Discovering Similar Passages within Large Text Documents

Demetrios Glinos

[1] Computer Science, University of Central Florida, Orlando, Florida, United States
[2] Advanced Text Analytics, LLC, Orlando, Florida, United States

Abstract. We present a novel general method for discovering similar passages within large text documents based on adapting and extending the well-known Smith-Waterman dynamic programming local sequence alignment algorithm. We extend that algorithm for large document analysis by defining: (a) a recursive procedure for discovering multiple non-overlapping aligned passages within a given document pair; (b) a matrix splicing method for processing long texts; (c) a chaining method for combining sequence strands; and (d) an inexact similarity measure for determining token matches. We show that an implementation of this method is computationally efficient and produces very high precision with good recall for several types of order-based plagiarism and that it achieves higher overall performance than the best reported methods against the PAN 2013 text alignment test corpus.

Keywords: passage retrieval, text alignment, plagiarism detection.

1 Introduction

The task of text alignment is to identify passages in one text document that correspond to passages in another document according to some measure of similarity. Text alignment is used in plagiarism detection, document deduplication, and passage retrieval for textual entailment determination, among other uses. Thus, depending on the context, it may be important to recognize that the passage *"This article discusses the famous Hamlet monologue of the main themes of the game."* may be a paraphrase of the passage *"This essay discusses Hamlet's famous soliloquy in relation to the major themes of the play."*

It should not be surprising that finding such a pair of passages is not a trivial task in general, even for the relatively simple case of identifying passages that are identical in both documents. Suppose we are given two 5,000-word documents, both of which contain the second sentence above. Suppose further that we are asked to find the common sentence but we have no information at all about it. Thus, we do not know how many words the sentence may have, nor its punctuation, nor even the topic of the sentence. How are we to find it?

A brute-force search to locate identical passages would involve comparing every possible passage of every valid length in one document with all possible passages of equal length in the other document. Moreover, since the documents could be complete duplicates, all passage lengths up to and including the common document length must be included in the search. Such a search will have a computational complexity of *O*(n^3)*.* Thus, for the 16-token second sentence above, there are 4,985 possible shingles of 16 consecutive words in the first document that will need to be compared against the same number of shingles in the second document, resulting in a total of approximately 25 million comparisons. A similar number of calculations would be required for each of the approximately 5,000 other valid passage lengths which, at an average number of 2,500 shingles, will require a total of over 60 billion passage comparisons.

The problem is all the more difficult when the given documents may not contain any similar passages at all, or where corresponding passages do exist, they differ due to paraphrasing, reordering, additions and deletions of words or phrases, and the use of synonyms and alternative grammatical constructions. Considering only that the corresponding passages may be of different lengths, computational complexity jumps to *O*(n^4) for the brute-force search.

As a result, practical text alignment methods must employ different methods for exploring the search space.

Current implementations typically involve layers of heuristics in a *seeding-extending-filtering* approach [5]. At the first level, heuristics are used to identify the anchor points in each document for possible corresponding passages. A second set of heuristics is then used to extend and merge these anchor points to form passages. A final set of heuristics filters the resulting passages to remove overlapping alignments, short passages, and passages that do not meet certain other criteria.

Thus, Torrejón and Ramos [9] find anchor points using a comprehensive set of 3-grams obtained by various transformations (termed "Contextual N-grams", "Surrounding Context N-grams", and "Odd-Even N-grams") on three-word shingles from which short and stop words have been eliminated and the remaining words have been stemmed. Extension is performed using an algorithm that takes into account common n-grams that appear within threshold distances from each other in the two documents, the distance depending on document length and various tuning parameters. Final filtering is performed using a "Granularity Filter" that joins adjacent passages that appear in both documents.

In a similar manner, Suchomel et al. [8] use word 4-grams, but also supplement them with stop word 8-grams to find the anchor points. This presents an ordering problem at the extension step, since in general there is no natural ordering of the combination of the two types of n-grams, which the authors term "features". For example, a stop word 8-gram can span multiple sentences, and hence overlap several word 4-grams. The authors resolve this using an algorithm that merges features into non-overlapping intervals using their character offsets and then retains intervals containing at least four features [7].

A different approach is is used by Kong et al. [3], for whom anchor points are sentences that exceed a given cosine similarity threshold after elimination of whitespace, punctuation, stop words, case transformation and stemming. The set of candidate sentence pairs is further winnowed based on the relative numbers of similar words in each sentence. At the extension step, adjacent sentence pairs that are within a threshold distance are merged using a "Bilateral Alternating Sorting" algorithm [4].

A problem closely related to the text alignment problem discussed here is the *sequence alignment* problem in bioinformatics, which involves matching biological sequences such as amino-acid chains in proteins and nucleotide sequences in DNA strands. The sequences for comparison typically involve thousands of bases in the query sequences and potentially millions in the database strings to which they are compared. The problem has been well studied and current practice is dominated by heuristically-based methods, such as BLAST ("Basic Local Alignment Search Tool") [1].

Prior to such heuristic methods, however, the dominant algorithm for sequence alignment was the Smith-Waterman algorithm [6], particularly as it was improved for efficiency by Gotoh [2]. The Smith-Waterman algorithm is of interest to us since it is a dynamic programming method, and as such, it has the desirable feature that it is guaranteed to find a maximal length alignment. Moreover, algorithm time complexity is low-order polynomial, as it is roughly *O(nm)* for comparing a sequence of length n against one of length m.

Our approach takes advantage of the fact that while typical text documents for comparison may be long, they are considerably shorter than the biological sequences that prompted the migration to heuristic methods in that domain. As a result, even for long text documents, we feel that using Smith-Waterman is both optimal and tractable, provided it is adapted to the text analysis environment.

We have therefore adapted and extended the algorithm in a number of ways. In particular, we have extended the algorithm by defining a recursive procedure for discovering multiple non-overlapping passages for a given document pair. We have also defined a matrix splicing procedure for dividing the computational task so that the method will scale with document size without exceeding memory constraints. And we have extended the algorithm to merge adjacent sequence strands and to cover inexact matches.

We believe our extensions and our approach are novel and have not been reported elsewhere.

We have tested an implementation of this approach against the 2103 PAN Workshop series text alignment test corpus [5]. PAN[1] is an evaluation lab, now in its eleventh year, for uncovering plagiarism, authorship, and social software misuse. The 2013 text alignment test corpus is a comprehensive collection of document pairs exhibiting different types of plagiarism, including direct copies, random obfuscation, cyclic translations, and summarization. This corpus includes test cases that are suitable for our algorithm, as well as non-order preserving plagiarism cases. Document sizes within the collection vary widely, and many large documents are included. Test results for our implementation are very encouraging and show that the proposed method performs well under evaluation test conditions, and indeed overall performance tops the best reported results from the 2013 evaluation despite addressing only a subset of the problem types.

[1] `http://pan.webis.de`

Although we have tested the method in a plagiarism detection context, the method is sufficiently general that it can be applied in other contexts readily, as it does not involve any plagiarism-specific tuning parameters.

In the sections that follow, we present the details of the method and the experimental results. Section 2 explains how the basic alignment algorithm works, with baseline extensions. Section 3 describes the recursive method for discovering multiple passages. Section 4 describes the matrix splicing procedure. Section 5 describes the corpus, measures, and results of the evaluation experiments. And finally, Section 6 presents our conclusions.

2 Basic Alignment Algorithm

Given two text documents A and B, we first read the documents as UTF-8 bytes and tokenize them into words, retaining all punctuation as separate tokens, except for " 's", which is retained as a single token. We also convert all non-printing ASCII characters to spaces, convert all newlines, tabs, and returns to spaces, and finally reduce all remaining tokens to lower case. The result is two sequences of tokens of generally different lengths, $A = a_0, a_1, ..., a_m$ and $B = b_0, b_1, ..., b_n$.

We then apply the basic Smith-Waterman algorithm [8], as simplified by Gotoh [9], to build up recursively a match matrix M, as follows[2]:

$$M(i,j) = max \begin{cases} M(i-1,j-1) + match(a_i,b_j) \\ M(i-1,j) + gap \\ M(i,j-1) + gap \\ 0 \end{cases} \tag{1}$$

where $match(a_i, b_j) = +2, \text{if } a_i = b_j; \text{and } -1$ otherwise; and where $gap = -1$ is the gap penalty.

The algorithm produces a matrix of non-negative integer values from which the maximal alignment is obtained using a straightforward traceback procedure that starts from the largest value in the matrix and simply reverses the matrix generation algorithm above to recover the path that produced the maximal value. If the alignment sequence that is produced contains at least 40 tokens of each input sequence, it is retained as an alignment detection.

We adapt the algorithm for text alignment by extending it in two ways. First, we provide a mechanism for joining directly adjacent subsequences. We do this by defining a parameter *chain*, which is configured to some small number. This parameter represents the number of "jumps" that are permitted in the traceback process, where a jump represents moving from a zero-value cell to an adjacent nonzero-value cell according to the traceback algorithm. This feature permits merging sequence strands in both documents that are separated by only one token in each document string. We use a chain value of 2 for our implementation,

[2] The matrix is of dimension *(m+1) by (n+1)* and is initialized with all zeroes in the first row and first column. The matrix is built row by row, proceeding left to right.

which we arrived at informally during development against the training corpus, thus permitting two such jumps in building up our alignment sequences.

We also extend the basic algorithm by relaxing the equality requirement for a match and using a similarity determination instead. While the similarity function can include synonymy, we have elected for this study to use a simpler implementation in which we equate the determiners *the, a,* and *an*, and where we also equate these twenty-five commonly occurring prepositions: *of, in, to, for, with, on, at, from, by, about, as, into, like, through, after, over, between, out, against, during, without, before, under, around,* and *among.*

Table 1 shows the matrix elements that are generated using our modified algorithm for the two sample sentences in the introductory section. Except for the first row and column, which are all initialized to zeroes to prime the recursion, the rows and columns in the table correspond to the tokens in the two documents. The shaded cells in the table show the traceback path that is produced by backtracking from the maximal element at the lower right. No jumps were involved in traceback in this example. The table also includes additional elements at the end of each token sequence to show how the operation of the algorithm exhibits a tapering effect once past the maximal element.

Table 1. Match matrix elements for two sample sentences showing traceback defining maximal-length alignment and tapering effect past the maximal element

		This	essay	discusses	Hamlet	's	famous	soliloquy	in	relation	to	the	major	themes	of	the	play	.	tempus	fugit
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
This	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
article	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
discusses	0	0	0	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
the	0	0	0	2	2	1	0	0	0	0	0	2	1	0	0	2	1	0	0	0
famous	0	0	0	1	1	1	3	2	1	0	0	1	1	0	0	1	1	0	0	0
Hamlet	0	0	0	0	3	2	2	2	1	0	0	0	0	0	0	0	0	0	0	0
monologue	0	0	0	0	2	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0
of	0	0	0	0	1	1	1	0	3	2	2	1	0	0	2	1	0	0	0	0
the	0	0	0	0	0	0	0	0	2	2	1	4	3	2	1	4	3	2	1	0
main	0	0	0	0	0	0	0	0	1	1	1	3	3	2	1	3	3	2	2	0
themes	0	0	0	0	0	0	0	0	0	0	0	2	2	5	4	3	2	2	1	0
of	0	0	0	0	0	0	0	0	2	1	2	1	1	4	7	6	5	4	3	2
the	0	0	0	0	0	0	0	0	1	1	1	4	3	3	6	9	8	7	6	5
game	0	0	0	0	0	0	0	0	0	0	0	3	3	2	5	8	8	7	6	5
.	0	0	0	0	0	0	0	0	0	0	0	2	2	2	4	7	7	10	9	8
carpe	0	0	0	0	0	0	0	0	0	0	0	1	1	1	3	6	6	9	9	8
diem	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	5	5	8	8	8

3 Discovering Multiple Passages

It is quite possible that two text documents may contain more than one set of corresponding passages. Moreover, it is not generally possible to know *a priori* how many there may be, if indeed there are any at all. Further, corresponding passages may not possess the same relative ordering in one document that they possess in the other.

We therefore define a recursive procedure for finding multiple aligned passages. This procedure makes use of two auxiliary linear arrays corresponding to the tokens for each document. The arrays are initialized to indicate that at the start, all tokens in each document are available for inclusion in the next found alignment sequence. However, as each sequence is found, the array elements for the tokens that have been used are marked to block off those tokens from further consideration. This ensures that there can be no overlapping alignments.

The recursive procedure is illustrated in Figure 1. The matrix sketch on the left depicts a first maximal alignment that has been found in matrix region A. By construction, the alignment includes all tokens in the first sequence that run from the top row to the bottom row of region A. Similarly, the alignment also includes all tokens from the second sequence that run from the leftmost column to the rightmost column of region A. Because the tokens used for this alignment may not be used again, the sketch on the left shows in white the remaining regions of the matrix that may contain additional alignments. These are searched in recursive fashion.

Similarly, the sketch on the right shows the situation after a second maximal alignment is found in region B. This results in blocking off from further consideration additional tokens, as shown by the additional shaded regions.

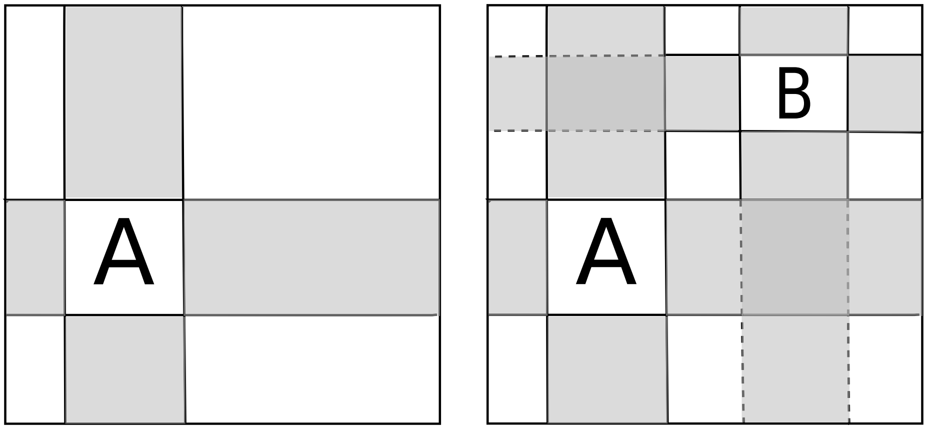

Fig. 1. Sketch of match matrix regions showing in white the remaining regions that may contain additional aligned passages after an alignment is first found in region A and subsequently an alignment is found in region B

As may be observed from Figure 1, it is the case in general that whenever an alignment is found within a region of the matrix, four subregions remain where additional alignments may potentially be found. These are searched in recursive fashion, adding to the auxiliary linear arrays each time a fresh alignment is found. The recursion ends when all regions have been searched.

4 Handling Long Documents

Because our method involves generating and processing a match matrix of a size equal to the product of the numbers of tokens in each of the two source documents, we recognize that there may be some text document pairs whose matrix requirements may exceed the available memory for storing such a data structure. Therefore, to ensure that our method scales with document size, we have devised a procedure for processing very large matrices, and by extension, very large documents, which we term *matrix splicing.*

Matrix splicing uses a system configuration parameter that represents the largest matrix size in terms of the total number of matrix elements that will be processed at one time. This value is system dependent and represents the memory available on the host system for the match matrix. It does not reflect any characteristic of the text documents themselves or the text analysis domain. We used a value of 50 million during initial development, which was adequate for our system, but we subsequently decreased it to 25 million to ensure that the matrix slicing component was exercised adequately. And since the system performed well on all test data with that value, we retained that value in the tests described in the next section.

It should be kept in mind that the value of 25 million represents token pairs, so that a matrix of this size will handle two 5,000-word documents, which correspond roughly to two 20-page papers, without invoking the splicing method. Nevertheless, as will be discussed in the next section, many documents much larger than these were encountered during testing, confirming the need for the method.

Matrix splicing proceeds as follows. First, the matrix needs for the document pair are computed and compared to the limiting value. If the needs do not exceed the value, all alignment processing is performed using a single matrix, which is examined recursively as described in the previous section. If the matrix needs exceed the limit, a first match matrix is constructed using all of the rows of the first document and as many of the columns of the second document as ensures that their product is within the system limit, after first reserving space in the matrix for a *carryover vector* of length equal to the number of rows in the matrix.

The carryover vector is initialized to all zeroes and represents, initially, the zeroeth column of the matrix, but it contains fields for additional information that may be needed later to stitch together two sequence segments that cross the boundary between two adjacent match matrices for the document pair.

Given an initialized carrover vector, the first match matrix is processed recursively as described in the previous section to identify valid text alignments within its boundaries. All detected alignments that do not begin at the rightmost boundary of the matrix are treated as strictly internal and are saved to be reported as a group with all other alignments for the document pair. However, all sequences, however short, that begin on the rightmost boundary of the current matrix are preserved in the carryover vector since they may be the beginnings of alignments that are completed in the next match matrix for the pair. It should be noted that only the matrix value and the row and column indices of the head of each such subsequence are needed to fully specify such a sequence, as they define the upper left corner of the bounding rectangle and the value that is used for applying the modified Smith-Waterman algorithm in the next match matrix.

Thus, after the first match matrix is processed, any fully internal alignments will be saved to be reported later, and any sequences that begin on the right boundary will be preserved in the corresponding entries of the carryover vector, which will also have zero values for all other entries. The next match matrix is then generated. However, this time, instead of all zeroes in the leftmost column, as in the simplest case, any nonzero values specified by the carryover vector will be used for values in the leftmost column in the matrix algorithm. During traceback, any alignment sequences that reach the leftmost column of the match matrix are then augmented by the information stored in the carryover vector at the boundary location. In this manner, alignment sequences are spliced together to form the combined sequences that would have been produced had the two documents been processed using a single match matrix. This process then repeats for however many match matrices as are needed to process the matrix needs for the two documents.

5 Experiments and Results

5.1 Test Corpus

To evaluate the performance of the method and algorithms described in the preceding sections, we selected the PAN Workshop 2013 test corpus. As described in [5], the corpus consists of document pairs on 145 topics that were processed both automatically and manually to produce document pairs that involve the following types of plagiarism: (i) no plagiarism; (ii) no obfuscation; (iii) random obfuscation; (iv) cyclic translation obfuscation; and (v) summary obfuscation. The corpus includes both order-preserving plagiarism problems suitable for testing our algorithm, as well as plagiarism problems (such as summarization) that our algorithm does not address.

Full details of the construction of the corpus are contained in [5]. Briefly, however, the document categories are described as follows: The no plagiarism category is self-explanatory. The no-obfuscation category represents cut-and-paste copying, although some differences in whitespace and line breaks are introduced. The random obfuscation category includes some amount of random text operations, such as word shuffling, adding or deleting words or phrases, and word replacement using synonyms. Cyclic translation involves translations of a document using automated translation services into two successive languages other than English, and then back into English. Finally, the summary category includes documents obtained from the Document Understanding Conference (DUC) 2006 corpus that have been processed to introduce noisy areas in addition to the summaries in the test documents.

The actual 2013 PAN text alignment test corpus comprises 5,185 document pairs from 3,169 source documents and 1,826 documents that contain suspected plagiarism. The test corpus includes 1,185 document pairs from the summary category and 1,000 from each of the other categories. The basic characteristics of the corpus are summarized in Table 2. The word counts in the table were obtained in each case by dividing the number of characters observed by five, which is the average word length for English.

Table 2. PAN 2013 Text Analysis Test Corpus basic characteristics

PAN 2013 Test Corpus	Chars	Words
Suspect Documents		
Min length	657	131
Max length	101,484	20,297
Mean length	14,650	2,930
Source Documents		
Min length	520	104
Max length	61,385	12,277
Mean length	4,570	914

5.2 Performance Measures

Full details for evaluating plagiarism detection performance are contained in [1]. We summarize them here.

The traditional measures of precision and recall for information retrieval systems are retained, but are adapted to the plagiarism detection context so that both measures reflect how well a detected alignment corresponds to the gold standard character offsets and lengths for both source and suspected documents.

The measures involve plagiarism *cases* and *detections*, which are defined as follows. A plagiarism *case* is a 4-tuple $s = \langle\, r_{off},\, r_{len},\, s_{off},\, s_{len} \,\rangle$ that represents the gold standard offsets and lengths for the corresponding passages in the suspect document ("r") and the source document ("s"), respectively. Similarly, a *detection* is represented by the 4-tuple $r = \langle\, r'_{off},\, r'_{len},\, s'_{off},\, s'_{len} \,\rangle$. All offsets and lengths are represented in characters.

Now, if both the suspect and source intervals of the detection have nonempty intersections with the corresponding intervals of the gold standard case, then we say that the case has been detected by the reported detection.

If we further define S to be the union of all gold standard cases and R to be the union of all reported detections, then we can define precision and recall as follows:

$$\text{precision(S,R)} = \frac{1}{|R|}\sum \frac{\bigcup(\text{s} \sqcap \text{r})}{|r|} \text{ and recall(S,R)} = \frac{1}{|S|}\sum \frac{\bigcup(\text{s} \sqcap \text{r})}{|s|} \tag{2}$$

$$\text{where s} \sqcap r = \begin{cases} s \cap r, \text{ if } r \text{ detects } s \\ \emptyset, \text{otherwise} \end{cases} \tag{3}$$

An additional measure called *granularity* is also used to reflect that it is undesirable for a detector to report multiple detections where there should be only one, thus:

$$granularity(S, R) = \frac{1}{|S_R|}\sum |R_S| \tag{4}$$

where $S_R \subseteq S$ are the subset of cases that are detected and R_S are the subset of reported detections that detect cases.

Finally, all of the above measures are combined in a composite measure called the *plagiarism detection score*, as follows:

$$plagdet(S, R) = \frac{F_1}{log_2(1 + granularity(S, R))} \tag{5}$$

where F_1 is the balanced harmonic mean of precision and recall.

5.3 Experimental Results

Our system implementation was run first against the the entire PAN 2013 text alignment test corpus in a test in which the source and suspect document plagiarism database categories were not known. Subsequent runs were then made against the document pairs for each of the individual categories. Table 3 below summarizes aggregate performance for the test runs. Table 4 presents the corresponding raw detection and plagiarism case information.

We examine first the runtime performance of the system. Overall, the system processed the 5,185 document pairs in approximately 23.5 minutes, corresponding to an average of 0.2590 seconds per document pair. This is quite respectable performance for a natural language processing system, although not real time. Our code was written in Java and was executed in a Linux virtual machine environment on a dual-core 2.66 GHz machine with 4 GB RAM. We anticipate that improved runtime performance can be obtained from migrating to a compiled language and better hardware. Runtime performance against the individual category types varied, with approximately equal time consumed for the three categories for which substantial numbers of alignments were detected.

Next, we turn to the functional performance of the system. Overall, the 0.8404 composite plagiarism detection score achieved by the system was 1.8% higher than the best reported result against the test corpus [5] despite poor performance against the summary obfuscation category. Nevertheless, overall precision was 0.9690, which exceeded the best reported by 7.4%. Overall recall, however, was dragged down by performance against the summary category and falls within the middle of the pack for reported results (see [5]).

Table 3. Aggregate performance against PAN 2013 test corpus

Target Corpus	PlagDet	Recall	Precision	Granularity	Runtime
No plagiarism	undefined	undefined	0.0000	undefined	1:41.2013
No obfuscation	0.9624	0.9603	0.9644	1.0000	6:47.717
Random obfuscation	0.7958	0.7073	0.9732	1.0413	5:09.562
Cyclic obfuscation	0.8441	0.7506	0.9730	1.0056	6:55.170
Summary obfuscation	0.0984	0.0560	0.9794	1.1099	2:52.770
Overall	0.8404	0.7588	0.9690	1.0177	22:23.481

Table 4. Case and detection counts for test runs against PAN 2013 test corpus

Target Corpus	Document Pairs	Reports	Detections	Cases	Cases Detected
No plagiarism	1,000	5	0	0	0
No obfuscation	1,000	1,160	1,159	1,206	1,159
Random	1,000	1,114	1,109	1,292	1,065
Cyclic	1,000	1,093	1,084	1,308	1,078
Summary	1,185	103	101	236	91
Overall	5,185	3,475	3,453	4,042	3,393

The anomalous performance against pairs in the summary obfuscation category is explained easily given the nature of our method and the type of obfuscation involved. Our method is for the detection of aligned texts. The key feature for any alignment is, by definition, that order is preserved. Summarization, by contrast, has an entirely different character. It is the nature of summarization that terms and concepts are taken from various locations within the source document, and there is no requirement that such terms and concepts appear in the summary in any particular order. Accordingly, summarization is inherently non-order preserving, and as such, an alignment method such as ours will generally fail to find an alignment. Nevertheless, we observe that the system did in fact detect 91 out of 236 summarization cases, and a manual examination of a number of these detections revealed source passages that appeared to us to indicate that perhaps some summaries were prepared by essentially paraphrasing coherent sections of the source text. While true summarization in context may be considered plagiarism, it does not present in our view a text alignment problem. Some of the random obfuscation and cyclic translation cases also exhibited these characteristics, although we did not have the opportunity to quantify the extent in this initial effort.

Overall, the test results show that our method performs best against the no-obfuscation category, where order is necessarily preserved, and that performance remains high and degrades only somewhat for the cyclic and random categories, which involve some measure of reordering of terms and concepts.

6 Conclusions

We have presented a general method for detecting text alignments across documents and have shown that it possesses both adequate runtime performance and is robust against alignment problems of varying difficulty. The method is capable of finding multiple algnments within a given document pair. The method is also scalable to handle very long documents. Based on our investigations, we believe that the method can serve as a component of a full plagiarism detection system, and can also be applied in a variety of other document processing contexts.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipmanl, D.J.: Basic local alignment search tool. Journal of Molecular Biology 215(2), 403–410 (1990)
2. Gotoh, O.: An Improved Algorithm for Matching Biological Sequences. Journal of Molecular Biology 162, 705–708 (1981)
3. Kong, L., Qi, H., Wang, S., Du, C., Wang, S., Han, Y.: Approaches for Candidate Document Retrieval and Detailed Comparison of Plagiarism Detection. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) CLEF (Online Working Notes/Labs/Workshop) (2012)
4. Kong, L., Qu, H., Du, C., Wang, M., Han, Z.: Approaches for Source Retrieval and Text Alignment of Plagiarism Detection–Notebook for PAN at CLEF 2013. In: Forner, P., Navigli, R., Tufis, D. (eds.) Working Notes Papers of the CLEF 2013 Evaluation Labs (September 2013)
5. Potthast, M., Gollub, T., Hagen, M., Tippmann, M., Kiesel, J., Rosso, P., Stamatatos, E., Stein, B.: Overview of the 5th International Competition on Plagiarism Detection. In: Forner, P., Navigli, R., Tufis, D. (eds.) Working Notes Papers of the CLEF 2013 Evaluation Labs (September 2013)
6. Smith, T., Waterman, M.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
7. Suchomel, S., Kasprzak, J., Brandejs, M.: Three Way Search Engine Queries with Multi-feature Document Comparison for Plagiarism Detection. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) CLEF (Online Working Notes/Labs/Workshop) (2012)
8. Suchomel, Š., Kasprzak, J., Brandejs, M.: Diverse Queries and Feature Type Selection for Plagiarism Discovery–Notebook for PAN at CLEF 2013. In: Forner, P., Navigli, R., Tufis, D. (eds.) Working Notes Papers of the CLEF 2013 Evaluation Labs (September 2013)
9. Torrejón, D., Ramos, J.: Text Alignment Module in CoReMo 2.1 Plagiarism Detector–Notebook for PAN at CLEF 2013. In: Forner, P., Navigli, R., Tufis, D. (eds.) Working Notes Papers of the CLEF 2013 Evaluation Labs (September 2013)