Behavior Research Methods
2008, 40 (2), 484-492
doi: 10.3758/BRM.40.2.484

A program for generating randomized simple and context-sensitive sequences

Gilbert Remillard
Morehead State University, Morehead, Kentucky

This article introduces Sequence Generation 2008 (SeqGen2008), a Windows-based sequence generator. SeqGen2008 can generate simple sequences satisfying user-defined event probabilities or frequencies. The program can also generate context-sensitive sequences satisfying user-defined transition matrices that specify the probabilities or frequencies with which distinct events are to follow specific contexts. An analysis of the properties and behavior of the algorithms employed by SeqGen2008 reveals that the algorithms are unbiased in their generation of sequences.

Sequences composed of *m* distinct events, with each event occurring one or more times, are common in psychological research. Typically, the events are stimuli or experimental conditions. Sequence generation, in many cases, involves ordering events in a sequence either by randomly selecting the events with replacement from a pool or by listing the events and randomly shuffling the list. These two simple approaches do not consider context (i.e., preceding events) when selecting the next event in the sequence. The disregard of context has two disadvantages. The first is a potential reduction in power via the introduction of noise in the data (van Casteren & Davis, 2006), and the second is a risk of introducing confounds. These will be discussed more fully in the next section of the article. Context-sensitive sequence generation, in contrast, permits tighter control over the sequential structure and so may be the better approach when sequences of events are generated in a within-subjects design (Emerson & Tobias, 1995; Remillard & Clark, 1999; van Casteren & Davis, 2006). This form of sequence generation, however, is less common than the two simple approaches described above, perhaps because it is more difficult to implement or it is not implemented by popular experiment generation software (e.g., E-Prime).

Another issue in the realm of sequence generation is the adequacy of the algorithms used to generate sequences. For example, there are numerous ways to randomly shuffle a list of events, but not all algorithms are unbiased (Castellan, 1992). The majority of studies in which participants have been presented with sequences of events have not described the algorithms that were used to generate the sequences. Consequently, one cannot ascertain whether the process of generating sequences was biased or unbiased in these studies. Even if acceptable algorithms were used, there is no evidence that they were implemented correctly.

The preceding discussion suggests a need for software that implements both context-sensitive sequence generation and unbiased algorithms. The present article introduces such software. The article is divided into four major sections. The first section describes five sequence generation methods—two common methods that disregard context and three context-sensitive methods that researchers might find useful. The second section presents a Windows application that implements the five methods. The third section describes the algorithms that underlie the various methods of sequence generation. The final section presents the results of Monte Carlo simulations. The simulations were run to ensure that the algorithms were implemented correctly and exhibited no bias in their generation of sequences.

Sequence Generation Methods

The *simple probabilities* method is a sequence generation method in which each distinct event has a fixed probability of being selected on each trial. For example, a researcher may wish to generate 500-trial sequences with Events 1–8 each having a 1/8 probability of being selected on each trial (see Table 1, tier 1), or Events 1–3 having probabilities of 8/10, 1/10, and 1/10, respectively, of being selected on each trial (see Table 1, tier 2). A disadvantage of the method is that observed relative frequencies (RFs) and expected RFs can differ considerably, especially when sequences are short.[1] However, observed RFs will tend to approach expected RFs as sequences become longer.

The *simple frequencies* method is a sequence generation method in which each distinct event occurs a prespecified number of times in the sequence. For example, a researcher may wish to generate 15-trial sequences with each of Events 1–5 occurring 3 times in a sequence (see Table 1, tier 3) or to generate 180-trial sequences with Events 1–4 occurring 60, 60, 30, and 30 times, respec-

G. Remillard, g.remillard@moreheadstate.edu

Table 1
Simple Probabilities and Simple Frequencies

Event	1	2	3	4	5	6	7	8	
Probability	1	1	1	1	1	1	1	1	/8
Event	1	2	3						
Probability	8	1	1	/10					
Event	1	2	3	4	5				
Frequency	3	3	3	3	3	/15			
Event	1	2	3	4					
Frequency	60	60	30	30	/180				

tively, in a sequence (see Table 1, tier 4). Clearly, simple frequencies generation can produce sequences with observed RFs that match expected RFs. For example, a researcher may wish to generate a 400-trial sequence with Events 1–5 having observed RFs of 2/10, 5/10, 1/10, 1/10, and 1/10, respectively. This could be achieved by generating a 400-trial sequence with Events 1–5 occurring 80, 200, 40, 40, and 40 times, respectively, in the sequence.

The simple probabilities and simple frequencies generation methods are common in psychological research. However, they fail to consider context. The event selected on a trial is generally independent of the events selected on preceding trials. This disregard of context has two disadvantages. The first is a potential reduction in power. For example, consider a researcher who generates twenty 90-trial sequences (one sequence per participant) with each of Conditions 1–3 occurring 30 times in a sequence. For each participant, the sequence is generated by randomly shuffling the 90 trials. Now, simply by chance, Condition 2 may be more likely to precede Condition 1 than to precede Condition 3 for some participants, and vice versa for other participants. As a result, the performance difference between Conditions 1 and 3 might vary across participants if there are order effects. This would inflate the error term (i.e., the participant × condition interaction) in an ANOVA and would diminish power. The second disadvantage of disregarding context is the risk of introducing confounds. For example, if, in the preceding scenario, Condition 2 is more likely to precede Condition 1 than to precede Condition 3 for the large majority of participants, this would represent a confound. The likelihood of this type of confounding will tend to increase as the number of participants decreases.

The next three sequence generation methods are context sensitive. The event selected on a trial is dependent on the events selected on preceding trials. The second method below can overcome the disadvantages outlined above.

The *contextual probabilities* method is a sequence generation method in which the probability of an event's being selected on trial t is dependent on the context (i.e., on the events selected on trials $t - n, \ldots, t - 2, t - 1$). Table 2 presents three transition matrices, each specifying the conditional probabilities with which events are to follow contexts. The first matrix has four events and four contexts of size 1. The row for Context 3 specifies that Events 1–4 have probabilities of 8/16, 4/16, 0/16, and 4/16, respectively, of being selected on trial t given that Event 3 is selected on trial $t - 1$. The probabilities for Context 3 could have been written in a different but equivalent manner—for example, 2 1 0 1 /4 or 50 25 0 25 /100. The zeroes in the matrix indicate that the researcher does not wish an event to occur twice in succession.

The second matrix has three events and nine contexts of size 2. The row for Context 1–2 specifies that Events 1–3 have probabilities of 5/10, 2/10, and 3/10, respectively, of being selected on trial t, given that Events 1 and 2 are selected on trials $t - 2$ and $t - 1$, respectively. Finally, the third matrix has two events and six contexts of size 3. The row for Context 2–1–2 specifies that Events 1 and 2 each have a probability of 25/50 of being selected on trial t, given that Events 2, 1, and 2 are selected on trials $t - 3$, $t - 2$, and $t - 1$, respectively. Again, the probabilities could have been written in a different but equivalent manner; for example, 1 1 /2 or 50 50 /100. The zeroes in the matrix indicate that the researcher does not wish an event to occur three times in succession. The contextual probabilities generation method has been used in studies to examine the human capacity for acquiring sequential structure (e.g., Schvaneveldt & Gomez, 1998; Soetens, Melis, & Notebaert, 2004).

A disadvantage of the method is that observed conditional RFs (CRFs) and expected CRFs can differ considerably, especially when sequences are short.[2] However, observed CRFs will tend to approach expected CRFs as sequences become longer.

The *contextual frequencies–exact* method is a sequence generation method in which each distinct event follows a given context a prespecified number of times in the sequence. For example, the first matrix in Table 2 specifies that Events 1–4 will immediately follow Context 4 precisely 4, 4, 8, and 0 times, respectively, in the sequence. The second matrix specifies that Events 1–3 will follow Context 3–1 precisely 3, 5, and 2 times, respectively, in the sequence. Finally, the third matrix specifies that Events 1

Table 2
Transition Matrices With Conditional Probabilities or Conditional Frequencies–Exact/Block

	Event				
Context	1	2	3	4	
1	0	8	4	4	/16
2	4	0	4	8	/16
3	8	4	0	4	/16
4	4	4	8	0	/16
1–1	3	5	2	/10	
1–2	5	2	3	/10	
1–3	5	3	2	/10	
2–1	2	5	3	/10	
2–2	5	3	2	/10	
2–3	5	2	3	/10	
3–1	3	5	2	/10	
3–2	5	2	3	/10	
3–3	5	3	2	/10	
1–1–2	25	25	/50		
1–2–1	25	25	/50		
1–2–2	50	0	/50		
2–1–1	0	50	/50		
2–1–2	25	25	/50		
2–2–1	25	25	/50		

and 2 will each follow Context 1–1–2 precisely 25 times in the sequence.

One strength of this method of generation is that it can elimate the disadvantages associated with simple frequencies generation. For example, rather than generating sequences by randomly shuffling 90 trials, the researcher in the scenario described earlier could generate sequences in which each condition follows every condition, say, 10 times or each condition follows every possible pair of conditions, say, 3 times (as in the second matrix of Table 2, but with 3 3 3 /9 for each context). More generally, one can generate sequences in which each of *m* distinct events follows every possible context of size *n* (of which there are m^n) precisely *f* times. The result is that each event is preceded by each context precisely *f* times. Emerson and Tobias (1995) have discussed other benefits of such a design; for example, it permits one to examine the effects of context on event performance.

Another strength of the contextual frequencies–exact generation method is that it produces sequences with observed CRFs that match the expected CRFs. This type of control over the sequential structure has been used fruitfully in studies in which the human capacity for acquiring sequential structure has been examined (e.g., Remillard, 2003, 2008; Remillard & Clark, 2001). Unfortunately, the method is not possible with some transition matrices (see the Internal Implementation section below). An alternative, in that case, could be the sequence generation method described next.

The *contextual frequencies–block* method is a sequence generation method in which each distinct event follows a given context a prespecified number of times across every *n* occurrences of the context. For example, the first matrix in Table 2 specifies that across every 16 occurrences of Context 2, Events 1–4 will immediately follow the context 4, 0, 4, and 8 times, respectively. The second matrix specifies that across every 10 occurrences of Context 2–2, Events 1–3 will follow the context 5, 3, and 2 times, respectively. Finally, the third matrix specifies that across every 50 occurrences of Context 1–2–2, Events 1 and 2 will follow the context 50 and 0 times, respectively.

The contextual frequencies–block generation method will usually produce a sequence with observed CRFs that are closer to the expected CRFs than will contextual probabilities generation, because the former method guarantees that expected CRFs for a context (e.g., 0/16, 8/16, 4/16, 4/16 for Context 1 in the first matrix of Table 2) will be observed across every *n* occurrences (e.g., 16 occurrences) of the context. Moreover, a smaller *n* for a context (e.g., 0 2 1 1 /4) will produce a sequence with observed CRFs for the context that are closer to the expected values than will a larger *n* for the context (e.g., 0 50 25 25 /100). If it is crucial that observed CRFs be as close as possible to expected CRFs and contextual frequencies–exact generation is not possible, contextual frequencies–block generation would be a viable alternative.

The availability of software that implements at least one of the generation methods above is limited. van Casteren and Davis (2006) developed a program called Mix that shuffles events in a sequence subject to user-specified constraints. Mix can implement simple frequencies generation. Also, the ability of the user to specify constraints endows Mix with context sensitivity. However, Mix would not be capable of implementing the three context-sensitive methods above.

Emerson and Tobias (1995) created a program that implemented contextual frequencies–exact generation for transition matrices in which each of *m* distinct events follows every possible context of size *n* (of which there are m^n) precisely *f* times. Remillard and Clark (1999) subsequently developed a program that could be applied to a broader array of transition matrices. The program was designed for the now antiquated DOS operating system. Both programs implemented an efficient but biased algorithm, as well as an inefficient but unbiased brute force algorithm. The software described below implements a relatively efficient unbiased algorithm.

Sequence Generation 2008 (SeqGen2008)

SeqGen2008 is a Windows application that implements the five sequence generation methods described above (see Figure 1). The user (1) selects a generation method, (2) selects an input file, (3) specifies the length of a sequence (i.e., the number of trials) and the number of sequences to be generated, and (4) generates the sequences. If contextual frequencies–exact or simple frequencies is selected, the sequence length label and text box disappear, because the length of a sequence is determined by the input file frequencies. The stages of the generation process and error messages, if any, are displayed in the status window.

The program takes one input file and produces three files: a sequence file, a test file, and an error file. An input file for simple probabilities or simple frequencies generation consists of a header specifying the number of distinct events, followed by a listing of the probabilities or frequencies for each event. An input file for contextual probabilities, contextual frequencies–exact, or contextual frequencies–block generation consists of a header specifying the number of contexts, the context size, and the number of distinct events, followed by a listing of the contexts and the probabilities or frequencies.

The program copies every character it reads from the input file to an error file. If there is a problem with the input file (e.g., an invalid character or value), an error message directs the user to the error file and the location of the error. If there is no problem reading the input file, the error file is deleted, and the user does not see it. The sequence file contains the generated sequences.

The test file's main purpose is to reassure the user that the program's implementation of the selected generation method using the selected input file was as expected. The program randomly chooses a sequence from the sequence file, calculates various quantities using the selected sequence, and reports the quantities in a test file. For example, for contextual probabilities and contextual frequencies–block generation, the program reports (1) the number of occurrences of each context in the selected sequence, (2) the observed CRFs, and (3) the minimum, maximum, and mean absolute discrepancies between the observed CRFs and the nonzero expected CRFs. The dis-

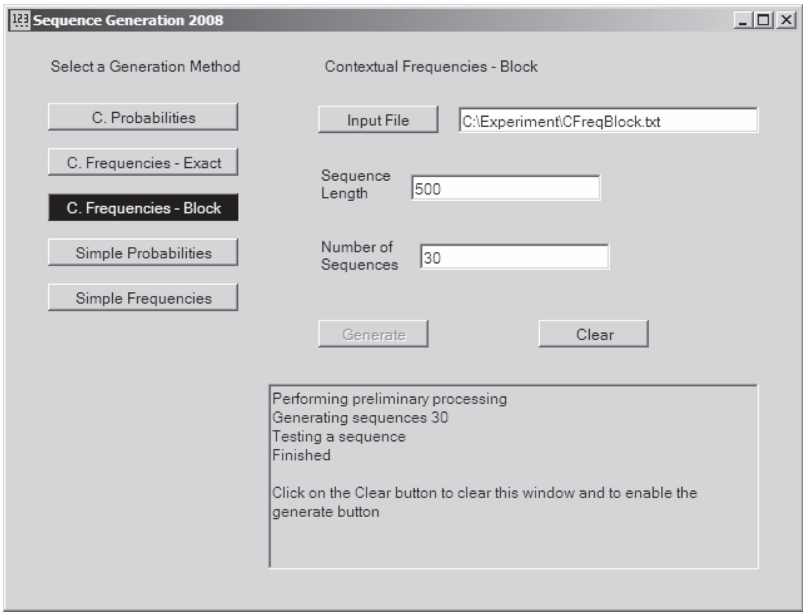

Figure 1. SeqGen2008's graphical user interface.

crepancies can range from 0 to 1. The mean discrepancy should, generally, be smaller for longer sequences than for shorter sequences and for contextual frequencies–block generation than for contextual probabilities generation.[3]

Availability. The program and further details about the input and output files can be obtained by following the link *Sequence Generation 2008* at people2.moreheadstate.edu/fs/g.remillard/.

Machine requirements. The program has been tested using Windows XP and Windows Vista. The program is written in C#. Consequently, Microsoft .NET Framework 1.1 or 2.0 must be installed on the user's computer. Windows Vista has .NET Framework 2.0 installed. Windows XP, however, may or may not have .NET Framework 1.1 or 2.0 installed.[4] If the framework is not installed, the user can download Microsoft .NET Framework Version 1.1 Redistributable Package or Microsoft .NET Framework Version 2.0 Redistributable Package (x86), free of charge, from the Microsoft Web site.

Internal Implementation

This section describes the program's internal implementation of the five sequence generation methods.

Simple probabilities generation. Let $f_1/n, f_2/n, \ldots, f_m/n$ ($n = f_1 + f_2 + \cdots + f_m$) be the probabilities associated with Events 1–m, respectively. The program's implementation of simple probabilities generation is equivalent to creating an array holding $f_1, f_2, \ldots, f_m$ occurrences of Events 1–m, respectively, and, on each trial, randomly choosing with replacement one of the n elements in the array. Thus, Event i has probability f_i/n of being selected on each trial. For example, the program's implementation of the probabilities in tier 2 of Table 1 is equivalent to creating an array holding 8, 1, and 1 occurrences of Events 1–3, respectively, and, on each trial, randomly choosing with replacement 1 of the 10 elements in the array.

Simple frequencies generation. Let $f_1, f_2, \ldots, f_m$ ($n = f_1 + f_2 + \cdots + f_m$) be the frequencies associated with Events 1–m, respectively. The program's implementation of simple frequencies generation is equivalent to creating an array holding $f_1, f_2, \ldots, f_m$ occurrences of Events 1–m, respectively, and, on each trial, randomly choosing *without* replacement one of the remaining elements in the array. Thus, Events 1–m will occur f_1–f_m times, respectively, in a sequence of length n.

The algorithm above is unbiased. Each distinct sequence of length n in which Events 1–m occur f_1–f_m times, respectively, has the same probability of being generated. To see this, first note that there are $n!$ ways of ordering the n elements initially placed in the array. However, different orderings may represent the same sequence if elements in the array are not all distinct. For example, if the array contains the six elements 1_1, 1_2, 1_3, 2_4, 2_5, 2_6, then 1_3–1_1–1_2–2_4–2_6–2_5 and 1_2–1_3–1_1–2_4–2_5–2_6 are two different orderings of the elements that represent the same sequence 1–1–1–2–2–2. In fact, there are $n!/(f_1!f_2!\ldots f_m!)$ distinct sequences in the list of $n!$ orderings, because each distinct sequence occurs $f_1!f_2!\ldots f_m!$ times in the list (Roberts, 1984, pp. 47–50). Now, random selection without replacement ensures that each of the $n!$ orderings is equally likely to be generated (Castellan, 1992). Because each distinct sequence occurs with the same frequency in the list of $n!$ orderings, it follows that each distinct sequence has the same probability of being generated.

The implementation of the three context-sensitive methods relies on graph theory. Transition matrices, such as those in Table 2, can be considered graphs with contexts as vertices (Emerson & Tobias, 1995; Remillard & Clark, 1999). We say that Context A (i.e., a_1–a_2–$\cdots$–a_n)

can be succeeded by Context B (i.e., b_1–b_2– . . . –b_n), or Context B can be preceded by Context A, if the last $n - 1$ elements of Context A are the first $n - 1$ elements of Context B (i.e., $a_2 = b_1$, $a_3 = b_2$, . . . , $a_n = b_{n-1}$) and the last element of Context B (i.e., b_n) is an event that can immediately follow Context A. Thus, Context 1–2–1 can be succeeded by Context 2–1–1 in the third matrix of Table 2, Context 3–1 can be succeeded by Context 1–2 in the second matrix, and Context 2 can be succeeded by Context 4 in the first matrix. An arc (i.e., arrow) is drawn from Context A to Context B if A can be succeeded by B. The number of arcs from A to B is determined by the corresponding value in the transition matrix. For example, the left side of Figure 2 shows the graph of the third transition matrix in Table 2. There are 25 arcs from Context 1–1–2 to Context 1–2–2, 50 arcs from Context 1–2–2 to Context 2–2–1, and so on. The right side of Figure 2 shows a portion of the graph of the second matrix in Table 2. There are 2 arcs from Context 1–2 to Context 2–2, 3 arcs from Context 2–2 to Context 2–2 (i.e., three loops), and so on. Finally, an examination of the first matrix in Table 2 shows that there are 4, 4, and 8 arcs from Context 4 to Contexts 1, 2, and 3, respectively.

For each context-sensitive method, the program determines whether or not a transition matrix is strongly connected prior to generating sequences. A matrix is strongly connected if, for every pair of Contexts A and B, A can reach B via a path of successor contexts (i.e., a path of arcs) and B can reach A. For example, Context 1–2–1 in Figure 2 can reach Context 1–2–2 via the path of successor Contexts 1–2–1, 2–1–1, 1–1–2, 1–2–2. Similarly, Context 3–2 in Figure 2 can reach Context 2–1 via the path of successor Contexts 3–2, 2–2, 2–1. A strongly connected matrix guarantees that each context can appear in a sequence. If Context A cannot reach Context B (i.e., the matrix is not strongly connected), and Context A is the starting context, it will be impossible for Context B to appear in the generated sequence. The program displays an error message if the transition matrix is not strongly connected.[5]

Contextual probabilities generation. The program randomly selects a starting context and adds the context to an array. An arc exiting the just-added context is randomly chosen with replacement, and the succeeding context to which the arc points is added next to the array. The preceding step is repeated until the number of contexts in the array is sufficient to satisfy the user-specified sequence length. For example, if Context 2–1–2 in Figure 2 has just been added to the array, one of the 50 arcs exiting the just-added context would be randomly chosen with replacement, and the succeeding context to which the arc points (e.g., say, 1–2–1) would be added next to the array. An arc exiting the just-added Context 1–2–1 would be randomly chosen with replacement, the succeeding context to which the arc points added next to the array, and so on. After the array is loaded, the events in the starting context are written to the sequence file, followed by the last event of each subsequent context in the array. For example, if 2–2–1 was the starting context, and 2–1–1, 1–1–2, and 1–2–2 were the next three contexts in the array, the sequence would begin with 2 2 1, followed by 1, 2, and 2.

Contextual frequencies–exact generation. Implementation requires that each arc be chosen precisely once. Thus, an Eulerian circuit must be constructed. Such a circuit exists if and only if (1) the transition matrix is strongly connected and (2) for each context, the number of arcs entering the context equals the number of arcs exiting the context (Roberts, 1984, p. 461). The third matrix in Table 2 is an Eulerian matrix (i.e., it satisfies both criteria), and so an Eulerian circuit exists. Figure 2 shows that the matrix is strongly connected and that, for each context, the number of arcs entering the context (e.g., 50 for Context 1–1–2) equals the number of arcs exiting the context (e.g., 50 for Context 1–1–2). The first matrix in Table 2 is also Eulerian. However, the second matrix in Table 2 is not Eulerian. The right side of Figure 2 shows

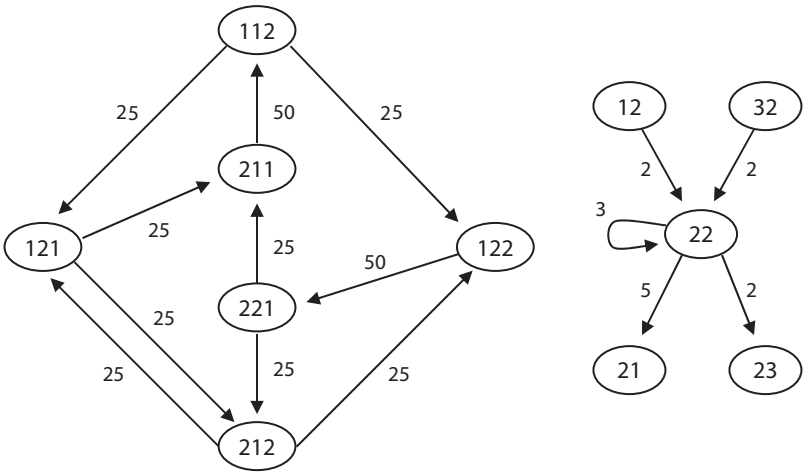

Figure 2. Left: The graph of the third transition matrix in Table 2. Right: A portion of the graph of the second transition matrix in Table 2. Shown are the arcs entering and exiting Context 2–2.

that the 7 arcs entering Context 2–2 are not equal to the 10 arcs exiting the context (the loop represents 3 entering and 3 exiting arcs). The program displays an error message if either criterion is not satisfied.

The program constructs an Eulerian circuit in three steps. First, a starting context is selected. The probability of selecting Context C as the starting context is

$$\frac{d^+(\mathrm{C})}{\sum_i d^+(\mathrm{C}_i)},$$

where $d^+(\mathrm{C})$ is the number of arcs exiting Context C and the denominator is the sum of the number of exiting arcs taken over all contexts. Thus, the greater the number of arcs exiting a context, relative to other contexts, the higher the probability of selecting the context as the starting context. Let $\mathrm{C_S}$ be the context selected as the starting context.

Next, an arborescence rooted at $\mathrm{C_S}$ is constructed. An arborescence rooted at $\mathrm{C_S}$ is a subgraph of the original graph in which (1) the subgraph consists of all the contexts in the original graph, (2) there is exactly one arc exiting each context in the subgraph, except for $\mathrm{C_S}$, for which there are no exiting arcs, and (3) every context can reach $\mathrm{C_S}$ via a path of successor contexts in the subgraph. The algorithm to construct an arborescence rooted at $\mathrm{C_S}$ is the one proposed by Propp and Wilson (1998, p. 202; see also Jiang, Anderson, Gillespie, & Mayne, 2007). The algorithm is one of the most efficient algorithms for randomly generating an arborescence.

The final step is the generation of the Eulerian circuit. The starting Context $\mathrm{C_S}$ is added to an array. An arc exiting the just-added context is randomly chosen *without* replacement—*except* for the arc in the arborescence, which is chosen only after all other exiting arcs have been chosen—and the succeeding context to which the chosen arc points is added next to the array. The preceding step is repeated until a context is added that has no remaining exiting arcs. This final context will be $\mathrm{C_S}$, because the number of arcs entering a context is equal to the number of arcs exiting the context and, when $\mathrm{C_S}$ was initially placed in the array, an exiting arc was used up before an entering arc. Thus, at some point, $\mathrm{C_S}$ will be entered (i.e., added to the array), but it will not be possible to exit. For all other contexts, an entering arc is used up before an exiting arc. So, whenever one of these contexts is entered (i.e., added to the array), it is possible to exit. Thus, the array begins and ends with $\mathrm{C_S}$. The array is also guaranteed to contain an Eulerian circuit by virtue of choosing arcs in the arborescence last (Jiang et al., 2007; Kandel, Matias, Unger, & Winkler, 1996).

To complete the process, the events of $\mathrm{C_S}$ are written to the sequence file, followed by the last event of each subsequent context in the array. The resulting sequence will begin and end with the same context. For example, if 1–2–1 was the starting context, the sequence will begin and end with 1 2 1. Also, the length of the sequence will be the sum of the frequencies in the transition matrix plus the size of the context (e.g., 65 and 303 elements for the first and third matrices in Table 2, respectively). Finally, and most important, the resulting sequence represents a random selection from the set of all distinct sequences that begin and end with $\mathrm{C_S}$ and that satisfy the frequencies in the transition matrix (Jiang et al., 2007; Kandel et al., 1996).

Let N_C be the number of distinct sequences that begin and end with Context C and that satisfy the frequencies in the transition matrix. Therefore,

$$N = \sum_\mathrm{C} N_\mathrm{C}$$

is the number of distinct sequences that satisfy the frequencies in the transition matrix. The algorithm described above is unbiased. Each distinct sequence will have a probability of $1/N$ of being generated. A proof of this assertion is presented in the Appendix.

The unbiased algorithm presented here is considerably more efficient than the unbiased, brute force algorithm implemented by Emerson and Tobias (1995) and Remillard and Clark (1999). The brute force algorithm involved moving through the graph from one context to another until a context was encountered that had no remaining exiting arcs, and restarting the process if an Eulerian circuit had not been constructed. I tested the efficiency of the present algorithm by applying it to a transition matrix in which each of six events followed each of the 46,656 possible contexts of size 6 precisely once. The algorithm was applied five times using a Pentium 2.80-GHz machine. Seven seconds were required each time to generate ten 279,942-element sequences. In sharp contrast, the brute force algorithm failed to generate a single sequence after running for 1 h.

Implementation of contextual frequencies–exact generation requires that a transition matrix be Eulerian. One of the criteria is that, for each context, the number of arcs entering the context must equal the number of arcs exiting the context. This will be achieved if, for each Context a_1–a_2– · · · –a_n, the sum of the frequencies for the context is equal to the sum of the frequencies in column a_n limited to contexts ending with x–a_1–a_2– · · · –a_{n-1}, where x is a placeholder for the first element in a context. For example, the sum of the frequencies for Context 1–2–1 in Table 2 is 50, and this is equal to the sum of the frequencies in column 1 limited to contexts ending with x–1–2 (i.e., 25 and 25 for Contexts 1–1–2 and 2–1–2, respectively). Similarly, the sum of the frequencies for Context 2 in Table 2 is 16, and this is equal to the sum of the frequencies in column 2 limited to contexts ending with x (i.e., 8, 0, 4, and 4 for Contexts 1–4, respectively). The second matrix in Table 2 is not Eulerian. The sum of the frequencies for Contexts 1–3 is 10, and this is not equal to the sum of the frequencies in column 3 limited to contexts ending with x–1 (i.e., 2, 3, and 2 for Contexts 1–1, 2–1, and 3–1, respectively). Transition matrices in which each of m distinct events follows each of the m^n possible contexts of size n exactly f times are Eulerian. Finally, Remillard and Clark (1999) have described a general procedure for taking a transition matrix that specifies conditional probabilities and producing an Eulerian matrix with the smallest possible frequencies that satisfies the conditional probabilities.

Contextual frequencies–block generation. The program proceeds as in contextual probabilities generation, with one exception: An arc exiting the just-added context is randomly chosen *without* replacement. When all arcs exiting a context have been chosen, the arcs for the context are restored, and selection without replacement begins anew.

Random selection without replacement ensures an unbiased process. For example, the second matrix in Table 2 specifies that across every 10 occurrences of Context 3–1, Events 1–3 will immediately follow the context three, five, and two times, respectively. There are 10!/(3!5!2!) = 2,520 distinct ways of arranging three 1s, five 2s, and two 3s to follow 10 occurrences of Context 3–1. Random selection without replacement guarantees that each distinct arrangement has the same probability of occurrence (see the Simple Frequencies Generation section for the rationale).

The sequence generation algorithms described above use C#'s built-in pseudorandom number generator. Simulations, reported in the next section, indicate that the generator is sufficient. The simulations provide no evidence that the algorithms are biased in their generation of sequences.

Monte Carlo Simulations

Simulations were conducted to ensure that the algorithms were implemented correctly and exhibited no bias in their generation of sequences. None of the 33 simulations described below produced significant results. Thus, there is no evidence that the algorithms are biased.

Simple probabilities generation. Suppose Events 1–3 have probabilities of 3/11, 5/11, and 3/11, respectively, of being selected on each trial. There are $3^6 = 729$ distinct sequences of length 6 that can be generated. If the sequence generation algorithm is unbiased, the probability of generating sequence s_1–s_2–$\cdots$–s_6, $p(s_1$–s_2–$\cdots$–$s_6)$, should equal the product $p(\mathrm{s}_1) \cdot p(s_2) \cdot \cdots \cdot p(s_6)$ where $p(s_i)$ is the probability of selecting s_i on each trial. For example, $p(2\text{–}1\text{–}3\text{–}3\text{–}2\text{–}1) = (5/11) \cdot (3/11) \cdot (3/11) \cdot (3/11) \cdot (5/11) \cdot (3/11)$. Moreover, the expected frequency of s_1–s_2–$\cdots$–s_6 should be $N \cdot p(s_1$–s_2–$\cdots$–$s_6)$, where N is the number of sequences of length 6 generated in the simulation. Setting $N = 300{,}000$, I obtained $\chi^2(728) = 707.71$, $p = .698$. Thus, the observed frequencies did not differ significantly from the expected frequencies across the 729 sequences. Five other simulations were performed with different numbers of events, different event probabilities, and different sequence lengths. The number of distinct sequences ranged from 64 to 1,024, and N was 300,000 in each case.

Simple frequencies generation. Suppose Events 1–4 are to occur four, one, two, and one times, respectively, in an eight-trial sequence. There are 8!/(4!1!2!1!) = 840 distinct sequences that can be generated. If the sequence generation algorithm is unbiased, the probability of generating a given sequence should be 1/840. Moreover, the expected frequency of the sequence should be $N \cdot (1/840)$, where N is the number of eight-trial sequences generated in the simulation. Setting $N = 300{,}000$, I obtained $\chi^2(839) = 826.67$, $p = .613$. Thus, the observed frequencies did not differ significantly from the expected frequencies across the 840 sequences. Three other simulations were performed with different numbers of events and different event frequencies. The number of distinct sequences ranged from 30 to 840, and N was 300,000 in each case.

Table 3
Transition Matrix With Conditional Probabilities or Conditional Frequencies–Block

	Event			
Context	1	2	3	
1	2	3	1	/6
2	1	1	1	/3
3	1	2	3	/6

Contextual probabilities generation. Consider the transition matrix in Table 3. There are $3^6 = 729$ distinct sequences of length 6 that can be generated. If the sequence generation algorithm is unbiased, the probability of generating sequence s_1–s_2–$\cdots$–s_6, $p(s_1$–s_2–$\cdots$–$s_6)$, should equal the product $p(s_1) \cdot p(s_2 \mid s_1) \cdot p(s_3 \mid s_2) \cdot \cdots \cdot p(\mathrm{s}_6 \mid \mathrm{s}_5)$, where $p(\mathrm{s}_1)$ is the probability of selecting s_1 as the starting context, and $p(s_j \mid s_i)$ is the conditional probability of selecting s_j given that s_i was selected on the preceding trial. For example, $p(1\text{–}3\text{–}1\text{–}3\text{–}2\text{–}2) = p(1) \cdot p(3 \mid 1) \cdot \cdots \cdot p(2 \mid 2) = (1/3) \cdot (1/6) \cdot (1/6) \cdot (1/6) \cdot (2/6) \cdot (1/3)$. Moreover, the expected frequency of s_1–s_2–$\cdots$–s_6 should be $N \cdot p(\mathrm{s}_1$–s_2–$\cdots$–$\mathrm{s}_6)$, where N is the number of sequences of length 6 generated in the simulation. Setting $N = 300{,}000$, I obtained $\chi^2(728) = 710.85$, $p = .668$. Six other simulations were performed with different numbers of events, different context sizes, and different sequence lengths. The number of distinct sequences ranged from 512 to 1,024, and N was 300,000 in each case.

Contextual frequencies–exact generation. Consider the transition matrix in which each of two distinct events follows each of the eight possible contexts of size 3 precisely once. Hutchinson and Wilf's (1975) formula indicates that there are 256 distinct sequences of length 19 that satisfy the frequencies in the matrix. If the sequence generation algorithm is unbiased, the probability of generating a given sequence should be 1/256. Moreover, the expected frequency of the sequence should be $N \cdot (1/256)$, where N is the number of sequences of length 19 generated in the simulation. Setting $N = 300{,}000$, I obtained $\chi^2(255) = 254.62$, $p = .495$. The transition matrix used in this simulation is one on which Emerson and Tobias's (1995) efficient algorithm displayed a bias.

Seven other simulations were performed with different numbers of events and different context sizes. Three of the simulations used transition matrices in which each cell in a matrix had the same frequency value. The remaining four simulations used matrices with unequal cell frequencies. Two of these matrices were ones on which Remillard and Clark's (1999) efficient algorithm displayed a bias. For six of the seven simulations, the number of distinct sequences ranged from 36 to 1,600, and N was 300,000 in each case. For one simulation, the number of distinct

sequences was 65,536, and N was 10,000,000. This simulation involved the transition matrix in which each of two distinct events follows each of the 16 possible contexts of size 4 precisely once. Emerson and Tobias's (1995) efficient algorithm displayed a bias with this matrix.

Contextual frequencies–block generation. The matrix in Table 3 specifies that across every six occurrences of Context 1, Events 1–3 will immediately follow the context two, three, and one times, respectively. There are 6!/(2!3!1!) = 60 distinct ways of arranging two 1s, three 2s, and one 3 to follow six occurrences of Context 1. If the sequence generation algorithm is unbiased, each arrangement should have a probability of 1/60 of being selected for any six-occurrence block of Context 1. A similar analysis for Context 2 suggests that each of the six distinct arrangements should have a probability of 1/6 of being selected for any three-occurrence block of Context 2. A long sequence was generated, and the frequency of each arrangement was counted over the first $N = 300{,}000$ blocks of Context 1 and Context 2. The results were $\chi^2(59) = 59.87$, $p = .444$, and $\chi^2(5) = 4.53$, $p = .476$, for Contexts 1 and 2, respectively. Thus, the observed frequencies did not differ significantly from the expected frequencies. Three other simulations were performed with different transition matrices. Each simulation examined two contexts as above.

Conclusion

Castellan (1992) has cautioned that

> researchers should always take extreme care in choosing algorithms in their research. Only well-documented algorithms should be used. Documentation does not mean that the algorithm has been described in print—even in a reputable journal or book. Appropriate documentation includes an analysis of the properties and behavior of the algorithm. (p. 76)

SeqGen2008 implements well-documented algorithms for generating sequences satisfying simple, user-defined event probabilities or frequencies, or more complex, user-defined transition matrices. Theoretical (i.e., mathematical) and empirical (i.e., simulation) analyses indicate that the algorithms are unbiased in their generation of sequences.

AUTHOR NOTE

Correspondence should be addressed to G. Remillard, Department of Psychology, Morehead State University, 601 Ginger Hall, Morehead, KY 40351 (e-mail: g.remillard@moreheadstate.edu).

REFERENCES

Castellan, N. J., Jr. (1992). Shuffling arrays: Appearances may be deceiving. *Behavior Research Methods, Instruments, & Computers*, **24**, 72-77.

Emerson, P. L., & Tobias, R. D. (1995). Computer program for quasi-random stimulus sequences with equal transition frequencies. *Behavior Research Methods, Instruments, & Computers*, **27**, 88-98.

Hutchinson, J. P., & Wilf, H. S. (1975). On Eulerian circuits and words with prescribed adjacency patterns. *Journal of Combinatorial Theory A*, **18**, 80-87.

Jiang, M., Anderson, J., Gillespie, J., & Mayne, M. (2007, June). *uShuffle: A useful tool for shuffling biological sequences while preserving the k-let counts*. Paper presented at the 2007 International Conference on Bioinformatics and Computational Biology, Las Vegas.

Jungnickel, D. (2005). *Graphs, networks, and algorithms* (2nd ed.). Berlin: Springer.

Kandel, D., Matias, Y., Unger, R., & Winkler, P. (1996). Shuffling biological sequences. *Discrete Applied Mathematics*, **71**, 171-185.

Propp, J. G., & Wilson, D. B. (1998). How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, **27**, 170-217.

Remillard, G. (2003). Pure perceptual-based sequence learning. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **29**, 581-597.

Remillard, G. (2008). Implicit learning of second-, third-, and fourth-order adjacent and nonadjacent sequential dependencies. *Quarterly Journal of Experimental Psychology*, **61**, 400-424.

Remillard, G., & Clark, J. M. (1999). Generating fixed-length sequences statisfying any given *n*th-order transition probability matrix. *Behavior Research Methods, Instruments, & Computers*, **31**, 235-243.

Remillard, G., & Clark, J. M. (2001). Implicit learning of first-, second-, and third-order transition probabilities. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **27**, 483-498.

Roberts, F. S. (1984). *Applied combinatorics*. Englewood Cliffs, NJ: Prentice Hall.

Schvaneveldt, R. W., & Gomez, R. L. (1998). Attention and probabilistic sequence learning. *Psychological Research*, **61**, 175-190.

Soetens, E., Melis, A., & Notebaert, W. (2004). Sequence learning and sequential effects. *Psychological Research*, **69**, 124-137.

van Casteren, M., & Davis, M. H. (2006). Mix, a program for pseudorandomization. *Behavior Research Methods*, **38**, 584-589.

NOTES

1. The observed RF of event E in a sequence is the number of occurrences of E in the sequence divided by the length of the sequence. The expected RF of E is established a priori (e.g., as in Table 1).

2. The observed CRF of event E given Context C in a sequence is the number of times E follows C in the sequence, divided by the number of occurrences of C in the sequence. The expected CRF of E given C is established a priori (e.g., as in Table 2).

3. The mean discrepancy can be substantially smaller for the contextual frequencies–block method than for the contextual probabilities generation method. For example, I generated a 2,000-trial sequence for each transition matrix in Table 2, using the two sequence generation methods. The mean discrepancies for the three transition matrices were .0153, .0199, and .0145, respectively, using contextual probabilities generation, and .0011, .0028, and .0036, respectively, using contextual frequencies–block generation. The discrepancies in the latter case were 7.2%, 14.1%, and 24.8% of the discrepancies in the former case. Repeating the process five times—twice with a 2,000-trial sequence and three times with a 1,000-trial sequence—the discrepancies using contextual frequencies–block generation were 4.1%–31.0% ($M = 14.4\%$) of the discrepancies using contextual probabilities generation.

4. One can determine whether or not a Windows XP machine has .NET Framework 1.1 or 2.0 by clicking on Start => Control Panel => Add or Remove Programs and examining the list of currently installed programs for Microsoft .NET Framework 1.1 or 2.0.

5. The breadth-first search (BFS) algorithm is used to establish whether or not a transition matrix is strongly connected. The algorithm is applied in two passes, with the first context in the matrix as the source in each pass. If the algorithm succeeds on the first pass, the first context can reach every other context via a path of successor contexts. Otherwise, the matrix is not strongly connected, and the process stops. If the first pass succeeds, the arcs in the graph are reversed, and BFS is applied to the reverse graph. If the algorithm succeeds on this pass, every context can reach the first context via a path of successor contexts in the original graph, and so the matrix is strongly connected. Otherwise, the matrix is not strongly connected.

APPENDIX
Proof That the Algorithm for Contextual Frequencies–Exact Generation Is Unbiased

Let N_C be the number of distinct sequences that begin and end with Context C and that satisfy the frequencies in an Eulerian transition matrix. Therefore

$$N = \sum_C N_C$$

is the number of distinct sequences that satisfy the frequencies in the matrix.

Before proceeding with the proof, two facts are needed. First, for a Context C,

$$N_C = \frac{a_C \cdot \left[d^+(C)\right]! \prod_{v \neq C} \left[d^+(v) - 1\right]!}{\prod_f f!},$$

where a_C is the number of arborescences rooted at C, $d^+(\ )$ is the number of arcs exiting the context that is enclosed in parentheses, and the denominator is a product taken over all frequencies in the matrix (Kandel et al., 1996). Second, the number of arborescences rooted at a context in an Eulerian matrix is the same for each context; that is, in an Eulerian matrix, $a_{Ci} = a_{Cj}$ for $C_i \neq C_j$ (Jungnickel, 2005, p. 121).

Proceeding with the proof, let S be a sequence that begins and ends with Context C_x and that satisfies the frequencies in the matrix, and let $P_{Gen}(S)$ be the probability that the algorithm generates S. Therefore, $P_{Gen}(S) = P$(algorithm selects C_x as the starting context) $\cdot$ P(algorithm generates S given that C_x is the starting context). It can be shown that the latter term in the product is equal to

$$\frac{1}{N_{Cx}}$$

(Jiang et al., 2007; Kandel et al., 1996). Thus,

$$P_{Gen}(S) = \frac{d^+(C_x)}{\sum_C d^+(C)} \cdot \frac{1}{N_{Cx}}.$$

Now I show that the product is equal to

$$\frac{1}{N}.$$

Let C_i be an arbitrary context. Because the matrix is Eulerian, $a_{Cx} = a_{Ci}$, and so

$$\frac{N_{Cx}}{N_{Ci}} = \frac{\left[d^+(C_x)\right]!\left[d^+(C_i) - 1\right]!}{\left[d^+(C_i)\right]!\left[d^+(C_x) - 1\right]!} = \frac{d^+(C_x)}{d^+(C_i)},$$

or equivalently,

$$d^+(C_i) = \frac{d^+(C_x)}{N_{Cx}} \cdot N_{Ci}.$$

Thus,

$$\sum_C d^+(C) = \frac{d^+(C_x)}{N_{Cx}} \cdot \sum_C N_C = \frac{d^+(C_x)}{N_{Cx}} \cdot N.$$

Substituting this value for

$$\sum_C d^+(C)$$

in the equation for $P_{Gen}(S)$ above yields

$$P_{Gen}(S) = \frac{1}{N}.$$

This completes the proof.

(Manuscript received June 19, 2007;
revision accepted for publication November 7, 2007.)