Exceedingly Simple Permutations and Combinations

Jan de Leeuw

Version 002, Februari 23, 2016

Contents

Note: This is a working paper which will be expanded/updated frequently. The directory [deleeuwpdx.net/pubfolders/nextPC](http://deleeuwpdx.net/pubfolders/nextpc has a pdf copy of this article, the complete Rmd file with all code chunks, and the R and C source code.

1 Problem

Most permutation and combination function in the various `R` packages generate a list or matrix with all permutations of order n or all choices of m from n objects. If n is large the list or matrix will be large and to fill it will take a lot of time. Usually you will need the permutations or combinations for some higher purpose, and you just sit there twiddling your thumbs while the matrix or list is constructed, or while your `R` memory is filled to the point of exploding.

In this note we give simple functions, using the `.C` interface, to compute the next permutation or combination in the lexicographic order. The functions take a vector argument and return

a vector. You could store the returned values in a matrix or list, but for some applications it makes more sense to use and discard them. That obviously saves memory. For an application of the permutation function, see De Leeuw (2005), for an application of the combination function see De Leeuw, Groenen, and Mair (2016).

To fill a matrix or list we would have to use these functions in an `R` loop, so they probably are not too efficient in terms of time. We sacrifice time to gain memory. On the other hand, using the `nextPermutation()` and `nextCombination()` functions will be easy to parallelelize.

2 Examples

```
nextPermutation (1:3)
```

```
## [1] 1 3 2
```

```
nextPermutation (c(1,3,2))
```

```
## [1] 2 1 3
```

```
nextPermutation (nextPermutation (c(1,3,2)))
```

```
## [1] 2 3 1
```

```
nextPermutation (3:1)
```

```
## NULL
```

```
nextCombination(1:3,5)
```

```
## [1] 1 2 4
```

```
nextCombination(c(1,4,5), 5)
```

```
## [1] 2 3 4
```

```
nextCombination(nextCombination(c(1,4,5), 5), 5)
```

```
## [1] 2 3 5
```

```
nextCombination(3:5,5)
```

```
## NULL
```

3 Code

3.1 C

```
void
swap(int *x, int i, int j)
{
  int           temp;
  temp = x[i];
  x[i] = x[j];
  x[j] = temp;
}


void
nextPermutation (int *x, int *nn)
{
  int             i, j, n = *nn;
  i = n - 1;
  while (x[i - 1] >= x[i])
    i--;
  if (i == 0)
    return;
  j = n;
  while (x[j - 1] <= x[i - 1])
    j--;
  swap(x, i - 1, j - 1);
  j = n;
  i++;
  while (i < j) {
    swap(x, i - 1, j - 1);
    j--;
    i++;
  }
}

void nextCombination (int* n, int* m, int* next) {
```

```
  int i, j, mm = *m - 1, nn = *n;
  for (i = mm; i  >= 0; i--) {
    if (next[i] != nn - mm + i) {
      next[i]++;
      if (i < mm) {
        for (j = i + 1; j <= mm; j++)
          next[j] = next[j - 1] + 1;
      }
      return;
    }
  }
}
```

3.2 R

```
dyn.load("nextPC.so")

nextPermutation <- function (x) {
  if (all (x == (length(x):1))) return (NULL)
  z <- .C("nextPermutation", as.integer(x), as.integer(length(x)))
  return (z[[1]])
}

nextCombination <- function (x, n) {
  m <- length (x)
  if (all (x == ((n - m) + 1:m))) return (NULL)
  z <- .C("nextCombination", as.integer(n), as.integer (m), as.integer(x))
  return (z[[3]])
}
```

4 NEWS

001 02/22/16

- First release

002 02/23/16

- Added some verbiage
- Added references to MDS applications

References

De Leeuw, J. 2005. "Unidimensional Scaling." In *The Encyclopedia of Statistics in Behavioral Science*, edited by B. S. Everitt and D. C, 4:2095–97. New York, N.Y.: Wiley.

De Leeuw, J., P. Groenen, and P. Mair. 2016. "More on Inverse Multidimensional Scaling." 2016.