A Combinatorial Method for Analyzing Sequential Firing Patterns Involving an Arbitrary Number of Neurons Based on Relative Time Order

Albert K. Lee and Matthew A. Wilson

Picower Center for Learning and Memory, RIKEN-MIT Neuroscience Research Center, Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Submitted 24 October 2003; accepted in final form 3 June 2004

INTRODUCTION

Finding meaningful patterns hidden within large amounts of data is now a key problem in many areas of science. In neuroscience, it is believed that information processing in the brain requires coordinated activity across many neurons. However, the code by which multiple neurons interact to perform their computations is still largely unknown and remains one of the fundamental problems in the field. With the recent development of techniques for simultaneously recording the spiking activity of large numbers of individual neurons, the search for complex multicell firing patterns that could help reveal this neural code has become possible. Here we develop a new approach for analyzing sequential firing patterns involving an arbitrary number of neurons based on relative firing order. Specifically, we develop a combinatorial method for quantifying the degree of matching between a “reference sequence” of N distinct “letters” (representing a particular target order of firing by N cells) and an arbitrarily long “word” composed of any subset of those letters including repeats (representing the relative time order of spikes in an arbitrary firing pattern). The method involves computing the probability that a random permutation of the word’s letters would by chance alone match the reference sequence as well as or better than the actual word does, assuming all permutations were equally likely. Lower probabilities thus indicate better matching. The overall degree and statistical significance of sequence matching across a heterogeneous set of words (such as those produced during the course of an experiment) can be computed from the corresponding set of probabilities. This approach can reduce the sample size problem associated with analyzing complex firing patterns. The approach is general and thus applicable to other types of neural data beyond multiple spike trains, such as EEG events or imaging signals from multiple locations. We have recently applied this method to quantify memory traces of sequential experience in the rodent hippocampus during slow wave sleep.

To fully exploit these experimental advances, there is a corresponding need for new methods to analyze complex firing patterns across multiple simultaneous spike trains (i.e., N simultaneous continuous-time discrete stochastic processes). In practice, this may consist of 10–100 simultaneous spike trains (1 train per neuron) each sampled every 0.1 ms, with the sample value “1” (a spike) occurring in each train with average frequency 0.1–10 Hz (the remaining sample values being “0”), and recorded for several hours. The cross-correlation function is a widely used method for detecting interactions between a pair of spike trains. A central peak indicates a tendency for coincident spikes, while an off-center peak indicates a tendency for one cell to fire after the other with a particular delay. However, only a few studies exist that describe ways to analyze interactions among the spike trains of more than 2 neurons (Abeles 1991; Abeles and Gerstein 1988; Abeles et al. 1993; Aertsen et al. 1989; Gerstein and Aertsen 1985; Gerstein and Perkel 1969; Gerstein et al. 1985; Grün et al. 2001; Güntig et al. 2003; Lee and Wilson 2000; Louie and Wilson 2001; Martignon et al. 2000; Nakahara and Amari 2002; Palm et al. 1988). Furthermore, most of these methods analyze firing patterns in terms of a set of precise interspike intervals (i.e., exact time delays between the successive spikes).

We develop a new approach for analyzing sequential firing patterns involving an arbitrary number of neurons based on relative firing order. Sequences are very important for neural processing. Examples include memories of a sequence of experienced events, a sequence of actions in an overall movement, or sequential recruitment of different brain areas during a task, as well as spike sequences in models of pattern recognition (Hopfield 1995). Relative time order of (as opposed to exact time intervals between) events within a sequence is important for dealing with possibilities such as timescaling (both uniform and nonuniform), noise (resulting in variability in timing), and order-sensitive plasticity rules (Bi and Poo 1998; Gustafsson et al. 1987; Levy and Steward 1983; Markram et al. 1997). It can also help reduce the sample size problem associated with analyzing complex patterns based on exact time intervals.

Specifically, we develop a combinatorial method for quantifying the degree of matching between a “reference sequence” of N distinct “letters” (representing a particular target order of firing by N cells) and an arbitrarily long “word” composed of those letters (representing the relative time order of spikes in an arbitrary firing pattern). The method involves computing the

Address for reprint requests and other correspondence: A. K. Lee, E18-301, 50 Ames St., Massachusetts Institute of Technology, Cambridge, MA 02139 (E-mail: albert@cortical.mit.edu).

The costs of publication of this article were defrayed in part by the payment of page charges. The article must therefore be hereby marked “advertisement” in accordance with 18 U.S.C. Section 1734 solely to indicate this fact.

First published June 22, 2004; 10.1152/jn.01030.2003.
probability that a random permutation of the word’s letters would by chance alone match the reference sequence as well as or better than the actual word does, assuming all permutations were equally likely. Lower probabilities thus indicate better matching. For more complex words, we derive formulae for computing upper and lower bounds of this probability. Using the probabilities from across a heterogeneous set of words (such as those produced during the course of an experiment), the overall degree and statistical significance of sequence matching can be computed without the use of Monte Carlo techniques. Words may contain repeated letters and need not contain each letter in the reference sequence. Thus this method can sample whether traces of a particular sequence exist in words that do not necessarily contain every letter (i.e., it naturally handles missing spikes). Variations of our approach emphasizing different aspects of matching are described, as well as variations that modify the equally likely permutation assumption.

Although we have initially motivated our method in terms of single spikes from individual neurons, our approach is general and can be applied to other types of neural data, such as multispike bursts from multiple cells, EEG events from multiple locations, or suprathreshold signals across multiple pixels of an optical or functional magnetic resonance image (fMRI). That is, what we will often refer to as a single spike from a single cell could just as well refer to a burst of spikes from a single cell, a particular pattern from a subset of cells, an EEG event from a given location, or any other type of signal. Thus we describe our approach in abstract terms. Each type of event (e.g., a spike) is captured in the abstract term “letter,” and a sequence of such events from multiple sources is represented by a “word” (i.e., a string of letters with different sources represented by different letters).

We have recently applied our method to quantify memory traces of sequential experience in the rodent hippocampus during subsequent slow wave sleep (SWS) (Lee and Wilson 2000). We found recurring bursts of activity involving up to 6 neurons firing in an order that directly related to the rat’s previous spatial experience. This case illustrates many of the conditions the method was designed to handle: a clear reference sequence (i.e., the sequence of place fields traversed by the rat) that we tested for traces of in sleep, missing letters, repeated letters, errors, and variable interletter and overall timescales.

SUPPORTING METHODS

Hippocampal data

The hippocampal data analyzed here consist of the same set of cells used in our previous work involving the relative-order analysis of hippocampal activity during slow wave sleep (SWS), and the experimental methods are described in detail there (Lee and Wilson 2000). Briefly, an experiment consisted of a rat first sleeping (PRE), then repeatedly running back and forth along a narrow track for chocolate reward at each end (RUN), then sleeping again (POST). A rodent hippocampal “place” cell fires selectively when the animal is in a particular location (the cell’s “place field”) of an environment (Muller 1996; O’Keefe and Dostrovsky 1971). We extracted 2 sets of time periods from RUN, those in which the rat was running in one direction (POS laps), and those in the opposite direction (NEG laps). During each set of laps in a given direction, the rat repeatedly passed through a fixed sequence of place fields of 5 to 11 simultaneously recorded hippocampal CA1 place cells. For the relative-order analysis, each such sequence of cells formed a reference sequence (with cells labeled 1, 2, 3, 4, . . . representing the order in which the peaks of their place fields were traversed, 1 being traversed first), and we tested the activity of these same cells during PRE and POST SWS for traces of this RUN sequence (Lee and Wilson 2000). (Note that for rats running along narrow tracks, many place cells fire only in one direction and those that fire in both directions do not necessarily have fields in the same location. Thus the sequences in each direction generally consist of different sets of cells and are not simply reversed versions of each other.) The results of both the relative-order and exact time interval analysis presented herein represent results pooled across the cells from both directions of 3 rats (1 experiment per rat). All of the hippocampal spike rasters displayed herein come from the POST SWS activity of the 10 simultaneously recorded place cells from one direction of one rat [with reference sequence 1234 . . . and corresponding to the “RAT1” POS direction cells reported in Lee and Wilson (2000)].

Exact time interval analysis

For comparison with our relative-order method, we searched our hippocampal data for all exact time interval firing patterns that satisfied the following criteria: the pattern must consist of spikes from ≥4 distinct cells, must have duration (i.e., the time from the first to last spike) ≥500 ms, and must repeat at least once (i.e., occur at least twice). The first step in our procedure was to perform burst filtering on the spike train of each cell. This involved eliminating all spikes from a cell that followed another spike from that cell by less than max_isi (which was chosen to be 0, 15, or 50 ms), thus leaving only the first spike of each “burst,” as well as isolated single spikes. We then reduced the times of all the remaining spikes to 1-ms precision (from the original 0.1-ms precision). With these remaining spikes, we used a modified version of the algorithm of Abeles and Gerstein (1988) to search for exact time interval patterns that repeated at least once. To allow for some variability in timing, we binned the time shifts between corresponding spikes in each pair of candidate patterns into 5-ms bins (except in the case of max_isi = 0 ms, i.e., no burst filtering) before checking for pattern matching. Finally, for a firing pattern P₂ to be considered to be a repeat of a firing pattern P₁, either their first spikes had to be separated by ≥100 ms, or the last spike of P₁ had to occur before the first spike of P₂. This was done to eliminate repeats that were presumably attributed to longer stretches of continuing activity (up to 100 ms) than were handled by the burst filtering. The search was done separately for each set of cells from one direction of one experiment. For example, for the 10 cells from the POS direction laps of one rat, we searched for repeating patterns in the activity of these 10 cells during POST SWS and the RUN POS laps (but not the RUN NEG laps). Then for the 11 cells from the NEG direction laps of this rat, we searched for repeats in the activity of these 11 cells during POST SWS and the RUN NEG laps. We then pooled the number of repeats found across both directions of the 3 rats.

J Neurophysiol • VOL 92 • OCTOBER 2004 • www.jn.org
**Simulated spike trains**

To illustrate parsing schemes for other types of data (Fig. 5, A and B), spike trains were simulated as follows. A step function representing the strength of input (expressed in terms of the target average firing rate) into the cell as a function of time was created. A Gamma process of order 5, with its rate adjusted to match the current target firing rate, was used to determine the time of the next spike. Each spike was followed by an absolute refractory period of 1 ms, after which time the Gamma process was restarted for the next spike. Time was quantized into 0.01-ms bins.

**MAIN METHOD AND RESULTS**

**The basic problem**

Consider the simultaneous spike trains from N neurons over a given time period. How could one detect and quantify meaningful firing patterns that may occur in this data? One type of meaningful pattern is a sequence of spikes, each one from a different cell. We set out to test whether an arbitrary pattern of spikes from multiple cells (e.g., the spikes fired during a 500-ms window) contains significant matching to a particular sequence of this type.

First, we cast the problem in its most general form. Because (as mentioned above) what we call a single spike from a single cell could represent any of a number of different types of events, we denote each event from a particular source (e.g., a spike fired by cell 1) by a “letter” (e.g., “1”). Because we are concerned with the relative time order of events from multiple sources, the timing of events can be reduced to an ordered string of letters (with different letters representing different sources). Then the basic problem we address can be stated as follows. We start by defining a “reference sequence” as an ordered list of an arbitrary number (N) of distinct (i.e., nonrepeating) “letters.” Then we ask: how can one quantify the degree of matching between a particular reference sequence and an arbitrary string consisting of only those letters? This arbitrary string, which we call a “word,” may contain repeated letters (e.g., a cell may fire more than one spike) and also need not contain all the letters in the reference sequence (e.g., a particular cell may not fire a spike at all), but every letter in a word must appear in the reference sequence. For example, the word 4271 could represent a period in which cells 4, 2, 7, and 1 each fired a spike in that order (Fig. 1). Because the letters are just labels and because we restrict ourselves to reference sequences containing only one event from each source, a reference sequence of length N can be represented as 1234⋯N. We will assume this form for all reference sequences (unless explicitly stated otherwise). An example of an invalid reference sequence is 123456789, given that no repeats are allowed.

**Our basic approach**

Our approach to the above sequence matching problem is based on combinatorics and probability. First we describe the idea with simple cases. For instance, given a 2-letter word in which both letters are distinct (i.e., the 2 letters are different), there are 2 possible ways in which the letters could be ordered (e.g., 12 or 21), and thus a 1/2 chance that the letters are in the same order as in the reference sequence, assuming each ordering is equally likely. Given a 3-letter word in which all 3 letters are distinct, there are 3! = 6 ways in which the letters could be ordered (e.g., 123, 132, 213, 231, 312, or 321), and thus a 1/6 chance that the letters are in the same order as in the reference sequence, again assuming each of the possible orderings are equally likely. [Note that 3! = 3 × 2 × 1 and, in general, n! = (n − 1) × (n − 2) × ⋯ × 3 × 2 × 1.] Now we extend this process to words of arbitrary length and letter composition. Because longer words could contain imperfect matches to the reference sequence that are still highly improbable based on chance, we will include imperfect matches in our analysis. For instance, although it is highly unlikely to find the word 123456789 by chance, it is also highly unlikely to find a word such as 12546789 by chance. Just as in the simpler cases, chance is defined as the probability of getting a match as good as the best one actually found in that word or better (where in the simple 2- and 3-letter words above there were no matches better than having the 2 or all 3 letters in order), assuming all n! permutations of the letters of that length n word are equally likely. The lower this probability, the more unlikely, and thus the higher the degree of matching between the word and the reference sequence. Note that the probability is conditional on the specific letters that were actually observed in the word. Later we show that this is a reasonable approach.

The reason for computing the probability that a word would contain a match as good as or better than the best match found in that word, assuming all possible permutations of its constituent letters were equally likely, is that we are now dealing with imperfect matches. It is exactly analogous to the statistical procedure of determining a P value, such as in computing the probability that one would observe 80 heads in 100 tosses of a coin, assuming the coin is fair. To evaluate how unlikely this is by chance, one computes the probability of getting 80 or more heads, assuming the null hypothesis of a fair coin. Here “80 heads” corresponds to the best match found in a word, “more than 80 heads” corresponds to the better matches, and “fair coin” corresponds to the permutations being equally likely.

Thus we need 1) a precise definition of matching that covers perfect as well as various imperfect matches, and 2) a way to rank matches from best to worst so that the phrases “best match found” and “better match” make sense.
Match definition

We define matches with 2 numbers, x and y, in the following manner. A word W (which may have repeated letters) contains an \((x, y)\) match to a given reference sequence S (which by definition cannot have repeated letters) if there exist \(x + y\) consecutive letters in W of which at least \(x\) letters are in strictly increasing order in S (but not necessarily consecutive in S). According to this definition, if \(S = 123456789\), the word 11377 does not contain a \((5, 0)\) match, but does contain a \((3, 0)\) match [as well as \((3, 1), (3, 2), (2, 0),\) and other matches], and the word 13436892 contains a \((6, 1)\) match [as well as \((6, 2), (5, 1), (5, 2), (5, 3), (4, 1),\) and other matches]. Essentially, our notation means that there are at least \(x\) letters in order, with at most \(y\) interruptions in between those matching letters.

Ranked match lists

Given a word W of length \(n\) and containing \(k\) distinct letters \((n \geq k)\), the best possible match to the reference sequence S by optimally rearranging the letters of W is a \((k, 0)\) match (i.e., \(k\) consecutive letters in W in the same order as in S and with no interruptions). In our approach, in the case that \(n > k\), the ordering of the \(n - k\) remaining letters does not matter. [Likewise, given any \((x, y)\) match, the order of the \(n - x\) remaining letters does not matter.] Thus there could be several different permutations of W containing the best possible match. For instance, if \(S = 123456789\) and \(W = 42717\), both 12477 and 71247 contain the best possible match. One reason we do not consider the specific arrangement of the remaining \(n - k\) letters is that, as discussed later, we presume (but do not require) a preprocessing (i.e., “parsing”) stage in which the letters and words are extracted from the raw data such that \(n\) is usually not much greater than \(k\).

Now how do we rank the possible matches? Given a word W of length \(n\) and containing \(k\) distinct letters, the list of all possible matches is shown in Fig. 2. To begin with, we will not consider matches \((1, y)\) for any \(y\) or \((2, y)\) for \(y > 0\). This is because every word contains a \((1, y)\) match, and any permutation containing a \((2, y)\) match with \(y > 0\) also contains a \((2, 0)\) match.

In terms of ranking matches, it makes intuitive sense that \((I)\) \((x, 0) > (x, 1) > (x, 2) > \ldots\) for any \(x\), and also that \((II)\) \((x, y) > (x - 1, y) > (x - 2, y) > \ldots\) for any \(x\) and \(y\), where “\(>\)” means “better than.” For instance, if \(x = 6\), \((I)\) says a \((6, 0)\) match is better than a \((6, 1)\) match; that is, getting 6 in a row in order is better than having 6 in order with one interruption. If \(x = 6\) and \(y = 0\), \((II)\) says a \((6, 0)\) match is better than a \((5, 0)\) match; that is, getting 6 in a row in order is better than having 5 in a row in order. Thus \((I)\) and \((II)\) are constraints that any ranking of matches should obey.

However, these constraints do not address how to rank \((x_1, y_1)\) and \((x_2, y_2)\) in the case that \(x_1 > x_2\) but \(y_1 > y_2\). This is partially addressed by the following. The intuition behind constraints \((I)\) and \((II)\) can be stated as the single constraint \((c^*):\) the ranking of matches must be such that a word that contains a worse match must not automatically also contain a better match. For instance, every word that contains an \((x, y)\) match automatically contains an \((x - 1, y)\) match; thus an \((x, y)\) match cannot be considered worse than an \((x - 1, y)\) match. However, \((c^*)\) also includes cases not covered by \((I)\) and \((II)\). For instance, according to \((c^*)\), \((6, 1) > (4, 0)\), because the 1 interruption in the \((6, 1)\) match will always still leave a \((4, 0)\) match somewhere. Even if the interruption is placed in the middle, say in 123Z456, no matter what Z is, either 123Z or Z456 will be a \((4, 0)\) match. Similarly, \((6, 3) > (4, 1)\).

Still, these constraints do not force a unique ranking of all the possible matches listed above. That is, several different rankings obey the intuitive constraint \((c^*)\). Two such rankings are described next.

\((I)\) THE ‘HORIZONTAL’ ‘H’ RANKING. Given any 2 possible matches \([\text{where an } (x, y) \text{ match is possible if and only if } x \leq k \text{ and } x + y \leq n, \text{ assign } (x_1, y_1) > (x_2, y_2) \text{ if either } I)\) \(x_1 > x_2\) \(\text{, or } II)\) \(x_1 = x_2 \text{ and } y_1 < y_2\). That is, the better matches are the ones with more letters in strictly increasing order regardless of interruptions, or the ones with the least interruptions given an equal number of letters in strictly increasing order. Thus the best possible match is the \((k, 0)\) match, followed by \((k, 1)\), then \((k, 2)\), and so on, until \((k, n - k)\). This is called “Horizontal” because matches are ranked following left to right horizontal lines, starting with the top row, in Fig. 2.

\((D)\) THE ‘DIAGONAL’ ‘D’ RANKING. Given any 2 possible matches, assign \((x_1, y_1) > (x_2, y_2)\) if either \(I)\) \(x_1 = x_2 \text{ and } y_1 < y_2\) \(\text{, or } II)\) \(x_1 > x_2\). Furthermore, consider only matches \((x, y)\) such that \(x - y = 2\). That is, the better matches are the ones with the greater difference between the number of letters in strictly increasing order and the number of interruptions, or the ones with the greater number of letters in strictly increasing order if the differences are equal (i.e., this ranking balances rewarding longer matches with penalizing more interruptions). Thus the best possible match is the \((k, 0)\) match, followed by \((k, 1)\) \(\text{, and so forth, then ending with } (2, 0)\). This is called “Horizontal” because matches are ranked following left to right horizontal lines, starting with the top row, in Fig. 2.

\((D)\) THE ‘DIAGONAL’ ‘D’ RANKING. Given any 2 possible matches, assign \((x_1, y_1) > (x_2, y_2)\) if either \(I)\) \(x_1 = x_2 \text{ and } y_1 < y_2\) \(\text{, or } II)\) \(x_1 > x_2\). Furthermore, consider only matches \((x, y)\) such that \(x - y = 2\). That is, the better matches are the ones with the greater difference between the number of letters in strictly increasing order and the number of interruptions, or the ones with the greater number of letters in strictly increasing order if the differences are equal (i.e., this ranking balances rewarding longer matches with penalizing more interruptions). Thus the best possible match is the \((k, 0)\) match, followed by \((k, 1)\) \(\text{, and so forth, then ending with } (2, 0)\). This is called “Horizontal” because matches are ranked following left to right horizontal lines, starting with the top row, in Fig. 2.

\((D)\) THE ‘DIAGONAL’ ‘D’ RANKING. Given any 2 possible matches, assign \((x_1, y_1) > (x_2, y_2)\) if either \(I)\) \(x_1 = x_2 \text{ and } y_1 < y_2\) \(\text{, or } II)\) \(x_1 > x_2\). Furthermore, consider only matches \((x, y)\) such that \(x - y = 2\). That is, the better matches are the ones with the greater difference between the number of letters in strictly increasing order and the number of interruptions, or the ones with the greater number of letters in strictly increasing order if the differences are equal (i.e., this ranking balances rewarding longer matches with penalizing more interruptions). Thus the best possible match is the \((k, 0)\) match, followed by \((k, 1)\) \(\text{, and so forth, then ending with } (2, 0)\). This is called “Horizontal” because matches are ranked following left to right horizontal lines, starting with the top row, in Fig. 2.

\((D)\) THE ‘DIAGONAL’ ‘D’ RANKING. Given any 2 possible matches, assign \((x_1, y_1) > (x_2, y_2)\) if either \(I)\) \(x_1 = x_2 \text{ and } y_1 < y_2\) \(\text{, or } II)\) \(x_1 > x_2\). Furthermore, consider only matches \((x, y)\) such that \(x - y = 2\). That is, the better matches are the ones with the greater difference between the number of letters in strictly increasing order and the number of interruptions, or the ones with the greater number of letters in strictly increasing order if the differences are equal (i.e., this ranking balances rewarding longer matches with penalizing more interruptions). Thus the best possible match is the \((k, 0)\) match, followed by \((k, 1)\) \(\text{, and so forth, then ending with } (2, 0)\). This is called “Horizontal” because matches are ranked following left to right horizontal lines, starting with the top row, in Fig. 2.
Diagonal because matches are ranked following upper-right to lower-left diagonals, starting with the upper leftmost such diagonal [i.e., that containing just (k, 0)], in Fig. 2.

We prove that both these rankings obey the constraint (c*) in Appendix B. In Appendix C we evaluate these and other possible rankings in terms of the probabilities that a random permutation contains each type of match. Intuitively, less-probable matches should generally be considered better. We have primarily used the D ranking scheme in our previous work, but we also showed that the H ranking gave similar results there (Lee and Wilson 2000). Both schemes have merit, as does a simplified H ranking scheme in which \( f_i \) and Wilson 2000). Both schemes have merit, as does a similarly used the D ranking scheme in our previous work, but we emphasize.

### Match probability calculation

Given any match ranking scheme, our combinatorial method provides a straightforward way for computing probabilities (and thus the significance) of sequence matching in words of arbitrary length and letter composition (including words with repeated letters and words with matches that have interruptions), and for reference sequences of arbitrary length. The method can be described simply as follows. Given a particular reference sequence and an arbitrary word of length \( n \), find the best match in the ranked match list that is actually found in the word. Then compute how rare such a match is by chance (i.e., assuming all \( n! \) permutations of the word's letters were equally likely) by determining the fraction of the \( n! \) permutations that contain a match that is as good as that match or better according to the ranked match list. We illustrate this procedure with examples.

Given reference sequence \( S = 123456789 \), say we observe the word 524679. For this word \( n = 6, k = 6 \). Using the D ranking scheme, the ranked match list (best to worst) is: \((6, 0) > (5, 0) > (5, 1) > (4, 0) > (4, 1) > (3, 0) > \ldots \). The best match from this list actually found in the word is a \((5, 0) \) match (i.e., 24679). Out of the 6! = 720 possible permutations of the 6 letters, there are 10 permutations whose best match is a \((5, 0) \) match (not necessarily 24679), and 1 permutation with a better match, the perfect \((6, 0) \) match 245679. Thus the probability of getting a \((5, 0) \) match or better in a word of this letter composition by chance (assuming all permutations were equally likely) is very small: \( 1/6! = 0.015 \) (Fig. 3).

Instead, say we observe the word 51426987. For this word \( n = 8, k = 7 \). Again using the D ranking scheme, the ranked match list (best to worst) is: \((7, 0) > (7, 1) > (6, 0) > (6, 1) > (5, 0) > (6, 2) > (5, 1) > (4, 0) > (5, 2) > \ldots \). The best match from this list actually found in the word is a \((5, 1) \) match: 14679, where \( Z = 9 \). Out of the 8! = 40,320 possible permutations of the 8 letters, there are 2,338 permutations that contain a \((5, 1) \) match or better, according to the ranked match list. Thus the probability of getting a \((5, 1) \) match or better in a word of this letter composition by chance (assuming all permutations were equally likely) is: \( 2,338/8! = 0.0580 \).

As these examples show, our method allows us to quantify matching in words that do not necessarily contain every letter in the reference sequence, that may have repeated letters, that may have matches with interruptions, and that may have additional letters outside the matching segment. All of these situations are handled in the same way with ranked match lists and permutations.

### The null hypothesis

To reiterate the null hypothesis, we assume that, given a word of particular length and letter composition, all possible orderings (i.e., permutations) of those letters (including any repeated letters) were equally likely. That is, we were assuming no particular ordering. We then compute the probability that such random ordering of these letters would by chance produce a match to a specific reference sequence that is at least as good as the best match observed in the actual ordering of letters in the word. As mentioned above, this is a conditional probability, that is, conditional on the letters actually observed in the word. This approach gives us the same answers as other reasonable null hypotheses.

For instance, consider the following null hypothesis. Assume each of the \( N \) letters in the reference sequence occurs exactly once in each word, but that we observe only \( n < N \) of them (e.g., some spikes are missing because of neural variability). Note this also means that \( n = k \) (i.e., words do not have repeated letters). Further assume that the relative position of the unobserved letters could be anywhere. Consider the probability that we would observe a match as good as or better than the best match found among any \( n \) observed letters, assuming all \( N! \) underlying permutations of the \( N \) letters were equally likely. The answer is the same as the original probability computed above. For example, given the reference sequence 123456789 and the word 524679, we observe only 6 letters, whereas the letters 1, 3, and 8 are missing. For each permutation of any 6 observed letters, there are \( 9! \) unique ways 3 unobserved letters could be located with respect to the positions of those 6 observed letters. The numerator in Fig. 3 thus becomes \((11 \times 9)!/6! \), whereas the denominator becomes \((6! \times 9!)/6! = 9! \) (representing every permutation of the 9 letters in the reference sequence), yielding the same probability 0.015. That is, 0.015 is the chance that any 6 letters selected from a random permutation of 9 distinct letters would contain a \((5, 0) \) match or better with respect to the reference sequence 123456789. However, if \( k < n \) (there are repeated letters), then...
the form this alternate null hypothesis should take is less clear, whereas our original approach handles repeated letters in a natural way.

Now consider a null hypothesis that is basically an inversion of our original one. As before, given a reference sequence \( S = 1234 \cdots N \) of length \( N \) and a word \( W \) of length \( n \) containing \( k = n \) distinct letters (\( n \leq N \)), find the best match in that word (according to the chosen match ranking scheme). Call that match \( F \). However, this time treat the observed word as fixed. Instead, ask what is the probability that a random reference sequence (i.e., any permutation of \( S \)) would have yielded a match in \( W \) that is as good as or better than \( F \) (i.e., the best match that had been found in \( W \) with respect to the original reference sequence \( S \)). Assuming all \( N! \) possible reference sequences were equally likely, the answer is the same as the original probability above. This is proven in Appendix D. For example, this means that the word \( 524679 \) contains a \((5, 0)\) possible 9-letter reference sequences (see Appendix D).

At a fundamental level, our conditional probability approach separates the issue of generating a word from that of the order within a word. One could imagine a null model that did not separate these issues and instead specified how each word was to be produced from among all possible words. This model might include the probability of producing a word of a given length, of getting a particular set of letters, of having certain letters repeat, of having these letters in a certain order, and all this perhaps as a function of previous words. For instance, consider a first-order Markov process with \( n + 1 \) states (one state for each letter plus a termination state), with the initial letter drawn from some probability distribution, and with transition probabilities for moving from each state to any other state (including itself) specified. This model would implicitly specify a null hypothesis of what letter orderings to expect (most likely not equally probable). It would also implicitly specify the expected probabilities of getting each type of \((x, y)\) match, and from this we could similarly compute a probability of observing a particular match or better with respect to a chosen match ranking. That is, our relative-order and \((x, y)\) match ranking approach could also be applied to such null models as these. However, our conditional probability approach sidesteps the complicated issue of how words are generated by taking the letters of each observed word as given. Instead we focus exclusively on the issue of order by assuming that every possible ordering of those observed letters was equally likely. This also means that the ordering of letters within each word is assumed to be independent of the ordering in every other word—an assumption used later in determining the significance of matching across a set of words. Our focus is on the order of letters and not how they are generated because we are concerned with sequences, not word length, letter composition, or other issues.

Is it a good idea to not assume any particular model of word generation? A modified version of the coin-flip analogy can again be useful here. Say we observe 100 consecutive trials of an unknown experiment with 2 possible outcomes (called “heads” and “tails”). Unlike with a real coin, here we do not know whether there is a fixed probability of getting a head on any given trial, whether trials are independent, or whether any aspect of the system is constant in time. How would one analyze such a potentially complicated system? One of the first things to measure would be the observed fraction of heads. Say it is 0.6 (i.e., 60 heads). To get a sense of whether this fraction is unusually large, we could compare it to the fraction expected from the simplest possible model: 100 independent trials each with fixed probability 0.5 of getting heads. Based on this model, the probability of observing a fraction \( \geq 0.6 \) is 0.03 (i.e., low). What can we conclude from this? We cannot conclude that the trials are independent, each with fixed probability 0.6 of getting heads. In fact, based on the fraction alone, we cannot conclude that any particular model of how outcomes are generated is likely to be correct. For instance, a deterministic sequence of “HHHTT” repeating forever would also produce the fraction 0.6. We can say only that it is unlikely for the simplest model—100 independent flips of a fair coin—to account for this observed fraction of heads. However, this knowledge is of value. This is exactly what we have done with respect to sequences. We have developed a way to measure the amount of sequence structure in an arbitrary word (and, as we shall show later, in a set of words, i.e., trials) by comparing it to the amount of sequence structure expected from the simplest possible model: the random model of ordering. It is analogous to measuring the fraction of heads. It answers the question of whether there is a bias in ordering at all, and how large the bias appears to be. This measure says nothing about how the observed amount of sequence structure comes about (i.e., how the words are generated). However, as we have seen with the coin-flip analogy, in the absence of constraints (such as those based on possible physical mechanisms), the space of possible models is vast. Thus here we avoid the complex issue of models of word generation.

Therefore if one asks what justifies comparing the data against such an unlikely null model (i.e., equally likely permutations), we can say the following. It is not because we believe the null model to be true, but because it is the first standard of comparison for ordering, just as a fair coin is the first standard for analyzing a series of coin flips. In fact, we have tested the equally likely assumption in real data using shuffles (and in some cases all possible permutations), as described below, and found the data consistent with such a model (Lee and Wilson 2000). There is a null model is not necessarily unlikely.

More efficient ways to compute exact, upper bound, and lower bound match probabilities

Although our combinatorial approach applies to words of arbitrary length and letter composition, this does not mean that it is always easy to calculate the corresponding probabilities, especially for longer words. Consider a word of length \( n \) with \( k \) distinct letters. Let \( m \) be an \( N \)-dimensional “multiplicity” vector whose entries \((m_1, m_2, m_3, \ldots, m_N)\) give the number of times each letter \((1, 2, 3, \ldots, N)\) occurs in this word. Thus \( k \) elements are nonzero, and \( \sum_{i=1}^{N} m_i = n \). A brute force probability calculation would require testing each of the \( n!/(n_1!n_2!\cdots n_k!) \) distinct permutations to see whether they contained the best match found or better. Therefore we have developed more efficient ways to calculate the exact probability for some important cases, and upper and lower bounds of the probability for other cases. In particular, we have a formula for computing
the exact probability of getting a \((k, 0)\) match (i.e., the best possible match) in any word (i.e., given any \(m\)). Furthermore, we have a formula for computing an upper bound probability for getting any \((x, y)\) match, and a lower bound probability for getting any \((z, 0)\) match where \(z < k\). (See APPENDIX E for the formulae and derivations.)

With these formulae we can compute both upper and lower bound probabilities of getting a given match or better in any word with respect to any ranking scheme. For example, consider again the reference sequence 123456789 and the word 51469784. For this word \(m = [1 0 0 2 1 1 1 1 1]\). Using the D ranking scheme, the ranked match list (best to worst) is: (7, 0) > (7, 1) > (6, 0) > (6, 1) > (5, 0) > (6, 2) > (5, 1) > (4, 0) > (5, 2) > ... , and (5, 1) is the best match found. We can use formula E2 to calculate an upper bound of the probability of getting a \((5, 1)\) match or better according to this ranking scheme as follows. The matches we must count are \((7, 0), (7, 1), (6, 0), (6, 1), (5, 0), (6, 2),\) and \((5, 1)\). Because formula E2 gives the probability of getting any \((u, v)\) match with \(u \equiv x\) and \(v \equiv y\), we use it twice—once with \(x = 5\) and \(y = 1\), and once with \(x = 6\) and \(y = 2\). The first gives an upper bound probability of getting a match inside A (Fig. 4, solid box), the second an upper bound probability of getting a match inside B (Fig. 4, dashed box). Adding these 2 upper bounds together gives us the desired upper bound. Here we get 0.1038 compared with the exact probability 0.0580. (Although not necessarily the tightest bound, it is good enough for practical use in many cases, e.g., in determining the significance of matching across many words, as shown below.) To compute a lower bound probability of getting a \((5, 1)\) match or better, find the worst \((z, 0)\) match as good as or better than \((5, 1)\), here \((5, 0)\), and compute a lower bound probability of getting this match using formula E3. This is the desired lower bound because it is a lower bound of getting a match that is as good as or better than \((5, 1)\). Here we get 0.0195 compared with the exact probability 0.0580. (Although again not necessarily the tightest bound, it is of practical use in many cases.) An analogous approach will work for any ranking scheme.

**Modified null hypotheses**

In computing probabilities, we have so far used the null hypothesis that all permutations are equally likely, but our approach can also handle other null hypotheses in which permutations have unequal weightings. All that is necessary is a way to weight the permutations in a manner appropriate to test what is desired.

![Diagram](Fig. 4. Example diagram illustrating part of process used in calculating an upper bound of the match probability. See text for details.)

For instance, suppose it is known that pairs of letters tend to occur in the same order as in the reference sequence S with a probability B, where B is called a bias. If \(B > 1/2\) then we would expect more sequence matching to S than based on equally likely permutations. To quantify the degree of sequence matching beyond that expected based on the bias, we weight more those permutations with more pairs of letters in order. The exact weighting is a matter of choice.

For example, we could weight each \(n\) letter permutation by \(B^f \times (1 - B)^j / H\), where \(f\) is the number of adjacent letter pairs in the permutation that are in the same order as in S, \(r = \) the number in the opposite order, and \(H\) is a normalization factor to make the weights of all permutations sum to 1. For \(4271, f = 1\) (from 27) and \(r = 2\) (from 42 and 71). (Note repeated adjacent letters would be ignored; thus \(f + r \leq n - 1\).) If \(B > 1/2\), this weighting would tend to increase the expected probability of getting a given match or better. [A more complicated version could involve bias values that are specific to each letter pair, e.g., \(B(3, 5) = \) the probability of observing the pair 35 given that one observes a 35 or 53.] This weighting scheme assumes a process in which each letter depends only on the letter immediately before it, with the resulting match probability indicating the amount of longer sequence matching that cannot be accounted for by the random chaining together of the shorter pair biases. However, in general there is no reason to believe such a process is the correct one.

Alternatively, we could weight each permutation by \(B^f \times (1 - B)^j / H\), where \(f\) is the number of all \(n \times (n - 1)/2\) letter pairs in the permutation that are in the same order as in S, \(r = \) the number in the opposite order, and \(H\) is again for normalization. This treats each of the \(n \times (n - 1)/2\) letter pairs as independent, but because they cannot all be independent, it gives an upper bound of the effect of a bias \(B > 1/2\), at least as far as perfect matches [i.e., \((n, 0)\) matches in words of length \(n\)] are concerned. We have used this latter weighting in our work on hippocampal sequences in SWS (Lee and Wilson 2000). There we set \(B = \) the average bias observed across all 2-letter words with 2 distinct letters. One could also set \(B = \) the average bias observed across all words regardless of length [with each \(n\)-letter word contributing \(n_i\) letter pairs to the average, where \(n_i\) is the number of all \(i\)’s \(n \times (n - 1)/2\) possible letter pairs in which the 2 letters are distinct].

Finally, consider weighting each permutation \(i\) by \(h \times b^{(i)}\), where again \(f(i) = \) the number of all \(n \times (n - 1)/2\) letter pairs in permutation \(i\) that are in the same order as in S, and \(h\) and \(b\) are fit to 2 equations

\[
\sum_{\text{all permutations } i} h \times b^{(i)} = 1 \tag{1}
\]

and

\[
\sum_{\text{all permutations } i} [h \times b^{(i)}] 	imes [f(i)/n_2] = B \tag{2}
\]

where \(h > 0, b > 0,\) and \(0 < B < 1\). Because each permutation’s weight, \(h \times b^{(i)}\), is the probability (according to the null hypothesis) of that permutation occurring, \(\text{Eq. 1}\) is the requirement that probabilities sum to 1. In \(\text{Eq. 2}\), \(f(i)/n_2\) = the fraction of all distinct letter pairs in permutation \(i\) that are in the same order as in S. Thus \(\text{Eq. 2}\) requires the probability-weighted expected pair bias (according to the null hypothesis) over all possible permutations to equal the observed bias \(B\), unlike in
Innovative Methodology

A. K. LEE AND M. A. WILSON

the previous weighting schemes. That is, imagine we are allowed to view only one randomly chosen pair of distinct letters from any permutation. Under this null hypothesis, the probability that such a pair is in the same order as in S is B. Furthermore, this scheme treats all pairs equivalently (unlike the first scheme), yet does not treat them as independent (unlike the second). If B > 1/2 then b > 1, so the form of the weights (i.e., powers of b with exponent f) corresponds to the idea that each additional pair in the same order as in S (i.e., incrementing f) makes a permutation more likely by a constant multiplicative factor b, just as in both previous weighting schemes [for which the constant multiplicative factor was B/(1 − B)].

Table 1 shows the different permutation weights for the simple case of W = 123 under the second (“all pairs upper bound”) and third (“all pairs”) weighting schemes with B = 0.6. For W = 524679 and the D ranking, the match probability [i.e., the probability of getting a (5, 0) match which is the best match found, or better] goes from 0.015 (equally likely) to 0.6. Thus such a match is considered that much less rare when a 0.6 pair bias is taken into account.

Other weighting schemes are possible, including those that account for triplet effects and other types of interactions (Lee and Wilson 2000). However, the general problem of how to “subtract out” the effects of lower-order interactions is a topic of further study for all higher-order analysis methods (Aerts et al. 1989; Güttig et al. 2003; Martignon et al. 1995, 2000; Nakahara and Amari 2002; Oram et al. 1999; Palm et al. 1988).

Significance of matching across many words

As mentioned above, the probability we calculate (whether with the equally likely permutation assumption or any other null hypothesis) is identical to a P value, thus giving us the significance of sequence matching for a single word. In a typical experiment there will be multiple observations, yielding a set of words, each of which will have a match probability associated with it. Some words may have low probability matches, whereas others may not, with a larger than expected proportion of low probability matches indicating general matching to the reference sequence.

An overall significance of sequence matching between such a set of words and the reference sequence can be determined in the following manner. First we define a “P ≤ P’ trial” as a word for which the best possible sequence match produced by optimally rearranging its letters [generally a (k, 0) match] has probability ≤ P’ of occurring (i.e., a word that could have contained a match of probability P’ or rarer based on its constituent letters). We define a “P ≤ P’ match” as a word for which the probability of getting the best match actually found in the word or better is ≤ P’. For example, given the reference sequence S = 123456789 and the D ranking, if we set P’ = 1/24, then the word 2471 is a P ≤ 1/24 trial (given that a rearrangement of its letters to 1247 contains a match of probability ≤1/24, i.e., 1/24 exactly) but not a P ≤ 1/24 match [given that the best match actually found, a (3, 0) match, has probability >1/24, i.e., 7/24], the word 524679 is both a P ≤ 1/24 trial and a P ≤ 1/24 match (given that the best match actually found has probability ≈1/24, i.e., 0.015; Fig. 3), and the word 123 is neither a P ≤ 1/24 trial or P ≤ 1/24 match. The expected P ≤ P’ match/trial ratio for a set of words is P’, assuming the ordering of letters within each word is independent of the ordering within all the other words. Given M matches out of T trials, the significance of the observed ratio M/T can be quantified with a Z score determined from the normal approximation to the binomial distribution

\[ Z = (M - T \times P')/\sqrt{T \times P' \times (1 - P')} \]  

(3)

The exact P value is given by the binomial distribution

\[ P (\text{observing} \geq M \text{ matches}) = \sum_{i=M}^{T} C(T, i) \times (P')^i \times (1 - P')^{T-i} \]  

(4)

where \(C(a, b) = a!/[b! \times (a - b)!]\) = the number of different sets of b objects that can be chosen from a total of a objects. This becomes difficult to compute for large T. For moderate sized P’ (e.g., 1/2 or 1/6), moderate sized Z, and sufficiently large T, the normal approximation is good and the Z score can be directly translated into a P value using the normal distribution. For small P’ (e.g., 1/24) and/or large Z, the normal approximation is not good, and one should use other approximations or bounds of the P value. For example, the binomial distribution can be approximated by a Poisson distribution for small P’, and thus an approximate P value is given by

\[ P (\text{observing} \geq M \text{ matches}) = \sum_{i=M}^{T} e^{-(T \times P')} \times (T \times P')^i / i! \]  

(5)

One should note the difference between the ratio (matches/trials) and the significance (Z score and P value). The comparison between the observed and expected match/trial ratios reveals what the effect size is likely to be, whereas the Z score (and P value) gives an idea of the confidence we should have in that effect size based on the amount of data we have.

This approach is valid for any chosen P’. Because we are generally concerned with finding at least moderately long matching sequences, we set P’ to values <1/2, such as 1/6, 1/24, or 0.01. Given that 1/3! = 1/6 is the probability of getting a (3, 0) match in an n = 3, k = 3 word, setting P’ = 1/6 means that we are focusing on matching sequences that generally have ≥3 letters in order (i.e., “3rd-order interactions”). Likewise, setting P’ = 1/4! = 1/24 focuses on matching sequences with generally 4 or more letters in order (i.e., “4th-order interactions”), and P’ = 0.01, 5 or more letters. Thus by setting P’ we can focus on sequence matches of any desired length (i.e., “nth-order interactions”) within the same framework. Note that because P ≤ P’ matches can have probabilities < P’, the match/trial ratio, Z score, and overall P value are actually

<table>
<thead>
<tr>
<th>Permutation</th>
<th>f</th>
<th>r</th>
<th>Equally Likely</th>
<th>All Pairs Upper Bound</th>
<th>All Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3</td>
<td>0</td>
<td>0.167</td>
<td>0.284</td>
<td>0.261</td>
</tr>
<tr>
<td>132</td>
<td>2</td>
<td>1</td>
<td>0.167</td>
<td>0.190</td>
<td>0.187</td>
</tr>
<tr>
<td>213</td>
<td>2</td>
<td>1</td>
<td>0.167</td>
<td>0.190</td>
<td>0.187</td>
</tr>
<tr>
<td>231</td>
<td>1</td>
<td>2</td>
<td>0.167</td>
<td>0.126</td>
<td>0.134</td>
</tr>
<tr>
<td>312</td>
<td>1</td>
<td>2</td>
<td>0.167</td>
<td>0.126</td>
<td>0.134</td>
</tr>
<tr>
<td>321</td>
<td>0</td>
<td>3</td>
<td>0.167</td>
<td>0.084</td>
<td>0.096</td>
</tr>
</tbody>
</table>

Here the bias B is set to 0.6. See text for details.
lower bounds of the true amount of matching at that particular $P'$ level. Thus a warning: to prevent a significant underestimate of the degree of matching, one should check the following 2 points. First, one should check that most of the $P \leq P'$ matches do not have probabilities that are far lower than $P'$. Second, one should check that most of the $P \leq P'$ trials could contain a match whose probability is simultaneously $\leq P'$ and close to $P'$. As an extreme illustration of the second kind of underestimate, suppose one chooses $P' = 1/24$ and that all the $P \leq P'$ trials are 5-letter words, each with 5 distinct letters. Assuming equally likely permutations, the best possible match (5, 0) has probability 1/120, whereas the second best match (4, 0) has probability 1/15. Thus the expected match/trial ratio is not 1/24 but rather 1/120, given that the only way for a word to be a $P \leq P'$ match is for it to contain a (5, 0) match.

In practice, we can use the exact, upper, and lower bound match probability formulae to quantify matching over a set of words in the following way. To begin with, it is important to distinguish 2 approaches with different goals. First, to compute a lower bound of the match/trial ratio, count a word as a $P \leq P'$ trial unless shown otherwise (i.e., unless some lower bound probability of the best possible match is $>P'$), and do not count a word as a $P \leq P'$ match unless shown otherwise (i.e., unless some upper bound probability of the best match found in the word is $\leq P'$). Alternatively, to compute an upper bound of the match/trial ratio, count a word as a $P \leq P'$ match unless shown otherwise (i.e., unless some lower bound probability of the best match found in the word is $>P'$), and do not count a word as a $P \leq P'$ trial unless shown otherwise (i.e., unless some upper bound probability of the best possible match is $\leq P'$, or unless it has already been counted as a $P \leq P'$ match). In the case of the standard equally likely permutation assumption, formula E1 for the exact probability of a (k, 0) match allows an unambiguous determination of whether a word is a $P \leq P'$ trial for both the upper and lower bound ratios. The lower bound ratio should be used as a conservative test to show that significant matching exists, and the upper bound ratio as a conservative test to show that significant matching is not present. In addition, in trying to show the latter, one should verify the 2 points mentioned above to prevent an underestimate. Note that it is not necessary to verify these points in computing a lower bound of matching because an underestimate is still a lower bound. In our hippocampal sequence learning work, we chose $P' = 1/24$, and we computed the lower bound ratio (and $Z$ score) to demonstrate significant matching between the experienced sequence and the set of words from the period subsequent to that experience. We computed the upper bound ratio (and $Z$ score) to demonstrate a lack of matching between the experienced sequence and the set of words from the preceding period. We also verified that most of the $P \leq P'$ matches from that preceding period did not have probabilities that were far lower than $P'$, and that most of the $P \leq P'$ trials could have contained a match whose probability was simultaneously $\leq P'$ and close to $P'$ (Lee and Wilson 2000). However, for the reasons mentioned above, it is generally easier to use this approach to demonstrate that significant matching exists than to demonstrate that it does not.

By considering matching over a set of words, our method allows one to detect traces of a particular reference sequence in a heterogeneous set of words that do not contain every letter and that may have few letters in common with each other. Such a situation may occur during spontaneous activity in which only a fraction of cells are active at a given time, as in the hippocampus during SWS (Lee and Wilson 2000). For instance, consider the following set of 4 words made from 7 letters: 125, 2367, 146, and 3455. Each word contains no more than 4 out of the 7 possible letters, and no pair of words has more than 1 letter in common. However, together these 4 words indicate the strongest traces of 3 out of the 7! = 5,040 possible reference sequences: 1234567, 1234657, and 1234675.

Shuffle analysis can be used to yield a complementary estimate of match significance across a set of words. First, compute the match/trial ratio with respect to a randomly shuffled version of the reference sequence (e.g., $S = 472936185$). The significance of matching to the original reference sequence could then be estimated from the distribution of match/trial ratios for many different shuffled reference sequences. For instance, one could determine that the match/trial ratio with respect to $S = 123456789$ is $Z$ SDs above the mean match/trial ratio of the shuffle distribution. This allows a measure of significance across words that does not assume any particular null hypothesis for the distribution of permutations, but rather uses the distribution across the observed words themselves. However, this does not allow one to measure the amount of sequence structure in an absolute sense (i.e., with respect to a simple, known null hypothesis, such as equally likely permutations). We computed the shuffle significance for our hippocampal data and found it to be similar to that based on the equally likely assumption. Furthermore, the shuffle distribution itself was consistent with the equally likely permutation and independent trials model (Lee and Wilson 2000).

Application to specific types of neural data

We have described our approach in terms of abstract symbols (letters and words) to focus only on the essential elements of relative order and sequence matching. The results are general and can be applied to a broad range of multiple-source neural data. To apply our results to a specific type of neural data one simply needs an appropriate (and preferably natural) “parsing” method to convert the data into letters and words. For instance, in an experiment composed of separate trials, the activity during each trial could give a different word. Within each trial, the order of letters (e.g., with each letter derived from the activity of a particular neuron or the signal at a local EEG site) could be based on the relative time of occurrence of signals (e.g., spikes or particular EEG events) following a particular event, such as the presentation of a stimulus, the beginning of a delay period, the beginning of a motor movement, and so forth. The reference sequence would be a presumed order of activity that one wanted quantitative evidence of within a set of trials. In Fig. 5, A and B we illustrate possible parsing schemes for both low and high firing rate cases using simulated spike trains. In the low-rate case, letters are assigned to the first spike fired by each cell. In the high-rate case, letters are assigned to the first moment the instantaneous firing rate rises sufficiently above the baseline rate. (A similar parsing scheme could be used for imaging data in which the intensity in a given region is substituted for the instantaneous firing rate.) If a cell does not fire any spikes (low-rate case) or significantly raise its firing rate above baseline (high-rate case)
Innovative Methodology

A. K. LEE AND M. A. WILSON

FIG. 5. Application of relative-order method to specific types of neural data. We illustrate how different types of data could be parsed into letters and words for relative-order analysis. A and B: simulated spike trains. Each step function gives the strength of input (in terms of target average firing rate) into each cell as a function of time, with the resulting output spike train above. See Supporting Methods for details of simulation. A: trial-by-trial data, low baseline firing rate case. Here we assign a letter to the first spike fired by each cell after the trial onset time (time 0). Then all the letters in the trial window (1,000 ms) are ordered to form a single word, here “12.” B: trial-by-trial data, high baseline firing rate case. Each time a spike occurs, the observed instantaneous firing rate (IFR, circles) is computed as the average firing rate over the period stretching from 2 spikes before to 2 spikes after the current spike. Mean and SD of the IFR for the 2 s before time 0 (during which the input is 10 Hz) is computed, and a threshold (dashed line) is set at 2 SDs over the mean. For each cell, a letter is assigned to the first spike after time 0 for which its IFR and the IFR of the next 3 spikes are all over the threshold. Then, as in A, all the letters in the trial window are ordered to form a single word, here “12.” C: continuous data [hippocampal CA1 pyramidal cell activity in slow wave sleep (SWS)]. First, for each cell, a letter is assigned to the first spike in each burst, as well as to isolated spikes (top panel). Then a single word is formed from each set of letters in which adjacent letters are separated by more than max_gap (here 100 ms). Thus one criterion for a good parsing method is that it produces a set of words with this property.

Sample size considerations

The significance of matching across many words depends on the amount of data available (i.e., the number of trials). An important aspect of our relative-order and probability approach is that it can greatly reduce the sample size problem associated with finding evidence for or against a particular complex firing pattern, especially compared with methods that classify patterns based on exact time intervals (Abeles 1991; Abeles and Gerstein 1988; Abeles et al. 1993; Aertsen et al. 1989; Gerstein and Perkel 1969; Grün et al. 2001; Güttig et al. 2003; Martignon et al. 2000; Nakahara and Amari 2002; Palm et al. 1988). That is, with larger n (i.e., more spikes in each pattern), it becomes harder to accurately estimate probability densities in the (n − 1)-dimensional space of exact (binned) intervals (i.e., there are too many possible patterns for a given amount of data).

Say we have N neurons and are concerned with patterns that occur within J ms. For simplicity, assume that each neuron contributes exactly one letter per J ms. First consider patterns with exact time intervals. If we divide J into U time bins, then there are...
possible patterns, with the time intervals quantized by J/U ms. (To see this, consider the i = 2 term. This term represents the number of ways to choose 2 particular neurons to occupy the first time bin, times the number of ways to place each of the remaining N − 2 neurons in any of the remaining U − 1 time bins. The sum is over the different numbers of neurons that can be chosen to occupy the first time bin.) In contrast, if we consider only relative firing order, then there are N! possible patterns. Suppose n = 5 neurons, J = 500 ms, and U = 100 bins (i.e., 5-ms bins). Then there are approximately 4.9 × 10^8 different patterns based on exact time intervals and 120 patterns based on relative firing order. For n = 10, this becomes 9.6 × 10^{18} versus 3.6 × 10^6 patterns.

However, our method can do much better than N!. In fact, it effectively reduces this number to approximately 1/P′, where P′ is the probability we set that determines which words are matches and trials. This is attributed to the result described above and in Appendix D. For example, say n = 8, choose the D ranking scheme, and we set P′ = 1/24 = 0.042. Say we observe an 8-letter word W1 with 8 distinct letters. The worst match W1 could contain and still be a P ≤ 1/24 match is (6, 2), which has probability 0.041 (which is nearly 1/24). According to Appendix D, this also means that W2 is a P ≤ 1/24 match with respect to 0.041 × 8! out of the 8! possible reference sequences. That is, although there are 8! possible relative-order patterns, each word is evidence for approximately P′ × 8! of them and evidence against (1 − P′) × 8! of them. This also applies to words with missing letters. Say a word contains only 4 out of the 8 letters (once each) (e.g., W3 = 1247). Only a (4, 0) match (which has probability 1/24) will make it a P ≤ 1/24 match. However, such a word will be evidence for 8!/4! = P′ × 8! reference sequences, given that W4 contains a (4, 0) match with respect to any reference sequence with the letters 1, 2, 4, and 7 in that order, regardless of where the missing letters are placed around them (there being 8!/4! unique ways to do so, e.g., S = 12345678 and 61284735). Therefore an estimate of the amount of sequence structure with respect to each of the N! possible relative-order patterns requires a sample size of only a few times 1/P′ trials (e.g., 100–200 trials for P′ = 1/24 regardless of the size of N, as opposed to a few times N! trials. Essentially, we have reduced the required sample size by allowing matches that are partial and imperfect to an extent determined by P′. (Note the approximate nature of 1/P′ is ascribed to the presence of words with repeated letters and words that cannot contain a match with probability exactly = P′.)

To illustrate the sample size issue, we compare our previous relative-order analysis of hippocampal spike activity during SWS (Lee and Wilson 2000) with an exact time interval analysis of the same data (see supporting methods). Consider the activity of the RUN sequence place cells from both directions of the 3 rats. Using our relative-order analysis (with parsing parameters max_isi = 50 ms, max_gap = 100 ms), in POST SWS we had found 35 P ≤ 1/24 matches to the RUN reference sequence out of 270 total P ≤ 1/24 trials (vs. 270/24 = 11 matches expected, P < 4 × 10^-5), providing strong evidence of RUN sequence structure in subsequent SWS. Because P′ was set to 1/24 = 1/4!, this analysis only considered matches with ≥4 distinct letters (cells) in order. We searched the same POST SWS data (98 min total) for exact time interval patterns consisting of spikes from ≥4 distinct cells firing within a maximum duration of 500 ms (which is longer than the matches found with relative-order analysis). Even when using 5-ms bins [compared with standard 1-to-3-ms bins (Abeles et al. 1993)], we found no patterns that repeat within POST SWS using a burst filtering max_isi of 50 ms (as used in the relative-order analysis), and only 6 patterns that repeat using max_isi = 15 ms. None of these patterns consisted of spikes from more than 4 distinct cells or occurred more than twice. Although one of these patterns matched (in relative order) the RUN reference sequence, not only were both occurrences detected by our relative-order method, but our method also detected several more spikes firing in the correct order (Fig. 6A). Furthermore, our method also detected 33 additional (P = 1/24) matches to the RUN sequences.

Alternatively, the POST SWS sequences may be an exact "memorization" of particular exact time interval patterns in RUN. Then a search should detect exact time interval patterns that occur at least once in both RUN (19 min total, counting only the laps) and POST SWS, but do not necessarily repeat within POST SWS. Again using 5-ms bins, we found only 3 such patterns using max_isi = 50 ms, and only 19 using max_isi = 15 ms. Each occurred just twice and none consisted of spikes from more than 4 distinct cells. Out of these 22 patterns, only 4 of these had all 4 cells firing in the same order as in the RUN reference sequence, and for 3 of those the POST SWS occurrence corresponded to the same P ≤ 1/24 match found with the relative-order analysis (plus in each case a 5th letter in order was missed). No corresponding exact time interval pattern in RUN was found for the remaining 33 POST SWS P ≤ 1/24 matches found by the relative-order analysis.

Note that burst filtering is generally used when searching for exact time interval patterns because bursts may be associated with intrinsic cellular mechanisms as opposed to interactions between neurons (Abeles et al. 1993). Here we relied on the binning parameter to account for possible timing variability. However, with 1-ms bins and no burst filtering, only 3 repeating patterns were found within POST SWS, and only 8 occur (once each) in both RUN and POST SWS. A previous study of exact time interval patterns in hippocampal data used even larger bins and found many repeating patterns, but did not focus on patterns with ≥4 distinct cells (Nádasdy et al. 1999).

An important reason exact time interval analysis results in so few repeats here is the low firing rates in this data. The mean rates for these cells in POST SWS are 0.31, 0.22, and 0.18 Hz for max_isi = 0, 15, and 50 ms, respectively, and in RUN: 2.2, 1.3, and 0.92 Hz. Furthermore, exact time interval analysis would not be able to capture the sequence similarity between the POST matches found by our relative-order method because of the variability in overall timescale, which ranges by a factor of ≥2 for each sequence (e.g., Fig. 6B). Relative order works here because it allows variability in timing and timescale, as well as different letters (cells) to be missing in each occurrence.

Although these results do not rule out an important role for exact time interval patterns in understanding hippocampal sequence learning, the key point is the extremely low number of exact time interval patterns that repeat in this data (regard-
suggests that relative-order methods may be essential for detecting and understanding hippocampal sequence learning.

**DISCUSSION**

To summarize, we have developed a new approach for analyzing spike train data from an arbitrary number of neurons. It can also be applied to many other types of multiple source neural data. In particular, we analyze sequential activity based on relative time order. We quantify the degree of matching between a reference sequence and a segment of activity (represented by a word) by computing the probability (which is equivalent to a $P$ value) of obtaining a match as good as or better than the best match actually observed in that word. This requires defining a measure of the degree of a match and establishing a relative ranking of various possible matches. Here we chose to define matches in terms of the number of letters in order ($x$) and the number of interruptions ($y$). We describe criteria that any ranking of these matches must satisfy and discuss some possible ranking schemes (e.g., horizontal and diagonal). Computing the probability also requires a null hypothesis that determines an expected distribution of matching. We generally choose to assume that all permutations of the letters of the observed word were equally likely. Thus the probability we compute is actually a conditional probability (i.e., conditional on the letters actually observed in each word). The degree (match/trial ratio) and significance ($Z$ score and $P$ value) of matching over a set of words can be computed from the corresponding set of probabilities. Furthermore, we derive formulae that make these match probability calculations for longer words more manageable.

Our method is designed to identify patterns composed of sequences of events (e.g., spikes, bursts of spikes, EEG events) with each event contributed by a different source (e.g., cell, EEG location). Our method does not cover all possible meaningful patterns, for instance, a sequence of spikes some of which come from the same cell (i.e., repeats in the reference sequence), (nearly) coincident spikes from a subset of cells regardless of relative order, and patterns with specific time intervals. However, the relative-order sequences our method does cover constitute a meaningful category of pattern. For example, the recurring sleep sequences we found in the rodent hippocampus had highly variable interletter and overall timescales, with the latter ranging by over a factor of 2, as well as being approximately 20 times faster than the originally experienced behavioral sequence. This, combined with an absence of such sequences in sleep before the behavior as well as other analyses, provides evidence of the rapid and precise encoding of sequential spatial experience (Lee and Wilson 2000).

In general, any method of quantifying pattern matching (regardless of whether the patterns are based on relative time order, exact time intervals, or any other characteristics) must have the following 2 features: 1) a way to rank (i.e., score) the different possible matches and 2) a null hypothesis describing the expected distribution of matching scores. Given these, the significance of a particular matching score is determined as the probability of obtaining that score or better with respect to the expected distribution of scores. Because we chose to define matches in terms of 2 parameters ($x$ and $y$), the way to rank different matches was not obvious, so we described different reasonable ways to rank them. [With a one-parameter matching

**FIG. 6.** Examples of hippocampal firing patterns detected by relative-order and exact time interval analysis. All examples come from the same set of cells during a single POST SWS session, with RUN reference sequence $S = 1234. . . A$: the only exact time interval pattern (of duration $\leq 500$ ms) in all the data that consists of spikes from $\geq 4$ distinct cells, occurred at least twice during POST SWS, and also matched the RUN reference sequence. This pattern consisted of spikes from exactly 4 distinct cells and occurred exactly twice, with both occurrences shown here. Open triangles represent spikes in the exact time interval pattern. Relative-order method (with max_isi = 50 ms and max_gap = 100 ms) not only detected both these patterns, but detected additional spikes (filled triangles) that further matched the RUN reference sequence, as well as many other patterns that matched the same RUN reference sequence, including both in $B$. $B$: example of 2 patterns with very different timescales that were both detected by the relative-order analysis as matching the same RUN reference sequence. Additional interletter timing variability can be seen across all 4 relative-order matches in $A$ and $B$. Timescale is the same in all rasters.

les of firing order), with the other patterns occurring either only once or not at all. Thus determining which particular exact time interval patterns are important is difficult, especially in low firing rate data such as these. One approach used to address this sample size problem is to ask how many patterns of $n$ spikes repeat $M$ times versus the expected number of such patterns (Abeles and Gerstein 1988; Abeles et al. 1993). However, this does not indicate which particular repeating patterns are special. In contrast, relative-order analysis has yielded strong evidence of learning of the RUN reference sequence in the same data. Furthermore, the variability in timing and timescale in the matching sequences we have found
score, the ranking would be implied, with better matches corresponding to higher scores. An example of such a one-parameter matching score is the simplified H ranking described above. Another is the Spearman rank correlation coefficient (Lehmann 1975), which could be adapted to measure the similarity between a word and a reference sequence.) For the null hypothesis we took the letters observed in each word as given and assumed that each permutation of these letters was equally likely. This has the advantages of simplicity and the availability of formulae to compute probabilities more efficiently (Appendix E). However, other choices could have been made. For instance, with our definition of matches, the words 12477 and 71247 both contained the best possible match with respect to the reference sequence $S = 123456789$. A different definition of matches could have ranked 12477 as better than 71247. Or we could have penalized interruptions more, or different interruptions differently (e.g., 124356 as better than 126345). (A measure like the Spearman rank correlation coefficient would have ranked 12477 as better than 71247, and 124356 as better than 126345, but it also would have ranked 432165 as better than 612345.) Such alternate definitions could have still used the same null hypothesis, or some other one. For instance, we could have chosen a model which specified how the words were generated from scratch. Such a model would implicitly specify a distribution of matching scores. Above we discussed reasons why we did not choose such a model and instead chose to compute conditional probabilities. Nevertheless, one should keep in mind the variety of choices and the reasons for particular decisions. Ultimately, choices are informed by the nature of the data being analyzed and by the types of patterns one is interested in looking for. One might even try several different ways of quantifying matching, then apply a Bonferroni correction to the lowest $P$ value.

Previous methods of analyzing firing patterns across multiple cells include correlation methods, such as the cross-correlation function between a pair of spike trains; the joint peristimulus time histogram (JPSTH), which can be applied to 3 spike trains (Aertsen et al. 1989; Gerstein and Perkel 1969; Palm et al. 1988); and the extension of the cross-correlation to an arbitrary number of spike trains (Grüning et al. 2001; Güttig et al. 2003; Martignon et al. 2000; Nakahara and Amari 2002). Other methods for analyzing the spike trains of an arbitrary number of neurons include an elegant algorithm to detect all repeating firing patterns consisting of spikes separated by exact time intervals (Abeles 1991; Abeles and Gerstein 1988; Abeles et al. 1993), gravity methods (Gerstein and Aertsen 1985; Gerstein et al. 1985), and template methods (Chi et al. 2003; Lee and Wilson 2000; Louie and Wilson 2001).

With the exception of gravity and template methods, these previous methods have classified patterns by the exact time intervals (within binning) between the spikes. Although patterns with exact time intervals that repeat may be of special significance for some types of neural processing (Abeles 1991), this can result in a sample size problem, especially with more complex patterns (i.e., there are too many possible patterns for a given amount of data). Furthermore, when firing rates are high (and thus the sample size problem is reduced), such methods will tend to find patterns that are embedded within much nonpattern activity. In contrast, by classifying patterns based on relative firing order, we can greatly reduce the sample size problem, even in data with low firing rates [e.g., hippocampal pyramidal cells in SWS (Lee and Wilson 2000)]. In addition, by using certain parsing schemes, our method can ignore patterns embedded in the midst of other activity. Because every letter counts, embedded matching patterns are detected only to the extent that the nonmatching activity explicitly does not increase by too much the probabilities of getting such a pattern.

Much important neural processing probably occurs in discrete events. However, cross-correlation methods provide time-averaged values of interactions, and thus cannot analyze the significance of individual events (words). Our method allows a determination of the significance of (matching within) such individual events through the calculated match probability. For instance, each recurring, low probability sequence match we found in hippocampal SWS activity may represent a discrete burst of learned information that is being broadcast to target structures elsewhere in the brain in a process of memory consolidation (Lee and Wilson 2000). Other methods can also allow an identification of important individual events (Abeles 1991; Abeles and Gerstein 1988; Abeles et al. 1993; Grüning et al. 2001).

Our approach allows a calculation of the degree and significance of high-order (i.e., long) sequence matching based on first principles (i.e., the equally likely permutation assumption and treating a set of words as independent Bernoulli trials) without the use of Monte Carlo techniques. We have verified both the validity of the null hypothesis and the accuracy of the calculated $Z$ score in real data by computing the distribution of significance ($Z$ scores) of matching for a set of words with respect to a large set of shuffled reference sequences (Lee and Wilson 2000). In contrast, methods to estimate the significance of patterns found with gravity techniques have not been presented (Aertsen and Gerstein 1985; Gerstein et al. 1985).

One goal of methods that analyze higher-order interactions (e.g., interactions involving more than 2 neurons) is to subtract out the influence of lower-order interactions. For instance, one may want to know whether a larger than expected number of longer sequences can simply be explained by the random chaining together of pairs of neurons that tend to fire in order. If so, this would not necessarily mean that the longer sequences were unimportant, but it would at least suggest a certain type of underlying mechanism. Our approach can account for pairwise biases by weighting permutations unequally instead of equally. The analogous correction for pairwise effects can be done in the 3-neuron JPSTH analysis (Aertsen et al. 1989; Palm et al. 1988). However, a general method of correction for all possible lower-order interactions is still an active area of research for all higher-order methods (Aertsen et al. 1989; Güttig et al. 2003; Martignon et al. 1995, 2000; Nakahara and Amari 2002; Oram et al. 1999; Palm et al. 1988), including our own. One issue is which effects one would want to adjust for. After deciding which lower-order interactions to account for, in our approach one could attempt to handle them by weighting the permutations appropriately. A recent study has described a case in which lower-order statistics such as the peristimulus time histogram (PSTH) and trial-by-trial spike count can account for the observed number of exact time interval patterns that repeat (Oram et al. 1999). Analogously, when using relative-order analysis on data with clear trial onset times, one should check the set of N PSTHs (one from each cell). If the set of PSTHs shows a sequence of peaks that resemble the reference se-
Innovative Methodology

A. K. LEE AND M. A. WILSON

sequence, this could be sufficient to explain the amount of sequence structure found (e.g., through Monte Carlo simulations). However, our method can detect sequence structure even when it would not show up in the PSTHs, such as in activity with large variability in timing and timescale, and activity in which only a fraction of trials have sequence matches (but matches that are rarer than their occurrence rate).

Finally, although all the other methods described above attempt to discover previously unknown patterns that may be significant, our method and the template method (Chi et al. 2003; Lee and Wilson 2000; Louie and Wilson 2001) specifically test for evidence of a particular reference pattern (sequence) within the data. That is, unlike the other methods, our approach does not automatically search all possible patterns (in our case, all $N!$ possible reference sequences constructed from permutations of $N$ letters) for interesting candidate patterns. Trying all possible reference sequences would be prohibitively time consuming for longer reference sequences (and even if it were possible, the significance of any findings would have to be determined in a different manner). Thus our method generally requires educated guesses of good candidate reference sequences. Once one has chosen a reference sequence, however, our method allows evidence for that sequence to be detected even with missing letters (by sampling using conditional probabilities) and errors (interruptions within the matching region and letters outside of it). Furthermore, because our method detects partial and imperfect matches, a candidate reference sequence need only be "close enough" to find any underlying sequence structure (where "close enough" is determined by setting $P'$).

In conclusion, a full understanding of brain computations will most likely require knowledge of complex interactions among large numbers of neurons and across multiple brain areas. However, the number of possible firing patterns grows exponentially with the number of neurons (or other sources) involved. The current ability to record from tens to over 100 neurons simultaneously (Ikegaya et al. 2004; Meister et al. 1991; Wessberg et al. 2000; Wilson and McNaughton 1993) already makes a search of all possible patterns virtually impossible for all known analysis methods. Thus one key challenge will be to decide which specific patterns to search for. One possibility is to limit the search to patterns of behavioral relevance, such as spatial sequences experienced by a rodent or songs learned by birds (Chi et al. 2003; Dave and Margoliash 2000; Lee and Wilson 2000; Louie and Wilson 2001). Given a manageable set of candidate patterns, methods such as ours can then be used to test them quantitatively for significance.

**APPENDIX A: KEY SYMBOLS**

- $S$: a reference sequence
- $N$: length of reference sequence (e.g., number of cells)
- $W$: a word
- $n$: number of letters in a word, including repeated letters
- $k$: number of distinct letters in a word
- $m$: multiplicity vector whose entries ($m_1, m_2, m_3, \ldots, m_N$) give the number of times each letter (1, 2, 3, \ldots, $N$) in the reference sequence occurs in a word
- $(x, y)$: a match consisting of $x + y$ consecutive letters of which at least $x$ letters are in strictly increasing order with respect to a given reference sequence (and are thus interrupted by at most $y$ letters)

$P'$ probability value that determines which words are considered trials (i.e., a "$P' \leq P'$ trial") and which words are considered matches (i.e., a "$P \leq P'$ match")

$C(a, b) = a!/b! \times (a - b)!$

**APPENDIX B: PROOFS THAT SHOW THE H AND D MATCH RANKINGS OBEY CONSTRAINT (c*)**

Recall the constraint (c*) on ranked match lists: the ranking of matches must be such that a word that contains a worse match must not automatically also contain a better match.

Given a word of length $n$ with $k$ distinct letters, we can assume without loss of generality that $N = k$, where $N$ = the length of the reference sequence. Then the $k$ distinct letters are 1, 2, 3, \ldots, $k$ with corresponding multiplicities $m_1, m_2, m_3, \ldots, m_k$. That is, $m$ is a length $k$ vector whose entries give the number of occurrences of each of the distinct letters in the word, and $\Sigma m_i = n$.

**H ranking**

To prove the H ranking obeys the constraint (c*), we need to show that for any match $(x, y)$ in the H ranked match list, and for any multiplicity vector $m$ (i.e., any arbitrary set of letters) which could be arranged to contain such a match as well as any set of matches better than it according to the H ranking scheme, there exists a permutation that contains such a match but none of those better than it. For any $(x, y)$ match in the H ranked match list except $(2, 0)$ [i.e., any $(x, y)$ match with $x \geq 3$], construct a permutation as follows.

Start with the $x$ letters $k - x + 1, k - x + 2, \ldots, k$ and arrange them in order. They constitute the $x$ matching letters (and are shown below inside the $[ ]$). From the remaining $n - x$ letters, let $k = (1, 2, 2, \ldots, 2)$ (for instance, if the number of 1's among the remaining $n - x$ letters is $\geq y$, then I consists of 1's). Divide the remaining $n - x - y$ letters into $G_1 =$ the letters $k - x + 1$ and $G_2 =$ the letters $k - x + 2$. Place the letters in $G_1$ in decreasing order in front of the matching letters, and the letters in $G_2$ in decreasing order behind the matching letters.

\[
\begin{align*}
\text{A} & \quad \text{B} \\
& \downarrow \quad \downarrow \\
[k - x + 1, k - x + 2, \ldots, k - 2k - 1k] \\
& \quad \quad [k - x, k - x + 2, \ldots, 21, 1]
\end{align*}
\]

Then insert the letters of I between the matching letters as follows. Insert any $k$s where A is, and all the other letters in decreasing order where B is. This permutation contains an $(x, y)$ match but no match better than it according to the H ranked match list because it contains no match $(u, v)$ with $u > x$ and no match $(u, v)$ with $u = x$ and $v < y$.

If $(x, y) = (2, 0)$, then simply construct the following permutation

\[
\begin{align*}
\text{k} & \quad \text{k} \quad \text{k} - 1 \quad \text{k} - 1 \quad \text{k} - 1 \quad \text{2} \quad \text{1} \quad \text{2} \quad \text{2} \quad \text{1} \quad \text{1} \quad \text{2}
\end{align*}
\]

This permutation contains a $(2, 0)$ match but no match better than it, given that it contains no match $(u, v)$ with $u > 2$.

**D ranking**

To prove the D ranking obeys the constraint (c*) is more complicated. As above, we need to show that for any match $(x, y)$ in the D ranked match list, and for any multiplicity vector $m$ whose letters could be arranged to contain such a match as well as any set of
matches better than it according to the D ranking scheme, there exists a permutation that contains such a match but none of those better than it. For any \((x, y)\) match in the D ranked match list \([i.e., (x, y) with x - y \geq 2]\), construct a permutation as follows.

Start exactly as in the H ranking proof

\[
k \cdots k k - 1 \cdots k - 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \\
\phantom{k \cdots k k - 1 \cdots k - 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \cdots k - x + 1 \\[k - x + 1 k - x + 2 \cdots k - 2 k - 1]
\]

The difference is where we insert the letters of I. There are \(x - 1\) gaps between adjacent matching letters. Given that \(x - y \geq 2\), there is at least one more gap than letters in I. We will insert no more than one letter from I in each gap, and we will insert letters in such a way that each insertion is an interruption. Because we will add y interruptions between \(x\) matching letters with no more than one interruption per gap, this will leave no matches \((u, v)\) such that \(u - v > x - y\). Furthermore, we will insert the letters such as to leave no match \((u, v)\) with \(u > x\). Together, these will satisfy the constraint.

Here is the insertion scheme. First, take all the letters in I that are \(\leq k - x + 1\) and place them in increasing order starting from the far right, one per gap. Say there are \(s\) such letters (labeled in increasing order \(i_1, \ldots, i_s\)), then we have

\[
k - x + 1 k - x + 2 \cdots k - s - 1 k - s k - s + 1 k - s + 2 \cdots i_1 \downarrow \uparrow i_2 \downarrow \uparrow \cdots \downarrow \uparrow i_s \downarrow \uparrow k - 2 k - 1 k
\]

With the remaining \(x - s\) matching letters, insert the remaining letters of I (relabeled \(i_1, \ldots, i_{s-a}\) in increasing order) in the following manner

\[
(3) \text{ insert } i_s \text{ if } i_s \leq k - s - 3 \\
(2) \text{ insert } i_{s-1} \text{ if } i_{s-1} \leq k - s - 2 \\
(1) \text{ insert } i_{s-a} \text{ if } i_{s-a} \leq k - s - 1
\]

Start with (1) (where \(i_s\) is the smallest remaining letter), then (2), and so forth, until the required condition is not met. After the first time the condition is not met, insert the remaining letters in the following manner

\[
(1) \text{ insert } i_{s-a} \\
(2) \text{ insert } i_{s-a+1} \\
(3) \text{ insert } i_{s-a+2} \\
k - x + 1 k - x + 2 \cdots k - x + 2 k - x + 3 k - x + 4 \cdots k - s
\]

Start with (1) (where \(i_{s-a}\) is the largest remaining letter), then (2), and so forth, until there are no letters left in I. This is the end of the construction.

For example, say \((x, y) = (8, 6)\), we have \([k - x + 1 k - x + 2 \cdots k - 2 k - 1]\) \([2, 3, 4, 5, 6, 7, 8, 9]\), and I contains \(1, 1, 3, 5, 8, 9\). The first insertion step gives \([2, 3, 4, 5, 6, 7, 8, 9]\), With the remaining \(x - s\) letters \([2, 3, 4, 5, 6, 7]\), we insert the remaining letters \(3, 5, 8, 9\) of I to give \([2, 9, 3, 8, 4, 5, 5, 6, 3, 7]\).

The permutation so constructed contains an \((x, y)\) match but no match better than it according to the D ranked match list, given that it contains no match \((u, v)\) with \(u - v > x - y\) and no match \((u, v)\) with \(u - v = x - y\) and \(u > x\).

APPENDIX C: COMPARISON OF THE H AND D MATCH RANKINGS IN TERMS OF PROBABILITIES

To help us evaluate the H and D ranking schemes, as well as other possible schemes, consider the probabilities of getting each type of match given different values for \(n\) (word length) and \(k\) (number of distinct letters). The probability for a particular match is computed as the fraction of the \(n!\) permutations of the letters that contain that match. Our intuition is that less-probable matches should be considered better. We consider a few cases to illustrate some key points (Fig. C1).

We see that (at least in these few cases) both the H and D ranking schemes roughly rank matches by increasing probability (i.e., better matches having lower probability), although this breaks down somewhat for the D ranking scheme for worse matches. We can also see examples of the potential shortcomings of ranking based strictly on probability. For instance, in the \(n = 9, k = 9\) case, \(P(7, 2) = 0.0043 < 0.0050 = P(6, 0)\). Even though \((7, 2)\) is less probable than \((6, 0)\), intuitively one might consider \((6, 0)\) to be a better match because the \((7, 2)\) match has one additional letter in order at the price of 2 more interruptions. There are also shortcomings with the H ranking scheme [which also places \((7, 2)\) as better than \((6, 0)\)]. For an extreme example of this shortcoming, if \(n = 100\) and \(k = 100\), the H ranking scheme places \((51, 20)\) as better than \((50, 0)\). On the other hand, the D ranking scheme balances rewarding larger \(x\) with penalizing larger \(y\) [where \(x\) and \(y\) refer to matches \((x, y)\)]. We have primarily used the D ranking.

### A

For \(n = 7, k = 7\), the probabilities for the following matches:

<table>
<thead>
<tr>
<th>((7, 0))</th>
<th>((6, 1))</th>
<th>((5, 2))</th>
<th>((4, 3))</th>
<th>((3, 4))</th>
<th>((2, 5))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00020</td>
<td>0.0073</td>
<td>0.0222</td>
<td>0.0591</td>
<td>0.0999</td>
<td>0.1417</td>
</tr>
<tr>
<td>0.2942</td>
<td>0.4016</td>
<td>0.4522</td>
<td>0.5998</td>
<td>0.8248</td>
<td>0.8921</td>
</tr>
<tr>
<td>0.9119</td>
<td>0.9149</td>
<td>0.99980</td>
<td>0.99980</td>
<td>0.99980</td>
<td></td>
</tr>
</tbody>
</table>

### B

For \(n = 9, k = 9\), the probabilities for the following matches:

<table>
<thead>
<tr>
<th>((9, 0))</th>
<th>((8, 1))</th>
<th>((7, 2))</th>
<th>((6, 3))</th>
<th>((5, 4))</th>
<th>((4, 5))</th>
<th>((3, 6))</th>
<th>((2, 7))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000028</td>
<td>0.000018</td>
<td>0.00056</td>
<td>0.00219</td>
<td>0.00439</td>
<td>0.00508</td>
<td>0.00529</td>
<td>0.00500</td>
</tr>
<tr>
<td>0.01749</td>
<td>0.0347</td>
<td>0.0497</td>
<td>0.0610</td>
<td>0.0776</td>
<td>0.2411</td>
<td>0.2785</td>
<td>0.3039</td>
</tr>
<tr>
<td>0.2829</td>
<td>0.5828</td>
<td>0.6751</td>
<td>0.7234</td>
<td>0.7400</td>
<td>0.7261</td>
<td>0.9250</td>
<td>0.9698</td>
</tr>
<tr>
<td>0.9822</td>
<td>0.9876</td>
<td>0.9858</td>
<td>0.9865</td>
<td>0.9866</td>
<td>0.9999972</td>
<td>0.9999972</td>
<td>0.9999972</td>
</tr>
</tbody>
</table>

### FIG. C1.

Exact probabilities that a random permutation of a word’s letters contains each type of possible match. Here random means that each permutation was equally likely. Results are shown for various example words. A: words 7 letters long with all 7 letters distinct. B: words 9 letters long with all 9 letters distinct. For each word the possible matches are shown above (compare with Fig. 2) and the corresponding probabilities are shown below.
scheme in our previous work, but we also showed that the H ranking gave similar results there (Lee and Wilson 2000). The H ranking scheme also has merit. For instance, in the n = 100, k = 100 case, one might want to rank (60, 30) as better than (31, 0) (which the H, but not D, ranking does). It is a matter of which aspects of matching one wants to emphasize.

One might wonder why not just use ranking based strictly on probability (i.e., lower probability matches always being better). There is a major practical difficulty in using such a scheme. Because the exact probabilities of matches are sensitive to the length and specific letter composition of each word, one would have to compute the exact probability of getting each type of match from scratch for each word encountered. This would be needed for determining the ranking of matches so that the best match contained in the word could be identified. For even moderately long words (e.g., n > 10) this would be very computationally expensive. On the other hand, the H and D ranking schemes give the ranking of matches before anything is calculated. Thus one needs to calculate the probability of only the best match found or better, and this can be approximated (as shown in the text) by using the formulae derived in APPENDIX E.

APPENDIX D: RECIPROCAL RELATIONSHIP BETWEEN WORDS AND REFERENCE SEQUENCES

For any word W without repeated letters (i.e., n = k), any reference sequence S of length N (n ≤ N), any match F contained by W, and any match ranking scheme, we prove the following result. Consider the fraction of the n! permutations of the word’s letters that contain a match F or better with respect to S. Now take any permutation S’ of the original reference sequence, and ask whether the original word W contains a match F or better with respect to S’. Consider the fraction of the N! permutations of S for which this is the case. We show that these two fractions are equal.

Without loss of generality, assume n = N (i.e., W consists of exactly one of each letter 1 through N). Let S* be the set of all N! possible reference sequences, and W* the set of all N! possible permutations of W. Furthermore, let d = the number of elements in W* that correspond to the single element 1234...N in S*, and d’ = the number of elements in S* that correspond to the single element W in W*. Note that when we say an element s of S* corresponds to an element w of W*, or that an element w of W* corresponds to an element s of S*, in both cases we mean that the permutation w contains a match F or better with respect to the reference sequence s. This is illustrated in Figure D1 for the case where n = 7, W = 3127564, and F = (4, 1). Because the letters are just labels, we can permute them without altering which elements correspond to each other, as long as we permute them in the same way in both S* and W*. By permuting any element of W* into W or any element of S* into 1234...N, we can see the following: each element s in S* corresponds to exactly d elements in W*, and each element w in W* corresponds to exactly d’ elements in S*. Now there are N! elements in S*, each of which corresponds to d elements in W*, thus W* as a whole is “hit” a total of N! × d times by S*. Again, because each element of W* is essentially equivalent (through label permutation) to every other element of W*, those N! × d hits must be distributed uniformly across W*.

Given that W* is also of size N!, this means each element of W* is hit by exactly d elements of S*. However, those d elements of S* are precisely the d’ elements that correspond to each element of W*. Thus d = d’, completing the proof.

Note that the proof does not require that the match relation be symmetric. In fact, even in the N = n case above, the match relation is not symmetric: W having a match F with respect to S does not imply S has a match F with respect to W. For example, W = 3127564 has a (4, 1) match with respect to S = 1234567, but W = 1234567 does not have a (4, 1) match with respect to S = 3127564. [This can be seen by permuting S = 3127564 into 1234567 by converting 3 to 1, 1 to 2, 2 to 3, 7 to 4, and so on. This converts W = 1234567 into 2317564, and 2317564 does not have a (4, 1) match with respect to 1234567.]

Now we show how the result also holds for N > n, as long as n = k. Consider the example used in the text: S = 1234...N, N = 9, W = 524679, F = (5, 0), and the D match ranking scheme. Figure 3 shows the 11 out of 6! permutations of W that contain a (5, 0) match or better with respect to S. Now for how many of the 9! possible 9-letter reference sequences does 524679 contain a (5, 0) match or better? Because 1, 3, and 8 are missing in W, we need only consider the order of 2, 4, 5, 6, 7, and 9 in each candidate reference sequence to determine whether there is a (5, 0) match or better. Thus the problem is reduced to the N = n case for which the proof applies. Therefore there are exactly 11 out of 6! orderings of those letters that a qualifying reference sequence must contain. The 3 missing letters can now be placed anywhere around those 6 letters to complete the reference sequences. There are 9!/6! unique ways to place the 3 letters, giving a total of 11 × 9!/6! reference sequences [i.e., (11 × 9!/6!)9/11! = 11!/6! = 1.5% of all 9! possible reference sequences].

In the case of words with repeated letters (n > k), the result generally does not hold. For example, given S = 123, n = 4, and k = 3, there can be a larger [W = 1323, F = (3, 1)], smaller [W = 1223, F = (3, 1)], or equal [W = 1233, F = (3, 0)] fraction of the 3! possible reference sequences that gives a match F or better compared with the fraction of word permutations that does so.

APPENDIX E: METHODS FOR COMPUTING EXACT, UPPER BOUND, AND LOWER BOUND MATCH PROBABILITIES ASSUMING ALL PERMUTATIONS ARE EQUALLY LIKELY

Recall the definition of match probability: given a particular reference sequence and an arbitrary word of length n, the probability of an (x, y) match—assuming all permutations are equally likely—is the fraction of the n! permutations of the n letters of that word which contain an (x, y) match.

Given a word of length n with k distinct letters, we can assume without loss of generality that N = k, where N = the length of the reference sequence. Then the k distinct letters are 1, 2, 3, …, k with corresponding multiplicities m1, m2, m3, …, mk. That is, m is a length k vector whose entries give the number of occurrences of each of the distinct letters in the word, and Σm = n.

Formula E1: exact probability of a (k, 0) match

Given any multiplicity vector m of length k, where k = the number of distinct letters, the exact probability that a random permutation contains a (k, 0) match, assuming all n! permutations of the n letters are equally likely, can be computed as follows.

Let M = the minimum entry in m (i.e., the multiplicity of the letter with smallest multiplicity). Thus each letter occurs at least M times. Because we have assumed m is of length k, M ≥ 1. Each permutation will contain exactly 0, 1, 2, …, or M (k, 0) matches. Out of the n! total possible permutations, let N(i) = the number of permutations simultaneously containing exactly i (k, 0) matches. Therefore the

![Figure D1](http://jn.physiology.org/)

**FIG. D1.** Example diagram illustrating part of proof in APPENDIX D. See text for details.
exact probability we seek is given by $[N(1) + N(2) + \cdots + N(M)]/n!$.

Consider trying to compute $N(1)$. Out of the $n$ letter positions for each permutation, there are $n - k + 1$ unique sets of $k$ consecutive positions in which to place a $(k, 0)$ match.

$$
\begin{array}{cccccccc}
1 & 2 & 3 & \cdots & n \\
* & * & * & \cdots & *
\end{array}
$$

Because each distinct letter is used exactly once in each $(k, 0)$ match, there are $\Pi_m$ ways to choose which of the $k$ letters (from among the repeats) to use in the $(k, 0)$ match. The remaining $n-k$ letters can be arranged in any of $(n-k)!$ ways. This gives a total of $(n-k+1) \times (\Pi_m) \times (n-k)!$ permutations with at least one $(k, 0)$ match. If $M = 1$, this gives $N(1)$ and we are done. However, if $M > 1$, this overcounts $N(1)$ because there will be permutations containing more than one $(k, 0)$ match, and this formula will count a permutation containing exactly $R$ $(k, 0)$ matches exactly $R$ times instead of once. Thus let $(n-k+1) \times (\Pi_m) \times (n-k)!$ be represented by $N(1)+1$, where $N(1)+1 = N(1)$ if $M = 1$. Then

$$
N(1+) = 1 \times N(1) + 2 \times N(2) + 3 \times N(3) + \cdots + M \times N(M)
$$

Now how do we calculate $N(R)$ for $R > 1$? For $N(2)$, for example, we use an analogous procedure to that for $N(1)+1$. First we need to count the number of unique sets of (nonoverlapping) pairs of $k$ consecutive positions in which to place 2 $(k, 0)$ matches. For $n = 8$ and $k = 3$ there are 6 such pairs

$$
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & n = 8 \\
* & * & * & # & # & # & # & *
\end{array}
$$

Call the number of such sets $Q(n,k,2)$. We must then multiply this by $(\Pi_m) \times (\Pi_m) \times (m_1 - 1)$, which gives us the number of ways to select the $2 \times k$ letters (among the repeats) to place in the 2 $(k, 0)$ matches. Finally we must multiply this by $(n-2 \times k)!$ for the number of ways to place the remaining $n-2 \times k$ letters. Again, if $M = 2$, this gives us $N(2)$ and we are done. However, if $M > 2$, this overcounts $N(2)$ because there will be permutations containing more than 2 $(k, 0)$ matches, and our formula will count a permutation containing exactly $R$ $(k, 0)$ matches exactly $R$ times instead of once, where $C(a, b) = a!/b!(a-b)!$. Thus let $Q(n, k, 2) \times (\Pi_m) \times [\Pi_m] \times (m_1 - 1) \times (n-2 \times k)!$ be represented by $N(2)+2$, where $N(2)+2 = N(2)$ if $M = 2$. Then

$$
N(2+) = 1 \times N(2) + 2 \times N(3) + 3 \times N(4) + \cdots + C(M, 2) \times N(M)
$$

Continuing in this fashion to compute $N(3)+\ldots, N(M)+\ldots$ [i.e., $N(q+) = Q(n, k, q) \times (\Pi_m) \times [\Pi_m] \times (m_1 - 1) \times \cdots \times [\Pi_m] \times (m_1 - q + 1) \times (n-q \times k)!$, where $Q(n, k, q) = C[n - q - (k - 1), q]$, we obtain

$$
\begin{align*}
N(1+) &= 1 \times N(1) + 2 \times N(2) + 3 \times N(3) + \cdots + C(R, 1) \times N(R) \\
N(2+) &= 1 \times N(2) + 2 \times N(3) + \cdots + C(R, 2) \times N(R) \\
N(3+) &= 1 \times N(3) + \cdots + C(R, 3) \times N(R) \\
\end{align*}
$$

Note that this sum has $C(k,x)$ terms, one for each of the ways to pick $x$ distinct letters out of the $k$ total distinct letters, whereas the product takes care of the repeats. Also note that if $x = k$ this expression reduces to

$$
N(M+) = C(M, M) \times N(M)
$$

Observe that

$$
[N(1+) - N(2+) + N(3+) - N(4+) + \cdots + (-1)^{M-1} \times N(M)]
$$

Therefore our exact probability answer is given by

$$
P = 1/n! \times [N(1+) - N(2+) + N(3+) - N(4+) + \cdots + (-1)^{M-1} \times N(M)] \quad (EJ)
$$

**Formula E2: upper bound probability of an (x, y) match**

Given any multiplicity vector $m$ of length $k$, where $k$ is the number of distinct letters, an upper bound of the probability that a random permutation contains an $(x, y)$ match (where $2 \leq x \leq k$ and $x + y \leq n$), assuming all $n!$ permutations of the $n$ letters are equally likely, can be computed as follows. In particular, this formula actually computes an upper bound of the probability that a permutation contains any $(x, y)$ match where $u \geq x$ and $v \leq y$.

The upper bound formula

$$
P \leq 1/n! \times X \times Y \times Z \quad (E2)
$$

is a product of 3 terms, where we will explain each term one at a time. First, $X$ is the number of unique sets of $x$ positions spanning no more than $x + y$ consecutive positions. Each such set represents one of the $x$ positions in which the $x$ matching letters can be placed. Now

$$
X = [n - (x + y) + 1] \times C(x + y - 1, y) + \sum_{i=0}^{y-1} C(x + i - 1, i)
$$

for $y > 0$ or

$$
X = [n - (x + y) + 1] \times C(x + y - 1, y)
$$

for $y = 0$.

This formula can be explained as follows. If $y = 0$, then the formula is just $n-x+1$, the number of unique sets of $x$ consecutive positions [as in $N(1)+1$ in formula E1 above]. If $y > 0$, the first term, $[n - (x + y) + 1] \times C(x + y - 1, y)$, consists of $n - (x + y) + 1$ is the number of unique sets of $x + y$ consecutive positions, and $C(x + y - 1, y)$ is the number of ways to place the $y$ interruptions in any of the $x + y - 1$ possible positions excluding the first position. By fixing the first position (i.e., not allowing the first position to be occupied by an interruption), we ensure that each set of $x$ positions within $x + y$ consecutive positions is counted only once. The second term accounts for the sets of $x$ positions spanning less than $x + y$ positions at the right end of the $n$ positions.

Second, $Y$ is the number of ways to pick $x$ distinct letters (including accounting for the repeats) to put in order (in those $x$ positions) for the $(x, y)$ match.

$$
Y = \sum_{\text{all } x \text{-tuples } (i_1, i_2, \ldots, i_x)} \left( \prod_{j=1}^{x} m_{i_j} \right)
$$

$Y$ is a summation over all $x$-tuples $(i_1, i_2, \ldots, i_x)$ in which $i_1 \leq i_2 \leq \cdots \leq i_x$. Note that this sum has $C(k, x)$ terms, and each of the ways to pick $x$ distinct letters out of the $k$ total distinct letters, whereas the product takes care of the repeats. Also note that if $x = k$ this expression reduces to

$$
J Neurophysiol • VOL 92 • OCTOBER 2004 • www.jn.org

Innovative Methodology
Finally, $Z = (n - x)! = \text{the number of ways to arrange the } n - x \text{ remaining letters in any order in the remaining } n - x \text{ positions. These } n - x \text{ positions include any interrupting positions within the match.}

The reason this is an upper bound is that it constructs all possible permutations containing an $(x, y)$ match, but by letting the remaining positions be any letter, each permutation may be constructed (and thus counted) more than once. By letting the remaining positions be any letter, and by allowing the interruptions to be outside the $x$ matching letters or absent, our construction also constructs (and counts) all permutations containing a $(u, v)$ match where $u \equiv x$ and $v \equiv y$.

**Formula E3: lower bound probability of a $(z, 0)$ match**

Given any multiplicity vector $\mathbf{m}$ of length $k$, where $k$ is the number of distinct letters, a lower bound of the probability that a random permutation contains a $(z, 0)$ match (where $2 \leq z < k$), assuming all $n!$ permutations of the $n$ letters are equally likely, can be computed as follows.

We start by constructing permutations containing at least one $(z, 0)$ match. There are $n - z + 1$ unique sets of $z$ consecutive positions in which to place the $z$ matching letters

$$Y = \sum_{\text{all } x\text{-tuples } (i_1, i_2, \ldots, i_x) \text{ where } 1 \leq i_1 < i_2 < \cdots < i_x \leq n} \left( \prod_{j=1}^x m_{i_j} \right)$$

ways to pick $z$ distinct letters (including accounting for the repeats) for the $(z, 0)$ match (note that this $Y$ is the same as in the upper bound formula), and $(n - z)!$ ways to arrange the remaining $n - z$ letters. We call the product, $(n - z + 1) \times Y \times (n - z)!$, of these 3 terms $N_{\text{fix}}(z)$ because it is constructed by fixing $z$ consecutive positions and letting the remaining positions contain any letters. As with $N(1)$ of formula E1, this counts some permutations more than once.

To see exactly how $N_{\text{fix}}(z)$ overcounts, we introduce the following notation. Let $N(w_0; w_1, w_2, \ldots, w_q)$, where $w_0 = w_1 = \cdots = w_{z-1} \equiv w_0$, be the number of permutations (out of the $n!$ total) containing a $(w_0, 0)$ match, a $(w_2, 0)$ match, etc.; and where each of these matches is as long (i.e., share no positions), and where each of these matches is as long as can be. Because of this last condition, for a fixed $w_0$, no permutation can belong to more than one such set. To see this, consider any permutation. Partition it into blocks of consecutive positions with strictly increasing letters. For instance, $3146232285679131145$ is partitioned into $3, 146, 23, 2, 28, 5679, 13, 1, 145$, revealing exactly $3 (1, 0)$ matches, $3 (2, 0)$ matches, $2 (3, 0)$ matches, and $1 (4, 0)$ match. If $w_0 = 3$, then this belongs to $N(3; 3, 3, 4)$.

$N_{\text{fix}}(z)$ overcounts by multiply counting any permutation that belongs to $N(z; w_0, w_2, \ldots, w_q)$ except for $N(z; z)$. For instance, $N_{\text{fix}}(z)$ counts a given permutation in $N(z; z + y)$ exactly $y + 1$ times, a permutation in $N(z; z, z + y)$ exactly $q$ times, and a permutation in $N(z; z + y_1, z + y_2, \ldots, z + y_q)$ (where $0 \leq y_1 \leq y_2 \leq \cdots \leq y_q \leq y$) exactly $(q_1 + 1) (q_2 + 1) \cdots (q + 1) + (y + 1)$ times.

To correct for this overcounting, we introduce $N(z; z)$, which constructs all permutations containing $\geq 2$ nonoverlapping $(z, 0)$ matches. [Again, by nonoverlapping we mean that the $2 (z, 0)$ matches cannot share any positions. For example, $123134$ is counted in $N(3; 3, 3, 3)$ but 1234 is not.] The number of unique sets of $(\geq 2)$ nonoverlapping pairs of $z$ consecutive positions in which to place 2 $(z, 0)$ matches is $\binom{n}{2} \times \left( (n - 2) \times (z + 1) \right) \times \left( z + 2 \right)$. The number of ways to select the $2 \times z$ letters (accounting for repeats) is a nested sum. The reduced multiplicity vector $\mathbf{m}'$ is constructed from the original multiplicity vector $\mathbf{m}$ by subtracting 1 from the multiplicity of each letter selected in the first (outer) $z$-tuple $(i_1, i_2, \ldots, i_z)$ and the “remaining” $z$-tuples $(r_1, r_2, \ldots, r_j)$ are the $z$-tuples that are available from this reduced multiplicity vector. Then

$$N_{\text{fix}}(z, z) = \frac{1}{2} \times \left( (n - 2) \times (z + 1) \right) \times \left( z + 2 \right) \times Y' \times (n - 2 \times z)!$$

Generally, $N_{\text{fix}}(z, z)$ also counts some permutations more than once. Note that $N_{\text{fix}}(z, z)$ is 0 if no permutation can contain 2 (nonoverlapping) $(z, 0)$ matches.

We propose that $N_{\text{fix}}(z) - N_{\text{fix}}(z + 1) - N_{\text{fix}}(z, z)$ will subtract off all the overcounting of $N_{\text{fix}}(z)$ and more, thus giving a lower bound. To see this, consider a given permutation in $N(z; z + y_1, z + y_2, \ldots, z + y_q)$ (where $0 \leq y_1 \leq y_2 \leq \cdots \leq y_q$), which $N_{\text{fix}}(z)$ counts exactly $q + y_1 + y_2 + \cdots + y_q$ times. $N_{\text{fix}}(z + 1)$ counts this permutation exactly $y_1 + y_2 + \cdots + y_q$ times, and $N_{\text{fix}}(z, z)$ counts it at least $(y_1 + y_2 + \cdots + y_q + q)! - \sum (y_i + 1)z)!/z)!$ times. To understand this last expression, consider that each of the $q (y_i + 1)$, 0 matches, where $y_i \geq 0$, contains $y_i + 1$ (overlapping) $(z, 0)$ matches. $N_{\text{fix}}(z, z)$ counts this permutation as many times as there are ways to select 2 simultaneous $(z, 0)$ matches out of these $q (y + 0)$ matches. How many ways are there to do this? If we restrict ourselves to selecting only 1 $(z, 0)$ match per $(z, y)$ match, then this amounts to asking what is the sum of the product of all possible pairs of values $y_1 + 1$, $y_2 + 1$, $\ldots$, $y_q + 1$. The square of the sum of these values, $\sum (y_i + 1)z)!/z)!$, gives us the sum we need, except that we must subtract all the $(y_i + 1)z)!/z)!$ terms, then divide by 2, given that the remaining terms have coefficient 2. The reason this undercounts $N_{\text{fix}}(z, z)$ is because we have ignored the cases where $y_i \geq z$, in which case one can simultaneously select both $(z, 0)$ matches from such a $(z + y)$ match. This expression has $q \times (q - 1)/2$ [i.e., $C(q, 2)!$ terms], each $\geq 1$, thus $N_{\text{fix}}(z, z)$ counts such a permutation at least $q \times (q - 1)/2$ times.

Thus $N_{\text{fix}}(z) - N_{\text{fix}}(z + 1) - N_{\text{fix}}(z, z)$ counts each permutation in $N(z; z + y_1, z + y_2, \ldots, z + y_q)$ no more than $(q + y_1 + y_2 + \cdots + y_q + 1)! - \sum (y_i + 1)z)!/z)!$ times. This reduces to $q \times (q - 1)/2$, which is never $> 1$, for any $q \geq 0, 1, 2, \ldots$ (i.e., it never overcounts). Therefore the lower bound probability is

$$P \geq 1/n! \times \left( N_{\text{fix}}(z) - N_{\text{fix}}(z + 1) - N_{\text{fix}}(z, z) \right)$$

If this formula gives a value $< 0$, then one can just set the lower bound probability to 0 because probabilities cannot be negative. Note that by setting $z = k$ [in which case $N_{\text{fix}}(z + 1) = 0$], one can see the relationship between this formula and formula E1.

Note

If the H ranking scheme is used, one may be able to compute tighter upper and lower bounds of the probability of obtaining an $(x, y)$ match or better using results from work on longest increasing (and decreasing) subsequences (Schensted 1961). If the simplified H ranking scheme is used, then one may be able to use these results to compute the exact probability for any match.

**Acknowledgments**

Special thanks go to W. Bradley for a detailed review of an earlier version of this manuscript. We are also grateful to W. A. Asaad, T. Davidson, M.

*J Neurophysiol* • VOL 92 • OCTOBER 2004 • www.jn.org

GRANTS

This work was supported by RIKEN and National Institutes of Health Grants to M. A. Wilson and a Howard Hughes Medical Institute predoctoral fellowship to A. K. Lee.

REFERENCES


