UvA-DARE (Digital Academic Repository)

Psychology of entertainment
van Driel, K.

Link to publication

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

Download date: 30 Nov 2018
In this chapter, an algorithm to measure agreement between sequences as proposed by Dijkstra and Taris is discussed. It is concluded that the 'optimal alignment' algorithm does not necessarily produce the optimal solution, that is, the minimal distance between two sequences.

In recent years, optimal alignment has gained increasing popularity among social scientists, mainly due to the work of Abbott (1990, 1995a), who first introduced the method to the social sciences. Optimal alignment refers to a family of algorithms for the calculation of the similarity between cases described by sequential data. This kind of measure is especially suitable for the classification of event history-data, such as job-careers or other sequential data like, for example, DNA that is built up as a sequence of nucleic-acids, placed in a certain order. If we want to know how much a bird and a monkey are genetically alike, we are not only interested in the similarity of the amount of particular nucleic acids in their DNA but especially in the similarity of the order in which the nucleic acids are placed. Unlike traditional methods, optimal alignment includes the order of codes in a sequence in the computation of the similarity, and not only the similarity of the codes as such. This
feature makes optimal alignment methods an attractive approach for the analysis of many kinds of longitudinal data.

The method of optimal alignment is based on the idea of changing one sequence into the other to establish similarity between the two. With the approach suggested by Abbott and Hrycak (1990) sequences may be modified with three kinds of edit-operations: deletion of codes, insertion of codes, and substitution of codes. With these edit-operations, both sequences are altered until they align, that is, consist of the same codes in the same order. The optimal solution is defined as the minimal number of alterations necessary. From this number, a measure of similarity may be obtained. However, determining the minimal number of alterations is rather complex, especially as the number of codes increases, because two sequences do not necessarily consist of the same number of codes (e.g., the DNA-string of a bird does not have the same length as a monkey’s). This problem is known in mathematics as the string-to-string correction problem (Wagner and Fischer, 1974), a combinatorial problem that can be solved with a mathematical technique known as dynamic programming (Kruskal, 1983). In the area of speech recognition these techniques are also often referred to as (dynamic) time-warping or elastic matching (Kruskal, 1983). Thus, the standard mathematical approach, like Abbott’s, produces optimal solutions.

Dijkstra and Taris (1995) proposed a new alignment algorithm based on a different set of edit-operations: deletion of codes and exchange of codes within a sequence. Instead of using a dynamic programming algorithm, they specify consecutive parts in the algorithm for deletion and exchange. The number of deletions and exchanges are multiplied by a proportionality constancy. In the deletion part of the algorithm, codes are deleted in two steps. In step 1, codes that occur only in one of both sequences are deleted. In step 2, codes that occur more often in one of the sequences than in the other, so-called superfluous codes, are deleted. In this way, two sequences are stripped of differing codes until they consist of the same number of similar codes but may still differ with respect to the order in which these codes occur. The next part of the algorithm, the exchange part, is then used to put the codes in the same order. For example, for the alignment of sequences L (long) = ABBC and S (short) = CBD, sequence L is stripped of code A during the first step, whereas
sequence $S$ is stripped of code $D$. In the second step, sequence $L$ is stripped of one superfluous code $B$, on which one exchange is needed to turn $BC$ into $CB$. This type of edit-operation, exchange of codes, gives the minimal number of exchanges of adjoining codes needed to turn one reduced sequence into the other reduced sequence. Finally, the number of deletions, and the number of exchanges are combined to overall measures of agreement, made proportional to different sequence features, as, for instance, the length of the sequences.

The appropriateness of some aspects of the DT algorithm has already been questioned by Abbott (1995b). In his comment on the algorithm, Abbott asked, for example, whether removal of superfluous codes is tenable from a substantial point of view. Hence, the approach taken by Dijkstra and Taris may not be applicable in many situations. We, however, argue that the presented algorithm is technically incorrect, so that even in situations in which removal of codes may be opportune, the algorithm produces incorrect answers. Specifically, Dijkstra and Taris (1995, p 226) claimed that their algorithms are exact, i.e., produce the optimal solution, that is, the minimal number of deletions and exchanges required to change one sequence in another. We will refute this claim by proving that DT is mathematically flawed because the deletion of superfluous codes is not optimal and, therefore, neither is the alignment.

The first step, removal of codes that occur in only one of the two sequences, is not relevant to the remainder of this article. The problem arises in the second step, the deletion of superfluous codes. This procedure is rather complex due to the requirement of optimality. This requirement demands sequences to be as similar as possible, in terms of exchanges, after deletion of the superfluous codes. The problem is not determining what the superfluous codes are but exactly which codes to remove. The description of this part of the algorithm presented below is from Dijkstra and Taris (1995; p226):

1. Determine the longer sequence (we call it $L$) and the shorter sequence (call it $S$).
2. Determine the superfluous codes of sequence L. For example, if $L = \text{"ABCCBCD"}$ and $S = \text{"ACBAD"}$, the [list of] superfluous codes of L are B, C, and C.

3. Set the starting point of sequence L to 0.

4. Determine the first code of sequence S.

5. Determine the first occurrence of this code, after the starting point in sequence L.

6. If before this code and after the starting point in sequence L, one or more superfluous codes occur, these codes are removed from sequence L and from the [list of] superfluous codes.

7. Set the starting point of sequence L to the code found in step 5.

8. Determine the next code from sequence S.

9. Repeat steps 5 to 8 until all codes of sequence S are scanned or no superfluous codes are left.

10. If there are superfluous codes left, scan sequence L on the occurrence of superfluous codes from the last code of sequence L backward. Remove these codes from sequence L (and from the [list of] superfluous codes).

11. If sequence L is shorter than sequence S, call the sequence that is now the shorter one sequence S and the other sequence L. Repeat steps 2 to 10.

The algorithm then calculates the number of exchanges needed to turn one of the reduced sequences into the other reduced sequence. Dijkstra and Taris claim that superfluous codes are deleted in such a way that both sequences become as similar as possible in terms of exchanges: “the code that results in a sequence that needs the least number of moves to transform it into the other sequence” (Dijkstra and Taris, 1995, p. 226).

Dijkstra and Taris (1995) illustrate this part of the algorithm with the following sequences as example:

$L = \text{ABCCBCD}$

$S = \text{ACBAD}$
Chapter 1: Nonoptimal Alignment

The superfluous codes of sequence L are one B and two Cs, whereas sequence S has one superfluous code, an A. By following the steps 1 to 11 described above, Dijkstra and Taris show that both sequences L and S are reduced to the following sequence: ACBD. This is in fact the optimal solution, as no further exchanges are necessary. However, the fact that the optimal solution is found in this example is not a sufficient condition for the claim of optimal alignment for all input. For other sequences, the optimal solution is not always found, as we will show with a counterexample. The sequences in our example (L = BCCBACD and S = ACBAD) are identical to the sequences in the Dijkstra-Taris example, with the exception of code A from the longer sequence, which is now moved from the first to the fifth position. Thus we maintain the exact same set of superfluous codes.

Let us apply the DT algorithm to these slightly modified sequences:

\[
\begin{align*}
L &= BCCBACD; \\
S &= ACBAD \\
2. \text{The superfluous codes of L are B, C, and C.} \\
3. \text{The starting point of sequence L is set to 0.} \\
4. \text{The first code of sequence S is an A.} \\
5. \text{The first occurrence of an A, after the starting point in sequence L, is on the fifth position.} \\
6. \text{Before the fifth position and after the starting point in sequence L, all superfluous codes occur. Codes B, C, and C are removed from the first, second, and third positions of sequence L and from the list of superfluous codes.} \\
7. \text{Since sequence L is now shorter than sequence S, the resulting sequences are renamed as follows:} \\
S_2 &= BACD \\
L_2 &= ACBAD
\end{align*}
\]

Steps 2 to 10 are repeated.

2. The superfluous code of \(L_2\) is an A.
3. The starting point of sequence L₂ is set to 0.

4. The first code of sequence S₂ is a B.

5. The first occurrence of a B, after the starting point in sequence L₂, is on the third position.

6. Before the third position, and after the starting point, in sequence L₂ the superfluous code occurs. Code A is removed from the first position of sequence L₂ and from the list of superfluous codes.

All superfluous codes have now been removed, with the following result:

L(reduced) = BACD
S(reduced) = CBAD

Now, two exchanges are needed to align these sequences. However, for optimal alignment a correct deletion algorithm should have identified the first, second, and sixth code of the original sequence L to be removed, instead of the first three codes. After the deletion of the redundant A from the original sequence S, both L(reduced) and S(reduced) would then become CBAD. In this case, additional exchanges are no longer needed. This proves that the DT algorithm does not produce the optimal solution.

The DT algorithm is implemented in the SEQUENCE program (Dijkstra, 1994) used for the analysis of sequential data. To illustrate the effect of this non-optimal solution we computed the overall agreement rₐ (Dijkstra and Taris, 1995) for the sequences in our example. Dijkstra and Taris define the value of the agreement as proportional to 1 minus, the number of exchanges divided by the number of codes of the reduced sequence incremented by 1. SEQUENCE, using two exchanges, gives an agreement of .3265, in which we, without exchanges, find an agreement of .4571. Similar differences were found for the other proposed measures, r₉ and r₇ (Dijkstra and Taris, 1995). The larger number of exchanges used by the DT algorithm results in a lower measure of agreement. In the example of Dijkstra and Taris the optimal solution is found accidentally, but in our example it is not. The similarity measures in the two examples can therefore not be compared. Thus, as it cannot be predicted,
when the DT algorithm produces optimal solutions and when not, the related agreement measures proposed by Dijkstra and Taris cannot be trusted.

In this article, it is shown that the claim that the DT algorithm produces the optimal solution does not hold. Where does DT go wrong? The DT approach applies two edit operations consecutively - first all the necessary deletions of codes and then all the necessary exchanges of codes. As the current example shows, the DT strategy excludes many other possible strategies, among which could be the optimal solution.\(^1\) When the alignment problem is simply recognized as a combinatorial problem, as it should be, it becomes clear that the deletion of elements cannot be performed independently of later exchanges, i.e., not consecutively as in the DT algorithm. However, an approach that would define the order of removal and replacements of codes would show more resemblance to the optimal alignment techniques questioned by Dijkstra and Taris (1995) than it would to the DT approach. In short, it seems unlikely that an optimal solution will be found for any input, by an algorithm based on the simple strategy of consecutive deletion and exchange operations.

As far as we know, there is no specific optimal alignment algorithm available that is based only on the deletion and exchange operations. An algorithm proposed by Lowrance and Wagner (1975) offers more possibilities to solve the problem and is provided with a solid mathematical foundation (Wagner, 1983). However, this algorithm is directed to even more complex combinatorial problems as it also allows for substitution and insertion operations as well as deletion and exchange.
1. An anonymous reviewer identified another problem with the DT-algorithm that does not arise in our example:

"Step 6 of DT is not well defined. Suppose one sequence has \( x \) copies of one code and the other has \( y \leq x \) copies. Then, there are \( x-y \) superfluous copies of that code in the first sequence. In step 6, the interval in \( L \) that is examined may contain more than \( x-y \) copies of that code, and the algorithm does not specify which \( x-y \) copies to remove. So step 6 is not well defined."

Consider sequence \( L \) (AADAAG) and sequence \( S \) (GAD). Sequence \( L \) has four copies of code A, whereas Sequence \( S \) has one copy of code A. The first occurrence of the first code of sequence \( S \) is on the sixth position of sequence \( L \). So three superfluous codes A will have to be removed from the first six positions of sequence \( L \). This interval contains all four codes A, of which three have to be removed. Because DT does not specify which three, possibly the first three are removed. This leads to a sub-optimal solution, since the remaining code A will have to be swapped with code D. This could have been avoided if the last three codes A had been removed in step 6.
Chapter 1: Nonoptimal Alignment

REFERENCES


