

Observer agreement for event sequences: Methods and software for sequence alignment and reliability estimates

VICENÇ QUERA

Universitat de Barcelona, Barcelona, Spain

ROGER BAKEMAN

Georgia State University, Atlanta, Georgia

AND

AUGUSTO GNISCI

Seconda Università degli Studi di Napoli, Naples, Italy

When sequences of discrete events, or other units, are independently coded by two coders using a set of mutually exclusive and exhaustive codes, but the onset times for the codes are not preserved, it is often unclear how pairs of protocols should be aligned. Yet such alignment is required before Cohen's kappa, a common agreement statistic, can be computed. Here we describe a method—based on the Needleman and Wunsch (1970) algorithm originally devised for aligning nucleotide sequences—for optimally aligning such sequences; we also offer the results of a simulation study of the behavior of alignment kappa with a number of variables, including number of codes, varying degrees of observer accuracy, sequence length, code variability, and parameters governing the alignment algorithm. We conclude that (1) under most reasonable circumstances, observer accuracies of 90% or better result in alignment kappas of .60 or better; (2) generally, alignment kappas are not strongly affected by sequence length, the number of codes, or the variability in the codes' probability; (3) alignment kappas are adversely affected when missed events and false alarms are possible; and (4) cost matrices and priority orders used in the algorithm should favor substitutions (i.e., disagreements) over insertions and deletions (i.e., missed events and false alarms). Two computer programs were developed: Global Sequence Alignment, or GSA, for carrying out the simulation study, and Event Alignment, or ELign, a user-oriented program that computes alignment kappa and provides the optimal alignment given a pair of event sequences.

Observer reliability is a central concern whenever behavior is coded by trained observers. If different observers view the same stream of behavior and code it using the same coding scheme but obtain substantially different data, then we cannot be confident about the objectivity or quality of their data and hence cannot trust results from analyses based on these data. When behavior is observed continuously and coded sequentially, investigators usually demand point-by-point, or local, agreement (Bakeman & Gottman, 1997), using a set (or sets) of mutually exclusive and exhaustive codes. For example, the codes "quiet-alert," "fussy," "crying," "rapid eye movement sleep," and "deep sleep" could be used to characterize an infant's state. However, a set of codes like these can be applied in more than one way. For some of these applications, methods for determining observer agreement are relatively straightforward, but for one relatively common and simple way of assigning codes, the methods are problematic. In the present paper, we consider this problem and offer a solution.

Before presenting the problematic application of determining observer agreement, it is useful to consider two unproblematic ways of assigning codes. One unproblematic application involves presenting coders with a transcript and asking them to assign codes to each unit of talk (or other unit) identified in the transcript. Another application involves presenting coders with a video recording of an infant and asking them to identify onset and offset times (e.g., to the nearest second) of the infant's different states, in which case we would think of the data as successive time units to which codes had been assigned (Bakeman & Quera, 1995). In such cases, the behavior that is to be observed is divided into units before coding, and coders need only assign codes to the specified units. For cases that fit these two applications, it has become conventional to report Cohen's kappa, an index that corrects for chance agreement and that assesses the level of agreement between observers about two mutually exclusive and exhaustive coding systems (Bakeman & Gottman, 1997; Cohen, 1960).

However, more simply, coders could be presented with a video recording and asked to identify events as they occur in sequence, without noting any information regarding time. Thus, coders simultaneously identify units of behavior (or *unitize*) and code. Cases that fit this application, which on its face seems quite simple, are in fact more complex than the first two with respect to determining observer agreement, because two observers may segment the stream of behavior differently, thus even the number of coded events might differ and their alignment in any case would be ambiguous. As Bakeman and Gottman (1997) wrote:

[Determining agreement about unitizing when] coding events, without any time information, is in some ways the most problematic [application] . . . If observers note only the sequence of events . . . then determining the agreement as to unit boundaries is more difficult. The two protocols would need to be aligned, which is relatively easy when agreement is high, and much more difficult when it is not, and which requires some judgment in any case. (p. 69)

Thus, determining agreement for what Bakeman and Quera (1995) call *event sequences* (a single stream of coded events without time information), which they characterize as the simplest of the sequential data types they define, is more difficult than determining agreement for seemingly more complex sequential data types.

In this article, we consider the issue of observer agreement for event sequences. First we describe methods for aligning such sequences, then we consider how their agreement might best be assessed, and finally we offer the results of simulation studies of the behavior of the agreement index we recommend.

Sequence Alignment

Event sequences are simply strings of codes. Two different strings of codes, each produced by an observer independently coding the same stream of behavior, may well differ in length because the observers unitized differently. The first question, therefore, is: How should these two sequences be aligned (i.e., which codes from Observer 1 should be matched with which codes from Observer 2) so that we can begin to assess the extent of agreement? What is desired is an algorithm that does not leave alignment to individual judgment, as Bakeman and Gottman (1997) suggested might be necessary. Such an algorithm, one that determines the optimal global alignment between two sequences by means of dynamic programming, is presented here.

The algorithm is adopted from sequence alignment and comparison techniques that are routinely used by molecular biologists to find similarities among DNA sequences in order to classify them and to search for patterns in the sequences themselves (Durbin, Eddy, Krogh, & Mitchison, 1998; Gusfield, 1997; Sankoff & Kruskal, 1999; Waterman, 1995). The techniques have also been used in speech processing, error detection and correction in computer science, and stratigraphic analysis (for a review, see Kruskal, 1999); for comparison of content in sociological research papers (Abbott & Barman, 1997); and for career

path similarity analysis (McVicar & Anyadike-Danes, 2000; Scherer, 2001). An alternative method for computing agreement between sequences was developed by Dijkstra & Taris (1995), although it does not guarantee optimal matching. Fichman (1999) presented a paper on the potential usefulness of a specific sequence alignment procedure for global matching of two event sequences corresponding to partners in an interaction. More recently, Fu (2001) developed ACT-PRO, a computer program that analyzes sequences of behavioral events using measures of sequence similarity based on alignment algorithms. To our knowledge, the use of these alignment techniques for the assessment of observer reliability has not been systematically explored to date, although Dijkstra (2007) included one common algorithm as a tool for comparing sequences in Sequence Viewer, a computer program for the analysis of sequences for sociological events.

A Sequence Alignment Algorithm

First, some notation and terminology: Let C be a coding scheme: $C = \{c_1, c_2, \dots, c_K\}$, where c_i is a code representing a discrete behavioral state, and K is the number of different, exhaustive, and mutually exclusive codes. For example, consider a coding scheme characterizing an infant's behavior in which letters are assigned to the different states: $C = \{A = \text{"quiet-alert"}; B = \text{"fussy or crying"}; C = \text{"rapid eye movement sleep"}; \text{and } D = \text{"deep sleep"}\}$. Assume that two observers independently code the following event sequences, $S_1 = \langle s_{11}, s_{12}, \dots, s_{1m} \rangle$ and $S_2 = \langle s_{21}, s_{22}, \dots, s_{2n} \rangle$, where m and n are the lengths of Sequence 1 and Sequence 2, respectively. For example, $S_1 = \langle \text{BBACDACDAB} \rangle$ and $S_2 = \langle \text{BACAACABABD} \rangle$. In general, $m \neq n$, but even if $m = n$, we cannot take for granted that s_{1i} is aligned or paired with s_{2i} for all i .

In order to align Sequence 1 with Sequence 2, some kind of correspondence between their respective codes must be established; that is, the sequences need to be globally aligned. Alignment proceeds step by step, by considering the transformations required to convert one sequence into the other. The more transformations required, the greater the distance between the two sequences. We begin with S_1 and apply step-by-step transformations that build a new sequence. When the transformation is complete, the new sequence is identical to S_2 , and a key question is how many steps were required to effect this transformation. At each step, there are four possibilities: (1) an agreement or identity transformation, in which a code from S_1 is paired with an identical code from S_2 and the common code is inserted into the new sequence; (2) a disagreement or substitution, in which a code from S_1 is paired with a different code from S_2 and the S_2 code is inserted into the new sequence; (3) a deletion, in which a code from S_1 is not paired with any code from S_2 and a hyphen (instead of the S_1 code) is inserted into the new sequence; and (4) an insertion, in which a code from S_2 is not paired with any code from S_1 and the S_2 code is inserted into the new sequence (with a hyphen inserted into the S_1 sequence). From the point of view of Observer 1, deletions and insertions are errors of omission and commission, respectively, on the part of Observer 2.

Given two event sequences, many different global alignments are usually possible. In fact, given two sequences with lengths m and n , the number of possible global alignments, if code substitutions but not insertions and deletions (indels) are permitted, is

$$\binom{m+n}{\min(m,n)}$$

(Ewens & Grant, 2001); this equation represents the lower bound for the total possible number of alignments that permit both indels and substitutions (see also Waterman, 1995). For example, if $m = n = 100$, that lower bound equals

$$\binom{200}{100} \cong 9 \cdot 10^{58},$$

hence looking for the best alignment by listing all possible alignments is essentially an impossible task. Our objective is to determine an optimal alignment—one that requires the fewest possible transformations and that results in the highest number of matching codes. Results depend on whether we permit or disallow substitutions, which is discussed below. In one example, disagreements are permitted; in the second, only errors of omission and commission are allowed. For example, for sequences S_1 and S_2 , two possible global alignments in which the first permits and the second disallows disagreements are

BBACDAC–DAB–	BBAC–DAC--DAB–
: :	
–BACAACABABD	–BACA–ACAB–ABD

These alignments were obtained by means of a dynamic programming algorithm, described subsequently, which provides optimal alignments. For both of these alignments, hyphens in the first sequence indicate that codes were inserted into the second one, whereas hyphens in the second sequence indicate that codes from the first sequence were deleted. Vertical lines indicate identity matches, and colons indicate substitution matches.

Longest Common Subsequence

When substitutions were allowed, five transformations were required, and the longest common subsequence of identical or similar codes was BAC[DA]AC[DB]AB (nine codes long), where [DA] indicates either D or A from the first and second sequences, respectively—likewise [DB]. When substitutions were disallowed, seven transformations were required, and the longest common subsequence of identical codes was BACACAB (seven codes long). When two sequences are optimally aligned, the longest common subsequence of agreement (LCSA) is obtained (see, e.g., Hirschberg, 1997; Needleman & Wunsch, 1970). If disagreements are permitted, then the LCSA consists of identical or similar codes; if they are not, then the LCSA consists of identical codes only. In both cases, the length of the LCSA is denoted Λ . Other, more complex sequence alignment methods exist (e.g., multiple global alignment of many sequences; see Gusfield, 1997; Lawrence et al., 1993), but because we are interested in finding the distance

between two sequences coded by independent observers, we will focus on algorithms and measures for pairwise, or simple global, alignment of two sequences.

Distance Between Sequences

The distance between two sequences, δ , can be defined as the number of code insertions, I , deletions, D , and substitutions, S , that is required to convert one sequence into the other; that is, $\delta = I + D + S$ (Gusfield, 1997; Kruskal, 1999; Levenshtein, 1965). The *Levenshtein distance* (also known as the *edit distance*) between two sequences is defined as the minimum number of code insertions, deletions, and substitutions necessary to convert one sequence into the other (but see Hirschberg, 1997); in other words, it is the minimum number of transformations required to convert one sequence into the other. The smaller the Levenshtein distance, the greater the LCSA length will be.

Costs and Generalized Levenshtein Distance

Every possible transformation can be assigned a particular cost or weight. For a coding scheme with K codes, a $(K+1) \times (K+1)$ cost matrix (or weight matrix, or scoring function; see, e.g., Durbin et al., 1998; Giegerich & Wheeler, 1996) is defined, where $\mathbf{W} = \{w_{ij}\}$ for $i = 0, 1, \dots, K$, and $j = 0, 1, \dots, K$; where $w_{ij} > 0$ ($i > 0, j > 0$) is the cost of substituting code j for code i ; where w_{i0} ($i > 0$) is the cost of deleting code i ; and where w_{0j} ($j > 0$) is the cost of inserting code j . Substitution costs along the upper left to lower right diagonal of \mathbf{W} are zero, by definition. Then the generalized Levenshtein distance (or GLD), or alphabet-weight edit distance (Gusfield, 1997), is the sum of the number of insertions, deletions, and substitutions, weighted by their corresponding costs:

$$\Delta = \sum_{s=1}^{s=I} w_{0j_s} + \sum_{t=1}^{t=D} w_{i_t 0} + \sum_{u=1}^{u=S} w_{i_u j_u},$$

where w_{0j_s} , $w_{i_t 0}$, and $w_{i_u j_u}$ are the costs of the s th insertion, the t th deletion, and the u th substitution in the sequences, respectively. If $w_I = w_D = w_S = 1$ (i.e., if all insertions, deletions, and substitutions are assigned identical costs), then $\Delta = I + D + S$. If $w_I = w_D = 1$ and $w_S = 2$, then a code insertion followed by a code deletion has the same cost as a code substitution. If $w_I = w_D = 1$ and $w_S > 2$, then substitutions will not occur; an insertion followed by a deletion would cost less than a substitution. The distance between sequences defines a true metric space of sequences; that is, the Levenshtein distance has the reflexive, symmetrical, and triangle inequality properties (Waterman, 1995). Costs can be chosen according to specific theoretical assumptions, or they can be estimated from the data. In any case, costs are usually symmetrical; that is, $w_{ij} = w_{ji}$. Costs can be estimated so that codes that occur rarely have higher indel costs than codes that occur often. As a general procedure, Mannila and Ronkainen (1997) propose assigning empirical indel costs that are inversely proportional to the code simple or unconditional frequencies in the sequences being compared. Alternatively, costs could be assigned in direct proportion to the codes' unconditional frequencies, because it could be argued that

	Delete	$c_1 = A$	$c_2 = B$	$c_3 = C$	$c_4 = D$
Insert	—	1	1	1	1
$c_1 = A$	1	—	1	1	1
$c_2 = B$	1	1	—	1	1
$c_3 = C$	1	1	1	—	1
$c_4 = D$	1	1	1	1	—

Figure 1. A weight or cost matrix that weighs all insertions, deletions, and substitutions equally.

mistakes such as inserting or deleting common codes, as opposed to rare codes, should have more weight on Δ , since detecting common events is easier than detecting rare ones.

Weight Matrices

The weight matrix permits us to assign a different weight to each insertion, deletion, and substitution, which provides far more flexibility than may ever be desired or desirable. In the context of DNA sequences, various schemes for proportional weights might make sense, but in the context of observer agreement, we think that almost always, absent a strong rationale, weights should be limited to simple integers. Here we consider three possibilities. All agreements on the diagonal are set to zero, by definition. In addition, (1) all other weights are set to one (see Figure 1), thus giving disagreements, omissions, and commissions equal weight; (2) omissions and commissions are set to one, but disagreements are set to two, thus counting a disagreement as the equivalent of two omission–commission errors; and (3) omissions and commissions are set to two, but disagreements are set to one, thus giving omission–commission errors twice the weight of disagreements. Of these three, we think the last option may best reflect what investigators expect of observer agreement.

In a similar fashion, although disallowing substitutions is discussed as an option in the literature we have cited, we would rarely recommend this option when observer agreement is considered. Almost always, observers are as likely, if not more likely, to disagree about how to categorize an event as to commit errors of omission or commission; it rarely makes sense, therefore, to disallow disagreements. Nonetheless, comparing results obtained when disagreements are permitted with results obtained when disagreements are disallowed, as we occasionally do in this article, can be instructive. Agreement is worse when disagreements are not allowed, and so this case represents a lower bound on agreement.

Needleman and Wunsch Algorithm

The algorithm that provides the optimal matching or alignment between two sequences was developed independently by several researchers from different fields during the 1970s (Kruskal, 1999), and has been reinvented since then (see, e.g., Mannila & Ronkainen, 1997). Molecular biologists call it the Needleman and Wunsch

(1970) algorithm (hereafter referred to as the NW algorithm); it belongs to a broad class of methods known as *dynamic programming*, in which the solution for a specific subproblem can be derived from the solution for the subproblem immediately preceding it. Applying dynamic programming to sequence alignment means that finding the optimal alignment between two sequences does not require checking all possible alignments, but rather only a very small portion of them. Dynamic programming proceeds step by step; at each step, three different possible subalignments, which would add to the subalignments accumulated from previous steps, are considered, and two are discarded (why three possible subalignments are chosen will be discussed shortly). Thus, at each step, two thirds of the possible subalignments are discarded. As a consequence, the method is exact (that is, it guarantees the optimal solution) without being exhaustive (that is, it does not explore all possible alignments; Galisson, 2000).

The goal of the algorithm is to determine an optimal alignment—that is, the steps required to transform one sequence into the other with the lowest generalized Levenshtein distance and the longest common subsequence. Thus the algorithm uses the cost matrix described earlier, with different costs resulting in different alignments. The algorithm uses three additional matrices, one to accumulate distances, one to accumulate lengths, and one for pointers, as described shortly. Each of these matrices has $m+1$ rows and $n+1$ columns, indexed $0, 1, \dots, m$, and $0, 1, \dots, n$, respectively, where m is the length of \mathbf{S}_1 and n the length of \mathbf{S}_2 . Row 0 indicates insertions; rows 1 to m are labeled with the codes in the \mathbf{S}_1 sequence; column 0 indicates deletions; and columns 1 to n are labeled with the codes in the \mathbf{S}_2 sequence. The labeling of the sequences is arbitrary; results are the same regardless of which sequence is labeled \mathbf{S}_1 .

The distance matrix (**D**) accumulates generalized Levenshtein distances; in particular, when complete, $D_{mn} = \Delta(\text{GLD})$ for \mathbf{S}_1 and \mathbf{S}_2 . The length matrix (**L**) accumulates common subsequence lengths; in particular, when complete, $L_{mn} = \Lambda$ (the length of the LCSA) for \mathbf{S}_1 and \mathbf{S}_2 . The pointer matrix (**P**), as described shortly, indicates which of three preceding cells [for cell (r, c) , preceding cells are cell $(r, c-1)$, cell $(r-1, c-1)$, and cell $(r-1, c)$, indicated with \leftarrow , \nwarrow , and \uparrow , respectively] contributes to the computation of L_{rc} and D_{rc} . Pointers are used to build the alignment by tracing them back from cell (m, n) .

The NW algorithm works as follows. First, insertion and deletion lengths are initialized to zero, thus $L_{00} = L_{r0} = L_{0c} = 0$. Second, insertion and deletion distances are initialized to their accumulative costs, thus $D_{00} = 0$, $D_{r0} = D_{r-1,0} + w(s_{1r}, 0)$, $D_{0c} = D_{0,c-1} + w(0, s_{2c})$, where $w(s_{1r}, 0)$ is the cost of deleting the element at position r of sequence \mathbf{S}_1 , and $w(0, s_{2c})$ is the cost of inserting the element at position c of sequence \mathbf{S}_2 (e.g., D_{04} is the accumulative cost of inserting elements s_{21}, s_{22}, s_{23} , and s_{24} ; similarly, D_{30} is the accumulative cost of deleting elements s_{11}, s_{12} , and s_{13}). Third, insertion and deletion pointers are initialized so that $P_{r0} = \uparrow$ and $P_{0c} = \leftarrow$. (Thus, for the expressions in the preceding sentences, $r = 1$ to m and $c = 1$ to n). After row 0 and column 0 are initialized, itera-

$$D_{rc} = \min \left[D_{r-1,c-1} + w(s_{1r}, s_{2c}), D_{r-1,c} + w(s_{1r}, 0), D_{r,c-1} + w(0, s_{2c}) \right]$$

tions of the NW algorithm then fill in the remaining cells, beginning with row 1 and considering columns 1 to n , then row 2, etc. (or vice versa, beginning with column 1 and considering rows 1 to m).

For each cell examined, the distance between the subsequences up to that point (i.e., $s_{11} \dots s_{1r}$ and $s_{21} \dots s_{2c}$) is computed as shown at the top of this page. In other words, at cell (r, c) , we select among three possible transformations in our ongoing effort to transform S_1 into S_2 . We can substitute code s_{2c} for s_{1r} or delete code s_{1r} or insert code s_{2c} . The transformation selected is the one that results in the lowest generalized Levenshtein distance for the subsequence up to this point. Note that there are two kinds of substitution. One occurs when $s_{2c} = s_{1r}$ (an agreement); the weight is zero and therefore they will always be matched. The other occurs when $s_{2c} \neq s_{1r}$ (a disagreement); these two codes are matched if substituting one for the other increases the generalized Levenshtein distance less than deleting the first code or inserting the second one. The three possibilities at cell (r, c) are illustrated in Figure 2.

L_{rc} and P_{rc} are updated accordingly. If substitution was chosen, then $P_{rc} = \searrow$ and $L_{rc} = L_{r-1,c-1} + 1$, because one more code was added to the common subsequence obtained at cell $(r-1, c-1)$. If insertion was selected, then $P_{rc} = \leftarrow$ and $L_{rc} = L_{r,c-1}$, because no code was added to the common subsequence obtained at cell $(r, c-1)$. If deletion was chosen, then $P_{rc} = \uparrow$ and $L_{rc} = L_{r-1,c}$, because no code was added to the common subsequence obtained at cell $(r-1, c)$.

When transformations have equal costs, one must be selected. The one selected can affect the specific alignment, but because the generalized Levenshtein distance and the length of the common subsequence are not affected, the choice is inconsequential in terms of optimal alignment. Still, to execute the algorithm, we need to break ties, for which purpose we prioritize transformations (see *SEQALN* software, Hardy & Waterman, 1997). Four possible priority orders are (1) substitution, deletion, insertion; (2) substitution, insertion, deletion; (3) deletion, substitution, insertion; and (4) insertion, substitution, deletion.

Order 3 provides the upper envelope and Order 4 the lower envelope of optimal alignments, in terms of the trace described below. Orders 1 and 2 favor substitution. Because we think that disagreement about an event is at least as common, if not more common, than errors of omission and commission, and because generalized Levenshtein distance and length of the common subsequence are not affected in any case, as a general rule we recommend Order 1; later we consider the effect of making other choices.

Application of the NW algorithm to sequences S_1 and S_2 , using Priority Order 1 and with all costs (i.e., insertion, deletion, and substitution) identical and equal to one, gives the results shown in Figure 3. This is the alignment

result (the one permitting disagreements) presented earlier for those sequences:

BBACDAC–DAB–
 | | | : | | : | |
 –BACAACABABD

We can now see how the results ($\Delta = 5$ and $\Lambda = 9$) were obtained. In general, application of the NW algorithm provides the generalized Levenshtein distance ($\Delta = D_{mn}$) and the length of the longest common subsequence ($\Lambda = L_{mn}$) for two sequences, whereas the \mathbf{P} matrix is used to generate the alignment, tracing the pointers back from cell (m, n) . As an example, Figure 4 shows how the alignment is generated from the pointers in Figure 3.

Measures of Agreement From Pairwise GSA

The NW algorithm yields two indices—the length of the longest common subsequence of agreement and the generalized Levenshtein distance (symbolized with Λ and Δ , respectively)—that can be used as measures of observer agreement for event sequences. The upper bounds for both indices vary; specifically, the upper bound for Λ is the minimum of m and n (that is, the length of the shorter sequence), whereas the upper bound for Δ depends on sequence lengths and the specific costs used as well. When assessing observer agreement, investigators are usually interested in knowing how a computed measure of agreement departs from chance agreement. However, the distributions of the measures of chance agreement provided by the NW algorithm are “far from being completely understood” (Waterman, 1995, p. 255), although expected values and bounds of Λ and Δ for random sequences of infinite or very long lengths have been estimated when code substitutions are disallowed (see, e.g., Baeza-Yates, Gavaldà, Navarro, & Scheihing, 1999; Boutet de Monvel, 1998; Chvátal & Sankoff, 1999; Dančík, 1994; Deken, 1979, 1999; Paterson & Dančík, 1994; Sankoff & Mainville, 1999). Although those results are valid only for very long sequence lengths m and n , event sequences recorded during observation sessions are

	$c-1$	$s_{2c} = c$	
$r-1$	$D_{r-1,c-1} + w(s_{1r}, s_{2c})$ \searrow	$D_{r-1,c} + w(s_{1r}, 0)$ \downarrow	
$s_{1r} = r$	$D_{r,c-1} + w(0, s_{2c})$ \rightarrow	D_{rc}	

Figure 2. The three possible values for distance D_{rc} between two subsequences. The upper left cell $(r-1, c-1)$ indicates substitution of code s_{2c} for s_{1r} ; the upper right cell $(r-1, c)$ indicates deletion of code s_{1r} ; and the lower left cell $(r, c-1)$ indicates insertion of code s_{2c} ; the transformation that results in the lowest value for D_{rc} is selected.

$S_1 \backslash S_2$	Delete	$s_{21} = B$	$s_{22} = A$	$s_{23} = C$	$s_{24} = A$	$s_{25} = A$	$s_{26} = C$	$s_{27} = A$	$s_{28} = B$	$s_{29} = A$	$s_{2,10} = B$	$s_{2,11} = D$
Insert	0 0 ←	1 0 ←	2 0 ←	3 0 ←	4 0 ←	5 0 ←	6 0 ←	7 0 ←	8 0 ←	9 0 ←	10 0 ←	11 0 ←
$s_{11} = B$	1 0 ↑	0 1 ↖	1 1 ←	2 1 ←	3 1 ←	4 1 ←	5 1 ←	6 1 ←	7 1 ↖	8 1 ←	9 1 ↖	10 1 ←
$s_{12} = B$	2 0 ↑	1 1 ↖	1 2 ↖	2 2 ↖	3 2 ↖	4 2 ↖	5 2 ↖	6 2 ↖	6 2 ↖	7 2 ←	8 2 ↖	9 2 ←
$s_{13} = A$	3 0 ↑	2 1 ↑	1 2 ↖	2 3 ↖	2 3 ↖	3 3 ↖	4 3 ←	5 3 ↖	6 3 ←	6 3 ↖	7 3 ←	8 3 ←
$s_{14} = C$	4 0 ↑	3 1 ↑	2 2 ↑	1 3 ↖	2 3 ←	3 4 ↖	3 4 ↖	4 4 ←	5 4 ←	6 4 ←	7 4 ↖	8 4 ↖
$s_{15} = D$	5 0 ↑	4 1 ↑	3 2 ↑	2 3 ↑	2 4 ↖	3 4 ↖	4 5 ↖	4 5 ↖	5 5 ↖	6 5 ↖	7 5 ↖	7 5 ↖
$s_{16} = A$	6 0 ↑	5 1 ↑	4 2 ↖	3 3 ↑	2 4 ↖	2 5 ↖	3 5 ←	4 6 ↖	5 6 ↖	5 6 ↖	6 6 ←	7 6 ←
$s_{17} = C$	7 0 ↑	6 1 ↑	5 2 ↑	4 3 ↖	3 4 ↑	3 5 ↖	2 6 ↖	3 6 ←	4 6 ←	5 6 ←	6 7 ↖	7 7 ↖
$s_{18} = D$	8 0 ↑	7 1 ↑	6 2 ↑	6 3 ↑	4 4 ↑	4 5 ↖	3 6 ↑	3 7 ↖	4 7 ↖	5 7 ↖	6 7 ↖	6 8 ↖
$s_{19} = A$	9 0 ↑	8 1 ↑	7 2 ↖	6 3 ↑	5 4 ↖	4 5 ↖	4 6 ↑	3 7 ↖	4 8 ↖	4 8 ↖	5 8 ←	6 8 ←
$s_{1,10} = B$	10 0 ↑	9 1 ↖	8 2 ↑	7 3 ↑	6 4 ↑	5 5 ↑	5 6 ↖	4 7 ↑	3 8 ↖	4 8 ←	4 9 ↖	5 9 ←

Figure 3. Dynamic programming table containing values of the distance, length, and pointer matrices after applying the NW algorithm to sequences BBACDACDAB and BACAACABABD. Rows and columns correspond to positions in Sequence 1 and Sequence 2, respectively; all costs were set to one. Each cell contains three entries: The first value is the generalized Levenshtein distance between the subsequences up to positions r and c in the initial sequences; the second value is the length of the subsequence that is common to both subsequences; and the third is a pointer. Pointers indicate which neighbor cell contributed to the computation of distance and length for the current cell. Gray cells with values in bold show the cells in the trace back from the bottom right cell to the top left one.

often not very long; common lengths are in the hundreds and few are in the thousands. For that reason, we usually cannot use expected values of chance agreement for infinite n as a reference.

Therefore, in order to assess observer agreement once sequences are aligned, the sequences' agreements and disagreements can be tallied in a $(K+1) \times (K+1)$ agreement matrix of the sort used to compute Cohen's kappa. The first row indicates insertions (i.e., events not coded by Observer 1 that were coded by Observer 2), and the first column indicates deletions (i.e., events coded by Observer 1 that were not coded by Observer 2), which accounts for the $K+1$ dimension. Figure 5 shows such a matrix for the alignment between S_1 and S_2 given previously. A kappa statistic can be defined in the usual way (Cohen, 1960) as the probability of observed agreement minus chance agreement divided by one minus chance agreement, but with one qualification: Because the cell in the upper left corner is a structural zero (once sequences are aligned, no event can be coded as missed by both observers), the agreements expected by chance need to be computed with an iterative proportional fitting algorithm and not the usual closed-form formula. Thus, for clarity, we term this statistic *alignment kappa* and not Cohen's kappa; for the data in Figure 5, the alignment kappa is .48.

When substitutions are not allowed, the matter is more complex—although as discussed earlier, we think this constraint would rarely be imposed. With no substitutions allowed, all cells indicating disagreement are structural zeros, and the usual computations for expected agreement do not apply. With this pattern of structural zeros, the constraints are such that for the model of quasi-independence, an iterative proportional fitting algorithm, as is often used for log-linear analysis (see, e.g., Bakeman & Robinson's ILOG program, 1994), computes expected frequencies

identical to those observed; thus, the value of kappa is zero. More general solutions, based on iterative procedures and permutations, for evaluating how a computed measure of agreement between sequences departs from chance agreement have been proposed (Altschul & Erickson, 1985; Booth, Maindonald, Wilson, & Gready, 2004).

Values of Alignment Kappa Under Various Simulated Conditions

In a previous paper (Bakeman, Quera, McArthur, & Robinson, 1997), we used numerical simulation to study values of observer agreement (e.g., Cohen's kappa) for

Cell	Pointer	S_1	S_2	Transformation
(10,11)	←	–	D	insertion
(10,10)	↖	B	B	–
(9,9)	↖	A	A	–
(8,8)	↖	D	B	substitution
(7,7)	←	–	A	insertion
(7,6)	↖	C	C	–
(6,5)	↖	A	A	–
(5,4)	↖	D	A	substitution
(4,3)	↖	C	C	–
(3,2)	↖	A	A	–
(2,1)	↖	B	B	–
(1,0)	↑	B	–	deletion
(0,0)	↖	–	–	–

Figure 4. Trace back for obtaining the alignment between sequences BBACDACDAB and BACAACABABD.

timed event sequences generated by observers of known reliability, as specified in a simulation program. Thus, simulation provides an opportunity denied us in actual situations; it lets us know (or specify) the theoretical reliability of observers before we attempt to measure their actual reliability. Likewise, regarding event sequences, it is especially interesting to know how the agreement measures provided by the NW algorithm are related to theoretical observer reliability because this allows us to judge which values of alignment kappa, computed once the observed event sequences are aligned, can reasonably be regarded as indicators of good reliability.

Simulation Parameters

For the present simulation, we define observer reliability as accuracy—that is, the observer's correct detection and coding of an event that has occurred (Bakeman et al., 1997; Gardner, 1995). Given a coding scheme with K exhaustive and mutually exclusive codes, a $(K+1) \times (K+1)$ matrix $\mathbf{R} = \{\rho_{ij}\}$ of conditional probabilities is defined, with both rows and columns numbered 0 to K . For $i > 0$, $j > 0$, ρ_{ij} is the probability that the observer records code c_j given that a behavior that should be coded c_i has occurred. The sum of the values in each row of matrix \mathbf{R} equals one; diagonal cells contain accuracies, and off-diagonal cells contain errors. Elements in column 0 are missed-event probabilities (see, e.g., Kaye, 1980) or omission errors; that is, ρ_{i0} is the probability that the observer failed to detect a behavior that should have been coded c_i . Elements in row 0 are false alarm probabilities (Kaye, 1980) or commission errors; that is, ρ_{0j} is the probability that the observer assigned a c_j code when no behavior actually occurred. For reliable observers, ρ_{ii} are close to one for all i ($\rho_{00} = 0$ by definition). An observer may be more reliable with respect to certain behaviors and less reliable with respect to certain others, in which case diagonal elements would not be identical. Also, an observer who fails to code a behavior correctly may favor some false codings more than others, in which case off-diagonal elements within a row would not be identical. Finally, when two fallible observers are compared, it can be assumed that both are equally reliable or not; in the former case, a single reliability matrix describes both observers, whereas in the latter, two different reliability matrices should be assumed.

Two more probabilities are required for the simulation: (1) the unconditional probability of a false alarm (α)—that is, the probability that the observer may code any behavior when no behavior has actually occurred (when a false alarm occurs, the observer records a code c_j with probability ρ_{0j}); and (2) the probability of missing an event ($\varepsilon_i = \rho_{i0}$)—that is, the probability that the observer may not code anything when a behavior has occurred that should have been coded c_i . When two fallible observers are compared, it can be assumed that both α and ε_i are identical or different for the two observers. If we assume, as we do here, that both observers have the same accuracy, ρ , that ρ is the same for all codes, and that all codes can be missed with identical probability, ε , then the cells in the reliability matrix, \mathbf{R} , are

Observer 1	Observer 2					
	–	A	B	C	D	Sum
–	0	1	0	0	1	2
A	0	3	0	0	0	3
B	1	0	2	0	0	3
C	0	0	0	2	0	2
D	0	1	1	0	0	2
Sum	1	5	3	2	1	12

Figure 5. Agreement matrix when substitutions are allowed (all indel and substitution costs equal 1), obtained from the alignment between sequences BBACDACDAB and BACAACABAD.

as follows. (1) Diagonal cells are set as $\rho_{ii} = \rho$. (2) Cells in column 0 (missed events) are set as $\rho_{i0} = \varepsilon = (1-\rho)\omega$, where $0 \leq \omega \leq 1$ is a parameter. Therefore, we define the probability of missing an event as a decreasing function of the accuracy that is associated with that event; for example, given $\omega = .20$, values for ε are .08, .06, .04, .02, and 0 when ρ equals .60, .70, .80, .90, and 1.00, respectively. (3) Cells in row 0 (false alarms) are set as $\rho_{0j} = p_j$ —that is, the code unconditional probabilities. (4) Remaining off-diagonal cells are set as $\rho_{ij} = (1-\rho)(1-\omega)/(K-1)$.

GSA Simulation Program

In order to understand how alignment kappa is related to observer reliability, we developed a computer program (GSA for Windows, available from the authors upon request) for carrying out several series of simulations guided by values for variables we have used before (Bakeman et al., 1997). Specifically, we considered values for observer accuracy (ρ) of .60, .70, .80, .90, and 1.00; sequence lengths (μ) of 25, 50, and 100; values for the number of codes (K) of 3, 5, 10, and 20; and equiprobable, moderately variable, and highly variable simple probabilities for those codes. These probabilities were computed according to this formula (see Bakeman et al., 1997, Table 1):

$$p_i(F, K) = \frac{1 + 2(F-1)(i-1)/(K-1)}{FK},$$

where F is a factor governing variability of code unconditional probabilities and $p_i(F, K)$ is the simple probability of code c_i (for all i , 1 through K) as a function of factor F and number of codes K . When $F = 1$, codes are equiprobable, and $p_i(1, K) = 1/K$ for all codes. The higher the F , the higher the variability. In the simulations, $F = 1$ was used for equiprobable, $F = 2$ for moderately variable, and $F = 4$ for highly variable probabilities. For example, when $K = 5$ and $F = 4$, probabilities are .050, .125, .200, .275, and .350. Rarely would probabilities be so neatly graduated in an actual investigation, but the differences between smallest and largest probabilities represented by our three cases should provide some guidance when investigators encounter a similar range of differences in their simple probabilities.

Each simulation run started with the generation of a random latent event sequence with specified length μ

for number of codes K with variability F ; this represents the true state of affairs. Then, two observed or manifest event sequences were generated according to specified parameters for ρ , α , and ω ; these represent the work of fallible coders. Given a latent sequence, in order to generate codes for one observed sequence, for every code c_i in the latent sequence, first an extra code or false alarm may be generated with probability α ; extra codes are sampled according to their simple probabilities p_j . Then, the latent code may generate either one observed code with probability $1 - (1 - \rho)\omega$, or it may be missed with probability $\varepsilon = (1 - \rho)\omega$, thus not generating a code at all; if an observed code must be generated, then it is sampled according to the simple code probabilities. It can be demonstrated that the expected length of a manifest sequence equals $\mu[1 + \alpha - (1 - \rho)\omega]$. In our simulations, we considered values of $\alpha = 0$ and $.2$ and $\omega = 0$ and $.2$. Therefore, the lengths of the manifest sequences ranged, on average, from 23 (when $\mu = 25$, $\rho = .6$, $\alpha = 0$, $\omega = .2$) to 120 (when $\mu = 100$, $\rho = 1$, $\alpha = .2$, $\omega = .2$). In each simulation, the expected length of both manifest sequences was identical, because the same reliability parameters were assumed for the two observers; however, when both false alarms and missed events were possible, the actual lengths of the two manifest sequences could differ, as often happens with observers. The pair of manifest sequences was then aligned with the NW algorithm, an agreement matrix was obtained, and alignment kappa was computed for it. Finally, the mean alignment kappa was calculated for all the pairs of sequences that had been generated under the same conditions.

In order to align the sequences, three different costs matrices and four priority orders were applied to the NW algorithm. Costs matrices were (1) both indel and substitution costs = 1; (2) indel costs = 1, substitution costs = 2; and (3) indel costs = 2, substitution costs = 1. Code substitutions were possible in all three cases. Priority orders were those described above; since the priority order affects the trace back but not the total number of agreements between the sequences, tallies in the off-diagonal cells of the agreement matrix may vary when the priority order is varied, and thus different kappas can be obtained.

In sum, the following parameters were varied systematically in the simulations according to a $5 \times 3 \times 4 \times 3 \times 2 \times 2 \times 3 \times 4$ factorial design. (1) Observer accuracy: $\rho = .60, .70, .80, .90, 1.00$. (2) Latent sequence length: $\mu = 25, 50, 100$. (3) Number of codes: $K = 3, 5, 10, 20$. (4) Code variability: $F = 1$ (equiprobable), 2 (moderate), 4 (high). (5) False alarm probability: $\alpha = 0, .20$. (6) Parameter for missing event probability: $\omega = 0, .20$. Actual missing event probabilities were thus $\varepsilon = 0$ (for any value of ρ , when $\omega = 0$) and $\varepsilon = .08, .06, .04, .02$, and 0 (for $\rho = .60, .70, .80, .90, 1.00$, when $\omega = .20$, respectively). (7) Cost matrices: (a) indels = substitutions = 1; (b) indels = 1, substitutions = 2; (c) indels = 2, substitutions = 1. (8) Priority orders (i.e., Orders 1–4): (a) substitutions, deletions, insertions; (b) substitutions, insertions, deletions; (c) deletions, substitutions, insertions; (d) insertions, substitutions, deletions.

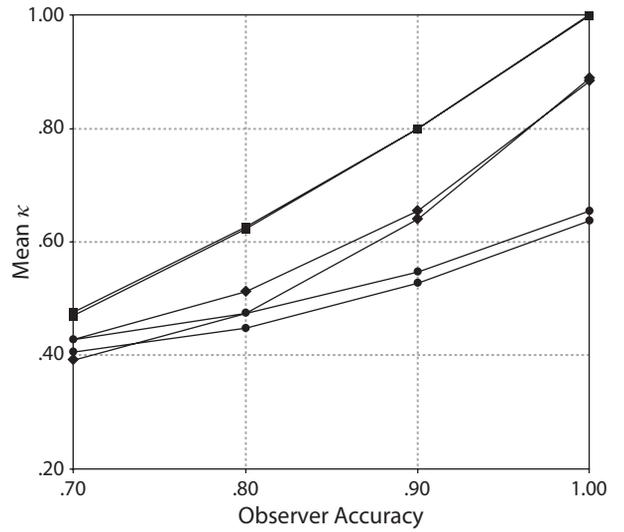


Figure 6. Mean values for kappa for varying observer accuracies, with identical indel and substitution costs = 1 and varying alignment priority orders. The top set of lines corresponds to a best case scenario ($\mu = 100, K = 20, F = 1, \alpha = \omega = 0$); the middle set corresponds to a worse case scenario ($\mu = 25, K = 3, F = 4, \alpha = \omega = 0$); and the bottom set corresponds to a worst case scenario ($\mu = 25, K = 3, F = 4, \alpha = \omega = .20$). Within each set, lines for Priority Orders 1 (lower line) and 3 (upper line) are shown.

For each combination of these parameters, 2,000 latent sequences were simulated, their corresponding manifest sequences were generated and aligned, and kappas were computed for the agreement tables obtained from the alignments. In total, 17,280,000 alignment kappas (8,640 design cells \times 2,000 pairs of manifest sequences) were calculated. Results were summarized by averaging the kappas within each design cell. Only the most relevant results will be detailed here; the complete table of mean kappas is available from the authors upon request.

Simulation Results

Figures 6, 7, 8, and 9 show mean values of kappa as a function of observer accuracy for selected combinations of parameters. Each point in the figures represents the mean value of kappa for 2,000 cases. Observer accuracies of .70, .80, .90, and 1 are displayed; for accuracies less than .70, essentially all mean values of kappa were below .50—values that are rarely regarded as acceptable.

Figures 6 and 7 portray three sets of two lines each. The top set we regard as a best case scenario: a longer sequence of many, equiprobable codes with no false alarms or missed events ($\mu = 100, K = 20, F = 1, \alpha = \omega = 0$). We regard the middle set as a worse case scenario: a shorter sequence of few codes varying in probability but with no false alarms or missed events ($\mu = 25, K = 3, F = 4, \alpha = \omega = 0$). The bottom set is a worst case scenario: like the middle set (worse case) but with the addition of false alarms and missed events ($\mu = 25, K = 3, F = 4, \alpha = \omega = .20$). We also considered a fourth case ($\mu = 100, K = 20, F = 1, \alpha = \omega = .20$); its results were slightly better than those of the worst case scenario, but

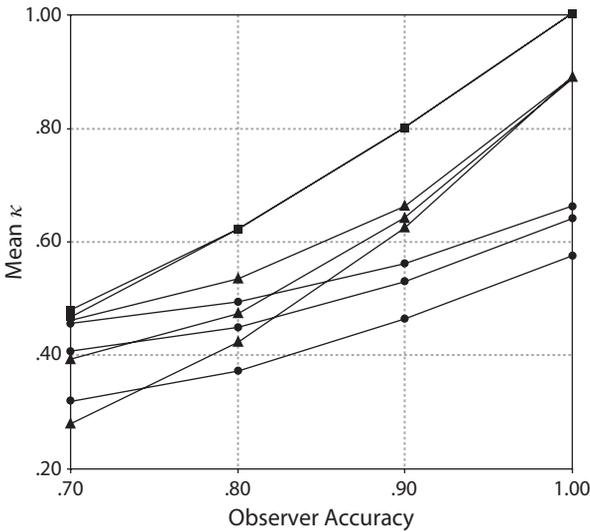


Figure 7. Mean values for kappa for varying observer accuracies with Alignment Priority Order 1 and varying sets of costs. The top set of (overlapping) lines corresponds to a best case scenario ($\mu = 100, K = 20, F = 1, \alpha = \omega = 0$); the middle set corresponds to a worse case scenario ($\mu = 25, K = 3, F = 4, \alpha = \omega = 0$); and the bottom set corresponds to a worst case scenario ($\mu = 25, K = 3, F = 4, \alpha = \omega = .20$). Within each set, lines for different costs are shown. Lower line, indel = 2 and substitutions = 1; middle line, indel = substitution = 1; upper line, indel = 1 and substitutions = 2.

sufficiently similar that we decided for simplicity not to consider the fourth case further.

The values graphed in Figure 6 indicate the effect of varying priority order (for these results, indel and substitution costs were identical; i.e., indel = substitution = 1). The lower line in each set represents Order 1 whereas the upper represents Order 3; Orders 2 and 4 gave essentially the same results as Orders 1 and 3, respectively, so they are not shown. The difference between Orders 1 and 3 increased somewhat as accuracy declined, but was never great. Consequently, we recommend using Order 1 when aligning sequences because it is slightly more conservative (i.e., it gives slightly lower values of alignment kappa). Moreover, Order 1, which specifies that substitutions are given priority over deletions and insertions, strikes us as more realistic than Order 3. Consequently, subsequent simulation results are given for Order 1 only. Otherwise, we conclude from Figure 6 that, except for the most extreme worst case scenarios, investigators can reasonably conclude that alignment kappas of .60 or better indicate observers who are at least 90% accurate.

The values graphed in Figure 7 indicate the effect of varying cost matrices. The lower line in each set favors substitutions (indel = 2 and substitutions = 1); the middle line favors neither (indel = substitution = 1, just as for the results in Figure 6); and the upper line favors insertions and deletions (indel = 1 and substitutions = 2). Costs have little effect on the best case scenario, but their effect increases for less-than-ideal scenarios, especially as observer accuracy declines. The lowest alignment

kappa values are obtained when substitutions are favored. Consequently, we recommend specifying indel = 2 and substitutions = 1 when aligning sequences, because this specification is slightly more conservative. Moreover, this specification, like Order 1, strikes us as more realistic than favoring insertions and deletions (disagreements are more likely than missed events and false alarms). Consequently, subsequent simulation results are given for indel = 2 and substitutions = 1 only. Otherwise, we conclude from Figure 7, as we did with Figure 6, that, except for the most extreme worst case scenarios, investigators can reasonably conclude that alignment kappas of .60 or better indicate observers who are at least 90% accurate.

The values graphed in Figure 8 provide a comparison of the effect of varying the number of codes and sequence lengths for equiprobable codes, keeping other factors constant (i.e., $F = 1, \alpha = \omega = 0$, substitutions favored over indels, and Priority Order 1). The top set of lines represents $K = 20$; the bottom, $K = 3$. The lower, middle, and upper lines in each set (separate lines are not always visible) represent $\mu = 25, 50$, and 100 , respectively. Alignment kappas were higher when more codes were defined (i.e., when $K = 20$), as expected, but sequence length had only a slight effect (when codes were few, alignment kappa became somewhat higher for longer sequences as observer accuracy declined). Under ideal circumstances (i.e., $K \geq 20$), the relationship between observer accuracy and mean kappa is almost linear, and accuracy can be estimated approximately as $\hat{\rho} \cong (3\kappa + 2)/5$, for $\kappa \geq .40$, Order 1, and substitutions favored over indels. We conclude that, all else being equal, alignment kappa is not much affected by sequence length; however, higher values can be expected when a greater versus a lesser number of codes is defined.

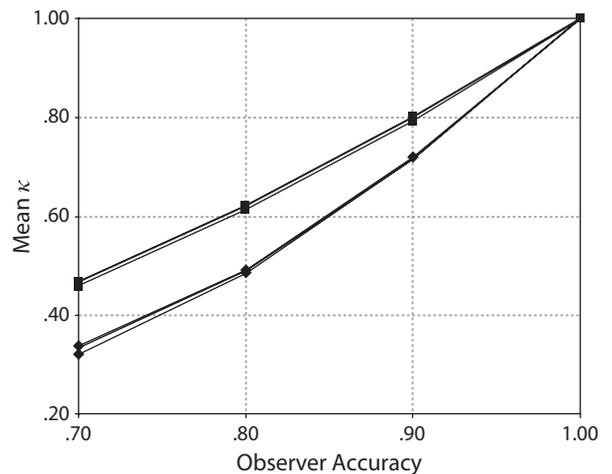


Figure 8. Mean values for kappa for varying observer accuracies and sequence lengths, with Alignment Priority Order 1, indel costs = 2 and substitution costs = 1, and equiprobable codes ($F = 1$); false alarms and missing events are not possible ($\alpha = \omega = 0$). The top set of lines corresponds to $K = 20$, and the bottom set corresponds to $K = 3$. Within each set, lines represent different latent sequence lengths; lower, middle, and upper lines = 25, 50, and 100, respectively, although separate lines are not always visible.

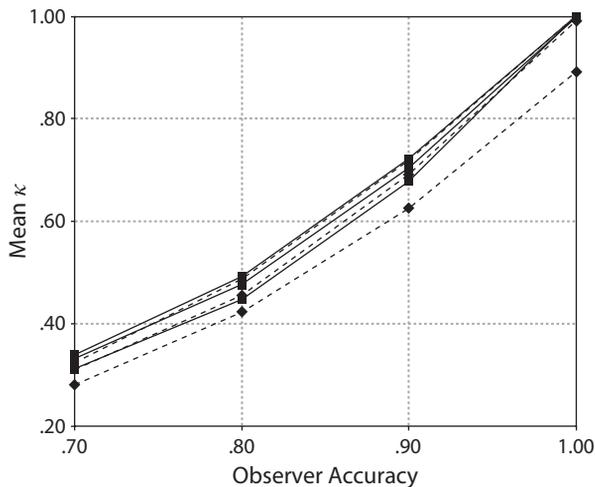


Figure 9. Mean values for kappa for varying observer accuracies and degrees of code variability, with $K = 3$ codes, Priority Order 1, and indel costs = 2 and substitution costs = 1; false alarms and missing events are not possible ($\alpha = \omega = 0$). Continuous lines represent $\mu = 100$, and dashed lines = 25. Lower, middle, and upper lines in each set represent $F = 4, 2,$ and $1,$ respectively.

The values graphed in Figure 9 provide a comparison of the effect of varying code variability and sequence length when few codes are defined, keeping other factors constant (i.e., $K = 3$, $\alpha = \omega = 0$, substitutions favored over indels, and Priority Order 1). The top set of lines represents $\mu = 100$; the bottom, $\mu = 25$ (although the sets overlap). The lower, middle, and upper lines in each set represent $F = 4, 2,$ and $1,$ respectively. Alignment kappas were a bit higher for longer sequences (i.e., when $\mu = 100$), as expected; likewise as expected, values of alignment kappa became somewhat higher as code variability decreased. We conclude that code variability has only a small effect on alignment kappa, other parameters being constant. As Figure 9 shows, when codes are highly variable, mean kappa is smaller than when the codes are equiprobable, but the effect of variability diminishes as sequence length increases. For long sequences, code variability has little effect on kappa, especially for high levels of observer accuracy; the effect presumably would be even smaller for values of K higher than the 3 assumed by Figure 9.

Summary and Recommendations

Sequences of discrete events, utterances, or some other unit that have been independently coded by two coders using a set of mutually exclusive and exhaustive codes can be aligned using a straightforward adaptation of Needleman and Wunsch's (1970) algorithm, an algorithm initially devised for aligning nucleotide sequences. Once aligned, an agreement kappa can be computed, which provides an estimate of observer agreement. We have termed this *alignment kappa*; it differs from the usual Cohen's (1960) kappa only in that the upper left cell of the agreement matrix is a structural zero, which means that expected frequencies cannot be computed with the

usual formula but instead require an iterative proportional fitting algorithm, as is commonly used in log-linear analyses (Bakeman & Robinson, 1994).

Using a computer program (GSA) that we devised for the purpose, we generated thousands of pairs of sequences; these were then aligned and their alignment kappa computed. We varied a number of parameters, including observer accuracy, latent sequence length, the number of codes defined, the variability of the probabilities that those codes were used, and the probabilities of false alarms and missed events (which, when not zero, result in pairs of sequences whose lengths differ). Other parameters that were varied were specific to the NW algorithm and included different cost matrices and various priority orders.

From the simulations, we concluded that under most reasonable circumstances, observer accuracies of 90% or better result in alignment kappas of .60 or better. All else being equal, kappa values were not strongly affected by sequence length, the number of codes, or the variability in their probability; however, they were adversely affected, as expected, by increasing probabilities of missed events and false alarms. We also concluded that, in the interest of being both conservative and realistic, cost matrices and priority orders should favor substitutions (i.e., disagreements) over insertions and deletions (i.e., missed events and false alarms), absent a strong rationale for doing otherwise.

We also developed a second user-oriented computer program, ELign, which aligns event sequences and computes alignment kappa. The user provides two files in Sequential Data Interchange Standard (SDIS; Bakeman & Quera, 1995) format; the first file contains one or more event sequences as coded by one observer, and the second contains the corresponding sequence or sequences as coded by a second observer. ELign allows the user to specify the cost matrices. Possibilities are: (1) indel costs = 2, substitution costs = 1; (2) indel costs = substitution costs = 1; (3) indel costs = 1, substitution costs = 2; (4) indel costs = 1, substitution costs = 3 (meaning no substitutions are permitted). Any of these default weights can be edited if the user has a reason for preferring other weights. As noted earlier, we recommend the first option, which is the program default. ELign also allows users to specify priority orders, although again as noted earlier, we recommend the program default Order 1, which favors substitutions.

ELign always displays the value of alignment kappa; however, the user can also request that all alignments, agreement matrices, and other matrices produced by the alignment algorithm be displayed. ELign is written in Pascal (Borland's Delphi); an executable version of the program can be downloaded at no cost from the authors' Web sites, www2.gsu.edu/~psyrab/BakemanPrograms.htm or www.ub.es/comporta/vquera/.

AUTHOR NOTE

Correspondence concerning this article should be addressed to V. Quera, Departamento de Metodología de las Ciencias del Comportamiento, Universidad de Barcelona, Campus Mundet, Paseo Valle de Hebrón, 171, E-08035 Barcelona, Spain (e-mail: vquera@ub.edu).

REFERENCES

- ABBOTT, A., & BARMAN, E. (1997). Sequence comparison via alignment and Gibbs sampling: A formal analysis of the emergence of the modern sociological article. *Sociological Methodology*, **27**, 47-87.
- ALTSCHUL, S. F., & ERICKSON, B. W. (1985). Significance of nucleotide sequence alignments: A method for random sequence permutation that preserves dinucleotide and codon usage. *Molecular Biology & Evolution*, **2**, 526-538.
- BAEZA-YATES, R. A., GAVALDÀ, R., NAVARRO, G., & SCHEIHING, R. (1999). Bounding the expected length of longest common subsequences and forests. *Theory of Computing Systems*, **32**, 435-452.
- BAKEMAN, R., & GOTTMAN, J. M. (1997). *Observing interaction: An introduction to sequential analysis* (2nd ed.). New York: Cambridge University Press.
- BAKEMAN, R., MCARTHUR, D., QUERA, V., & ROBINSON, B. F. (1997). Detecting sequential patterns and determining their reliability with fallible observers. *Psychological Methods*, **2**, 357-370.
- BAKEMAN, R., & QUERA, V. (1995). *Analyzing interaction: Sequential analysis with SDIS and GSEQ*. New York: Cambridge University Press.
- BAKEMAN, R., & ROBINSON, B. F. (1994). *Understanding log-linear analysis with ILOG: An interactive approach*. Hillsdale, NJ: Erlbaum.
- BOOTH, H. S., MAINDONALD, J. H., WILSON, S. R., & GREASY, J. E. (2004). An efficient Z-score algorithm for assessing sequence alignments. *Journal of Computational Biology*, **11**, 616-625.
- BOUTET DE MONVEL, J. (1999). Extensive simulations for longest common subsequences: Finite size scaling, a cavity solution, and configuration space properties. *European Physical Journal B*, **7**, 293-308.
- CHVÁTAL, V., & SANKOFF, D. (1999). An upper-bound technique for lengths of common subsequences. In D. Sankoff & J. B. Kruskal (Eds.), *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison* (2nd ed., pp. 353-357). Stanford, CA: CSLI Publications.
- COHEN, J. (1960). A coefficient of agreement for nominal scales. *Educational & Psychological Measurement*, **20**, 37-46.
- DANČÍK, V. (1994). Upper bounds for the expected length of longest common subsequences. *Bulletin of the European Association for Theoretical Computer Science*, **54**, 248.
- DEKEN, J. (1979). Some limit results for longest common subsequences. *Discrete Mathematics*, **26**, 17-31.
- DEKEN, J. (1999). Probabilistic behavior of longest-common-subsequence length. In D. Sankoff & J. B. Kruskal (Eds.), *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison* (2nd ed., pp. 359-362). Stanford, CA: CSLI Publications.
- DIJKSTRA, W. (2007). Sequence Viewer (Version 4.2a). [Computer software]. Retrieved from home.fsw.vu.nl/w.dijkstra/sequenceviewer.html.
- DIJKSTRA, W., & TARIS, T. (1995). Measuring the agreement between sequences. *Sociological Methods & Research*, **24**, 214-231.
- DURBIN, R., EDDY, S., KROGH, A., & MITCHISON, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University Press.
- EWENS, W. J., & GRANT, G. R. (2001). *Statistical methods in bioinformatics: An introduction*. New York: Springer.
- FICHMAN, M. (1999). *Finding patterns in sequences: Applying sequence comparison techniques to study behavior processes*. Unpublished manuscript, Carnegie Mellon University.
- FU, W.-T. (2001). ACT-PRO action protocol analyzer: A tool for analyzing discrete action protocols. *Behavior Research Methods, Instruments, & Computers*, **33**, 149-158.
- GALISSON, F. (2000, August). *Introduction to computational sequence analysis*. Tutorial presented at the 8th International Conference on Intelligent Systems for Molecular Biology, San Diego.
- GARDNER, W. (1995). On the reliability of sequential data: Measurement, meaning, and correction. In J. M. Gottman (Ed.), *The analysis of change* (pp. 339-359). Mahwah, NJ: Erlbaum.
- GIEGERICH, R., & WHEELER, D. (1996). *Pairwise sequence alignment*. VSNS BioComputing Division, Technische Fakultät, Universität Bielefeld: Available at www.techfak.uni-bielefeld.de/bcd/Curric/PrwAli/prwali.html.
- GUSFIELD, D. (1997). *Algorithms on strings, trees, and sequences: Computer science and computational biology*. New York: Cambridge University Press.
- HARDY, P., & WATERMAN, M. S. (1997). *The sequence alignment software library at USC*. Unpublished manuscript, University of Southern California.
- HIRSCHBERG, D. S. (1997). Serial computations of Levenshtein distances. In A. Apostolico & Z. Galil (Eds.), *Pattern matching algorithms* (pp. 123-141). New York: Oxford University Press.
- KAYE, K. (1980). Estimating false alarms and missed events from interobserver agreement: A rationale. *Psychological Bulletin*, **88**, 458-468.
- KRUSKAL, J. B. (1999). An overview of sequence comparison. In D. Sankoff & J. B. Kruskal (Eds.), *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison* (2nd ed., pp. 1-44). Stanford, CA: CSLI Publications.
- LAWRENCE, C. E., ALTSCHUL, S. F., BOGUSKI, M. S., LIU, J. S., NEUWALD, A. F., & WOOTTON, J. C. (1993). Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, **262**, 208-214.
- LEVENSHTEIN, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, **163**, 845-848.
- MANNILA, H., & RONKAINEN, P. (1997). Similarity of event sequences. In *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning: TIME'97* (pp. 136-139). Daytona Beach, FL.
- MCVICAR, D., & ANYADIKE-DANES, M. (2000). *Predicting successful and unsuccessful transitions from school to work using sequence methods*. Belfast, U.K.: Economic Research Institute of Northern Ireland.
- NEEDLEMAN, S. B., & WUNSCH, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**, 443-453.
- PATERSON, M., & DANČÍK, V. (1994). Longest common subsequences. In I. Prívára, B. Rován, & P. Ruzicka (Eds.), *Proceedings of 19th International Symposium on Mathematical Foundations of Computer Science* (pp. 127-142). Berlin: Springer.
- SANKOFF, D., & KRUSKAL, J. B. (Eds.) (1999). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison* (2nd ed.). Stanford, CA: CSLI Publications.
- SANKOFF, D., & MAINVILLE, S. (1999). Common subsequences and monotone subsequences. In D. Sankoff & J. B. Kruskal (Eds.), *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison* (2nd ed., pp. 363-365). Stanford, CA: CSLI Publications.
- SCHERER, S. (2001). Early career patterns: A comparison of Great Britain and West Germany. *European Sociological Review*, **17**, 119-144.
- WATERMAN, M. S. (1995). *Introduction to computational biology: Maps, sequences and genomes*. London: Chapman & Hall.

(Manuscript received September 3, 2005;
revision accepted for publication December 13, 2005.)