Behavior Research Methods
2006, 38 (4), 584-589

Mix, a program for pseudorandomization

MAARTEN VAN CASTEREN and MATTHEW H. DAVIS
MRC Cognition and Brain Sciences Unit, Cambridge, England

In psychological experiments involving multiple trials, the order in which individual trials are presented to participants influences the results obtained. For this reason, experimenters often create carefully constrained experimental lists or check randomly generated lists to avoid known causes of order artifacts (e.g., short-term stimulus or response repetition). Creating appropriately structured pseudorandom lists can be a difficult and time-consuming task in generating psychological experiments. Mix is a Windows program that can generate pseudorandomized orders according to complex, user-specified constraints. Mix can be used to generate a novel item order for each individual participant, even for complex experiments in which stimulus and/or response repetition is an experimental variable of interest, or for which automated randomization would not normally be possible. The program also contains a number of other practical features for generating files for use with a variety of experiment control software. A Mix executable for Windows, a complete manual, and terms of use are available at www.mrc-cbu.cam.ac.uk/personal/maarten.van-casteren/Mix.htm. Use is limited to academic or other nonprofit applications.

The speed or accuracy of responses to items in psychological experiments is influenced by many different factors. The independent variables that are deliberately manipulated by an experimenter are only a subset of the factors that will influence individual responses. All other sources of variability can be regarded as irrelevant noise as far as detecting the effect of an experimentally manipulated variable is concerned. When these extraneous sources of variation are randomly distributed, they enter the error term in many inferential statistics, leading to appropriately conservative significance tests. However, inadvertently linking extraneous variation to independent variables can contribute to false positive or false negative results that undermine scientific accuracy.

Accordingly, successful experiment design requires not only that independent variables be manipulated but also that these independent variables not be contaminated by sources of noise on response measurement. Tools that assist in controlling extraneous sources of noise in experimental psychology are therefore extremely valuable. In this article we introduce one such tool, Mix, a computer program that can be used to generate appropriate, pseudorandom orderings of experimental trials for psychological research. Mix is a Windows command line program, available as a free download from the Web for research and other noncommercial use.

Experimental factors that influence responses can be divided into two distinct sources, those that originate in the nature of experimental items and those that originate in the experimental context in which those items appear. For example, in experimental investigations of visual word recognition, item-related factors might be word frequency (Broadbent, 1967), semantic concreteness (Whaley, 1978), or word length (Weekes, 1997). It is well known that irrelevant item-related factors that could affect responses must be carefully matched in any experiment in which these item factors are not the topic of investigation (see, e.g., Cutler, 1981; Rastle & Davis, 2002). However, despite the fact that randomization of trial order is a cornerstone of experimental psychology, techniques for matching and controlling contextual influences on response time do not generally go beyond the application of random-number generators. In this article, we motivate a computerized solution to the problem of controlling contextual influences on response measurement: a Windows computer program, Mix, that can automatically generate pseudorandomly ordered experimental lists in which a number of different constraints on trial ordering are simultaneously satisfied. In this way, Mix provides experimental control over variance due to context effects in psychological experiments, both for situations in which contextual factors are a source of to-be-excluded noise and for those situations in which contextual factors are to be manipulated. Before describing the Mix program and its operation, we will introduce some of the important contextual factors that can introduce variance into experimental psychological investigations.

Contextual Influences on Responses in Psychological Experiments

That earlier items influence the speed and accuracy of current responses is an extremely common observation in experimental psychology. This is most clearly shown by priming effects in the recognition of words, faces, objects, and other stimuli. In both semantic and repetition prim-

Correspondence concerning this article should be addressed to M. van Casteren, MRC Cognition and Brain Sciences Unit, 15 Chaucer Road, Cambridge CB2 7EF, England (e-mail: maarten.van-casteren@mrc-cbu.cam.ac.uk).

ing, the magnitude of these contextual influences is an important dependent measure (for reviews, see Neely, 1991; Tenpenny, 1995). These contextual influences generally decay rapidly with time; for example, semantic priming is robust at short latencies but is only rarely observed if many items intervene between the prime and target (but see Becker, Moscovitch, Behrmann, & Joordens, 1997). Similarly, many repetition priming studies have shown a rapid decay of priming effects, with increased lag between first and second presentations (see, e.g., Bentin & Moscovitch, 1988), though some repetition priming effects can be extremely long lasting (Cave, 1997).

Testing the effect of priming, or ruling out priming effects as a confound, requires precise experimental control of the placement of specific trials in an experimental list. For instance, if semantic priming is to be assessed, prime and target trials should be adjacent. If semantic priming is to be avoided, there should be trials intervening between pairs of items with related meanings. For repetition priming experiments, the number of trials that intervene between first and second presentations may be an important experimental factor to control or manipulate. In either case, experimental design may well require direct control of the order of the experimental list. Typically, this sort of precise control cannot be achieved with the automated scrambling routines included in most experiment generation software.

There are a number of other influences of experimental context that one might also wish to manipulate or control. Any experimental situation in which different strategies may be involved in generating responses to trials in particular conditions may be prone to contextual influences. Responses will likely be slower when the immediately preceding trial comes from a different condition or requires a different type of decision (similar to the task-switch costs explored by Rogers & Monsell, 1995). In this case, contextual influences may come not only from repeating specific content from a previous trial, as in semantic or repetition priming, but also from priming a particular process or type of response. For example, as far as speeded reading aloud is concerned, a number of authors have shown that reading times for regularly and irregularly spelled words can be influenced by such considerations as whether preceding trials include a pseudoword or large numbers of irregularly spelled words (see Monsell, Patterson, Graham, Hughes, & Milroy, 1992; Zevin & Balota, 2000). Choice response-time tasks may also be affected by the composition of the preceding trial. For example, in the lexical decision task, it has been shown that the frequency and lexicality of the preceding trial have significant influence on response time (Perea & Carreiras, 2003). Similar influences may be observed in other experimental situations. In general, experimenters will wish to ensure that a succession of items from the same experimental condition does not occur in any of their experimental lists. This constraint may be difficult to achieve if random list orders are generated.

One final way in which the serial order of trials can affect the outcome of an experiment is through the correlated structure of residual fluctuations of response measurements in psychological experiments (Gilden, 2001). In simple terms, there is a stronger correlation between response times to items that are closer together in the experiment list than to items placed farther apart. For that reason, any situation in which only a limited number of item orders are used for a psychological experiment creates spurious correlations between items, for no other reason than that they happen to be presented close together in the experimental list. This influence of serial correlation on response times can be avoided only by using multiple lists, each differently ordered.

Given this large number of contextual influences on measured responses, experimental psychologists can be forgiven for trusting their fate to random number generators, and simply randomizing anew the order of items for each volunteer. However, although this approach to randomization prevents any consistent contextual influences, it may create additional noise by failing to prevent the appearance of long runs of trials from the same condition. Furthermore, if trial order is to be manipulated experimentally, as in priming studies, it is likely that optimal presentation orders cannot be created automatically by simple randomization. Researchers must then create pseudorandom trial orders by hand, a difficult and time-consuming task that increases the likelihood of errors being introduced. If generating a single experimental list is a time-consuming process, researchers may often decide to use for their experiment a reduced set of related lists, or even just a single list, a practice that has the unfortunate effect of ensuring that any errors or artifacts in such a single list will be amplified by being presented to several participants.

The only way to solve this complex set of problems is to generate multiple, novel, pseudorandomly ordered lists that each embody the ordering constraints required for a particular experiment. In that way, each participant can receive a novel experimental list that is carefully constructed to measure the effect of the intended independent variable on responses while minimizing the influence of extraneous sources of experimental noise. In this article, we introduce a Windows computer program called Mix that automates this process. In addition to generating the desired order of experimental trials, Mix incorporates a number of other features that allow it to be used to entirely automate the process of producing experimental files. This means that new experiment files can be generated for each individual participant, thereby removing any consistent, undesirable effect of order artifacts.

In describing the function and use of Mix, we first give a general description of the program and the way in which it is used, then we give an overview of all possible order constraints available in it. Then we explain the algorithm used to produce these constrained pseudorandomized orders, and finally we give some general guidelines for using the program.

A Description of Mix

Mix is a Windows utility that operates from two standard ASCII text files created using a text editor such as

Emacs or Notepad, or exported from applications such as Word or Excel. These two text files contain a script file that specifies, with simple commands, the pseudorandomization job that Mix has to perform, and an item file with the list of items to be randomized, along with their relevant properties. Mix is initiated from the Windows command prompt and will output a file (or files) with an item list that meets the constraints specified in the script file. Additional options allow for formatting commands and headers and footers to be added to this file for use with DMDX (Forster & Forster, 2003) or with other similar software for running experiments.

The item file presented to Mix is designed to have one item on each line. Each item represents a single trial in an experiment. Items can have any number of fields containing information, all of which can be used to constrain the pseudorandomizing process.

Mix has to be told which information in the item file is relevant to the randomization. This is specified in the script file in the following three steps.

1. It may first be necessary to distinguish between the different types of items in your experiment, if the experiment contains trials of different types that should be randomized differently or that have a different format in the item file.

2. Next, item properties have to be defined. This can be done for each type of item specifically, or for all items. A property can be any piece of information provided in the item file. In the case of a visual lexical decision experiment (Meyer & Schvaneveldt, 1971), a property of an item could be the fact that it is a word or a nonword. Other properties could be experimental condition, item status (e.g., target, filler, or practice item), item duration, and so on. A property is defined by assigning it to a field in the item file, or to a part of such a field.

3. Finally, constraints can be assigned to properties. By looking at the properties of all items and by checking all constraints that are specified for each of those properties, Mix can now decide if a constraint is violated.

Example

The visual lexical decision task mentioned earlier is a very simple example of how to use Mix. Assume that the items are provided in the following file:

```
table w
roof w
flower w
kloft nw
quelk nw
wug nw
```

The first field is a word item and the second tells you whether the item is a real word or a nonword. A script that randomizes these items but that allows a maximum repetition of three words or nonwords looks like this:

```
Property lexicality 2
Constraint lexicality MaxRep 3
```

The first line tells Mix that the property *lexicality* is to be found on the second field of the items file, and the second line sets a constraint on that property.

Overview of All Possible Constraints

Minimum distance constraint. This ensures that items that are identical on the relevant property are a certain minimum distance apart in the output file(s). This can be used to prevent unwanted semantic or category priming.

Maximum distance constraint. This ensures that the distance between an item and the nearest other item with the same value on its relevant property is smaller than the specified maximum distance. This can be used together with the minimum distance constraint to create a range of distances for repetition priming or for a memory recall task.

Maximum repetition constraint. This ensures that items with an identical value for the specified property are not repeated more than a maximum number of times. For example, it can prevent more than three adjacent occurrences of pseudowords in a lexical decision task.

Minimum repetition constraint. This ensures that items with the same property are put together in groups of a certain minimum size. When the experiment contains several types of trials with different tasks, this constraint can be used to prevent the participants' having to switch from one task to another too frequently.

Pattern constraint. This prevents regular patterns for the relevant property in the output. The user needs to set the window size and the maximum number of times a pattern of that size is allowed to occur, to prevent orders from being too easy to predict. This constraint is especially important when there are many other constraints, or other constraints that are difficult to meet. In these cases, the easiest order for the program to come up with could have a very regular structure. For this reason, it is advisable not to set constraints to unnecessarily strict values. When orders still become too predictable, the pattern constraint can be used.

Order constraint. This forces an item or class of items to always appear before or after another item or class of items. More complicated orders can be achieved by using multiple order constraints. This constraint could ensure, for instance, that prime and target are presented in the desired order in a semantic or repetition priming paradigm.

Numeric constraint. This requires that numeric values for adjacent items be checked, only allowing a certain minimum or maximum difference. Because all other constraints compare item properties as string values, they can only distinguish between identical and nonidentical properties. With this numeric constraint, you can actually prevent adjacent items from having properties that are too close or too different in value.

Enemies or friends. Properties can be defined to apply only to specific items. In this way, individual items can be designated "friends" or "enemies" because specific constraints can be put on two items that should, for example, not be too close together or too far apart, in order to prevent unwanted semantic priming between items that are related in meaning.

Fixing or banning items. Sometimes items have to stay in their initial location—for example, breaks in an experiment or initial fillers after a break. Mix has a feature that allows these items to be fixed in place. Other constraints that apply to these items are still enforced, and items around fixed items are selected in a way that satisfies these constraints. Items can also be banned from certain locations, ensuring, for example, that the first few items after a break are not target items.

Additional Information on Constraints

All the constraints described above can be used in any possible combination as long as they do not contradict one another. Mix tries to find an order that satisfies all constraints, but it fails if it is unable to find such an order after a certain number of attempts.

In most cases, it is possible to put a different value on a constraint for a specific property value. For example, in the case of the visual lexical decision experiment, you might want to set a maximum repetition value of four on the words but of only three on the nonwords.

Items can also be grouped together in blocks, in which case Mix can only reorder them within their block. However, constraints can still be made to work over block boundaries, if necessary.

Any constraint can be made conditional upon the value of another property, meaning that the constraint is checked only when two items have the same value for this conditional property. This makes it possible to set constraints between items that share certain properties.

Algorithm

Mix uses a combination of random shuffling and repair to create the requested output order(s). The list of items is first shuffled in an unconstrained pseudorandom way comparable to other random number generators; then it is processed from start to finish by a repair algorithm. During the execution of this algorithm, a single item is always under examination, as are two parts of the current list of items: the part that comes before the current item (containing the order that has been produced so far), and the part that comes after the current item. We call this last part the *heap*, since none of these items has yet been assigned a definite location in the list (though each has been assigned a randomly generated position).

The algorithm first checks for any violations of constraints for the item at the current location, in the context of the list of already assigned items. If there are no violations, as is likely early in the list, or where the initial random ordering was successful, the algorithm moves on to consider the next item. However, if the item at its current position in the list violates some of the user-specified constraints, the algorithm tries two different forms of repair operation. First, it tries to exchange the current item with one of the items on the heap in order to satisfy the currently violated constraint(s). If a successful repair is found, the current item is replaced by an appropriate item from the heap and the algorithm moves on to the next item. If no item from the heap is suitable (as might be the case toward the end of the list), then as a second step the algorithm tests to determine whether a successful exchange for the current item can be made by using one of the items already assigned to a position earlier in the list. If an appropriate substitution can be found, this earlier item is replaced by a new item from the heap (provided the heap still contains a correct replacement).

If no item that fits all of the constraints for the current location can be found on the list, a final attempt is made to repair the current list by using backtracking. This allows the program to go back and undo an earlier decision and follow a different repair process from that point on. However, if this backtracking still fails to find a list that matches the specified constraints on list ordering, the entire search process is restarted; that is, the whole list is randomly shuffled again and the search is restarted at the first item in the list. This restart procedure is attempted a certain number of times before the program reports a failure to create the requested order (by default, 50 attempts are made, though this is a user-specified parameter).

The algorithm described is reasonably efficient in that, with most problems, it produces a solution in an acceptable amount of time. The disadvantage is that it is not possible to do an exhaustive search with this approach (given the combinatorial explosion of possible list orderings), so a failure to find a solution can never be interpreted as indicating that no solution is possible, only that the number of correct orders is a very small subset of the number of possible orders. That is, even if the algorithm completes the requested number of restarts without finding a correct solution, this does not mean that the set of possible solutions is empty, but means only that the set is small in comparison with the size of the search space.

Other Features

Converting items. To enable researchers to generate complete experiment files with Mix, we added a feature with which the output can be made to fit a certain format. This somewhat resembles the *printf* function in the programming language C, enabling the insertion of fields from the item file into formatted strings. In this way, output files can be produced in a format required by stimulus presentation programs such as DMDX (see, e.g., Forster & Forster, 2003).

Headers and footers. A fixed header and/or footer can be added to the generated order. The content of a specified file is added at the front or back of the output file. This can contain either comments or information needed for the experimental software being used.

Blocks of items. Items can be grouped in blocks, and whether each block needs to be randomized or not can be specified. For each output file, the order of the blocks can be given, enabling counterbalancing of conditions or other item categories. Constraint checking can be set to work either only within blocks or over block boundaries.

Program

The current version of Mix is a Windows command line utility that accepts a script file and an items file, both in

plain text. However, since the program is written in standard C++, it can in principle be compiled on any operating system that provides a modern C++ compiler with a recent version of the Standard Template Library. Versions for operating systems other than Windows are not available yet, but Linux, Solaris, or Macintosh versions may be compiled, if there is sufficient user demand. The program is completely self-contained and does not need any other components to be installed. The source code will not be made available.

Guidelines for Using Mix in Experimental Research

When Mix is used to create orders for experiments, a new separate order should be generated for each participant. Using the same order more than once will introduce extraneous noise. This is very easily avoided with a tool like Mix. Even practice items, dummies, and fillers should be pseudorandomized in the same way as the experimental items.

The script file needed for Mix is an excellent piece of documentation about the way the experiment was created. For this reason, it should be stored with all other files for future reference and should contain as many comments as necessary.

Since Mix makes producing constrained pseudorandomized orders extremely easy, there is a certain risk of overuse. Adding too many constraints, or using constraints with values that are too strict, can lead to very predictable orders. Even a seemingly random order could still contain enough structure to influence the behavior of participants, given the ability of participants to learn repetitive sequences of responses without awareness (Reber, 1989). For example, if a certain class of items is constrained not to appear more than twice in a row, and there are only two types of items, then after two stimuli from the same class the next item is completely predictable. Participants might learn to use this information unconsciously over the course of the experiment and adapt their response strategy after such a sequence. This could even occur without the experimenters' being aware that they have specified such a strictly constrained list; for instance, if different properties of the experimental items are correlated, a constraint on one property can have an effect on the distribution of another property.

To prevent the introduction of predictability into pseudorandom sequences, Mix contains a routine that checks each generated order. Every property that has been defined is checked, whether or not there is a constraint attached to that property. This check tests whether or not the distribution of items after a particular sequence differs from the general distribution. This checking routine first goes through all items to determine the total distribution of the property, then goes through all items again, keeping track of all sequences, from length 1 to a maximum (user specifiable) length. The distribution of the next item(s) is calculated for each sequence and compared with the general distribution, using a chi-square statistical test. To correct for large numbers of statistical tests involved in checking a long experimental list, the list of *p* values that result from these statistical analyses is subjected to a false discovery rate correction for multiple comparisons (Benjamini & Hochberg, 1995). If any sequences lead to a nonrandom distribution of subsequent items, the user is notified and predictable sequences, along with the distributions for the next item, are written to a file for inspection. It is worth noting that Mix contains a special pattern constraint to prevent this problem occurring. This ensures that specific, predictable patterns do not occur with excessive frequency. However, since this constraint is computationally expensive, it is often better to exclude the constraint and to test for predictability after generating a pseudorandom list.

In general, it is recommended that users set constraints on the output order only if there is good reason to do so, and to keep a close eye on the notifications of the predictability routine mentioned above. In addition, it is a good idea to inspect at least one output file produced by Mix for a certain script, to confirm that the output list is indeed as expected.

Conclusion

Mix is a very useful tool that makes generating lists for behavioral experiments quicker and easier. Not only does Mix save time, however; used appropriately, it may also improve the quality of experimental results by preventing common order artifacts and thereby reducing the amount of unexplained variance in the results of behavioral research. We expect, therefore, that Mix could increase the statistical power of many standard experiments, though direct comparisons of empirical data would be required to confirm this.

REFERENCES

Becker, S., Moscovitch, M., Behrmann, M., & Joordens, S. (1997). Long-term semantic priming: A computational account and empirical evidence. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **23**, 1059-1082.

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B*, **57**, 289-300.

Bentin, S., & Moscovitch, M. (1988). The time course of repetition effects for words and unfamiliar faces. *Journal of Experimental Psychology: General*, **117**, 148-160.

Broadbent, D. E. (1967). Word-frequency effect and response bias. *Psychological Review*, **74**, 1-15.

Cave, C. B. (1997). Very long-lasting priming in picture naming. *Psychological Science*, **8**, 322-325.

Cutler, A. (1981). Making up materials is a confounded nuisance, or Will we be able to run any psycholinguistic experiments at all in 1990? *Cognition*, **10**, 65-70.

Forster, K. L., & Forster, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers*, **35**, 116-124.

Gilden, D. L. (2001). Cognitive emissions of 1/*f* noise. *Psychological Review*, **108**, 33-56.

Meyer, D. E., & Schvaneveldt, R. W. (1971). Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *Journal of Experimental Psychology*, **90**, 227-234.

Monsell, S., Patterson, K. E., Graham, A., Hughes, C. H., & Milroy, R. (1992). Lexical and sublexical translation of spelling to sound: Strategic anticipation of lexical status. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **18**, 452-467.

Neely, J. H. (1991). Semantic priming effects in visual word recogni-

tion: A selective review of current findings and theories. In D. Besner & G. W. Humphreys (Eds.), *Basic processes in reading: Visual word recognition* (pp. 264-336). Hillsdale, NJ: Erlbaum.

Perea, M., & Carreiras, M. (2003). Sequential effects in the lexical decision task: The role of the item frequency of the previous trial. *Quarterly Journal of Experimental Psychology*, **56A**, 385-401.

Rastle, K., & Davis, M. H. (2002). On the complexities of measuring naming. *Journal of Experimental Psychology: Human Perception & Performance*, **28**, 307-314.

Reber, A. S. (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology: General*, **118**, 219-235.

Rogers, R. D., & Monsell, S. (1995). Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, **124**, 207-231.

Tenpenny, P. L. (1995). Abstractionist versus episodic theories of repetition priming and word identification. *Psychonomic Bulletin & Review*, **2**, 339-363.

Weekes, B. (1997). Differential effects of number of letters on word and nonword naming latency. *Quarterly Journal of Experimental Psychology*, **50A**, 439-456.

Whaley, C. P. (1978). Word–nonword classification time. *Journal of Verbal Learning & Verbal Behavior*, **17**, 143-154.

Zevin, J. D., & Balota, D. A. (2000). Priming and attentional control of lexical and sublexical pathways during naming. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **26**, 121-135.

(Manuscript received June 10, 2005;
revision accepted for publication September 17, 2005.)