Shuffle: a program to randomize lists with optional sequential constraints

Christophe Pallier
LSCP, EHESS-CNRS, 54 bd Raspail, 75006 Paris, France

Abstract

This paper describes 'shuffle', a program that allows to randomize lists of items, with optional restrictions on successive items. It is useful, for example, to extract random samples from a dictionary, or to generate quasi-randomized lists of stimuli in which there are no long sequences of similar trials. 'Shuffle' reads lines from a text file and prints them (or a subset of them) in a random order. Restrictions on the random permutation can be enforced in the following way: each line is spliced into columns and one can specify maximum number successive lines with the same labels in given columns. 'Shuffle' is programmed in Awk, a scripting language which, we argue in this paper, is a valuable instrument in the psychologist's toolbox.

Introduction

Preparing an experiment often requires to extract random samples of items from lists and to randomize the order of such items. For example, when the experimental design allows it, it is good practice to present the stimuli in different random orders for each subject. The aim is to diminish the importance of potential between-trials effects which may be specific to a given list.

One potential between-trials effect may arise when a list contains long sequences of trials with similar characteristics. For example, successive trials may contain stimuli with similar features, or map systematically on the same response. It is known that participants are quite sensitive to such local repetitions in speeded responses tasks (Luce, 1986).

It is therefore often desirable to avoid long repetitions in random lists. The program we present here, named 'shuffle', allows to randomly permutate items in a list while avoiding such repetitions. (We are proposing 'shuffle' here as an stand-alone program, but it has been actually implemented as a command of the Expe experiment generator for many years (Pallier, Dupoux, & Jeannin, 1997b, 1997a).)

E-mail: pallier@lscp.ehess.fr. Web: http://www.ehess.fr/centres/lscp/persons/pallier/

Algorithm

Here is the pseudo-code for the generation of an unconstrained random permutation:

```
...read line[1] to line[N]
for i := N downto 2 do swap( line[i], line[random(i)] )
//  remark: 1 <= random(i) <= i
```

With this algorithm, all permutations of the lines are equiprobable (each has the probability $1/n!$; the demonstration is left to the reader.[1])

A constrained permutation is generated from a random unconstrained permutation very much in the way a human would do it by hand: it checks, from the first line downward, if all constraints are fulfilled. Every time a line is found that violates a constraint, the algorithm searches systematically for an acceptable line further down the list. If such a line is found, the algorithm swaps it with the line that violated the constraint. If no acceptable line can be found, the current permutation is abandoned and a new one is tried. This algorithm is not guaranteed to converge: By default, 'shuffle' gives up after it has tried as many permutations as there are lines in the input file; Yet the maximum number of trials can be specified by the user.

Requirements, Availability and Use

'shuffle' is written in the Awk programming language, a standard Unix tool (Aho, Kernighan, & Weinberger, 1988; Doughertyby, Robbins, & Estabrook, 1997; Robbins, 1997). We recommend the use of 'gawk', the GNU version of the Awk, under which we developed and tested shuffle. gawk is available for free, and for many platforms, including GNU/Linux www.gnu.org and MS DOS (http://www.delorie.com). A nice feature of gawk is that there is no hard coded limit on the number of input lines nor on the size of lines of the input file (the only limit is the virtual memory of your computer: we routinely use 'shuffle' to extract random sample from a 8 Mb dictionary).

Shuffle is distributed under the terms of the GNU General Public License 2 (GPL, 1991). It can be freely downloaded from the software page on the author's web site[2]. The installation is pretty straightforward as Awk is an interpreted language: shuffle's source code is contained in a single, human-readable file, which is interpreted by gawk. The package also contains a two-pages manual in text, postscript and troff formats (the latest is for Unix' `man` command). Unix users just need to copy 'shuffle' in a directory of the path to further have access to the 'shuffle' command.

To MS-DOS users, the instructions recommend to copy the files `gawk.exe` and `shuffle` in a directory, say, `c:\awk`, and to define shuffle as a macro, by adding to the file `c:\autoexec.bat` the following line:

```
doskey shuffle=c:\awk\gawk.exe -f c:\awk\shuffle $*
```

[1] Tip: you can use the recurrence principle.

[2] http://www.ehess.fr/centres/lscp/persons/pallier. This page is declared in altavista: if for any reason the page has moved, a web search on altavista with the keywords 'Christophe Pallier shuffle' should yield the new location.

Figure 1. Listing 1: a few examples of shuffle's use

```
shuffle -?               # to display a short help
shuffle sample.txt
shuffle n=10 sample.txt  # limits output to 10 lines
shuffle n=10 sample.txt  # new permutation of 10 lines
shuffle n=5 seed=134 sample.txt # sets the seed of the random generator
shuffle n=5 seed=134 sample.txt # you get the same permutation as before...
shuffle n=8 constr='2' sample.txt # no more than 2 succesive lines with
                                  # the same value in column 1
shuffle n=8 constr='0 3' sample.txt # no more than 3 succesive lines with
                                    # the same value in column 2
shuffle n=8 constr='2 2' sample.txt # no more than 2 succesive lines with
                                   # the same values in column 1 or column 2
shuffle n=8 constr='1 1' sample.txt # not much room for randomness
```

After rebooting, the command 'shuffle' is available at the DOS prompt. This is a better solution than calling 'gawk -f shuffle' in a batch, because DOS batches fail with redirections.

Listing 1 shows a few examples of the use of 'shuffle'. Note that in these examples, 'shuffle' just displays its result. To save it in a file, one has to use the redirection operator '>': Supposing that you want to shuffle the file '`mylist.txt`' and put the result in, say, '`newlist.txt`', you would type '`shuffle mylist.txt >newlist.txt`'.

The choice of Awk

We would like to justify our choice of Awk to implement shuffle. It would not be difficult to write a graphical version of 'shuffle' in Visual Basic (which would allow to run it from within Word) or in Java (which would allow it to be run on a web page). This would make it even simpler to use for some users (but would prevent the use of shuffle in batch scripts). However, in our experience the Awk language is perhaps the most useful and efficient language for the tasks routinely performed by psychologists, from searching dictionaries or small databases, constructing lists of stimuli, or analyzing data.[3]

Awk provides powerful constructions for manipulating a wide variety of data in reasonably efficient ways, especially thanks to its regular expressions and its associative arrays. It is has a simple syntax, is interpreted, and has a relatively short learning curve (people with rudiments of programming can learn in it matter of a few hours). Many tasks that our students spent hours trying to do with spreadsheets or database programs, such as Excel and Access, can typically be performed much more quickly with Awk. Awk has no graphics capabilities by itself, but with the conjoint use of gnuplot, another free, multi-platform, program[4], it can replace spreadsheet program such as Excel for exploratory data analysis.

[3]See also (Loui, 1996): " Most people are surprised when I tell them what language we use in our undergraduate AI programming class. That's understandable. We use GAWK. [...] Their surprise turns to puzzlement when I confide that (a) while the students are allowed to use any language they want; (b) with a single exception, the best work consistently results from those working in GAWK."

[4]Gnuplot can be downloaded at `http://www.cs.dartmouth.edu/gnuplot_info.html`

One big advantage of Awk is that one can check how a student has produced some results by inspecting the Awk scripts she/he had to write. In the 70s and 80s, psychology students used to learn and use BASIC to perform many small tasks. We contend that Awk could advantageously replace BASIC today, and that it can be a valuable tool in the psychologist's toolbox [5]

We will illustrate this on two simple examples. While 'shuffle' enforces local constraints, one also often want to make sure that items in different conditions have similar distributions in experimental lists. The Awk script shown in listing 2 prints the means and the standard deviations of the ranks of the items in the different conditions (the condition is supposed here to be given, in the list of items, as a label in the fist column). Note that one may write a script calling repeatedly shuffle and the script in listing 2 until the mean ranks of the different conditions do not differ more than by a prespecified number.

Figure 2. Listing 2: awk script to compute the mean and standard dev. of the ranks of different conditions

```
{ line++
  n[$1]++
  sumranks[$1] += line
  ssranks[$1] += line*line
}
END{
 print "condition mean_rank stdev_rank";
 for (i in n)
  printf "%8s %8.1f %8.1f\n",i,sumranks[i]/n[i], \
         sqrt(ssranks[i]/n[i]-(sumranks[i]*sumranks[i])/(n[i]*n[i]))
}
```

The second example shows how to compute the token frequency of words in a text file. This code, given in listing 3, illustrates the power of associative arrays (arrays with strings of characters, rather than just numbers, as index).

Figure 3. Listing 3: awk script to compute the token frequency of all words in a text file (ignoring punctuation)

```
BEGIN { RS="[\n\t '{}\(),.:;\"]+" } # word separators
{ count[$0]++ }
END{ for (token in count) print token,count[token]; }
```

References

Aho, A. V., Kernighan, B. W., & Weinberger, P. J. (1988). *The Awk programming language.* Addison-Wesley Pub. Co.

Doughertyby, D., Robbins, A., & Estabrook, G. (1997). *Sed & Awk.* O'Reilly & Associates.

[5] Over the last 8 years, we have managed to convert almost all our colleagues to using Awk.

GPL. (1991). *Gnu general public license.* (Version 2. Published by the Free Software Fondation, `http://www.fsf.org`)

Loui, R. P. (1996). *Why GAWK for AI.* (Draft for ACM SIGPLAN Patterns (Language Trends), included in gawk-3.0.3 distribution)

Luce, R. D. (1986). *Response times: Their role in inferring elementary mental organization.* New York: Oxford University Press.

Pallier, C., Dupoux, E., & Jeannin, X. (1997a). Expe: an expandable programming language for on-line psychological experiments. *Behavior Research Methods, Instruments, & Computers*, *29*(3), 322-327.

Pallier, C., Dupoux, E., & Jeannin, X. (1997b). *Expe 6 reference manual* (Tech. Rep.). available at http://www.ehess.fr/centres/lscp/expe/expe.html.

Robbins, A. D. (1997). *Effective AWK programming* (1.0.3 ed.). Free Software Foundation, Boston, MA. (available in the gawk-3.0.3 distribution from www.gnu.org)

Appendix: shuffle source code

```
#!/bin/gawk -f
#
# shuffle
#
# Shuffle performs a random permutation on the lines from the standard input,
# with possible restrictions.
#
# Version 1.0
# Author: Christophe Pallier (pallier@lscp.ehess.fr)
# Date: 13 Mar. 1999
#
# This program is copyrighted under the terms of the GNU
# license (see the file COPYING and http://www.gnu.org).
#
# input variables: constr='c1 c2 ... ' seed=xxx n=yyy iter=uuu

BEGIN {
  srand();  # set random seed to current time (may be overidden by 'seed'
  STDERR = "/dev/stderr";
  if (ARGV[1]=="-?" || ARGV[1]=="-h") { usage(); exit 1; }
  if (ARGV[1]=="-l") { license(); exit 1; }
}

# stores input in a table
{ nlines++; table[nlines]=$0; }

END{
  split(constr,constraint);
  if (seed) srand(seed);

# here starts the real work...

  loop = 0;
  MaxLoops=iter?iter:nlines;
  output_nlines=n?n:nlines;

  do {

    permute(table,nlines);
    bad_permut = 0;  # the permutation is not yet bad...
    loop++;

# try to respect constraints, swapping lines when necessary

    nf=split(table[1],prev);
    for (k=1;k<=nf;k++) rep[k]=1; # rep[k]=number of repetitions in column [k]

    for (i=2;(!bad_permut)&&(i<=output_nlines);i++) {
      if (DEBUG) {
```

```
50          printf ("Search line %d:",i);
51          for (o=1;o<=nlines;o++) printf "%s ",table[o];
52        }
53        pass_line=0;
54        for (j=i;(!pass_line)&&(j<=nlines);j++) {
55          if (DEBUG) printf (" l%d ",j);
56          nf=split(table[j],current);
57          fail=0;
58          for (k=1;(!fail)&&(k<=nf);k++) {
59            if ((constraint[k]) && (prev[k]==current[k]) && (rep[k]+1>constraint[k])) {
60  # constraint[k] is violated
61              fail=1;
62              if (DEBUG) printf "@"> STDERR;
63            }
64          } # next k
65          if (fail==0) pass_line=1; # bingo !
66        } # next j
67  # now, if pass_line==1, line j fulfills the constraints
68        if (pass_line) {
69          j--;
70          if (DEBUG) { printf "accept line %d\n",j; }
71          if (j>i) swap(table,i,j);
72          for (k=1;k<=nf;k++) {
73            if (prev[k]==current[k]) rep[k]++; else rep[k]=1;
74            prev[k]=current[k];
75          }
76        }
77        else {
78  # it is not possible to rearrange this permutation;
79  # let's try another one.
80          bad_permut = 1;
81          if (DEBUG) printf "#\n" > STDERR;
82        }
83      } # next i
84
85    } while ( (bad_permut) && (loop<MaxLoops))
86
87
88      if (bad_permut) print "shuffle: could not find a permutation after" loop "iterations" > ST
89      else {
90        if (DEBUG) printf "Converged in %d loops\n",loop > STDERR;
91        for (i=1;i<=output_nlines;i++) print table[i];
92      }
93  }
94
95
96  function swap(array,i,j)  # exchange array[i] and array[j]
97  {
98    if (i!=j) {
99      temp=array[i];
100     array[i]=array[j];
```

```
101       array[j]=temp;
102     }
103   }
104
105   function permute(array,size) # random permutation of array's elements
106   {
107     for (i=size;i>1;i--) swap(array,i,1+int(rand()*i));
108   }
109
110   function usage()
111   {
112
113     usage_str = "" \
114     "Shuffle performs a random permutation on the lines from the standard input,\n" \
115     "with possible restrictions.\n" \
116     "Usage:\n   shuffle [constr='n1 n2...'] [n=<num>] [seed=<num>] [iter=<num>]\n" \
117     "Options:\n" \
118     " constr='n1 n2...' : set constraints on maximum number of repetitions in columns 1, 2 ...\n
119     " n=<num>     : outputs 'n' lines\n"\
120     " seed=<num>  : sets the seed for the random number generator to number <num>\n"\
121     " -h|? : shows this help\n" \
122     " -l   : displays the license\n\n"\
123     "shuffle reads the lines from the standard input or from a file given\n"\
124     "as argument, and outputs them in a random order, with possible restrictions:\n"\
125     "\nIf constr is not specified on the command line, all permutations are\n"\
126     "equiprobable. Otherwise, constr is a string of numbers separated by\n"\
127     "spaces, where the ith number specifies the maximum\n"\
128     "number of repetitions in ith column of the output. All the\n"\
129     "permutations fullfilling the constraints are equiprobable.\n\n"\
130     "Author: Christophe Pallier (pallier@lscp.ehess.fr).\n"\
131     "Date: 13 March 1999\n" \
132     "This program can be used and distributed under the GNU License (shuffle -l)";
133     print usage_str;
134   }
135
136   function license() {
137       lic=" Shuffle - performs a random permutation on the lines from the standard input,\n" \
138       " with possible restrictions.\n" \
139       " Copyright (C) 1999 Christophe Pallier (pallier@lscp.ehess.fr)\n\n"\
140       " This program is free software; you can redistribute it and/or\n"\
141       " modify it under the terms of the GNU General Public License\n"\
142       " as published by the Free Software Foundation; either version 2\n"\
143       " of the License, or (at your option) any later version.\n"\
144       "\n This program is distributed in the hope that it will be useful,\n"\
145       " but WITHOUT ANY WARRANTY; without even the implied warranty of\n"\
146       " MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n"\
147       " GNU General Public License for more details.\n"\
148       "\n You should have received a copy of the GNU General Public License along\n"\
149       " with this program; if not, write to the Free Software Foundation, Inc.,\n"\
150       " 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA. (www.gnu.org)";
151       print lic;
```

```
152         exit 1;
153     }
```