# Efficiently Enumerating the Subsets of a Set

J. Loughry[*]
*Lockheed Martin Space Systems Company*

J.I. van Hemert[†]
*Leiden Institute of Advanced Computer Science*
*Leiden University*

L. Schoofs[‡]
*Intelligent Systems Lab*
*Department of Mathematics and Computer Science*
*University of Antwerp, RUCA*

12 December 2000

**Abstract**

The task of constructing the subsets of a set grows exponentially with the size of the set. The two most common methods of enumerating the subsets of a set, lexicographic ordering and Gray codes, in practice are sub-optimal when the sets become large. An algorithm is presented for rapidly finding the smallest subset $T_{\min} \subseteq S$ satisfying some condition $P$. The algorithm generates a sequence of all subsets of a set of $n$ elements in which the number of elements in each subset is monotonically increasing. Time complexity of the algorithm is only slightly greater than that of lexicographic ordering. The algorithm will find the smallest subset containing up to $k < n$ items before either lexicographic ordering or a binary reflected Gray code sequence have even looked at all $n$ elements in the set.

## 1 Introduction

The task of constructing the subsets of a set grows exponentially with the size of the set. We describe a method for generating a sequence of subsets in which the number of elements in each subset increases monotonically. No other known method of enumerating subsets generates a monotonic sequence. The method we present will find the smallest subset $T_{\min} \subseteq S$ satisfying some condition $P$

[*]Department 3740, Mail Stop X-3741, P.O. Box 179, Denver, Colorado 80201-0179 USA.
[†]Neils Bohrweg 1, 2333 CA Leiden, The Netherlands
[‡]Groenenborgerlaan 171, B-2020 Antwerpen, Belgium

in much less time than either lexicographic or Gray code ordering. An algorithm is given for efficiently generating sequences of this type. The technique is applicable to the solution of Constraint Satisfaction Problems (CSP).

### 1.1 Paper Organization

The first part of this paper introduces the idea of enumerating the subsets of a set by means of a sequence of binary numbers. The known systematic methods of enumerating the subsets of a set, lexicographic ordering and Gray codes, are described, along with their drawbacks when sets become large. Then a new method is described, one that generates the subsets of a set in monotonically increasing order of size. An algorithm is presented for efficiently generating sequences of this type.

## 2 Enumerating Subsets

We can express the idea of a $k$-subset ($T_k$) of a set $S$ having $n$ elements by means of a $n$-digit binary number in which exactly $k$ of the digits are 1 [4]. If a given element of the set $s_i \in T_k$, then the $i$th most-significant bit of the $n$-digit binary number is 1. The total number of subsets is $2^n$. Thus, to enumerate $\mathcal{P}(S)$, the set of all subsets of $S$, it suffices to count from 0 to $2^n - 1$ in binary.

There are $2^n!$ possible sequences. As $n$ becomes large, the order in which we construct the subsets becomes significant. A good solution must have the following characteristics:

- It must be computationally efficient.
- All subsets containing $k$ elements should be enumerated before any subsets containing $k+1$ elements; and furthermore, the number of elements in each subset should be monotonically increasing.

The two most commonly used methods of enumerating the subsets of a set are lexicographic ordering and Gray codes [2]. Unfortunately, neither of these methods satisfies the second requirement. We will shortly describe a new method that does.

### 2.1 Lexicographic Ordering

Lexicographic ordering is simply the familiar counting sequence

$$1, 2, 3, \ldots, 2^n - 1.$$

Table 1 shows a lexicographic sequence, and Figure 1 illustrates how the size of the subset varies over time. The problem with lexicographic ordering is that it never considers the last element in the set until halfway through the sequence. If you have a set of $n$ items, you must construct $2^{n-1}$ permutations of the first $n - 1$ items before even considering the $n$th item in the set. The performance of this sequence for large values of $n$ is poor.

Table 1: Subsets in lexicographic order. The last item in the set, represented by the most significant bit of the binary representation, is not even considered until halfway through the sequence. If the number of elements in the set is large, this could take a very long time.

| *Binary Number* | *Subset of* { ○, ◇, ⋆, ● } |
|---|---|
| 0000 | $\phi$ |
| 0001 | { ○ } |
| 0010 | { ◇ } |
| 0011 | { ○, ◇ } |
| 0100 | { ⋆ } |
| 0101 | { ○, ⋆ } |
| 0110 | { ◇, ⋆ } |
| 0111 | { ○, ◇, ⋆ } |
| 1000 | { ● } |
| 1001 | { ○, ● } |
| 1010 | { ◇, ● } |
| 1011 | { ○, ◇, ● } |
| 1100 | { ⋆, ● } |
| 1101 | { ○, ⋆, ● } |
| 1110 | { ◇, ⋆, ● } |
| 1111 | { ○, ◇, ⋆, ● } |

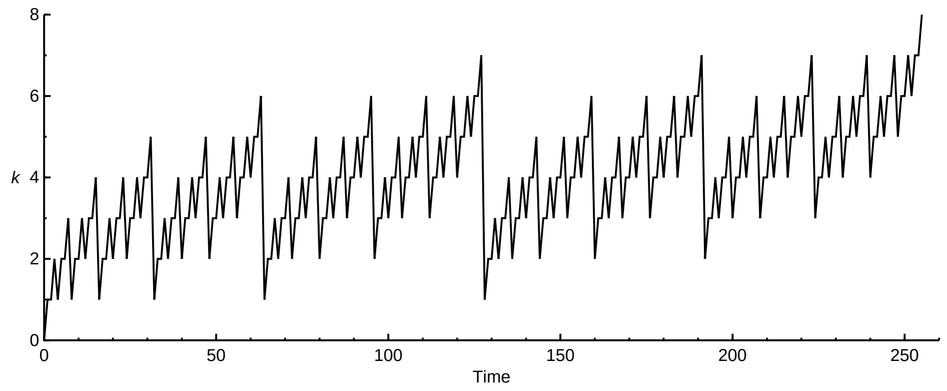

Figure 1: Lexicographic ordering amounts simply to counting in binary.

## 2.2 Gray Codes

Gray codes are a rearrangement of the binary numbers such that adjacent values differ in exactly one bit position[1]. There are many such sequences; the one shown here is called a *binary reflected Gray code*. A Gray code sequence is computationally optimal in the sense that it minimizes the number of set operations required. Any subset in $\mathcal{P}(S)$ can be constructed from its immediate predecessor in the sequence merely by adding or removing one item. But Table 2 shows that the Gray code sequence is no better than lexicographic ordering when it comes to constructing a reasonable sequence of subsets. (Figure 2 shows this behavior geometrically.) As in lexicographic ordering, the $n$th element in the set is not even considered until $2^{n-1}$ subsets have already been constructed. The performance of this method for large values of $n$ is also poor.

Table 2: Subsets in Gray code order. Just as was the case with a lexicographic sequence, the last item in the set, represented by the most significant bit of the binary representation, is not even considered until fully half of all possible subsets have been examined.

| *Binary Number* | *Subset of* $\{\circ, \diamond, \star, \bullet\}$ |
|---|---|
| 0000 | $\phi$ |
| 0001 | $\{\circ\}$ |
| 0011 | $\{\circ, \diamond\}$ |
| 0010 | $\{\diamond\}$ |
| 0110 | $\{\diamond, \star\}$ |
| 0111 | $\{\circ, \diamond, \star\}$ |
| 0101 | $\{\circ, \star\}$ |
| 0100 | $\{\star\}$ |
| 1100 | $\{\star, \bullet\}$ |
| 1110 | $\{\diamond, \star, \bullet\}$ |
| 1111 | $\{\circ, \diamond, \star, \bullet\}$ |
| 1101 | $\{\circ, \star, \bullet\}$ |
| 1001 | $\{\circ, \bullet\}$ |
| 1011 | $\{\circ, \diamond, \bullet\}$ |
| 1010 | $\{\diamond, \bullet\}$ |
| 1000 | $\{\bullet\}$ |

## 2.3 Banker's Sequence

If $n$ is large, then systematically constructing all $2^n$ subsets of $S$ can take a while. If the problem is to find the smallest subset $T_{\min} \subseteq S$ that satisfies some

[1]One way of constructing an $n$-bit Gray code is to find a Hamiltonian path through adjacent vertices of an $n$-dimensional hypercube with sides of length 1.

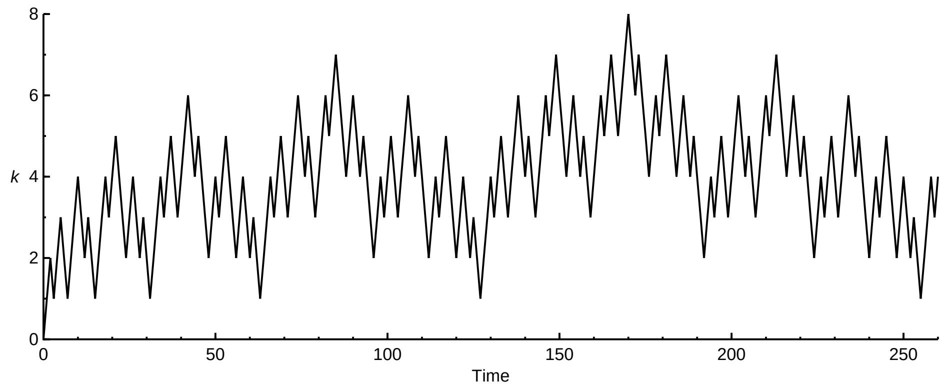

Figure 2: Gray codes are computationally optimal, but they share the weakness of lexicographic ordering that not all members of the set are examined until the sequence is half finished.

predicate $P$, then we would like to examine subsets in monotonically increasing order by size.

Consider the ordering presented in Table 3, in which we examine the subsets of $S$ in order by the number of elements in each subset. Ideally we would like to examine all $k$-subsets before considering any subsets of size $k+1$ or larger. So we first construct all the subsets consisting of a single element. Then we examine all 2-subsets, all 3-subsets, and so on, until $k \to n$ and we have constructed all $2^n$ subsets.[2]

The new sequence, being a permutation of the numbers between 0 and $2^n - 1$, depends on $n$; for $n = 6$, one such sequence (in decimal) is:

$$\begin{aligned} B_6 = \{ & 0, 32, 16, 8, 4, 2, 1, \\ & 48, 40, 36, 34, 33, 24, 20, 18, 17, 12, 10, 9, 6, 5, 3, \\ & 56, 52, 50, 49, 44, 42, 41, 38, 37, 35, 28, 26, 25, 22, 21, 19, 14, 13, 11, 7, \\ & 60, 58, 57, 54, 53, 51, 46, 45, 43, 39, 30, 29, 27, 23, 15, \\ & 62, 61, 59, 55, 47, 31, 63\}. \end{aligned} \tag{1}$$

No reference to this sequence was found in the literature [1, 3]. We refer to this ordering as a Banker's sequence, because it was discovered while looking for a solution to the problem of matching "adjustments" in a bank's books at the end of the day. Figure 3 illustrates how the sequence proceeds systematically through increasingly larger subsets of $S$.

[2] or until we find a solution, or until we run out of time.

Table 3: Banker's sequence. Every element of the set is examined within the first $n$ iterations. Correspondingly larger subsets are constructed in monotonically increasing order of size.

| *Binary Number* | *Subset of* $\{\circ, \diamond, \star, \bullet\}$ |
|---|---|
| 0000 | $\phi$ |
| 1000 | $\{\circ\}$ |
| 0100 | $\{\diamond\}$ |
| 0010 | $\{\star\}$ |
| 0001 | $\{\bullet\}$ |
| 1100 | $\{\circ, \diamond\}$ |
| 1010 | $\{\circ, \star\}$ |
| 1001 | $\{\circ, \bullet\}$ |
| 0110 | $\{\diamond, \star\}$ |
| 0101 | $\{\diamond, \bullet\}$ |
| 0011 | $\{\star, \bullet\}$ |
| 1110 | $\{\circ, \diamond, \star\}$ |
| 1101 | $\{\circ, \diamond, \bullet\}$ |
| 1011 | $\{\circ, \star, \bullet\}$ |
| 0111 | $\{\diamond, \star, \bullet\}$ |
| 1111 | $\{\circ, \diamond, \star, \bullet\}$ |

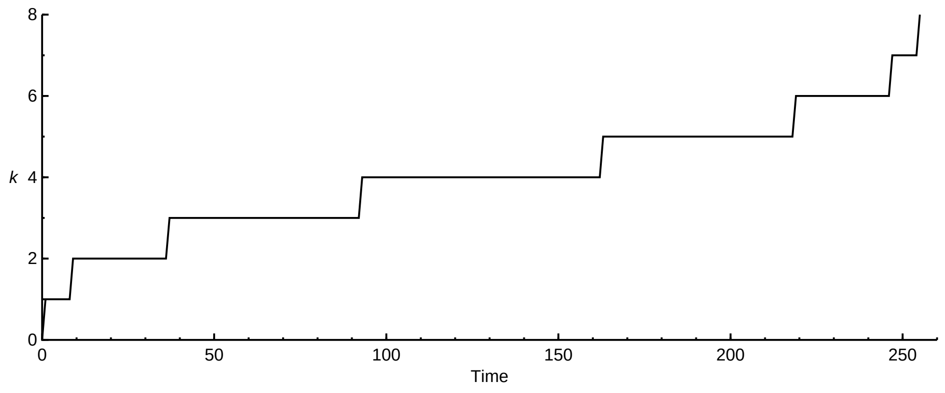

Figure 3: The banker's sequence looks at the smaller subsets first.

## 3 Efficiently Enumerating Subsets

Neither lexicographic ordering nor Gray codes are particularly well suited to looking for a *minimum* subset. Both sequences will eventually enumerate $\mathcal{P}(S)$, but they tend to waste a lot of time up front constructing large subsets. For $n = 100$, there are $2^{100}$ subsets, about $10^{30}$. Assuming a computer capable of checking $10^8$ subsets per second, a Gray code solution would require about $10^{22}$ seconds, or about $4 \times 10^{14}$ years to complete. The running time of a lexicographic ordering is about twice that. The problem is computationally intractable without an intelligent method of ordering.

Table 4 shows the number of set operations (bit transitions) required to completely search a smaller set of $n = 30$ elements. The running time of the banker's sequence is comparable to that of a lexicographic ordering. However, the banker's sequence is guaranteed to find the smallest subset of up to $k \leq 29$ items before either of the other two orderings have even finished examining all $n = 30$ items in the set.

Table 4: Comparison of the running times of three different algorithms for a set of $n = 30$ elements.

| *Method of Ordering* | *Total Bit Transitions* |
|---|---|
| Gray Code | 1,073,741,824 |
| Lexicographic | 2,147,483,617 |
| Banker's Sequence | 2,863,311,486 |

## 4 Generating a Banker's Sequence

There are many ways to generate a banker's-type sequence. The following algorithm, which generates a Banker's sequence for a set of $n$ items, implements it in one possible way.

```
#include <iostream.h>
#include <stdlib.h>

void output(int string[], int position);
void generate(int string[], int position, int positions);

int length;

// This function takes "string", which contains a description
// of what bits should be set in the output, and writes the
// corresponding binary representation to the terminal.
// The variable "position" is the effective last position.
```

```
void
output(int string[], int position)
{
  int * temp_string = new int[length];
  int index = 0;
  int i;

  for (i = 0; i < length; i++)
  {
    if ((index < position) && (string[index] == i))
    {
      temp_string[i] = 1;
      index++;
    }
    else
      temp_string[i] = 0;
  }

  for (i = 0; i < length; i++)
    cout << temp_string[i];

  delete [] temp_string;
  cout << endl;
}

// Recursively generate the banker's sequence.

void
generate(int string[], int position, int positions)
{
  if (position < positions)
  {
    if (position == 0)
    {
      for (int i = 0; i < length; i++)
      {
        string[position] = i;
        generate(string, position + 1, positions);
      }
    }
    else
    {
      for (int i = string[position - 1] + 1; i < length; i++)
      {
         string[position] = i;
         generate(string, position + 1, positions);
      }
    }
  }
  else
    output(string, positions);
}

// Main program accepts one parameter: the number of elements
// in the set.  It loops over the allowed number of ones, from
// zero to n.

int
main (int argc, char ** argv)
{
  if (argc != 2)
  {
    cout << "Usage: " << argv[0] << " n" << endl;
    exit(1);
  }
  length = atoi(argv[1]);

  for (int i = 0; i <= length; i++)
  {
    int * string = new int[length];
    generate(string, 0, i);
    delete [] string;
  }
  return (0);
}
```

### 4.1 How the Algorithm Works

The algorithm works by... (*need table from Jano for explanation*).

Interestingly, the sequence generated by Algorithm 1, expressed in binary, is bitwise symmetric about the middle, sort of like a binary reflected Gray code.

## 5 Conclusions and Future Work

We presented a method for generating a sequence of subsets in which the number of elements in each subset increases monotonically. There is no other known method of enumerating subsets that generates a monotonic sequence. The method we presented is useful for finding the smallest subset $T_{\min} \subseteq S$ that satisfies some condition $P$. It will find a solution much faster than either lexicographic or Gray code ordering, because it avoids generating reams of large subsets until it has examined all of the smaller possible subsets first.

An algorithm was presented for efficiently generating sequences of subsets in monotonically increasing order by size. The technique is applicable to the solution of Constraint Satisfaction Problems (CSP).

In the future it would be nice to have a direct mapping from the natural numbers $\mathcal{N}$ such that if $a_{\in\mathcal{N}} < b_{\in\mathcal{N}}$ then the number of 1's in the binary representation of $a$ is less than the number of ones in the binary representation of $b$.

## References

[1] Donald E. Knuth. *The Art of Computer Programming.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1973.

[2] F. Ruskey. Information on subsets of a set. `http://sue.csc.uvic.ca/~cos/inf/comb/Subset-Info.html`, 1999.

[3] N.J.A. Sloane. The on-line encyclopedia of integer sequences. `http://www.research.att.com/~njas/sequences/`, 1999.

[4] E.W. Weisstein. CRC concise encyclopedia of mathematics. `http://www.astro.virginia.edu/~eww6n/math/Subset.html`, 1999.