_r \leq g_r + 1\). Although the improvement in the execution times is significant compared with the program in Semb\textsuperscript{6}, 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:

3. Performance evaluation

In order to evaluate the performance of our newly proposed algorithms and to compare them with those of Er\textsuperscript{2} and Semb\textsuperscript{6}, 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. \(c_1 = 1112113421\).](https://example.com/figure1.png)

<table>
<thead>
<tr>
<th>(n)</th>
<th>Er’s</th>
<th>%</th>
<th>Semb’s</th>
<th>%</th>
<th>Setpart1</th>
<th>%</th>
<th>Setpart2</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>11.1</td>
<td>100</td>
<td>23.0</td>
<td>100</td>
<td>207</td>
<td>10.5</td>
<td>94.5</td>
<td>8.8</td>
</tr>
<tr>
<td>13</td>
<td>73.4</td>
<td>100</td>
<td>147.4</td>
<td>100</td>
<td>201</td>
<td>68.6</td>
<td>93.5</td>
<td>57.2</td>
</tr>
<tr>
<td>14</td>
<td>494.3</td>
<td>100</td>
<td>998.9</td>
<td>100</td>
<td>202</td>
<td>464.2</td>
<td>93.9</td>
<td>392.5</td>
</tr>
<tr>
<td>15</td>
<td>3489.9</td>
<td>100</td>
<td>7134.7</td>
<td>100</td>
<td>204</td>
<td>3271.7</td>
<td>93.7</td>
<td>2774.9</td>
</tr>
<tr>
<td>16</td>
<td>25820.2</td>
<td>100</td>
<td>52900.2</td>
<td>100</td>
<td>204</td>
<td>23839.7</td>
<td>92.3</td>
<td>20449.2</td>
</tr>
</tbody>
</table>

Table 1. A comparison of the actual running times (measured in seconds of CPU time) of Er’s, Semb’s and our Setpart algorithms for generating all partitions of \(\{1, \ldots, n\}\) on a VAX 8800 computer
Table 2. A comparison of the actual running times (measured in seconds of CPU time) on a Sun-4/280 computer

<table>
<thead>
<tr>
<th>n</th>
<th>Er's</th>
<th>%</th>
<th>Sembas</th>
<th>%</th>
<th>Setpart1</th>
<th>%</th>
<th>Setpart2</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3.4</td>
<td>100</td>
<td>19.4</td>
<td>606</td>
<td>7.6</td>
<td>238</td>
<td>7.0</td>
<td>219</td>
</tr>
<tr>
<td>13</td>
<td>21.7</td>
<td>100</td>
<td>124.9</td>
<td>576</td>
<td>48.4</td>
<td>223</td>
<td>44.5</td>
<td>203</td>
</tr>
<tr>
<td>14</td>
<td>146</td>
<td>100</td>
<td>845.4</td>
<td>579</td>
<td>322.7</td>
<td>222</td>
<td>297.0</td>
<td>203</td>
</tr>
<tr>
<td>15</td>
<td>1030.4</td>
<td>100</td>
<td>6031.4</td>
<td>585</td>
<td>2285.1</td>
<td>222</td>
<td>2113.8</td>
<td>205</td>
</tr>
<tr>
<td>16</td>
<td>7577.1</td>
<td>100</td>
<td>45065.8</td>
<td>595</td>
<td>16825.8</td>
<td>222</td>
<td>15627.6</td>
<td>206</td>
</tr>
</tbody>
</table>

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 and Sembas. 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: it enables an efficient adaptive and cost-optimal parallel algorithm to be devised, as described in another paper by the same authors.

Acknowledgement

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

B. DJOKIĆ, M. MIYAKAWA, S. SEKIGUCHI, J. SEMBA and I. STOJNEMOVIC

References


Correspondence

A proposal for the renaming of the scientific branch of computer science

Dear Sir,

The creation, expansion and evolution of the scientific branch of Computer Science which is unique in world history is entirely due to the computer. The term Computer Science means, of course, the science of computers.

For this reason, perhaps, European scientists decided on the term Informatique, which was later accepted by nearly all the European countries. This term was derived from the combination of the French words Information and Automatique.

Personally I have been preoccupied with these terms, ‘Computer science’ and ‘Informatique’, because the first is based on the instrument (Computer), and the second on the kind of processing (Automatique).

In reality we have identified the machine or the kind of processing with the science itself. But our problem is, of course, the information itself and its change and not the computer or the kind of processing. It is as unsuitable to use the term ‘Computer Science’ as it would be to name, for example, Astronomy, ‘Telescope Science’, or for Microbiology to be called ‘Microscope Science’.

For all these reasons we suggested in the late sixties the term ‘pliroforiki’ (πληροφορική) to be adopted in Greece. This word is derived from the Greek work pliroforia (πληροφορία = information) with the suffix iki (-ική). This suffix gives the word the precise meaning we need. That is, this term defines exactly the object of this modern science, which is the study of information and its change.

The richness and the exactness of the Greek language made it in the past a source of suitable terms for most of the scientific areas. Many of these terms are similar in form to the above term pliroforiki. Such examples are: ‘logic’ (λογική), ‘arithmetic’ (αριθμητική), ‘mechanics’ (μηχανική), ‘statistics’ (στατιστική), ‘physics’ (φυσική), etc.

The term pliroforiki has been fully accepted in Greece in recent years. So I believe that an attempt should be made to establish this term internationally, because the term not only expresses completely the above mentioned branch of science, but is also consistent with corresponding terms of other scientific areas which have been established for many years.

Hence the precise term and consequently its longevity is ensured.

I should be deeply obliged if you would publish this letter in your journal in order to invite wider discussions by specialists in this branch of science.

Yours faithfully,

N. APOSTOLATOS

Od. Andrountou 7-Ag. Paraskevi,
Athens 15341, Greece

Editor’s Comment

What’s in a Name

In his letter, published above, Professor Apostolatos expresses a concern that neither of the phrases “Computer Science” (English language) nor “Informatique” (French language) reflects adequately the nature of the subject that the phrases are intended to represent. His contention is that the two phrases are too closely associated with computer hardware, whereas the central focus of computer-related studies is (or should be) information itself.

To my mind the principal weakness in the argument put forward by Professor Apostolatos is the contention that computer scientists should downgrade the importance of the tools with which they are working in relation to the medium. This seems akin to the suggestion that sculptors should rename their subject according to whether they are working with clay or bronze or wood or stone rather than in relation to the art of design. Information and information systems have been with us since the beginning of man’s evolution; the tools for generating computer-based information systems have been with us only for some 45 years. Surely the thing that distinguishes the subject which we are considering is the emergence of the tools together with the techniques for