Journal of Modern Applied Statistical Methods

Volume 1 | Issue 2 Article 59

11-1-2002

A Program for Generating All Permutations of {1, 2, ..., n}

Robert DiSario
Bryant College

Recommended Citation
DiSario, Robert (2002) "A Program for Generating All Permutations of {1, 2, ..., n}," *Journal of Modern Applied Statistical Methods*: Vol. 1 : Iss. 2 , Article 59.
DOI: 10.22237/jmasm/1036109640

Journal of Modern Applied Statistical Methods
Fall 2002, Vol. 1, No 2, 518-522

1538 – 9472/02/$30.00

A Program for Generating All Permutations of $\{1, 2, \ldots, n\}$

Robert DiSario
Bryant College

A Visual Basic program that generates all permutations of {1, 2, …, n} is presented. The procedure for running the program as an Excel macro is described. An application is presented which involves selecting permutations which meet a specific constraint.

Key words: Visual Basic, permutation.

Introduction

A Visual Basic program for generating all combinations of *n* elements taken *m* at a time was presented in Stamatopoulos (2002). The present work presents a program for generating all permutations of *n* elements. Applications involving combinations and permutations often arise in designing experiments and in other areas. As an example, the program was used to find all permutations that meet a specific requirement.

The procedure given in the present work meets the requirements stated in Stamatopoulos (2002) for algorithms which implement automatic enumeration: a) all possible cases are exhausted; b) none of the permutations need to be stored – the current case that has been formulated is the basis for generating the next one. Therefore it presents a practical means for generating permutations.

Methodology

The program consists of a main module, Macro1( ), and 3 functions: Permute( ), Findlarg( ) and Sort( ). The main module handles input and output (input from Excel ; output to a text file), dimensions and initializes an array, and calls Permute( ). Findlarg( ) returns the largest element to the right of a given position in an array. Sort( ) sorts the elements to the right of a given position in an array. Permute( ) takes as input a permutation of {1,2,…,n} and creates the next permutation in the "natural sequence". For an example of the "natural sequence" of permutations of {1,2,3,4} see the output below. Permute( ) also returns 0 when the final permutation in the natural sequence has been created. A general description of Permute( ) follows. A listing of the program, written in Visual Basic, appears in an appendix.

Robert DiSario is an Assistant Professor in the department of mathematics at Bryant College. He received a Ph.D. in statistics from Boston University in 1996. His academic interests include applied statistics and combinatorics. E-mail him at rdisario@bryant.edu

Description of *Permute( )* function

Permute(x(),n)
Set *bigfix* = *n*.
Note: *bigfix* is an element that serves as a reference point in the array.
Top:
Find position of *bigfix* (call it *bigindx*). Check whether array is in descending order from *bigindx* to the right. If descending, work left. Else, work right.
Work left: (refers to left of *bigfix*)
If nothing to left of *bigfix*, then done (this is the last permutation in natural sequence).
Else the element to the left of bigfix, x(bigindx-1), needs to be changed. Switch it with the smallest element on its right

```
which is bigger than it. Then sort
the elements from bigindx to the
right.
Permute( ) is done (indicated by
done = 1).
end Work left

Work right: (refers to right of
bigfix)
Find the largest element on the
right of bigfix. Set bigfix equal
to this largest element.
Permute( ) is not done (indicated
by done = 0)
end Work right

Return to top:
```

Results

Application 1

As a first example, the program was used to generate all 24 permutations of the set {1,2,3,4}. The results are shown in Table 1. This output reveals the order referred to above as the "natural sequence". Note that the output file contains a single column of permutations, but that Table 1 has been reformatted into 6 columns to save space.

Application 2

As a typical application, experimenters are often interested in the order of presentation of experimental conditions or stimuli. In some cases, the orders used must be selected according to very specific considerations. Furthermore, the experimenter may desire to use a different order for each of the subjects or replications. As an example, suppose an experimenter wants a list of all the permutations of {1,2,3,4,5} in which "1" is not next to "2", "2" is not next to "3", "3" is not next to "4", and "4" is not next to "5". The program was modified (as described below) to check each permutation to determine whether or not it meets this constraint. The list of all such permutations appears in Table 2.

Table 1. "Natural Sequence" of Permutations of {1,2,3,4}. Read down then across.

1 2 3 4	1 4 2 3	2 3 1 4	3 1 2 4	3 4 1 2	4 2 1 3
1 2 4 3	1 4 3 2	2 3 4 1	3 1 4 2	3 4 2 1	4 2 3 1
1 3 2 4	2 1 3 4	2 4 1 3	3 2 1 4	4 1 2 3	4 3 1 2
1 3 4 2	2 1 4 3	2 4 3 1	3 2 4 1	4 1 3 2	4 3 2 1

Table 2. All permutations of {1,2,3,4,5} with the property that adjacent elements are not consecutive integers.

1 3 5 2 4	2 4 1 3 5	2 5 3 1 4	3 1 5 2 4	3 5 2 4 1	4 2 5 1 3	5 2 4 1 3
1 4 2 5 3	2 4 1 5 3	3 1 4 2 5	3 5 1 4 2	4 1 3 5 2	4 2 5 3 1	5 3 1 4 2

To select only those permutations that meet the constraint, the section of the program that prints the permutation was modified. First the permutation was checked to see if it satisfies the constraint. Then printing was conditional on the outcome of this check. This was accomplished by setting a "satisfy" flag to 0 if the constraint was not met and to 1 if the constraint was met. The specific lines that were changed (both original and modified) are presented in Appendix III. A similar approach could be used to select permutations according to other constraints.

References

Stamatopoulos, C. (2002). Generation of combinations using *Excel*. *Journal of Modern Applied Statistical Methods*, 1, 191-194.

Appendix I

The BASIC code that appears in Appendix II can be run as an Excel macro. The procedure for doing this is described in Stamatopoulos (2002). Note that before pasting the program lines into the Visual Basic editor, it is necessary to first delete two lines which are automatically generated by Excel: *Sub Macro1( )* and *End Sub*.

The program can be assigned to a control key. It will read a value of *n* from the cell *B4* in *Sheet1* of the *Excel* workbook. It outputs the permutations to a text file called *perms.txt*.

Appendix II

Program listing

```
Sub Macro1()
'Open file for output.
'Read n from worksheet
'Set initial permutation {1,2,...,n}
Open "c:\perms.txt" For Output As #1
n = Range("B4")
ReDim x(n)
For i = 1 To n
 x(i) = i
Next i

'Notdun=0 iff current permutation is n, n-1, ..., 1
notdun = 1
Do While (notdun)
 For i = 1 To n
'Print current permutation
  Print #1, x(i);
 Next i
'Print line feed
 Print #1, ""
'Find next permutation and note whether it is the final one
 notdun = permute(x(), n)
Loop
Close
End Sub

Function permute(x(), n)
'Creates the next permutation in the "natural sequence"
'Returns 0 if permutation is n, n-1, ..., 1
'Default is to return 1
permute = 1
bigfix = n
'Done = 1 indicates next permutation is complete, 0 not.
done = 0
Do While (done = 0)
 done = 1

'Find the index of bigfix
 For i = 1 To n
  If x(i) = bigfix Then bigindx = i
 Next i
 descend = 1
 If bigindx <> n Then
 For i = bigindx To n - 1
  If x(i) < x(i + 1) Then descend = 0
```

```
 Next i
 End If
 If descend And bigindx = 1 Then permute = 0
 If descend Then
'Work left
  current = x(bigindx - 1)
  candidx = bigindx
'Find element to switch with x(bigindx-1)
  For i = bigindx To n
   If x(i) > current And x(i) < x(candidx) Then candidx = i
  Next i
'Switch them
  temp = x(candidx)
  x(candidx) = x(bigindx - 1)
  x(bigindx - 1) = temp
  temp = sort(x(), bigindx)
 End If
'End of work left

'Work right
 If descend = 0 Then
  done = 0
  bigfix = findlarg(x(), bigindx + 1)
 End If
'End of work right
Loop
End Function

Function findlarg(x(), start)
'Finds largest x(i) from i = start to i = n
candid = x(start)
ub = UBound(x)
For i = start To ub
 If x(i) > candid Then candid = x(i)
Next i
findlarg = candid
End Function

Function sort(x(), start)
'Sorts x() from i = start to i = n
ub = UBound(x)
For i = start To ub
For j = i To ub
If x(i) > x(j) Then
temp = x(i)
x(i) = x(j)
x(j) = temp
End If
Next j
Next i

End Function
```

Appendix III

Program modification used to select permutations meeting constraint described in application 2.

Original code:

```
For i = 1 To n
'Print current permutation
  Print #1, x(i);
 Next i
'Print line feed
 Print #1, ""
```

Modified code:

```
'Check whether permutation meets constraints
satisfy = 1
For i = 2 To n
If Abs(x(i) - x(i - 1)) = 1 Then satisfy = 0
Next i

If satisfy Then
 For i = 1 To n
' print current permutation
  Print #1, x(i);
 Next i
' print line feed
 Print #1, ""
End If
```