

A recursive algorithm for forming the constrained elemental cardinality Power Set

Mads Dyrholm
Center for Visual Cognition
University of Copenhagen
Denmark

September 30, 2009

Abstract

A Power Set with elemental cardinality constraint is the largest subset, of a Power Set, in which all elements satisfy a cardinality constraint. Sets with an equality constraint on the elemental cardinality can be formed directly, as opposed to filtering an unconstrained Power Set, hence avoiding a significant computational overhead.

1 Introduction

Given a set \mathcal{S} , the Power Set of \mathcal{S} , denoted $\mathcal{P}(\mathcal{S})$, is the set of all possible subsets of \mathcal{S} . For example,

$$\mathcal{S} = \{A, B, C\} \quad \Leftrightarrow \quad \mathcal{P}(\mathcal{S}) = \{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}.$$

The notion of a Power Set can be used to express convoluted combinatorics in a very elegant way. For instance, in the probabilistic modeling work of [Kyllingsbæk, 2001, Kyllingsbæk, 2006], multiple integrals over products is made analytically tractable by substituting sums over Power Sets. However, in that particular work, an elemental cardinality constraint is enforced in order to make an elegant presentation.

Let $\mathcal{P}_j(\mathcal{S})$ denote the Power Set with elemental cardinality- j constraint, that is $\mathcal{P}_j(\mathcal{S})$ is the largest subset of $\mathcal{P}(\mathcal{S})$ in which all elements (each being a subset of \mathcal{S}) has cardinality equal to j . Clearly, $\mathcal{P}_j(\mathcal{S})$ can be obtained by forming $\mathcal{P}(\mathcal{S})$ and then filtering it, but since $\mathcal{P}(\mathcal{S})$ can be enormous compared to $\mathcal{P}_j(\mathcal{S})$, such filtering can demand a significant overhead. As a result, a principled way of establishing $\mathcal{P}_j(\mathcal{S})$ has been desired. This paper describes an algorithm for doing so.

2 The algorithm

An expansion of $\mathcal{P}(\mathcal{S})$ is typically made by declaring a counter variable with as many bits available as there are elements in \mathcal{S} . The counter is then incremented as a binary word from all bits zero, representing the empty set, to all bits one, representing the inclusion of all elements of \mathcal{S} .

The present algorithm for $\mathcal{P}_j(\mathcal{S})$ can be described in a similar manner; using a counter variable, however, this variable instead of being binary, has always exactly j bits enabled. The counter is

initialized by letting j ones be shifted all the way to the left and the remaining bits set to zero. First note that $\mathcal{P}_j(\mathcal{S})$ can be formed by shifting the j bits around without them ever crossing. Hence after initialization, only the rightmost bit can be shifted. For each shift of this bit towards the word boundary, a new possible position is open for the next bit to the left of it. This argument then again applies to the next bit down the line yielding a recursive principle — starting with the leftmost bit as the “slowest” level of recursion, and the depth of recursion is j .

The algorithm is defined in the following pseudocode, where, instead of actually moving bits around, the elements of an array representing \mathcal{S} are copied directly to a subset array which is then passed to a user defined code once for each element of $\mathcal{P}_j(\mathcal{S})$. In the following algorithm definition, S is the array which represents the set \mathcal{S} . Freedom is left to the implementer to define S more precisely — it could for example be an array of pointers to elements of \mathcal{S} , or an array of integers indexing the elements of \mathcal{S} .

Algorithm 1 for forming $\mathcal{P}_j(\mathcal{S})$

Require: $j \geq 0$; an array S s.t. $\text{length}(S) \leq j$; a temporary array E of size j .

Ensure: User defined code will be executed for each element of $\mathcal{E} \in \mathcal{P}_j(\mathcal{S})$.

```

1: POWERSETJ( $S, j$ )                                     ▷ Start the algorithm

2: procedure POWERSETJ( $U, s$ )
3:   if  $s > 0$  then                                     ▷ Recur deeper?
4:      $l \leftarrow \text{length}(U)$ 
5:     for  $i \leftarrow 1, l - s + 1$  do
6:        $E(s) \leftarrow U(i)$ 
7:       POWERSETJ( $U(1 + i : l), s - 1$ )                 ▷ Recursion
8:     end for
9:   else
10:    User defined handling of  $E$  representing  $\mathcal{E} \in \mathcal{P}_j(\mathcal{S})$ .
11:   end if
12: end procedure

```

3 Example

The companion implementation of the algorithm is written in the `Matlab` language. As an example, the following command will form $\mathcal{P}_j(\mathcal{S})$ for $\mathcal{S} = \{1, 2, 3, 4\}$ and $j = 2$

```

>> powersetj(1:4,2);
     2     1
     3     1
     4     1
     3     2
     4     2
     4     3

```

4 Matlab source

The following listing shows a Matlab implementation of the algorithm.

```

function E=powersetj(U,s,E,j)
% POWERSETJ
%
% Synopsis
% =====
%
% powersetj(S,j)
%
% -- Author: Mads Dyrholm --
% Center for Visual Cognition, University of Copenhagen.
% August 2009
%
% Purpose
% =====
%
% Run through Pj(S).
%
% Arguments
% =====
%
% S - Array for which the Pj(S) should be expanded.
%
% j - Scalar integer defining the elementary
% cardinality constraint of the Power Set.

if nargin==2
    j = s;
    E = zeros(1,j);
    powersetj(U,s,E,j);
    return
end

if s>0
    for i=1:length(U)-s+1
        E(s) = U(i);
        E=powersetj(U(1+i:end),s-1,E,j);
    end
else
    if length(E)==0, disp( 'Empty set ' ); end
    disp(E);
end

```

References

- [Kyllingsbæk, 2001] Kyllingsbæk, S. (2001). Item based fitting of whole report data. Available from <http://www.psy.ku.dk/tva>.
- [Kyllingsbæk, 2006] Kyllingsbæk, S. (2006). Modeling visual attention. *Behavior Research Methods*, 38:123–133.