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 to randomize lists of items, with the option of setting constraints 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 a set of lines and reprints them (or a subset of them) in a random order. It is possible to set constraints on the output so that no more than a certain number of lines with similar labels are repeated, or, on the contrary, so that a minimum number of lines separates two lines with the same labels.

Introduction

Experimental psychologists routinely need to create randomized lists. A first example is the selection of the material for psycholinguistics experiments, which typically requires extracting random samples of words from dictionaries. Another example is that of creating different randomizations of trials' order for each participant in an experiment.

Presenting the stimuli in different random orders to each participant aims to diminish the impact of list-specific potential between-trials interactions. Some between-trials interactions arise when series of trials contains long sequences with similar characteristics; For example, successive trials may contain stimuli that have similar features, or map systematically onto the same response. It is known that response times of participants are sensitive to such local repetitions (?).

This paper presents `shuffle`, a program designed to extract random samples and to 'quasi-randomize' the order of items in lists, avoiding long runs of similar items.

Installing shuffle

The "shuffle" package can be freely downloaded from the software page on the author's web site: http://www.pallier.org. It consists of two programs: `shuffle` and `shuffle.tk`. The first is meant to be used from a command line, while the other provides a graphical interface.

E-mail: `pallier@lscp.ehess.fr`. Web: `http://www.pallier.org`

Both programs are written in Perl (cf. http://www.perl.com, (?)). Therefore, before using them, an interpreter for the language Perl must be installed on your computer (cf. http://www.activestate.com for the Windows version). Moreover the Tk module for Perl must be installed to use the graphical version (?).

One of the advantages of having shuffle written Perl, is that it can run under Linux, MacOSX, Windows (cf. http://www.activestate.com)... The other is that, if necessary, one can modify the source code to tune it to ones' needs. "shuffle" is distributed under the terms of the GNU General Public License 2 (?).

Once Perl is setup, to install the "shuffle" package, you need to:

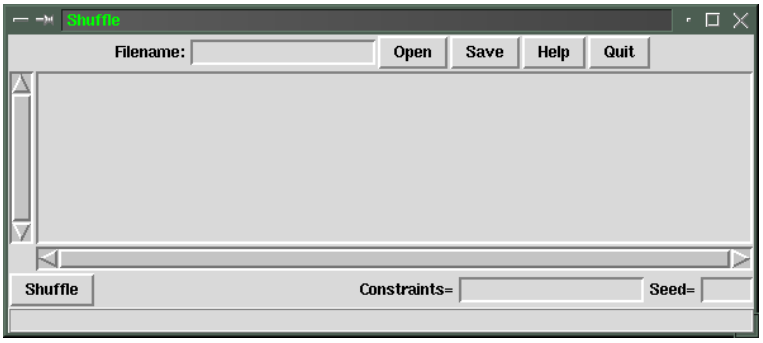
1. copy `shuffle` and `shuffle.tk` in a directory listed in the PATH variable (e.g. `/bin` under Linux, `c://windows` under Windows.).

2. adapt, if necessary, the first line of each program, to reflect to actual location of perl on your system. The original files assume that the perl program is located at `/usr/bin/perl`. You may have to change this to, for example, `/usr/local/bin/perl` (To locate perl, under unix/linux, just type 'which perl' on a command line).

Using 'shuffle'

`shuffle` can either be used as a command-line utility, or as program with a graphical interface. New users will probably prefer the version with the graphical interface, but the command line version has additional options, no limitation on the size of the input and embodies the unix philosophy of designing small, portable and efficient tools that can easily be used as building blocks in more complex programs. It will therefore appeal more to advanced users.

Typing '`shuffle.tk`' on a command line launches the graphical version.



The center of the window features a text area where the original list can be entered, with one item per line. The list can be edited directly, or pasted from the clipboard. Alternatively, it can be read from a text file by typing first the filename in the box entitled 'Filename' and then pressing the 'Open' button.[1]

[1]The user should be aware that there is a limitation on the size of the text that can be put in a text area in a graphical interface. This limitation varies from one operating system to the other, so no general limit can be given, but we should be cautious when pasting more than, e.g. 32000 characters. If in doubt, use the command-line version.

Pressing the 'shuffle' button shuffles the lines in the text area in a random order. The "seed" box can be used to enter a number which determine the permutation that will be used. This is useful if one want to apply the same permutation to several lists.

If the "constraints" box is empty, all permutations are equiprobable (the formal demonstration is given below). To avoid certain permutations however, it is possible to write a series of numbers in this box. This option is meaningful only when the input text is in tabular format, that is when each line contains the same number of items separated by spaces. The items in some or all of the columns may be considered as "labels" corresponding to levels of experimental factors. It is then possible to reject permutations in which the labels are repeated, across successive lines, more than a fixed number of times. For example, if one column contains the labels 'WORD' or 'PSEUDO' to indicate the level of the factor 'LEXICALITY', one may want to avoid permutations in which, say, more than four lines in row contain the same label (that is belong to the same level of the factor LEXICALITY).

The i^{th} number in the constraint box corresponds to the i^{th} column in the input. When set to '0', there is no constraints in the relevant column. When set to a positive number, it means the maximum number of repetitions (across successive lines) in the relevant column. When set to a negative number, e.g. $-n$, it means that a least n lines must separate two occurrences of the same label.

The latter option is useful to create lists where a small of targets appear in the middle of a larger number of filler items: on can create a column containing either the label 'TARGET' or 'FILLERxxx' where 'xxx' varies for each filler. If the constraint '-10' is associated to this columns, targets will be separated by at least 10 fillers.

The command line utility will be described briefly, supposing the input list is in a text file named 'corpus.txt' in the current working directory. The command:

```
shuffle corpus.txt
```

prints the lines from `corpus.txt` in a random order. In order to save this result in a text file, one must use the redirection operator '`>`': the following command:

```
shuffle corpus.txt >neworder.txt
```

produces a random permutation of the lines of `corpus.txt` and stores it in the file `neworder.txt`.

Often, one does not need a complete permutation, but rather only a random sample from the original list. The variable 'n', passed as argument to `shuffle`, permits to specify the maximum number of output lines:

```
shuffle n=10 corpus.txt
```

yields 10 lines picked at random from `corpus.txt`. The variable `seed` can also be set on the command line. This can be useful to apply the same permutation to different files:

```
shuffle seed=123 corpus.txt
shuffle seed=123 corpus2.txt
```

Finally, to specify constraints on the repetition of labels in columns, one must pass the variable 'constr' on the command line:

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 successive lines with
                                   # the same values in column 1 or column 2
shuffle n=8 constr='1 1' sample.txt # not much room for randomness
```

```
shuffle constr='0 0 4 -6' corpus.txt
```

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 '>'.

Proof of equiprobability

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:  random(i) return a number in the interval [1;i]
```

We are going to use the recurrence principle to demonstrate that this algorithm makes all permutations of the lines equiprobable (that is, each has the probability $1/n!$). It helps to note that the previous code is the iterative equivalent of the following recursive algorithm:

```
function perm(n) {
 swap(line[n],line[random(n)])
 if n>1 then perm(n-1)
}
```

For a permutation of size 2: line[2] is swapped with either line[1] or line[2], each case having probability 1/2. Therefore the two permutations, (1,2) and (2,1), are equiprobable.

Let us suppose that the algorithm generates equiprobable permutations up until rank $n-1$ (that is, any given line has probability $1/n-1$ to occur in any given output rank). To generate a permutation of rank n, the algorithm first swaps line(n) randomly with one of the lines between 1 and n (therefore each has the same probability, $1/n$, to occur in output rank n), and then applies itself recursively to the $n-1$ first elements. By hypothesis, all n-1 elements have equal probability to be assigned any given output rank between 1 and n-1. Finally, a given line (with rank between 1 and n) has probability 1/n to be assigned to output rank n and probability $(n-1)/n \times 1/(n-1) = 1/n$ to be assigned to any given output rank between 1 and $n-1$.

Algorithm for constrained permutations

A constrained permutation is generated very much in the way a human would do it by hand. First an unconstrained permutation of the input lines is generated; then the algorithm 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: the constrains may be to strong, that is, impossible to fullfill. For example, suppose that the input list is composed of 2/3 of items of type 1, and 1/3 of items of type 2; if you require that successive lines always be of different types, then the algorithm will fail. Yet, if half of the items in the input are of type 1 and half of type 2, then the algorithm will always find one of the two solutions (permutations that alternate between types 1 and 2.)

Conclusion

I wrote shuffle for my own needs. I distribute it in the hope that it can be useful to others. I do not garantee that it works as intended. I would appreciate any bug reports, about the programs or the documentation.

References

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

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

Schwartz, R. L., & Phoenix, T. (2001). *Learning perl.* O'Reilly.

Walsh, N. (1999). *Learning Perl/Tk.* O'Reilly.

Appendix: shuffle source code

```
#!/usr/bin/perl
eval 'exec /usr/bin/perl -S $0 ${1+"$@"}'
    if $running_under_some_shell;
                        # this emulates #! processing on NIH machines.
                        # (remove #! line above if indigestible)

eval '$'.$1.'$2;' while $ARGV[0] =~ /^([A-Za-z_0-9]+=)(.*)/ && shift;
                        # process any FOO=bar switches

# shuffle
#
# Shuffle performs a random permutation on the lines from the standard input,
# with possible constraints on successive lines.
#
# Version 1.0 (perl)
# Author: Christophe Pallier (pallier@lscp.ehess.fr)
# Date: 13 Mar. 1999 (awk) 20 July 2000 (perl)
#
# 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

$[ = 1;                 # set array base to 1
$, = ' ';               # set output field separator
$\ = "\n";              # set output record separator

$ExitValue = 0;

#####################################
# process command line options

if ($ARGV[1] eq '-?' || $ARGV[1] eq '-h') {
    &usage();
    $ExitValue = 1; last line;
}

if ($ARGV[1] eq '-l') {
    &license();
    $ExitValue = 1; last line;
}

@constraint = split(' ', $constr);

# set random seed
```

```
50  if ($seed) {
51      srand($seed);
52  } else {
53   #  srand($_);
54  }
55
56  $ignore_empty_lines=1;
57
58  ################################################
59  # reads the input lines and store them in 'table'
60
61  while (<>) {
62    chop;
63    if ($ignore_empty_lines) { $table[++$nlines] = $_ unless ($_ eq ""); }
64    else  { $table[++$nlines] = $_; }
65  }
66
67  $n = ($n eq "")?$nlines:$n;
68  $iter = ($iter eq "")?$n:$iter;
69
70  &shuffle(*table,$constr,$n,$iter) || die "Could not converge";
71
72  for ($i = 1; $i <= $n; $i++) {
73          print $table[$i];
74  }
75
76  exit $ExitValue;
77
78
79  ############################ subroutine shuffle ###############
80  sub shuffle {
81    local(*table,$constr,$n,$iter)=@_;
82  # return 0 if no solution, 1 else.
83  #  print @table;
84
85
86  $loop = 0;
87  $MaxLoops = $iter ? $iter : $nlines;
88  $output_nlines = $n ? $n : $nlines;
89
90  do {
91      &permute(*table, $nlines);
92      $bad_permut = 0;    # will be set to 1 if the permutation is bad...
93      $loop++;
94
95      # scan the table line by line
96      # swapping lines to try and respect the constraints
97
98      $nf = (@prev = split(' ', $table[1]));
99      for ($k = 1; $k <= $nf; $k++) {
100         $rep[$k] = 1;   # rep[k]=number of repetitions in column [k]
```

```
101     }
102     for ($i = 2; (!$bad_permut) and ($i <= $output_nlines); $i++) {
103         $pass_line = 0;
104         for ($j = $i; (!$pass_line) and ($j <= $nlines); $j++) {
105             $nf = (@current = split(' ', $table[$j]));
106             $fail = 0;
107             for ($k = 1; (!$fail) and ($k <= $nf); $k++) {
108 
109                 if ($constraint[$k]>0) {
110                     if (($prev[$k] eq $current[$k]) and
111                         ($rep[$k] + 1 > $constraint[$k])) {
112                         $fail = 1;
113                     }
114                 }
115 
116                 if ($constraint[$k]<0) {
117                     $back=$i-(-$constraint[$k]);
118                     if ($back<1) { $back=1; };
119                     for ($ll = $back; ($ll < $i) and (!$fail);$ll++) {
120                         @tmp=split(' ',$table[$ll]);
121                         if ($tmp[$k] eq $current[$k]) {
122                             $fail=1;
123                         }
124                     }
125                 }
126 
127             } # next k
128             if ($fail == 0) {
129                 $pass_line = 1;
130             }
131         } # next j
132         # now, if pass_line==1, line j fulfills the constraints
133         if ($pass_line) {
134             $j--;
135             if ($j > $i) {
136                 &swap(*table, $i, $j);
137             }
138             for ($k = 1; $k <= $nf; $k++) {
139                 if ($prev[$k] eq $current[$k]) {
140                     $rep[$k]++;
141                 }
142                 else {
143                     $rep[$k] = 1;
144                 }
145                 $prev[$k] = $current[$k];
146             }
147         }
148         else {
149             # it is not possible to rearrange this permutation;
150             # let's try another one.
151             $bad_permut = 1;
```

```
152             }
153         }
154         # next i
155     } while ( $bad_permut & ($loop < $MaxLoops));
156
157     if ($bad_permut) { 0 }
158     else { 1 }
159
160     }
161
162     sub swap {
163         local(*table,$i,$j)= @_;
164     # exchanges the ith and jht element from table
165         if ($i != $j) {
166             $temp = $table[$i];
167             $table[$i] = $table[$j];
168             $table[$j] = $temp;
169         }
170     }
171
172     sub permute {
173         local(*array, $size) = @_;
174     # random permutation of array's elements
175         for ($i = $size; $i > 1; $i--) {
176             &swap(*array, $i, 1 + int(rand(1) * $i));
177         }
178     }
179
180     sub usage {
181         $usage_str = '' .
182           "Shuffle performs a random permutation on the lines from the standard input,\n"
183           . "with possible restrictions.\n" .
184           "Usage:\n   shuffle [constr='n1 n2...'] [n=<num>] [seed=<num>] [iter=<num>]\n"
185           . "Options:\n" .
186           " constr='n1 n2...' : set constraints on maximum number of repetitions in columns 1, 2 ...\
187           . " n=<num>     : outputs 'n' lines\n" .
188           " seed=<num>  : sets the seed for the random number generator to number <num>\n"
189           . " -h|? : shows this help\n" . " -l   : displays the license\n\n" .
190           "shuffle reads the lines from the standard input or from a file given\n"
191           . "as argument, and outputs them in a random order, with possible restrictions:\n"
192           . "\nIf constr is not specified on the command line, all permutations are\n"
193           . "equiprobable. Otherwise, constr is a string of numbers separated by\n"
194           . "spaces, where the ith number specifies the maximum\n" .
195           "number of repetitions in ith column of the output. All the\n" .
196           "permutations fullfilling the constraints are equiprobable.\n\n" .
197           "Author: Christophe Pallier (pallier\@lscp.ehess.fr).\n" .
198           "Date: 13 March 1999\n" .
199           'This program can be used and distributed under the GNU License (shuffle -l)';
200         print $usage_str;
201     }
202     sub license {
```

```
203       $lic =
204         " Shuffle - performs a random permutation on the lines from the standard input,\n"
205         . " with possible restrictions.\n" .
206         " Copyright (C) 1999 Christophe Pallier (pallier\@lscp.ehess.fr)\n\n" .
207         " This program is free software; you can redistribute it and/or\n" .
208         " modify it under the terms of the GNU General Public License\n" .
209         " as published by the Free Software Foundation; either version 2\n" .
210         " of the License, or (at your option) any later version.\n" .
211         "\n This program is distributed in the hope that it will be useful,\n" .
212         " but WITHOUT ANY WARRANTY; without even the implied warranty of\n" .
213         " MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n" .
214         " GNU General Public License for more details.\n" .
215         "\n You should have received a copy of the GNU General Public License along\n"
216         . " with this program; if not, write to the Free Software Foundation, Inc.,\n"
217         . ' 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA. (www.gnu.org)';
218       print $lic;
219       $ExitValue = 1; last line;
220   }
221
222
```