Reply of M. Verhelst

It is correct that optimal flowcharts can only be derived from decision tables showing the complete set of 2^m theoretically possible decision rules (where m is the number of conditions in the table), together with the frequencies of each such rule (some of these frequencies being eventually 0) and also the time to test each condition. Such an algorithm has been derived by Reinwald and Soland [3].

As I remarked in my paper [1, p. 974], the concept of optimization which I used is somewhat different. Indeed my algorithms start from a decision table, which will be usually found in practice, that is, where some of the rules have already been combined by using dashes, and where the impossible rules are absent. In such a case, there are two possibilities: either we expand the table to a complete one, look for the additional information, and apply the Reinwald and Soland algorithm; or we take the table as it stands and aim at optimization, using whatever information is available. I have chosen the latter alternative, because to my feeling, the computational work in applying the Reinwald and Soland algorithms soon becomes prohibitive for practical decision tables with a relatively large number of conditions. This is, however, not to say that the work of Reinwald and Soland should not be considered as very valuable and useful.

Even so, it is true that the lower bound S proposed in my paper is not valid in all cases. I realized this some time ago, and have corrected it in a later publication which unfortunately appeared in Dutch [6]. There I stated (p. 42) that "the lower bound S is only valid on condition that, in combining simple rules to complex ones, no starred answers have disappeared in the process." This is exactly what happened in the example provided above by King and Johnson. In such cases, which can easily be detected by expanding the dashes in the table, I suggested considering p and q as being each 50 percent of the originally combined frequency (since the option is not to look for further information) and to choose to calculate S from the original table or from the decomposed one depending upon which one shows the lowest S. The table showing the lowest S would then be used as an input for applying the algorithms.

For all these reasons, it should be preferred to rebaptize my "optimum-finding" algorithm as a second "optimum-approaching" one, which can readily be expected to perform better than the other optimum-approaching algorithm presented in the same paper, but which as a penalty will require more computation time for deriving the flowchart.

As I stated in the conclusion to my paper, the additional computation effort needed will usually not be justified by the small reduction of execution time obtained. —M. Verhelst

Short Communications
Programming Techniques

A Numbering System for Combinations

Gary D. Knott
National Institutes of Health

Key Words and Phrases: combinatorics, coding system, storage mapping function
CR Categories: 4.9, 5.30

It is often necessary to generate the sequence of combinations of s things chosen from among n things, or to generate a random member of this sequence. Various algorithms have been given for these tasks. Algorithm 94 [1] is a good combination generator, and methods for generating a random combination are discussed in [2, pp. 121–125].

Another facility which can be useful upon occasion is to be able to represent a combination by an integer, i.e. to have a combination numbering system. This involves defining a function, r, such that for any combination, p, considered as a vector, $r(p)$ is a unique integer corresponding to p. r must be invertable so that p can be obtained from $r(p)$. Moreover it is desirable that the range of r be a segment of consecutive integers and that r preserve lexicographic ordering.

Such a function allows a combination to be represented as a single integer; this saving of space and collapsing of structure is sometimes convenient. It also can be used to generate all combinations or a random combination. This frugality of methodology must be balanced, however, against the fact that both derived schemes are inferior to the methods referenced above. We present a function and its inverse below which realizes the combinational numbering system just described.

Let $p = p_1 \cdots p_s$ be a combination chosen from $\{0, 1, \ldots, n-1\}$ such that $p_1 < p_2 < \cdots < p_s$. Define

$$v_{ns}(p) = \sum_{1 \le i \le s} \binom{p_i}{i}.$$

If the reverse of p is lexicographically less than the reverse of q, where both p and q are s-combinations

Author's address: Building 12A, DCRT, National Institutes of Health, Bethesda, MD 20014.

from $\{0, 1, \ldots, n-1\}$, then $v_{ns}(p) < v_{ns}(q)$. Moreover $v_{ns}(01 \cdots (s-1)) = 0$ and $v_{ns}((n-s) \cdots (n-1)) = \binom{n}{s} - 1$, so v_{ns} maps the $\binom{n}{s}$ possible combinations onto $0, 1, \cdots, \binom{n}{s}-1$, preserving reverse lexicographic order.

Let $c_n(p)$ be the combination $q_1 \cdots q_s$ of $\{0, 1, \ldots, n-1\}$ defined by: $q_i = n - 1 - p_{s+1-i}$. $c_n(p)$ is called the complement of p.

Now v_{ns}^{-1} can be defined recursively as follows: $v_{ns}^{-1}(k) = p_1 \cdots p_s$, where $p_i = d(k - \sum_{i<j\leq s} \binom{p_j}{j}, i)$ and $d(x, i)$ is computed by: for $j \leftarrow n - 1$ step -1 until $i - 1$ do if $x \geq \binom{j}{i}$ then return (j).

Define rank(p) as the ordinal position of p in the sequence of lexicographically-ordered combinations of s elements from $\{0, 1, \ldots, n-1\}$, so $1 \leq \text{rank}(p) \leq \binom{n}{s}$. Then rank $(p) = \binom{n}{s} - v_{ns}(c_n(p))$, also, rank$^{-1}(k) = c_n(v_{ns}^{-1}(\binom{n}{s} - k))$. For example, let $n = 5$ and $s = 3$. Then we have:

p	$c_n(p)$	$v_{ns}(p)$	$rank(p)$
012	234	0	1
013	134	1	2
014	034	4	3
023	124	2	4
024	024	5	5
034	014	7	6
123	123	3	7
124	023	6	8
134	013	8	9
234	012	9	10

The function v_{ns} appears in [3, ex. 56 of sect. 1.2.6]. The work by Brown [4] may seem to be applicable here, but in reality, it solves the related problem of a numbering system for the permutations of the combinations of s elements chosen from n elements. Now we may present programs for computing rank and rank^{-1}.

The following nonlocal entities are assumed to exist and to be initialized as indicated.

integer n: all combinations are chosen from $\{0, 1, \ldots, n-1\}$.
integer s: all combinations have s components.
integer array $e[0:n]$: e is used as a vector of "logarithms" in computing binomial coefficients. $e[i]$ is an exponent of i.
integer array $f[0:n]$: $f[i]$ is the greatest prime factor of i ($f[0]=1$). f is used in computing binomial coefficients.
integer array $r[0:n]$: $r[i] = i/f[i]$. r is used in computing binomial coefficients.

The arrays, f and r, may be initialized with the following procedure.

```
procedure initialfandr;
begin integer i, j;
for i := 0 step 1 until n do f[i] := 1;
for i := 2 step 1 until n do begin
  if f[i]=1 then for j := i step i until n do f[j] := i;
  r[i] := i÷f[i] end
end initialfandr
```

The following procedures are the required code for implementing rank and rank^{-1}.

```
integer procedure evaluate;
begin integer i,u,v;
u := 1; v := 1;
if e[0]≠0 then evaluate := 0 else
begin for i := n step -1 until 2 do
    if i≠f[i] then begin e[f[i]] := e[f[i]] + e[i];
                         e[r[i]] := e[r[i]] + e[i];
                         e[i] := 0
                   end else
    if e[i]>0 then u := u × f[i] ↑ e[i]
              else v := v × f[i] ↑ -e[i];
  evaluate := u÷v
end
end evaluate;

integer procedure rank(p); integer array p;
begin integer sum, i,j,k;
sum := 0;
for i := 0 step 1 until n do e[i] := 0;
for i := step 1 until s do begin
  e[i] := e[i]-1;
  k := n-1-p[s-i+1];
  for j := k-i+1 step 1 until k do e[j] := e[j]+1;
  sum := sum + evaluate;
  for j := k-i+1 step 1 until k do e[j] := e[j]-1
                          end;
for i := n-s+1 step 1 until n do e[i] := e[i]+1;
rank := evaluate - sum
end rank;
procedure rankinverse (p,k); integer array p; integer k; value k;
begin integer i,j,m;
j := if s ≤ n-s then s else (n-s);
for i := 1 step 1 until j do e[i] := -1;
for i := 0, j+1 step 1 until n-j do e[i] := 0;
for i := n-j+1 step 1 until n do e[i] := 1;
k:= evaluate-k: e[n-s] := e[n-s]+1; e[n] := e[n]-1;
m := evaluate; j := n-1;
for i := s step -1 until 1 do begin
  while k<m do begin m := m×(i-i)÷j; j := j-1 end;
  k := k-m; p[s-i+1] := n-1-j; m := m×i÷(if j=0 then 1
  else j); j := j-1
                          end
end rankinverse
```

The logarithmic method used in computing binomial coefficients was suggested in [5]. It is possible that an alternate approach, based upon the additive recursion equation for binomial coefficients, would be faster and perhaps require less space. The direct use of logarithms in floating point may also be feasible when sufficient accuracy is available.

Received January 1973

References

1. Kurtzberg, Jerome. Algorithm 94. *Comm. ACM 5*, 6 (June 1962), 344.
2. Knuth, Donald E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., 1969.
3. Knuth, Donald E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison Wesley, Reading, Mass., 1968.
4. Brown, Richard M. Decoding combinations of the first n integers taken k at a time. *Comm. ACM 3*, 4 (Apr. 1960), 235–236.
5. McKay, J.K.S. On the evaluation of multiplicative combinatorial expressions. Letter to the editor. *Comm ACM 11*, 6 (June 1968), 392.