Behavior Research Methods, Instruments, & Computers
1999, 31 (2), 235-243

Generating fixed-length sequences satisfying any given *n*th-order transition probability matrix

GILBERT REMILLARD
University of Manitoba, Winnipeg, Manitoba, Canada

and

JAMES M. CLARK
University of Winnipeg, Winnipeg, Manitoba, Canada

An experimental design involving sequences of m distinct events can be conceptualized as an nth-order transition probability matrix specifying the probabilities with which each of the m distinct events is to follow certain n-grams. This paper describes a general method for constructing sequences of shortest possible length that satisfy any such matrix and presents a computer program that randomly generates such sequences.

The participants in many psychological experiments are exposed to sequences in which each of m distinct events occurs a number of times. A lexical decision task, for example, involves the presentation of words or nonwords ($m = 2$), with participants making a timed response to indicate the lexical status of the current item. In a serial reaction time task, a stimulus typically appears in one of four locations, and participants press one of an equal number of corresponding keys ($m = 4$). In the Stroop task, participants name the ink color (generally one of four possibilities) of one of four possible color words appearing on successive trials ($m = 16$, if all combinations are permitted, or $m = 12$, if colors and words cannot be identical on a given trial).

In such designs, the speed and correctness of responding on a particular trial can depend on the pattern of stimuli on preceding trials and on the probability of events following one another. Researchers will therefore need to control and/or vary systematically the transition probabilities between successive events. For example, detecting a particularly subtle effect in lexical decision could benefit from ensuring that words of different types equally often follow specific sequences of words and nonwords; using serial reaction time tasks to study implicit learning requires manipulation of transition probabilities between successive stimuli; and a Stroop study may require specific types of transitions (e.g., negative priming trials in which a distractor color word on one trial becomes the target ink color on the following trial).

Any such design can be conceptualized as an nth-order transition probability matrix, in which the matrix specifies the probabilities with which each of m distinct events is to follow certain n-grams (i.e., runs of length n). A sequence satisfying an nth-order transition probability matrix would consist of the n-grams listed in the matrix and would encompass the transition probabilities outlined in the matrix. The top panel in Table 1 is an example of a first-order transition probability matrix. A sequence satisfying the matrix would consist of the four unigrams 1, 2, 3, and 4 and would encompass the transition probabilities outlined in the matrix. Thus, in the sequence, the probability of 2 following the unigram 1 would be 3/5 (i.e., .60).

The top panel in Table 2 is an example of a second-order transition probability matrix. A sequence satisfying the matrix would consist of the eight bigrams 1–2, 1–3, 2–1, 2–4, 3–1, 3–4, 4–2, and 4–3 and would encompass the transition probabilities outlined in the matrix. Thus, in the sequence, the probability of 2 following the bigram 3–4 would be 2/5. Finally, a sequence satisfying the third-order transition probability matrix in Table 3 (ignoring the xs and the numbers in square brackets, for the moment) would consist of the 27 trigrams 1–1–1, 1–1–2, . . . , 3–3–3 and would encompass the transition probabilities outlined in the matrix. Thus in the sequence, the probability of 3 following the trigram 2–1–3 would be 1/6.

Emerson and Tobias (1995) developed a computer program that could generate sequences in which each of m distinct events followed each of the m^n possible n-grams with probability $1/m$. Thus, their program could generate sequences satisfying transition probability matrices such as those in Table 4. However, researchers (e.g., Bertelson, 1961; Kornblum, 1969; Remillard & Clark, 1999; Schvaneveldt & Gomez, 1996) may want sequences that satisfy an nth-order transition probability matrix in which not all m^n possible n-grams are considered (e.g., Table 2), or

This research was supported by a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship to G.R. and by an NSERC Operating Grant to J.M.C. Turbo Pascal (Version 5.0) source code listings of the two programs described in this article, the program to set up the equations and the sequence-generation program, can be downloaded off the World-Wide Web by entering the URL http://www.uwinnipeg.ca/~clark and choosing the Programs link. Correspondence concerning this article should be addressed to G. Remillard, Department of Psychology, University of Winnipeg, Winnipeg, MB R3B 2E9, Canada (e-mail: gremilla@uwinnipeg.ca).

Table 1
First-Order Transition Probabilities, Unknowns Representing the Number of Times in *S* that the Unigrams Are Followed and Preceded by an Event, and Frequencies in *S*

	Next Event			
Unigram	1	2	3	4
		Probabilities		
1	0	3/5	2/5	0
2	1/2	0	0	1/2
3	1/2	0	0	1/2
4	0	2/5	3/5	0
		Unknowns		
1	0	(3/5)*x*1	(2/5)*x*1	0
2	(1/2)*x*2	0	0	(1/2)*x*2
3	(1/2)*x*3	0	0	(1/2)*x*3
4	0	(2/5)*x*4	(3/5)*x*4	0
		Frequencies		
1	0	6	4	0
2	5	0	0	5
3	5	0	0	5
4	0	4	6	0

Table 2
Second-Order Transition Probabilities, Unknowns Representing the Number of Times in *S* that the Bigrams Are Followed and Preceded by an Event, and Frequencies in *S*

	Next Event			
Bigram	1	2	3	4
		Probabilities		
1–2	3/5	0	0	2/5
1–3	2/5	0	0	3/5
2–1	0	3/5	2/5	0
2–4	0	2/5	3/5	0
3–1	0	3/5	2/5	0
3–4	0	2/5	3/5	0
4–2	2/5	0	0	3/5
4–3	3/5	0	0	2/5
		Unknowns		
1–2	(3/5)*x*1	0	0	(2/5)*x*1
1–3	(2/5)*x*2	0	0	(3/5)*x*2
2–1	0	(3/5)*x*3	(2/5)*x*3	0
2–4	0	(2/5)*x*4	(3/5)*x*4	0
3–1	0	(3/5)*x*5	(2/5)*x*5	0
3–4	0	(2/5)*x*6	(3/5)*x*6	0
4–2	(2/5)*x*7	0	0	(3/5)*x*7
4–3	(3/5)*x*8	0	0	(2/5)*x*8
		Frequencies		
1–2	117	0	0	78
1–3	52	0	0	78
2–1	0	99	66	0
2–4	0	60	90	0
3–1	0	96	64	0
3–4	0	60	90	0
4–2	48	0	0	72
4–3	108	0	0	72

in which not all transition probabilities are equal (e.g., Tables 1–3). In the absence of a general method for constructing sequences that satisfy such *n*th-order transition probability matrices, researchers may resort to trial and error, limit themselves to transition probability matrices for which sequences are easy to construct, or implement the transition probabilities as the sequence is being presented to participants so that, in the long run, probabilities will match those in the transition probability matrix.

The present article describes a general procedure for generating sequences of shortest possible length that satisfy *any* given *n*th-order transition probability matrix. In producing sequences that satisfy an *n*th-order transition probability matrix, two types of sequences must be distinguished. One type of sequence begins and ends with the same *n*-gram. The other type of sequence begins and ends with different *n*-grams. This article will focus on generating sequences of the first type.

The current article consists of three main sections. Assume that *S* is a sequence of shortest possible length that begins and ends with the same *n*-gram and that satisfies the given *n*th-order transition probability matrix. The first section describes the process for determining the frequencies in *S* with which each of the *m* distinct events follows each of the *n*-grams. The second section describes a sequence-generation algorithm that uses the frequencies to generate sequences of shortest possible length that begin and end with the same *n*-gram and that satisfy the given *n*th-order transition probability matrix. The final section describes a computer program that implements the sequence-generation algorithm.

Determining the Frequencies in *S*

Since *S* begins and ends with the same *n*-gram, it follows that, for every *n*-gram n_i in *S*, the number of times

Table 3
Third-Order Transition Probabilities, Unknowns Representing the Number of Times in *S* that the Trigrams Are Followed and Preceded by an Event, and Frequencies in *S* (in Square Brackets)

	Next Event		
Trigram	1	2	3
1–1–1	(3/6)*x*1 [54]	(2/6)*x*1 [36]	(1/6)*x*1 [18]
1–1–2	(1/3)*x*2 [24]	(1/3)*x*2 [24]	(1/3)*x*2 [24]
1–1–3	(3/6)*x*3 [18]	(2/6)*x*3 [12]	(1/6)*x*3 [6]
1–2–1	(3/6)*x*4 [24]	(2/6)*x*4 [16]	(1/6)*x*4 [8]
1–2–2	(1/3)*x*5 [16]	(1/3)*x*5 [16]	(1/3)*x*5 [16]
1–2–3	(3/6)*x*6 [24]	(2/6)*x*6 [16]	(1/6)*x*6 [8]
1–3–1	(3/6)*x*7 [18]	(2/6)*x*7 [12]	(1/6)*x*7 [6]
1–3–2	(1/3)*x*8 [8]	(1/3)*x*8 [8]	(1/3)*x*8 [8]
1–3–3	(3/6)*x*9 [6]	(2/6)*x*9 [4]	(1/6)*x*9 [2]
2–1–1	(3/6)*x*10 [27]	(2/6)*x*10 [18]	(1/6)*x*10 [9]
2–1–2	(1/3)*x*11 [12]	(1/3)*x*11 [12]	(1/3)*x*11 [12]
2–1–3	(3/6)*x*12 [9]	(2/6)*x*12 [6]	(1/6)*x*12 [3]
2–2–1	(3/6)*x*13 [18]	(2/6)*x*13 [12]	(1/6)*x*13 [6]
2–2–2	(1/3)*x*14 [12]	(1/3)*x*14 [12]	(1/3)*x*14 [12]
2–2–3	(3/6)*x*15 [18]	(2/6)*x*15 [12]	(1/6)*x*15 [6]
2–3–1	(3/6)*x*16 [27]	(2/6)*x*16 [18]	(1/6)*x*16 [9]
2–3–2	(1/3)*x*17 [12]	(1/3)*x*17 [12]	(1/3)*x*17 [12]
2–3–3	(3/6)*x*18 [9]	(2/6)*x*18 [6]	(1/6)*x*18 [3]
3–1–1	(3/6)*x*19 [27]	(2/6)*x*19 [18]	(1/6)*x*19 [9]
3–1–2	(1/3)*x*20 [12]	(1/3)*x*20 [12]	(1/3)*x*20 [12]
3–1–3	(3/6)*x*21 [9]	(2/6)*x*21 [6]	(1/6)*x*21 [3]
3–2–1	(3/6)*x*22 [12]	(2/6)*x*22 [8]	(1/6)*x*22 [4]
3–2–2	(1/3)*x*23 [8]	(1/3)*x*23 [8]	(1/3)*x*23 [8]
3–2–3	(3/6)*x*24 [12]	(2/6)*x*24 [8]	(1/6)*x*24 [4]
3–3–1	(3/6)*x*25 [9]	(2/6)*x*25 [6]	(1/6)*x*25 [3]
3–3–2	(1/3)*x*26 [4]	(1/3)*x*26 [4]	(1/3)*x*26 [4]
3–3–3	(3/6)*x*27 [3]	(2/6)*x*27 [2]	(1/6)*x*27 [1]

Table 4
First-, Second-, and Third-Order Transition Probability Matrices Where Each of Four, Three, and Two Events Follows Each of the Four, Nine, and Eight Possible Unigrams, Bigrams, and Trigrams with Probability 1/4, 1/3, and 1/2, Respectively

	Next Event			
Sequence	1	2	3	4
Unigram				
1	1/4	1/4	1/4	1/4
2	1/4	1/4	1/4	1/4
3	1/4	1/4	1/4	1/4
4	1/4	1/4	1/4	1/4
Bigram				
1–1	1/3	1/3	1/3	
1–2	1/3	1/3	1/3	
1–3	1/3	1/3	1/3	
2–1	1/3	1/3	1/3	
2–2	1/3	1/3	1/3	
2–3	1/3	1/3	1/3	
3–1	1/3	1/3	1/3	
3–2	1/3	1/3	1/3	
3–3	1/3	1/3	1/3	
Trigram				
1–1–1	1/2	1/2		
1–1–2	1/2	1/2		
1–2–1	1/2	1/2		
1–2–2	1/2	1/2		
2–1–1	1/2	1/2		
2–1–2	1/2	1/2		
2–2–1	1/2	1/2		
2–2–2	1/2	1/2		

that n_i is followed by an event is equal to the number of times that n_i is preceded by an event. For example, in the sequence 1–2–4–3–2–2–1–4–3–3–1–2, which begins and ends with the bigram 1–2, the bigram 4–3 is followed by an event twice and preceded by an event twice. Similarly, the bigram 1–2 is followed by an event once and preceded by an event once. The above property of S will be used to determine the frequencies in S with which each of the m distinct events follows each of the n-grams.

To illustrate the process of determining frequencies in S, consider the first-order transition probability matrix in the top panel of Table 1. As a first step, four unknowns, $x1, x2, x3$, and $x4$, are introduced that represent the number of times in S that the unigrams 1, 2, 3, and 4, respectively, are followed and preceded by an event (see the middle panel of Table 1). Consequently, 2 and 3 follow the unigram 1 $(3/5)x1$ and $(2/5)x1$ times, respectively. Similarly for the other rows. By the first column and the rows corresponding to unigrams 2 and 3, events 2 and 3 precede the unigram 1 $(1/2)x2$ and $(1/2)x3$ times, respectively. Thus, $x1 = (1/2)x2 + (1/2)x3$. Setting each of the four unknowns equal to its corresponding column sum results in the following system of four equations in four unknowns:

$$\begin{aligned}
(1)\quad & x1 = (1/2)x2 + (1/2)x3\\
(2)\quad & x2 = (3/5)x1 + (2/5)x4\\
(3)\quad & x3 = (2/5)x1 + (3/5)x4\\
(4)\quad & x4 = (1/2)x2 + (1/2)x3.
\end{aligned}$$

The general solution to the system of equations in terms of $x4$ is $x1 = x4$, $x2 = x4$, and $x3 = x4$, where $x4$ can be any number (i.e., $x4$ is arbitrary).[1] In reality, $x4$ cannot be any number. It is a frequency and must, therefore, be a positive integer. Moreover, it must be a multiple of two and five, since $(1/2)x2 = (1/2)x4$ and $(3/5)x1 = (3/5)x4$ are also frequencies. Since S is a sequence of shortest possible length, $x4$ must be the smallest number that is a multiple of two and five.[2] Setting $x4 = 10$, the remaining unknowns become $x1 = 10$, $x2 = 10$, and $x3 = 10$. In the bottom panel of Table 1, the unknowns are replaced with their numerical values. These are the frequencies in S with which each of the four events follows each of the four unigrams.

As a second illustration, consider the second-order transition probability matrix in the top panel of Table 2. Eight unknowns—$x1, x2, x3, x4, x5, x6, x7$, and $x8$—are introduced that represent the number of times in S that the eight bigrams 1–2, 1– 3, 2–1, 2–4, 3–1, 3–4, 4–2, and 4–3, respectively, are followed and preceded by an event (see the middle panel of Table 2). Consequently, 1 and 4 follow the bigram 1–2 $(3/5)x1$ and $(2/5)x1$ times, respectively. Similarly for the other rows. By the second column and the rows corresponding to bigrams 2–1 and 3–1, events 2 and 3 precede the bigram 1–2 $(3/5)x3$ and $(3/5)x5$ times, respectively. Thus, $x1 = (3/5)x3 + (3/5)x5$. Setting each of the eight unknowns equal to its corresponding column sum results in the following system of eight equations in eight unknowns:

$$\begin{aligned}
(1)\quad & x1 = (3/5)x3 + (3/5)x5\\
(2)\quad & x2 = (2/5)x3 + (2/5)x5\\
(3)\quad & x3 = (3/5)x1 + (2/5)x7\\
(4)\quad & x4 = (2/5)x1 + (3/5)x7\\
(5)\quad & x5 = (2/5)x2 + (3/5)x8\\
(6)\quad & x6 = (3/5)x2 + (2/5)x8\\
(7)\quad & x7 = (2/5)x4 + (2/5)x6\\
(8)\quad & x8 = (3/5)x4 + (3/5)x6.
\end{aligned}$$

The general solution to the system of equations in terms of $x8$ is $x1 = (13/12)x8$, $x2 = (13/18)x8$, $x3 = (11/12)x8$, $x4 = (5/6)x8$, $x5 = (8/9)x8$, $x6 = (5/6)x8$, and $x7 = (2/3)x8$, where $x8$ is arbitrary. Clearly, $x8$ must be a multiple of 3, 6, 9, 12, and 18. The unknown, $x8$, must also be a multiple of 5, since $(3/5)x8$ is a frequency. Since S is a sequence of shortest possible length, $x8$ must be the smallest number that is a multiple of 3, 5, 6, 9, 12, and 18. Setting $x8 = 180$, the remaining unknowns become $x1 = 195, x2 = 130, x3 = 165, x4 = 150, x5 = 160, x6 = 150$, and $x7 = 120$. In the bottom panel of Table 2, the unknowns are replaced with their numerical values. These are the frequencies in S with which each of the four events follows each of the eight bigrams.

As a final illustration of the process of determining frequencies in S, consider the third-order transition probability matrix in Table 3. Twenty-seven unknowns, $x1, x2, \ldots, x27$, are introduced that represent the number of times in S that the trigrams 1–1–1, 1–1–2, . . . , 3–3–3, respectively, are followed and preceded by an event. Con-

sequently, 1, 2, and 3 follow the trigram 1–1–1 $(3/6)x1$, $(2/6)x1$, and $(1/6)x1$ times, respectively, and similarly for the other rows. By the first column and the rows corresponding to trigrams 1–1–1, 2–1–1, and 3–1–1, events 1, 2, and 3 precede the trigram 1–1–1 $(3/6)x1$, $(3/6)x10$, and $(3/6)x19$ times, respectively. Thus $x1 = (3/6)x1 + (3/6)x10 + (3/6)x19$. Setting each of the 27 unknowns equal to its corresponding column sum results in the following system of 27 equations in 27 unknowns:

$$
\begin{aligned}
&(1) \quad x1 = (3/6)x1 + (3/6)x10 + (3/6)x19\\
&(2) \quad x2 = (2/6)x1 + (2/6)x10 + (2/6)x19\\
&(3) \quad x3 = (1/6)x1 + (1/6)x10 + (1/6)x19\\
&(4) \quad x4 = (1/3)x2 + (1/3)x11 + (1/3)x20\\
&\quad\ \vdots\\
&(26) \quad x26 = (2/6)x9 + (2/6)x18 + (2/6)x27\\
&(27) \quad x27 = (1/6)x9 + (1/6)x18 + (1/6)x27.
\end{aligned}
$$

The general solution to the system of equations in terms of $x27$ is $x1 = (18)x27$, $x2 = (12)x27$, $x3 = (6)x27$, $x4 = (8)x27$, . . . , $x26 = (2)x27$, where $x27$ is arbitrary. The unknown $x27$ must be a multiple of six, since $(1/6)x27$ is a frequency. Since S is a sequence of shortest possible length, $x27$ must be the smallest number that is a multiple of six. Setting $x27 = 6$, the remaining unknowns become $x1 = 108$, $x2 = 72$, $x3 = 36$, $x4 = 48$, . . . , $x26 = 12$. Replacing the unknowns with their numerical values yields the numbers in square brackets in Table 3. These are the frequencies in S with which each of the three events follows each of the 27 trigrams. It is possible to go on to define fourth- or higher order transition probability matrices and determine the frequencies in S.

If S actually exists, the above procedure will determine the frequencies in S with which each of the m distinct events follows each of the n-grams. An important question to ask, therefore, is how does one know if S actually exists. As it turns out, an answer is provided by the form of the solution to the system of equations. A proof of the following theorem appears in the Appendix.

Theorem 1. There exists a sequence beginning and ending with the same n-gram that satisfies the given nth-order transition probability matrix if and only if the solution to the system of equations is of the form $x_i = (p_i/q_i)x_a$, $i = 1, 2, \ldots, r$, where r is the number of unknowns, p_i and q_i are positive integers, and x_a is the arbitrary unknown.

The general solution to each of the above systems of equations was such that one unknown was arbitrary (i.e., $x4$ in the first system, $x8$ in the second system, and $x27$ in the third system), and all unknowns were expressed in terms of the arbitrary unknown. Therefore, by Theorem 1, S exists for the first-, second-, and third-order transition probability matrices in Tables 1, 2, and 3, respectively. As an example of a situation for which S does not exist, consider the first-order transition probability matrix in the top panel of Table 5. The general solution to the system of equations is $x1 = x2$ and $x3 = x4$, where $x2$ and $x4$ are arbitrary. Clearly, the form of the solution does not match that in Theorem 1, and so S does not exist for the first-order transition probability matrix.

Table 5
An Unsatisfiable Transition Probability Matrix

	Next Event			
Unigram	1	2	3	4
	Probabilities			
1	1/2	1/2	0	0
2	1/2	1/2	0	0
3	0	0	1/2	1/2
4	0	0	1/2	1/2
	Unknowns			
1	$(1/2)x1$	$(1/2)x1$	0	0
2	$(1/2)x2$	$(1/2)x2$	0	0
3	0	0	$(1/2)x3$	$(1/2)x3$
4	0	0	$(1/2)x4$	$(1/2)x4$

If the third row in the top panel of Table 5 was 0, 1/2, 1/2, and 0, rather than 0, 0, 1/2, and 1/2, the general solution to the system of equations would be $x1 = x2$, $x3 = 0$, and $x4 = 0$, where $x2$ is arbitrary. Again, the form of the solution does not match that in Theorem 1, so S does not exist for the revised first-order transition probability matrix.

As a final note, when nth-order transition probability matrices are like those in Table 4, where each of m events is to follow each of the m^n possible n-grams with probability $1/m$, a system of equations does not have to be set up and solved to determine the frequencies in S with which each of the m events follows each of the n-grams. The frequency matrix would simply consist of the value one in each cell.

A computer program to set up the equations. The first step in determining the frequencies in S with which each of the m distinct events follows each of the n-grams is to set up a system of equations, as was described above. The Turbo Pascal (Version 5.0) program we developed will take a transition probability matrix as input and generate the required system of r equations in r unknowns $(x1, x2, \ldots, xr)$ where r is the number of rows in the probability matrix. The unknowns $x1, x2, \ldots, xr$ represent, respectively, the number of times in S that the n-grams associated with rows 1, 2, . . . , r are followed and preceded by an event. Thus, the program will generate equations just like those described in the previous section.

To use the program, an input file containing the transition probability matrix must first be created. The input files for the transition probability matrices in Tables 1 and 2 appear in the left column of Table 6. The row labels (i.e., n-grams) are entered first, followed by the transition probabilities for each row. Note that each probability is specified as a pair of numbers separated by a space. Thus, 3/5 is specified as 3 5, and 0 is specified as 0 0. The user next enters five values at the beginning of the source code. The five values are nrows and ncols (the number of rows and columns in the transition probabil-

Table 6
Input Files for the Transition Probability and Frequency Matrices in Tables 1 and 2

Probability				Frequency			
Table 1 Matrices							
1				1			
2				2			
3				3			
4				4			
0 0	3 5	2 5	0 0	0 6	4 0		
1 2	0 0	0 0	1 2	5 0	0 5		
1 2	0 0	0 0	1 2	5 0	0 5		
0 0	2 5	3 5	0 0	0 4	6 0		
Table 2 Matrices							
1 2				1 2			
1 3				1 3			
2 1				2 1			
2 4				2 4			
3 1				3 1			
3 4				3 4			
4 2				4 2			
4 3				4 3			
3 5	0 0	0 0	2 5	117	0	0	78
2 5	0 0	0 0	3 5	52	0	0	78
0 0	3 5	2 5	0 0	0	99	66	0
0 0	2 5	3 5	0 0	0	60	90	0
0 0	3 5	2 5	0 0	0	96	64	0
0 0	2 5	3 5	0 0	0	60	90	0
2 5	0 0	0 0	3 5	48	0	0	72
3 5	0 0	0 0	2 5	108	0	0	72

ity matrix), lsize (the size of the row labels, i.e., *n*), infile (the path and name of the input file), and outfile (the path and name of the output file where equations are to be stored). The values of nrows, ncols, and lsize would be, respectively, 4, 4, and 1 for the transition probability matrix in Table 1; 8, 4, and 2 for the matrix in Table 2; and 27, 3, and 3 for the matrix in Table 3. When the program is run, the message "Setting up equations" is displayed, together with a counter that counts down from nrows to zero.

Once a system of equations has been set up, the system can be used as input to a program designed to solve systems of equations (see note 1). The solution to the system of equations is then examined, to determine whether *S* actually exists (see Theorem 1). If *S* exists, then, as outlined in the previous section, the solution is used to establish a frequency matrix describing the frequencies in *S* with which each of the *m* distinct events follows each of the *n*-grams.

The Sequence-Generation Algorithm

Having ascertained that *S* exists and having determined the frequencies in *S*, the sequence-generation algorithm can then be used. The algorithm uses the frequencies in *S* to generate sequences of shortest possible length that begin and end with the same *n*-gram and that satisfy the given *n*th-order transition probability matrix. The sequence-generation algorithm is illustrated, using the frequency matrix in the bottom panel of Table 1. First, a starting unigram (i.e., a starting row) is chosen, say row(1).[3] Thus, 1 becomes the 1st element in the sequence. The next element is chosen without replacement from row(1)—say, 3. Thus, the sequence is now 1–3. The next element is then chosen without replacement from row(3)—say, 4. The sequence is now 1–3–4. The process is continued in this manner until all 40 entries in the frequency matrix have been chosen. The final length of the sequence would be 41 elements, and it would be a sequence of shortest possible length that begins and ends with the same unigram (see below) and that satisfies the first-order transition probability matrix in Table 1.

As another illustration of the sequence-generation algorithm, the frequency matrix in the bottom panel of Table 2 is considered. First, a starting bigram (i.e., a starting row) is chosen—say, row(2–1). Thus, 2–1 are the first 2 elements in the sequence. The next element is chosen without replacement from row(2–1)—say, 3. The sequence is now 2–1–3. The next element is then chosen without replacement from row(1–3), say, 1. The sequence is now 2–1–3–1. The process is continued in this manner until all 1,250 entries in the frequency matrix have been chosen. The final length of the sequence would be 1,252 elements, and it would be a sequence of shortest possible length that begins and ends with the same bigram (see below) and that satisfies the second-order transition probability matrix in Table 2.

As a final illustration, the frequencies in Table 3 are considered. First, a starting trigram (i.e., a starting row) is chosen—say, row(1–3–2). Thus, 1–3–2 are the first 3 elements in the sequence. The next element is chosen without replacement from row(1–3–2)—say, 3. The sequence is now 1–3–2–3. The next element is then chosen without replacement from row(3–2–3)—say, 1. The sequence is now 1–3–2–3–1. The process is continued in this manner until all 972 entries in the frequency matrix have been chosen. The final length of the sequence would be 975 elements, and it would be a sequence of shortest possible length that begins and ends with the same trigram (see below) and that satisfies the third-order transition probability matrix in Table 3.

The sequence-generation algorithm will always terminate at the starting row where an element is to be chosen but there are none left to choose. This is a consequence of the fact that, for every *n*-gram n_i, the number of times that n_i is followed by an event is equal to the number of times that n_i is preceded by an event. Thus, a sequence will begin and end with the same *n*-gram. For example, if row(1) was the starting row in Table 1, the sequence will start and end with the unigram 1. If row(2–1) was the starting row in Table 2, the sequence will begin and end with the bigram 2–1. Similarly, if row(1–3–2) was the starting row in Table 3, the sequence will begin and end with the trigram 1–3–2.

Often, the sequence-generation algorithm will terminate without having chosen every entry in the frequency matrix. That is, an element must be chosen from the starting row, and there are none left to choose, but there

are still some unchosen elements in other rows. One option is to restart the algorithm and choose elements in a different order. However, such a brute force approach can be inefficient, because there can be numerous failures and considerable processing (e.g., many elements already chosen) prior to each failure. The following theorem states a necessary and sufficient condition ensuring the algorithm's success.

Theorem 2. Let row(n_k) be the starting row, and for each row(n_i), let last(n_i) be the last entry in row(n_i) to be chosen by the sequence-generation algorithm. The algorithm will choose every entry in the frequency matrix if and only if, for each row(n_i), there exists a *path* from row(n_i) to row(n_k) via last entries.

To illustrate the theorem, consider the frequency matrix in Table 1. If last(1) = 2, last(2) = 4, last(3) = 4, and last(4) = 3 and row(3) is the starting row, the algorithm will choose every entry in the matrix. This is because, for each row(i), there exists a path from row(i) to row(3) via last entries. For example, 1–2–4–3 is a path from row(1) to row(3) via the last entries last(1) = 2, last(2) = 4, and last(4) = 3. If the starting row was row(2), however, it would be impossible for the algorithm to choose every entry in the matrix, because there does not exist a path from row(4) to row(2) via last entries [last(4) = 3 and last(3) = 4].

As another illustration of the theorem, consider the frequency matrix in Table 2. If last(1–2) = 4, last(1–3) = 4, last(2–1) = 2, last(2–4) = 3, last(3–1) = 3, last(3–4) = 2, last(4–2) = 1, and last(4–3) = 4, and row(2–1) is the starting row, the algorithm will choose every entry in the matrix. This is because, for each row(i–j), there exists a path from row(i–j) to row(2–1) via last entries. For example, 1–2–4–3–4–2–1 is a path from row(1–2) to row(2–1) via the last entries last(1– 2) = 4, last(2–4) = 3, last(4–3) = 4, last(3–4) = 2, and last(4– 2) = 1. If the starting row was row(3–1), however, there is no path from row(2–4) to row(3–1) via last entries [last(2–4) = 3, last(4–3) = 4, last(3–4) = 2, last(4–2) = 1, last(2–1) = 2, and last(1–2) = 4], so the algorithm would terminate without choosing every entry in the frequency matrix. A proof of Theorem 2 appears in the Appendix.

Determining last entries. Theorem 2 suggests one way of making the sequence-generation process more efficient. Prior to employing the sequence-generation algorithm, a starting row(n_k) is chosen, and last entries are selected, so that the condition in Theorem 2 is satisfied. The sequence-generation algorithm will then be guaranteed to succeed. Although the method of determining last entries described below can have numerous failures and restarts, there is little processing prior to each failure, since only last entries are considered. Consequently, the determination of last entries followed by the sequence-generation algorithm is more efficient than the pure brute force approach described earlier, in which the sequence-generation algorithm is continually restarted. As we shall see in the next section, the former can be considerably more efficient than the latter.

The method of determining last entries proceeds in cycles. Failure on any cycle causes the process to return to the first cycle. The first cycle tries to find a path from the first row to the starting row(n_k) via last entries. For example, row(1–2) in Table 2 is the first row. Choose a last entry for row(1–2)—say, last(1–2) = 4. Next, choose a last entry for row(2–4)—say, last(2–4) = 3. Now, choose a last entry for row(4–3), and so on. Last entries are chosen in this manner until the resulting path leads either to the starting row(n_k) or to a dead end (i.e., to a row whose last entry was chosen on the current cycle). If it leads to a dead end, the first cycle is restarted. If the path leads to the starting row, a second cycle is begun with some row(n_j) for which a last entry has not yet been chosen. The second cycle tries to find a path, via last entries, from row(n_j) either to the starting row(n_k) or to a row whose last entry was chosen on a previous cycle, since there is a path from that row to the starting row via last entries. If the second cycle leads to a dead end (i.e., to a row whose last entry was chosen on the current cycle), the process returns to the first cycle. If the second cycle is successful, there is a third cycle that tries to find a path, via last entries, from some row(n_i), for which a last entry has not yet been chosen, either to the starting row(n_k) or to a row whose last entry was chosen on a previous cycle. The process stops when last entries have been chosen for all the rows. The result is that, for each row(n_i), there exists a path from row(n_i) to the starting row(n_k) via last entries. The sequence-generation algorithm is then guaranteed to succeed.[4]

The Sequence-Generation Program

The sequence-generation program implements both the brute force approach and the last-entries approach, as described in the preceding section. The user first creates an input file containing the frequency matrix. The input files for the frequency matrices in Tables 1 and 2 appear in the right column of Table 6. The row labels (i.e., *n*-grams) are entered first, followed by the frequencies for each row. The user next enters seven values at the beginning of the source code. The seven values are nrows and ncols (the number of rows and columns in the frequency matrix), lsize (the size of the row labels, i.e., *n*), nseqs (the number of sequences to be generated), infile (the path and name of the input file), outfile (the path and name of the output file where sequences are to be stored), and method (1 for the brute force approach, and 2 for the last-entries approach). The values of nrows, ncols, and lsize would be, respectively, 4, 4, and 1 for the frequency matrix in Table 1; 8, 4, and 2 for the matrix in Table 2; and 27, 3, and 3 for the matrix in Table 3.

The program has six main data structures that are dynamically allocated. Since it is easier to work with row numbers than with row labels, the rl array contains the row labels and is used to convert row labels to row numbers, and vice versa. For example, row(2–4) in Table 2 would be converted to row(4). The su array contains the successors of each row in the frequency matrix. If row(j)

succeeds row(i) q times in the frequency matrix, then j will appear q times in the ith row of su. For example, in the frequency matrix in Table 2, row(3) [i.e., row(2–1)] succeeds row(1) 117 times. Consequently, the number 3 would appear 117 times in the first row of su. The tsu array keeps track of the total number of entries in each row of su, and the isu array is an index for each row in su. Finally, the sq array contains the generated sequence, and the la array is used in the method of determining last entries to keep track of which rows were involved in which cycles.

When the program is run, it first creates the various arrays. This is signaled by the message "Creating arrays" and a counter that counts down from nrows to zero. When the arrays have been created, the message "Generating sequences" is displayed, together with a counter that counts up from zero to nseqs.

The brute force approach generates a sequence by performing three tasks consecutively. First, a starting row is chosen at random. Next, the elements in each row of the su array are randomly permuted (the rearrange_su_1 procedure). Finally, the sequence-generation algorithm is employed, using the su array, and the resulting sequence placed in the sq array (the output_sequence_1 procedure). If the algorithm fails (i.e., it does not choose every element in the su array), the process is restarted at step 1, where a starting row is chosen at random. If the algorithm succeeds, a random location in the sq array is chosen, and the sequence is written to the output file, beginning at the randomly selected location. Thus, the outputted sequence is a cyclic shift of the sequence in the sq array. This is done to safeguard against possible biases in the elements that might tend to appear early and late in a sequence. Also, in writing the sequence to the output file, row numbers are converted back to row labels, using the rl array.

The last-entries approach also generates a sequence by performing three tasks consecutively. First, a starting row is chosen at random, and the method of determining last entries is implemented (the est_last_entries procedure). Last entries are always chosen at random. Next, all but the last entries in each row of the su array are randomly permuted (the rearrange_su_2 procedure). Finally, the sequence-generation algorithm is employed, using the su array, and the resulting sequence placed in the sq array (the output_sequence_2 procedure). From the sq array, the sequence is written to the output file, again with a random cyclic shift and with row numbers being converted back to row labels.

Program efficiency. In writing the program, we tried to minimize, as best we could, the number of operations and the memory capacity required to generate a sequence. With respect to memory requirements, the six data structures require 2*(lsize + 8)*nrows + 4*E bytes, where E is the total number of entries in the frequency matrix.[5] For the frequency matrix in Table 2, the data structures would require 2*(2 + 8)*8 + 4*1,250 = 5,160 bytes.

To test the program, we gave it a large frequency matrix in which each of five events followed each of the 3,125 possible five-grams precisely once. Thus, the frequency matrix consisted of 3,125 rows, five columns, and the value one in each of the 15,625 cells. Running the program five times on a Pentium 133-MHz machine, the last-entries approach required 15–30 sec each time to generate ten 15,630-element sequences, whereas the brute force approach required 15–30 min each time. Thus, the former is considerably more efficient than the latter. With respect to memory, the program's data structures required 2*(5 + 8)*3125 + 4*15,625 = 143,750 bytes. In contrast, Emerson and Tobias's (1995) program would have required 39.2 million bytes of memory.[6] Thus, the current program is extremely efficient memorywise.

Although the last-entries approach is more efficient than the brute force approach, the latter is somewhat better at randomly generating sequences (see the Randomness of the Sequences Generated section in the Appendix). However, the bias in the sequences generated by the last-entries approach is slight and seems to occur only under some circumstances. Thus, the last-entries approach would probably be sufficient for most psychological applications.

Conclusion

Given any nth-order transition probability matrix, the present article has described a general procedure for generating sequences of shortest possible length that begin and end with the same n-gram and that satisfy the given transition probability matrix. First, the probability matrix is used as input to the first program we developed, which sets up a system of r equations in r unknowns, where r is the number of rows in the probability matrix, and each unknown, $x1, x2, \ldots, xr$, represents, respectively, the number of times in S that the n-gram associated with row $1, 2, \ldots, r$ is followed and preceded by an event. Next, the system of equations is used as input to a program designed to solve systems of equations. The solution is then examined, to determine whether S actually exists (see Theorem 1). If S exists, the solution is used to establish a frequency matrix describing the frequencies in S with which each of the m distinct events follows each of the n-grams. Finally, the frequency matrix is used as input to the second program we developed, which generates sequences of shortest possible length that begin and end with the same n-gram and that satisfy the given nth-order transition probability matrix.

Although the focus has been on generating sequences of shortest possible length, longer sequences can be generated by taking the smallest possible value of the arbitrary unknown (e.g., $x4 = 10$ in the first system of equations, $x8 = 180$ in the second system of equations, $x27 = 6$ in the third system of equations) and multiplying it by some positive integer. This is equivalent to multiplying the frequency matrix for S by the positive integer. For example, the entries in the frequency matrix in Table 1 could

be multiplied by the integer 3 to generate sequences that are 121 elements in length.

REFERENCES

Bertelson, P. (1961). Sequential redundancy and speed in a serial two-choice responding task. *Quarterly Journal of Experimental Psychology*, **13**, 90-102.

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.

Even, S. (1979). *Graph algorithms*. Rockville, MD: Computer Science Press.

Gibbons, A. (1985). *Algorithmic graph theory*. New York: Cambridge University Press.

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

Kornblum, S. (1969). Sequential determinants of information processing in serial and discrete choice reaction time. *Psychological Review*, **76**, 113-131.

Remillard, G., & Clark, J. M. (1999). *Implicit learning of first-, second-, and third-order transition probabilities*. Manuscript submitted for publication.

Schvaneveldt, R. W., & Gomez, R. L. (1996, October). *Attention and modes of learning: Evidence from probabilistic sequences*. Paper presented at the 37th Annual Meeting of the Psychonomic Society, Chicago.

NOTES

1. There are a number of commercially available software packages that solve systems of equations. Equations in the present article were solved with Maple V (Release 4).

2. Maple V can also determine the smallest common multiple of a set of given numbers.

3. The starting row is irrelevant. If S begins and ends with the n-gram n_i, there is also a sequence beginning and ending with the n-gram n_k not equal n_i that satisfies the given nth-order transition probability matrix. A cyclic shift of S beginning with n_k would be such a sequence. For example, 3–4–1–2–3 is a cyclic shift of 1–2–3–4–1 beginning with 3.

4. There are more efficient ways of determining last entries so that the condition in Theorem 2 is satisfied. However, we have found that some of these ways can be considerably more biased than the current approach. That is, the more efficient methods are less likely to yield sequences that represent a random sample from the space of all the possible sequences.

5. If the data structures require more memory than is available, the program will terminate with a "heap overflow" error message.

6. The value of 39.2 million bytes was calculated, using Emerson and Tobias's (1995) equation for needed memory capacity, $4m^{n-1}[m^{n-1} + 2(fm + 2)]$, with $m = 5$, $n = 6$, and $f = 1$.

APPENDIX

The proofs that follow make use of graph theory. Textbooks by Even (1979) and Gibbons (1985) provide the necessary background.

Proof of Theorem 1

Before beginning the proof, the following is defined: Given an nth-order transition probability matrix, the n-gram n_j is a successor of the n-gram n_i (or n_i is a predecessor of n_j) if and only if the last $n-1$ elements of n_i are the same as the first $n-1$ elements of n_j and the last element of n_j is an event that can follow n_i. For example, in Table 2, the bigram 2–4 is a successor of the bigram 1–2. In Table 1, the unigram 2 is a successor of the unigram 1. As a final example, the trigram 1–2–1 in Table 3 is a successor of the trigram 1–1–2.

Let $x_1, x_2, \ldots, x_r$ be the unknowns in the system of equations. x_i is the number of times that the n-gram n_i is succeeded and preceded by an n-gram. $x_i = 0$ (if n_i has no predecessors), or

$$x_i = c_{i1}x_{i1} + c_{i2}x_{i2} + \ldots + c_{ik}x_{ik}, \qquad \text{(A1)}$$

where $n_{i1}, n_{i2}, \ldots, n_{ik}$ are the predecessors of n_i, and $0 < c_{ij} \leq 1$ is the probability that n_i succeeds n_{ij}. The system of equations is homogeneous. Thus, there exists a solution. For each i, either $x_i = 0$ or x_i is a linear combination of unknowns designated as arbitrary.

To begin the proof, suppose there exists a sequence beginning and ending with the same n-gram that satisfies the given nth-order transition probability matrix. Therefore, for any two n-grams n_i and n_j, there is a path via successors from n_i to n_j. Also, every n-gram has a predecessor, and so each unknown can be written as in Equation A1. Moreover, in the solution to the system of equations, x_i does not equal 0 for every i, because x_i is the number of times in the sequence that the n-gram n_i is succeeded and preceded by an n-gram. Thus, for $i = 1, 2, \ldots, r$, x_i is a linear combination of at least one arbitrary unknown. Let x_i and x_j be two distinct unknowns. x_i can be expanded as in Equation A1. x_i can be further expanded by expanding the unknowns in the right side of Equation A1. Since there is a path via successors from n_j to n_i, successive expansions of the right side of Equation A1 will eventually include x_j. Thus, x_i depends on x_j. Since any two unknowns depend on one another, the solution to the system of equations can only have one arbitrary unknown. This, coupled with the fact that the coefficients in the system of equations were rational numbers, implies that, for $i = 1, 2, \ldots, r$, $x_i = (p_i/q_i)x_a$ where p_i and q_i are integers and x_a is the arbitrary unknown. Moreover, p_i/q_i must be positive, because x_i is the number of times in the sequence that the n-gram n_i is succeeded and preceded by an n-gram.

Conversely, suppose that the solution to the system of equations is of the form $x_i = (p_i/q_i)x_a$, $i = 1, 2, \ldots, r$, where r is the number of unknowns, p_i and q_i are positive integers, and x_a is the arbitrary unknown. Since none of the unknowns is equal to zero, each was written as in Equation A1. Let n_i and n_j be two n-grams. Now, $x_j = (p_j/q_j)(q_i/p_i)x_i$. Thus, x_j depends on x_i. Also, x_j can be expanded as in Equation A1. Since x_j depends on x_i, successive expansions of the right side of Equation A1 must eventually include x_i. This implies that there is a path via successors from n_i to n_j. Thus, for any two n-grams, there is a path from one to the other via successors.

To determine frequencies, let x_a be a multiple of the q_is and of the denominators of the probabilities in the probability matrix. The resulting frequency matrix is a digraph D, where the r n-grams are the vertices and entries in the frequency matrix indicate the number of edges between vertices. For example, in the bottom panel of Table 2, there are 117 edges from bigram 1–2 to bigram 2–1. For each vertex in D, indegree = outdegree. This is because, for each n-gram n_i, the number of times that n_i is succeeded by an n-gram is equal to the number of times that n_i is preceded by an n-gram. The digraph D is also strongly connected, for there is a path via successors between any two n-grams. Thus, there exists an Eulerian circuit in D. Tracing such a circuit yields a sequence beginning and ending with the same n-gram that satisfies the given nth-order transition probability matrix. This completes the proof.

Proof of Theorem 2

The frequency matrix is a digraph D, where each n-gram is a vertex and the entries in the matrix are the edges. For example, the frequency matrix in Table 1 is a digraph with 4 vertices and 40 edges. Row(1) indicates that there are 6 edges from unigram 1 to unigram 2 and 4 edges from unigram 1 to unigram 3. The frequency matrix in Table 2 is a digraph with 8 vertices and 1,250 edges. Row(1–2) indicates that there are 117 edges from bigram 1–2 to bigram 2–1, and 78 edges from bigram 1–2 to bigram 2–4. As a final example, the frequency matrix in Table 3 is a digraph with 27 vertices and 972 edges. Row(3–2–1) indicates that there are 12, 8, and 4 edges from trigram 3–2–1 to trigrams 2–1–1, 2–1–2, and 2–1–3, respectively.

For every vertex in the digraph D, indegree = outdegree. This is because, for every n-gram n_i, the number of times that n_i is followed by an event is equal to the number of times that n_i is preceded by an event. The sequence-generation algorithm traces a path through D.

To begin the proof, suppose that, for each row(n_i), there exists a path from row(n_i) to row(n_k) via last entries. Therefore, the set of edges [n_i, last(n_i)], excluding the edge [n_k, last(n_k)], constitutes a spanning in-tree with root at n_k. An Eulerian circuit in D can be constructed in a single tracing if the circuit is begun at n_k and edges belonging to the spanning in-tree are chosen last (Even, 1979, pp. 46–47; Gibbons, 1985, pp. 158–160). It follows that the algorithm will choose every entry in the frequency matrix.

Conversely, suppose that the algorithm chooses every entry in the frequency matrix. The algorithm has therefore traced an Eulerian circuit in D beginning and ending at n_k. The edge [n_i, last(n_i)], n_i not equal to n_k, is the last edge leading out of n_i in the circuit. Thus the set of edges [n_i, last(n_i)], n_i not equal to n_k, is a spanning in-tree with root at n_k (Even, 1979; Gibbons, 1985). Consequently, for each row(n_i), there is a path from row(n_i) to row(n_k) via last entries. This completes the proof.

Randomness of the Sequences Generated

To assess whether generated sequences represent a random sample from the space of all possible sequences, the brute force and last-entries approaches were tested on eight different frequency matrices. Five of the eight matrices were similar to those in Tables 1 and 2, in that cell frequencies were unequal. In the remaining three matrices, each of 2, 2, and 3 events followed, respectively, each of the 4, 8, and 3 possible bigrams, trigrams, and unigrams equally often.

For each matrix, M_i, the number of different possible sequences, N_i, that could be generated was calculated, using Hutchinson and Wilf's (1975) formula. Across the eight matrices, N_i ranged from 216 to 2,268. Next, a total of 250*N_i sequences was generated for each matrix. If the brute force and last-entries approaches are unbiased, each of the N_i different sequences should be generated approximately 250 times. For the brute force approach, the pattern of results was identical across the eight matrices. Ninety percent of the N_i different sequences had generation frequencies between 225 and 275, and 100% of the sequences had generation frequencies between 200 and 300. Results using the last-entries approach were similar to those of the brute force approach, except in two cases. For one matrix, 70% of the sequences had generation frequencies between 225 and 275, and for the other matrix, the percentage was 75. However, in both cases, over 95% of the sequences had generation frequencies between 200 and 300. Thus, the brute force approach appears to sample randomly from the space of all the possible sequences, whereas the last-entries approach is slightly biased under some circumstances.

(Manuscript received September 21, 1998;
revision accepted for publication March 1, 1999.)