AN ALGORITHM FOR GENERATING SUBSETS OF FIXED SIZE WITH A STRONG MINIMAL CHANGE PROPERTY

Peter EADES
Department of Computer Science, University of Queensland, St. Lucia, Queensland, Australia 4067

Brendan McKay
Australian National University, Canberra, A.C.T. 2600, Australia

Communicated by G.R. Andrews
Received 24 January 1984

Keywords: Combinatorial algorithm, minimal change algorithm, Hamilton path, combinations

1. Introduction

In this paper we present an algorithm for generating a list

\[ S_1, S_2, \ldots, S_m \quad (m = \binom{n}{k}) \]

of all subsets of size \( k \) of a set of size \( n \). The algorithm has two important properties: it is a minimal change algorithm in that successive subsets \( S_i \) and \( S_{i+1} \) are minimally different, and it is fast in the sense that the time required to produce the list is bounded by a constant multiple of \( \binom{n}{k} \).

There are many algorithms (see [4] for a survey), including the one presented in this paper, which have the following Minimal Change Property:

(MCP): if \( S_i \) and \( S_{i+1} \) are successively generated, then \( S_{i+1} \) is obtained from \( S_i \) by changing exactly one element.

That is, \( S_{i+1} \) and \( S_i \) have exactly \( k - 2 \) elements in common. Eades, Hickey and Read [5] note that (MCP) is not a sufficiently strong minimal change criterion when the subsets are represented as sorted arrays, that is, \( \{s_1, s_2, \ldots, s_k\} \) where \( s_1 < s_2 < \cdots < s_k \) is represented as the array \( (s_1, s_2, \ldots, s_k) \).

The difficulty is that although only one element of the set is changed, more than one entry of the array may have to change to retain the ordering of the entries.

The algorithm presented in this paper overcomes this difficulty. In fact, it generates all \( \binom{n}{k} \) elements of the set

\[ U_{n,k} = \{(s_1, s_2, \ldots, s_k) : 1 \leq s_1 < s_2 < \cdots < s_k \leq n\} \]

such that the following Strong Minimal Change Property holds:

(SMCP): if \( (s_1, s_2, \ldots, s_k) \) and \( (s'_1, s'_2, \ldots, s'_n) \) are successively generated elements of \( U_{n,k} \), then, for some \( m \), \( s_i = s'_i \) for all \( i \neq m \).

This property admits another description, relevant to the case where the subsets are stored as bitstrings. Suppose that a chord is a selection of \( k \) notes from a contiguous group of \( n \) keys on a piano. There are \( \binom{n}{k} \) chords. An algorithm with (SMCP) gives a person with \( k \) fingers a way of playing every possible chord, changing one note at a time, so that no two fingers are ever crossed.

The algorithm is presented in the next section,
and analysed in Section 3. Finally, in Section 4 we compare its performance to other algorithms with (MCP) and (SMCP).

2. The algorithm

In this section we describe a list L(n, k) of the elements of $U_{n,k}$ so that (SMCP) is satisfied. If $k < 0$ or $k > n$, then $L(n, k)$ is empty. If $k = 0$, then $L(n, k)$ consists of the single null vector ( ); if $k = n$, then $L(n, k)$ consists of the single vector (1, 2, ... , n); and if $k = 1$, then $L(n, k)$ is the list (1), (2), ... , (n). For $1 < k < n$ the list $L(n, k)$ is described in Fig. 2.

The list in Fig. 2 can be produced with Algorithm GENERATE in Fig. 3. The procedure FORWARD, when called with parameters $\text{POINTER} = p$, $\text{DIFFERENCE} = d$, operates on the array SUBSET as follows. Suppose that, prior to the call, SUBSET had value $s = (s_1, s_2, \ldots , s_k)$. Let $V_{p,d,s}$ be the set 
\[
\left\{ (s_1, \ldots , s_{p-1}, x_p, x_{p+1}, \ldots , x_k : \\
\quad d < x_p < x_{p+1} < \cdots < x_k < n \right\}.
\]
Note that $V_{1,0,()}$ is $U_{n,k}$. In fact, $V_{p,d,s}$ has $(k-n-d+1)$ elements and the correspondence between $V_{p,d,s}$ and $U_{n-d,k-p+1}$ is clear. The procedure FORWARD

![Fig. 1. Seven fingered person playing a chord with k = 7.](image)

![Fig. 2. The list L(n, k) for 1 < k < n.](image)

Algorithm GENERATE(n, k)

declare SUBSET: array [0..k] of integer

main program

SUBSET = (1, 2, ... , k)
PROCESS(0, 0)
FORWARD(1, 0)

procedure FORWARD(POINTER, DIFFERENCE)
if (POINTER < k) and (DIFFERENCE < n-k-1)
then
FORWARD(POINTER + 1, DIFFERENCE + 2)
PROCESS(POINTER + 1, n-k + POINTER + 1)
REVERSE(POINTER + 1, DIFFERENCE + 2)
PROCESS(POINTER, DIFFERENCE + 2)
FORWARD(POINTER, DIFFERENCE + 1)
else if POINTER = k
then
for LASTINARRAY = DIFFERENCE + 2 to n do
PROCESS(k, LASTINARRAY)

procedure REVERSE(POINTER, DIFFERENCE)
if (POINTER < k) and (DIFFERENCE < n-k-1)
then
REVERSE(POINTER, DIFFERENCE + 1)
PROCESS(POINTER, DIFFERENCE + 1)
FORWARD(POINTER + 1, DIFFERENCE + 2)
PROCESS(POINTER + 1, DIFFERENCE + 2)
REVERSE(POINTER + 2, DIFFERENCE + 2)
else if POINTER = k
then
for LASTINARRAY = n-1 downto DIFFERENCE + 1 do
PROCESS(k, LASTINARRAY)

procedure PROCESS(POSITIONCHANGED, NEWVALUE)
SUBSET[POSITIONCHANGED] = NEWVALUE
append SUBSET to the list L(n, k)
adds all elements of $U_{p,d}$ to $L(n, k)$ in such a way that (SMCP) holds for successively added vectors, and REVERSE does the same in reverse order.

It is interesting to note that FORWARD and REVERSE are oblivious to the value of SUBSET; that is, they do not need to use the present value of subset to compute the next value.

The algorithm can be optimised by a clever implementation of the stack in a way similar to [7] and [1]. Such a version is available from the authors.

3. Analysis

It is clear from the text that the time complexity of $\text{GENERATE}(n, k)$ is $O(k + \binom{n}{k} + c(n, k))$, where $c(n, k)$ is the number of calls to FORWARD or REVERSE. Now consider the invocation tree of calls to these procedures; since every node which is not a leaf has three children and produces at least two subsets, it follows that $c(n, k) \leq 1 + \frac{3}{2}\binom{n}{k}$.

**Proposition 3.1.** The time complexity of the algorithm $\text{GENERATE}(n, k)$ is $O(k + \binom{n}{k})$.

This shows that the algorithm is fast in the sense that the average time required to generate a subset is bounded by a constant, independent of $n$ and $k$ if $k < n$.

In fact, an inductive argument may be used to calculate $c(n, k)$ explicitly, as well as the average $p(n, k)$ of POSITIONCHANGED over all calls to PROCESS.

**Proposition 3.2**

(a) $p(n, k) = \frac{k(n - k)}{(n - k + 1)}$, and

(b) $c(n, k) = 1 + 3 \sum_{i=1}^{\lceil k/2 \rceil} \binom{n - 2i}{n - k - 1}$.

It is interesting to note that $c(n, k) = O(n^{k-1})$ for fixed $k$. This implies that all but $O(n^{-1})$ of the subsets are produced by the $\textbf{for}$ loops, explaining the exceptional speed when $k$ is much smaller than $n$.

4. Conclusion

In [6, p. 42] and [3] two other subset generating algorithms which satisfy (SMCP) are given. Both these algorithms are fast in the sense of the previous section. However, the tests of [2] indicate that, in practice, both are considerably slower than Algorithm $\text{GENERATE}$ when $k$ is small in comparison to $n$. When $k$ is close to $n$, the algorithm in [3] is of comparable speed to $\text{GENERATE}$, but the algorithm of [6] remains slower.

The algorithm of [7] is considerably faster than $\text{GENERATE}$; although it satisfies (MCP), it does not satisfy (SMCP).

We refer the reader to [2] for detailed comparisons.

**References**


