An Alignment-based Approach to Text Segmentation Similarity Scoring

Gerardo Ocampo Diaz and Jessica Ouyang

Department of Computer Science University of Texas at Dallas Richardson, TX 75083 {godiaz, jessica.ouyang}@utdallas.edu

Abstract

Text segmentation is a natural language processing task with popular applications, such as topic segmentation, element discourse extraction, and sentence tokenization. Much work has been done to develop accurate segmentation similarity metrics, but even the most advanced metrics used today, B, and WindowDiff, exhibit incorrect behavior due to their evaluation of boundaries in isolation. In this paper, we present a new segment-alignment based approach to segmentation similarity scoring and a new similarity metric A. We show that Adoes not exhibit the erratic behavior of B and WindowDiff, quantify the likelihood of B and WindowDiff misbehaving through simulation, and discuss the versatility of alignment-based approaches for segmentation similarity scoring. We make our implementation of A publicly available and encourage the community to explore more sophisticated approaches to text segmentation similarity scoring.

1 Introduction

Text segmentation is a natural language processing (NLP) task that consists of dividing a sequence of text elements into segments.

Let $T = e_1, e_2, e_3...e_n$ be a sequence of text elements (e.g. words, sentences, paragraphs, etc...). A segmentation S of T is given by a binary string $Q = [0|1]^{n-1}$ that encodes boundaries between the elements of T. The *i*th character of Q codifies the presence of a boundary (1) or lack thereof (0) between e_i and e_{i+1} in S. S contains m - 1boundaries and partitions T into m segments¹.

Measuring similarity between segmentations is not simple. The most straightforward approach is to frame a segmentation as a series of decisions made at every *potential boundary position* (PBP),

	0 ↓	0 ↓	$\stackrel{1}{\downarrow}$	0 ↓	0 ↓	
S	Dogs	are	cute	Very	fast	cars

Figure 1: Example segmentation with Q = 00100.

which exist between every pair of elements in T, and to calculate the average PBP agreement, but this does not match human intuition well.

Consider how S in Figure 1 compares with h1and h2 in Figure 2: h_1 agrees with S in 4 out of 5 positions (one missing boundary), while h_2 agrees with S in only 3 out of 5 positions (one missing and one "extra" boundary). Yet it is easy to agree that h_2 is actually closer to S, as it has simply "shifted" the boundary in S one unit to the right.

h_1	Dogs	are	cute	Very	fast	cars
h_2	Dogs	are	cute	Very	fast	cars

Figure 2: Alternate segmentations to S from Figure 1.

To address this, researchers have proposed a variety of similarity metrics that distinguish "soft" and "hard" errors (shifted versus missing/extra boundaries). However, existing metrics look at boundary errors in isolation; they do not consider the impact that errors have on segments around them.

r		 	 	 		 	 	
h_3		 	 	 		 	 	
h_4	1	 	 	 	 		 	

Figure 3: Three similar segmentations.

Consider how hypothesis segmentations h_3 and h_4 compare to a reference segmentation r in Figure 3. Both have a boundary that is shifted one PBP to the right, which results in an"extra" element in the segment to the left of the PBP and a missing element in the segment to the right. However, the

¹This definition corresponds to *single-type* segmentation. A *multi-type* version also exists where different boundary types are considered, enabling the encoding of different types of segments and even hierarchical relations between them.

resulting segment distortion is not the same. In h_3 , only 1/2 of the elements in the the first segment are correct, while 1/2 of the reference elements are missing from the second segment; in h_4 , the third segment has 4/5 correct elements, and the fourth segment has 1/4 missing elements. It is easy to argue then that h_4 is closer to r than h_3 , but current metrics are unable to distinguish between them.

We propose a new similarity metric based on segment alignment, which scores segmentations based on how well their *segments* match, rather than their boundaries (Section 3). We show that our metric aligns more closely with human intuition than existing metrics (Section 4) and quantify the errors encountered by those metrics (Section 5). Code for our new metric and relevant materials are made publicly available at https://github.com/sierra98x/resources.

2 Existing Metrics

Current segmentation similarity metrics fall into two categories: **window-based** metrics try to capture errors by sliding a window across the element sequence T and comparing the boundaries in both segmentations; in contrast, **edit-based** metrics try to find a sequence of boundary edit operations that would make both segmentations equal.

2.1 Window-Based Metrics

WindowDiff (Pevzner and Hearst, 2002) and P_k (Beeferman et al., 1999) are the most popular similarity metrics currently used.

 $\mathbf{P}_{\mathbf{k}}$ is defined as "the probability that a random pair of elements, k elements apart, will be classified inconsistently by two segmentations as belonging/not belonging in the same segment." Given an element sequence T of length n, a reference segmentation r, and an alternate segmentation h, a window of size k + 1 is slid across the elements (k is recommended by the authors to be half the average segment size in r); at every window position, the segmentations are compared based on the elements at the edges of the window, e_i and e_{i+k} ; if the segmentations disagree on whether the elements belong in the same segment, a penalty of 1 is added; finally, the penalty sum is divided by the number of windows:

$$P_k(r,h) = \frac{1}{n-k} \sum_{i=1,j=i+k}^{i=n-k} \delta(r_{i,j}) \neq \delta(h_{i,j})$$



Figure 4: Illustration of P_k and WindowDiff with k = 4 (Pevzner and Hearst, 2002). Penalized windows indicated by dashed lines.

where $\delta(x_{i,j})$ is true iff e_i , e_j are in the same segment in segmentation x.

There are a variety of situations where P_k penalizes errors inconsistently (Pevzner and Hearst, 2002): it penalizes missing boundaries more than extra boundaries, fails to penalize extra boundaries that are in close proximity to correct boundaries, and is also quite sensitive to the window size k.

WindowDiff improves on P_k by using a different penalty criteria. Instead of comparing the elements at the window edges, WindowDiff counts the number of boundaries between the edge elements and assigns a penalty of 1 if the number is inconsistent between segmentations:

WD
$$(r,h) = \frac{1}{n-k} \sum_{i=1,j=i+k}^{i=n-k} b(r_{i,j}) \neq b(h_{i,j})$$

where $b(x_{i,j})$ is the boundary count between e_i and e_j in segmentation x.

WindowDiff solves some of P_k 's inconsistency problems, but still produces unintuitive scores and penalizes errors at the edges of the element sequence less than those towards the middle (a weakness shared with P_k). WindowDiff is usually reported along with P_k rather than instead of it.

Lamprier et al. (2007) present a simple correction to WindowDiff: adding k - 1 extra elements at the beginning and end of the sequence T ensures that errors at every PBP are penalized an equal number of times. Further, they argue that WindowDiff is unfair because the expected score of a random segmenter depends on the number of boundaries in the reference r. To address this, they present two normalized versions of WindowDiff, **NWin** and **TNWin**, which take into account the expected WindowDiff scores of two random segmentations with the same cardinality as the reference and hypothesis segmentations being evaluated. Finally, Scaiano and Inkpen (2012) propose **WinPR**, which uses the element padding correction from (Lamprier et al., 2007) and categorizes the errors at each window into true positives (correct boundaries), false positives (extra boundaries), true negatives (correct empty PBPs), and false negatives (missing boundaries), allowing for finer-grained error analysis and the calculation of F1 scores.

Although WinPR is an improvement on WindowDiff, it has not been widely adopted by the community and, like NWin, depends on the correctness of WindowDiff; the improvements presented in WinPR and NWin do not offset the core theoretical issues with WindowDiff. The th Thus, throughout the rest of this paper, we will limit our discussion of window-based metrics to WindowDiff and P_k .

2.2 Edit-Based Metrics

Edit-based segmentation similarity metrics are based on ideas introduced by Damerau-Levenshtein string edit distance (Damerau, 1964; Levenshtein, 1966) and partially replicated by Generalized Hamming Distance (Bookstein et al., 2002). The general idea is that every segmentation can be framed as a sequence of boundaries, each placed at a specific position. If we define a set of edit operations (with costs) that can modify any sequence of boundaries, the distance between two segmentations can be measured as the cost of the optimal sequence of edit operations required to make the two segmentations equal. The optimal sequence of edit operations is equivalent to a boundary alignment between the segmentations.

Segmentation Similarity (Fournier and Inkpen, 2012) and Boundary Similarity (Fournier and Inkpen, 2012) are both based on the same set of boundary edit operations:

- Match: Mark a boundary as correct (no cost).
- Addition/Deletion: Insert or delete a boundary.
- K-Transposition: Shift a boundary to the left or right by a max of k units². Default $k = 1^3$.
- Substitution: Replace a boundary with one of a different type⁴.

Segmentation Similarity (S) (Fournier and Inkpen, 2012) assigns a constant cost to all edit



Figure 5: Example segmentation alignment with boundary edit operations (Fournier, 2013).

operations and normalizes the resulting distance based on the total number of possible boundaries for the given element sequence. The idea behind this normalization is to scale the cost based on the potential complexity of the segmentation in question; the intuition is that a constant cost is less impactful on a longer/more complex sequence than it is on a shorter/simpler one.

Let A_e , $T_e S_e$ be the sets of the optimal boundary addition/deletion, transposition, and substitution operations required to align a pair of segmentations, h_1 and h_2 , over a sequence of elements T. Further, let b be the number of boundary types (in the case of multi-type segmentation) available.

$$S(h_1, h_2, T) = 1 - \frac{|A_e| + |T_e| + |S_e|}{b(|T| - 1)}$$

Fournier and Inkpen argue that S a) produces scores that align favorably with human intuition compared to WindowDiff in three key examples, b) has reduced sensitivity to variations in segment sizes compared to WindowDiff, and c) produces more accurate inter-annotator agreement scores than WindowDiff in one dataset. It is also noted that S can be used for multi-type segmentation, where traditional window-based methods can not.

Boundary Similarity (B) (Fournier, 2013) improves S by introducing weighted-costs transpositions/substitutions, improving the edit distance normalization factor, and producing a confusion matrix from the edit operations to calculate F1 scores.

$$B(h_1, h_2, T) = 1 - \frac{|A_e| + t(T_e, k) + s(S_e, B_t)}{|A_e| + |T_e| + |S_e| + |M|}$$

where k is the maximum transposition distance, |M| is the number of matching boundary pairs between the two segmentations, B_t is the set of boundary types, and t and s are functions that return the weighted sums of T_e (transpositions) and S_e (substitutions). The normalization factor in B produces behavior that aligns more closely with human judgement than in S.

²If a boundary can not be transposed, it must be deleted and a boundary must be inserted at the corresponding location.

³When k > 1, Segmentation Similarity and Boundary Similarity allow transpositions across existing boundaries.

⁴Only required for multi-type segmentation.

When comparing scores generated by WindowDiff, P_k , S, and B on a handful of key examples, Fournier argues that B produces behavior that falls more in line with human intuition. B is further shown on one dataset to produce more reliable interannotator agreement scores when compared to S, as S-based inter-annotator agreement scores are shown to be inflated, and also to overcome WindowDiff's bias towards segmentations with few or tightly-clustered boundaries when evaluating three segmenters.

As we will demonstrate Section 4, however, B (and WindowDiff) disregards the impact of individual mistakes on the surrounding segments, which leads to scores that do not align well with human judgement in key scenarios.

3 An Alignment-Based Approach to Segmentation Similarity Scoring

In Section 1, Figure 2, we presented an example that showcased the importance of weighing boundary differences in terms of the impact they have on their corresponding segments. None of the current metrics attempt to do this, and they can not be easily modified to do so.

We propose to measure similarity between a pair of segmentations by comparing the segments defined in them. The intuition is straightforward: two segmentations are similar iff the segments defined by them are similar. Inspired by alignments from machine translation and string comparison, our approach measures segmentation similarity by *finding the maximum likelihood segment alignment and scoring its correctness*.

The concept of the most likely alignment is based on two key observations. First, it only makes sense to align overlapping segments. Second, the overlap between two segments is a good indicator for their "closeness", which tells us if they should be aligned. Thus, the maximum likelihood alignment (MLA) is one where every segment is aligned to its closest other segment.



Figure 6: Sample maximum likelihood alignment.

Consider the example alignment in Figure 6: h_2 has fuzzily merged the first two segments in h_1 into

a single segment, which results in the third segment from h_1 having a slightly shifted boundary in h_2 . Here, it does not make sense to align the third segment in h_1 with the first segment in h_2 ; even if they overlap, the third segment in h_1 overlaps mainly with the second segment in h_2 .

The MLA can be found greedily in $O(m_1 + m_2)$ time, where m_1 and m_2 are the number of segments in h_1 and h_2 , respectively. We only need to find the closest segment for any given segment⁵. Figure 7 shows pseudocode for generating the MLA⁶.

MLA(h1,h2,fn: c):

```
for each segment p in h1
| for each p-overlapping segment q in h2
| | closeness = c(p,q)
| r = max c(p,x) segment in h2
| align p (source) to r (target)
repeat for h2
```

return list of alignment edges

Figure 7: Maximum likelihood alignment algorithm.

The MLA depends on the closeness function c. For a generic alignment, where all elements in the element sequence are considered equal, we recommend a simple intersect ratio function i between two segments, x and y:

$$i(x,y) = \frac{\operatorname{intersect}(x,y)}{|x|}$$

The MLA explains the differences between a pair of segmentations in terms of boundaries: in Figure 8, missing/extra boundaries are indicated by the existence of segments with more than one aligned segment. The first segment in h_3 is aligned to two segments in h_4 because h_4 contains an extra boundary; similarly, the third segment in h_4 is aligned to two segments in h_3 because the third segment in h_4 is missing a boundary present in h_3 . Furthermore, the existence of pairs of aligned segments are indicators of matches or transpositions, such as the last segments in h_3 and h_4 .

⁵We resolve max closeness ties with Jaccard index scores (see next page); if the tie can not be broken, the left-most segment is chosen to align. Other tie-breaking strategies may be used.

⁶Pseudocode is not $O(m_1 + m_2)$; presented for brevity.



Figure 8: Maximum likelihood alignment.

Once the MLA has been generated, a function should be chosen to map the MLA to a similarity score. A simple approach is to assign a weight to every alignment edge, using a function g, and normalize by the number of edges in the MLA. This generic similarity score A is defined as:

$$A(h_1, h_2, c, g) = \frac{\sum_{\text{edge} \in \text{MLA}(h_1, h_2, c)} g(\text{edge})}{\# \text{edges in MLA}(h_1, h_2, c)}$$

A variety of edge weighting functions can be used: clustering similarity functions such as the rand index, or set similarity metrics such as the overlap coefficient, the Sørensen–Dice coefficient, or the Jaccard index. Both symmetric and asymmetric weighting functions can be used, as the edges generated by the MLA function are directed; we recommend the Jaccard index, since it guarantees a symmetrical segmentation similarity score. Further, the Jaccard version of A can be easily modified to distinguish between "soft" and "hard" mistakes by penalizing edges with weights under some threshold t. The Jaccard index, $J \in [0, 1]$, between two sets S and T is defined as:

$$J(S,T) = \frac{|\text{intersect}(S,T)|}{|\text{union}(S,T)|}$$

The MLA approach with similarity score function A compares favorably to WindowDiff, B, and similar metrics in terms of error analysis, as the MLA structure and edge weights provide information about segmentation differences in terms of both boundaries and segments.

Further, the separation between the MLA algorithm and the similarity score function A makes our approach quite versatile, as the MLA may instead be scored with a different, task-specific similarity scoring function. Consider the reference segmentation r and candidate segmentations h_1 and h_2 in Figure 9: h_1 and h_2 are equidistant to r under A with Jaccard (0.58), B, and WindowDiff. However, for a task like topic segmentation, h_2 may be preferred, as it contains "meta" topics that consistently match two topics each in r, whereas h_1 contains two correct topics, but one really bad third topic,

which is a mixture of four topics in r. Conversely, for a task like sentence segmentation, h_1 may be preferred, as it correctly identifies two sentences, where h_2 contains only incorrect sentences. The MLA could be used in conjunction with a similarity scoring function that imposes exponentially increasing penalties on segments with many alignments to favor h_2 , while a scoring function that considers only the highest weighted edge for any given segment would favor h_1 .

r				
h_1				
h_2	1			

Figure 9: Reference segmentation and two candidates.

Finally, as we will show in the following section, a straightforward implementation of A, using the intersect ratio i as the closeness function and the Jaccard index J as the edge weight function, behaves favorably compared to current metrics in a key set of examples.

4 Similarity Metric Behavior

In this section, we outline three erratic behaviors from B and WindowDiff, and compare these metrics against A in a series of example segmentations.

4.1 Cross-Boundary Transpositions

Since *B* and WindowDiff look at each boundary in isolation, they consider all boundary shifts with the same distance to be equally bad, resulting in pseudo-transpositions, where one boundary crosses over another, being penalized the same as standard transpositions. It is easy to argue against this, as a boundary shift that crosses another boundary is not a true transposition, but rather a pair of overand under-segmentations. This is illustrated in Figure 10, where h_1 is clearly closer to the reference segmentation r than is h_2 . h_1 transposes the leftmost boundary of r two units to the right, while h_2 pseudo-transposes the rightmost boundary two units to the left, crossing over the middle boundary. h_2 results in an oversegmentation of the second segment and undersegmentation of the third and fourth segments of r. A correctly identifies this behavior because it works on segment alignments; B and WindowDiff, however, incorrectly score h_1 and h_2 as being equally close to r.

r	1		I I I		1	1	 		1		
h_1	1					 	 		1		
h_2					1	1			1		
P	Pair A			<i>B</i> (k=1)			1 - WD (k=2)			2)	
(\mathbf{r}, h_1)		0.8	3	0.50		0.67					
(r, h_2)		0.6	0	0.50	0	0.67					

Figure 10: Cross-boundary pseudo-transposition.

r	 			1	1			 		 	 	 	
h_1				1		1			 	 	 	1	
h_2	 			1	1	1		 		 	1	1	1
P	air	A		В	8 (k	=1)	1	_ `	WI) (k=	2)
(r, h_1)		0.9	1	0.83			0.83						
(r, h_2)		0.7	9		0.8	3		0.83					

Figure 11: Constant cost transpositions.

4.2 Constant Cost Transpositions

B and WindowDiff measure the cost of a transposition based on its absolute distance, without considering the impact it has on the surrounding segments. This quickly leads to problematic behavior, as any given pair of segmentations that contain transpositions with the same distance will get the same score (assuming all other boundary operations are the same). Figure 11 illustrates this problem: Band WindowDiff score both segmentations equally, even though the impact of the transposed boundaries is not the same. Both h_1 and h_2 transpose a boundary by one unit, but in h_1 this results in a single extra/missing token in segments originally of size five, while in h_2 , the extra/missing token affects segments originally of size two, impacting them more significantly. A produces proper behavior here, weighing the distance of the transposition in relation to the corresponding segment sizes.

4.3 Vanishing Transpositions

Unlike A, the sensitivity of B and WindowDiff to "near-misses" (transpositions) is regulated by constants. B defines a maximum transposition distance, while WindowDiff utilizes a fixed window size. This causes problematic behavior, as boundary shifts beyond the maximum transposition distance for each metric look the same, which is compounded with the disregard for segment sizes mentioned in the previous subsection.

h_{1a}			
h_{1b}		i i i	
h_{2a}			
h_{2b}			
	•		
h_{3a}			
h_{3b}			
h_{4a}			
h_{4b} \downarrow \downarrow			
h_{5a}			
h_{5b}			
Pair	A	<i>B</i> (k=1)	1 - WD (k=2)
(h_{1a}, h_{1b})	0.70	0.33	0.69
(h_{2a}, h_{2b})	0.76	0.33	0.69
(h_{3a}, h_{3b})	0.83	0.33	0.69
(h_{4a}, h_{4b})	0.91	0.75	0.85
(h_{5a}, h_{5b})	1	1	1

Figure 12: Vanishing Transpositions

The five segmentation pairs in Figure 12 illustrate this behavior. Although the pairs are ordered by increasing similarity, B and WindowDiff score three out of five pairs equally using their default maximum transposition distance values of 1. The behavior of B is particularly concerning, as it jumps from a relatively high score of 0.75 for the fourth pair, to a very low score of 0.33 for the third pair. In contrast, A correctly matches the first segments and second segments in each pair and considers the relative impact of the transposition given the size of the segments involved.

4.4 Alignment with Human Intuition

We perform a simple experiment to verify whether human judgement of segmentation similarity is sensitive to the kind of errors previously described. We hand-craft 3 reference segmentations r, each with a pair of alternate segmentations, h_1 and h_2 , that exemplify each of the problems described in this section (cross-boundary transpositions, constant cost transpositions, and vanishing transpositions). We present each of these 3 instances to 6 NLP graduate students and ask them to indicate whether the alternate segmentations, h_1 and h_2 are equally similar to their reference segmentation r, or whether one is more similar than the other. For all 3 reference segmentations, all 6 students agree that h_1 and h_2 are *not* equally similar to r; in fact, they prefer the candidate segmentation that has the smallest relative impact on the segments being transposed. The document presented to students and the tally of their responses is available in Appendix A.

5 Error Quantification

We quantify the likelihood of WindowDiff and B behaving erroneously through simulation⁷. For the three main error types described in Section 4, we first instantiate every possible reference segmentation r for sequences of length $n \in [5, 20]$. We then try to find two alternate segmentations h_1 and h_2 that B or WindowDiff score as equally similar to r, but in fact are not. Finally, for both B and WindowDiff, we present the ratio of reference segmentations r of length n for which such error-producing pairs h_1 and h_2 exist. We also include A in our simulations and verify that it does not behave erroneously in any of the tested scenarios, so it is not included in our discussion.

To simplify our analysis, we use the Lampriercorrected version of WindowDiff (Lamprier et al., 2007), which pads the beginning and end of the sequence with k - 1 elements. The number of errors produced by this version is a lower bound on the number of errors produced by the original WindowDiff, which penalizes boundary mismatches at the edges less than those at the center.

5.1 Cross-Boundary Transpositions

We consider *B* and WindowDiff (*WD*) to behave erroneously if a pair of segmentations h_1 and h_2 are judged equally similar to *r*, where h_2 pseudotransposes a boundary by *x* units, crossing an existing boundary from *r*, and h_1 performs a standard transposition on any boundary, i.e., does not transpose across boundaries, also by *x* units. We only consider h_1 where the two segments on either side of the transposed boundary have Jaccard > 0.5with their corresponding original segments in *r*, i.e. the transposition can reasonably be considered a "soft" mistake where the affected segments are still more similar to the originals than not, in contrast to the pseudo-transposition in h_2 , where, by crossing a boundary, h_2 effectively oversegments one reference segment and undersegments another (Figure 10).



Figure 13: Ratio of potential cross-transposition errors for *B* and WindowDiff.

Figure 13 shows the percentage of the reference segmentation space for which such erroneous pairs h_1 , h_2 exist, for both *B* and *WD* with various sequence lengths n. First, note that B and WD behave similarly; both B and WD penalize transpositions based solely on distance, so it is expected that they would judge any erroneous pair h_1 , h_2 to be equidistant to r if both h_1 , h_2 transpose one boundary by the same distance. Second, the relationship between the number of segments m and the total number of elements in the sequence n reveals an interesting trend: when the number of segments is too low or too high, it is impossible to construct erroneous pairs. For example, erroneous pairs cannot be constructed for m = 2 because there is no "second" boundary to transpose across; similarly, when m approaches n, the segments become unit-sized and can no longer be involved in either normal or cross-boundary transpositions. Thirdly, the increasing-decreasing behavior of the curves stems from the "soft" transposition constraint that we impose on h_1 . It can be shown that the smallest possible sizes for two adjacent segments containing a "soft" transposition are 5 and 3; this is because the minimum (pseudo-)transposition distance required to cross a boundary is 2 units. Once m is large enough that the average segment size is less than 3, it becomes increasingly hard to find such adjacent segments of sizes 5 and 3, so the number of erroneous h_1 , h_2 pairs decreases steadily.

⁷We use the implementation of *B* from the segeval Python 3 package (Fournier, 2013) and WindowDiff from the Python 3 NLTK package (Bird et al., 2009). Simulation code is available at https://github.com/sierra98x/resources.

5.2 Constant Cost Transpositions

Here, *B* and *WD* behave erroneously if a pair of segmentations h_1 and h_2 are judged equally similar to *r*, where h_1 "soft" transposes a boundary by *x* units, and h_2 "hard" transposes any boundary by *x* units, i.e., the segments on either side of the transposition in h_2 have Jaccard < 0.5 with their corresponding original segments in *r*.



Figure 14: Ratio of potential constant cost transposition errors for *B* and WindowDiff.

The trend in Figure 14 can be explained as follows: if the ratio between the number of segments m and the sequence length n is too low, the segments are so large that it is rare to find a pair of segments such that transposing their boundary results in a "hard" error; conversely, when the ratio is too high, all segments are small, so it becomes increasingly hard to find "soft" transpositions. Like in Figure 10, B and WD follow the same trend because h_1, h_2 transpose by the same number of units; in addition, we see once again that the number of erroneous pairs starts decreasing once the average segment size (m/n ratio) drops below 3.

5.3 Vanishing Transpositions

Here, *B* and *WD* behave erroneously if a pair of segmentations h_1 and h_2 are judged equally similar to r, where h_1 transposes a boundary by x units and h_2 transposes the same boundary by y > x units. Again, we only consider h_1 with "soft" transpositions, where the two segments on either side of the transposed boundary have Jaccard > 0.5 with the corresponding original segments from r; h_2 may have a "soft" or "hard" transposition.

Figure 15 differs from Figures 13 and 14 in that *B* and *WD* behave differently for this error type, which makes sense, given that this is the only ex-



Figure 15: Ratio of potential vanishing transposition errors for B and WindowDiff.

periment where h_1 , h_2 do not transpose by the same number of units: recall that *B* has a fixed maximum transposition size (default value of 1) beyond which transpositions can not be distinguished, while *WD*'s maximum transposition size depends on the window size *k*, which is equal to half the average segment size, and thus a function of *m* and *n*. The global peak in error rate for *WD* occurs when *k* is so small that no transpositions are allowed; *B* makes fewer mistakes than *WD* because *B* always allows transpositions of size 1. The local minimum between the first and second maxima for *WD* is caused by the step-wise nature of the *k* function, since the window size must be a whole number.

6 Limitations

While we have seen that A performs favorably when compared to B and WindowDiff, further investigation may be warranted on the general MLA approach. First, the space of potential alignments for a given pair of segmentations can be quite large, and while a simple greedy intersection ratio approach generates sensible alignments, edge cases may exhibit undesirable behavior.

Consider Figure 16: h_0 deletes a boundary from r, while h_1 and h_2 transpose it different distances. However, A gives h_2 a worse score than h_0 ; this behavior is explained by the MLA between r and h_2 containing a diagonal alignment between the first segment in h_2 and the second segment in r, due to the first segment in r being very small — so small that transposing its boundary by two units is considered worse than deleting it. The intersect ratio closeness function in A uses segment size to distinguish "soft" and "hard" transpositions; as we

	r		
	h_0		
	h_1		
	h_2		1 I I I I I I I I
Pair	A	<i>B</i> (k=1)	1 - WD (k=2)
(\mathbf{r}, h_0)	0.67	0.50	0.88
(r, h_1)	0.79	0.75	0.88
(r, h_2)	0.58	0.33	0.62

Figure 16: Intricate behavior of A.

saw in Figure 12, when the segments are longer, A will match the left and right segments of a twounit transposition with the original segments in r, resulting in a similarity score greater than h_0 .

However, specific applications may lean towards favoring "soft" transpositions over deletions regardless of segment size, which would require a) a different segment-to-segment closeness function c, or b) maximizing some global MLA function, such as Maximum Spanning Tree.

Second, in Figure 9, we presented a reference segmentation r and two different candidate segmentations h_1 and h_2 that are scored equally by A, B, and WD. Here, the fact that A can not distinguish between them is not due to the MLA, as there is only one possible alignment between each candidate and r. Thus, it may be of interest to develop more sophisticated MLA scoring functions, in order to distinguish between h_1 and h_2 .

7 Conclusion

In this paper, we present a new alignment-based approach to text segmentation similarity scoring and present a new similarity metric A. We show that, unlike A, the most advanced segmentation similarity metrics, B and WindowDiff, behave erratically in three key scenarios. We discuss the versatility of alignment-based approaches when paired with different alignment and scoring functions, and show that A, B, and WindowDiff exhibit intricate behaviors that should be explored in the future. We make our implementation of A publicly available⁸ in hope that it encourages the NLP community to explore more sophisticated approaches to text segmentation similarity scoring.

References

- Doug Beeferman, Adam Berger, and John Lafferty. 1999. Statistical models for text segmentation. *Machine learning*, 34(1):177–210.
- Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".
- Abraham Bookstein, Vladimir A. Kulyukin, and Timo Raita. 2002. Generalized hamming distance. *Inf. Retr.*, 5(4):353–375.
- Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun.* ACM, 7(3):171–176.
- Chris Fournier. 2013. Evaluating text segmentation using boundary edit distance. In *Proceedings of the* 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1702–1712, Sofia, Bulgaria. Association for Computational Linguistics.
- Chris Fournier and Diana Inkpen. 2012. Segmentation similarity and agreement. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 152–161, Montréal, Canada. Association for Computational Linguistics.
- Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, and Frederic Saubion. 2007. On evaluation methodologies for text segmentation algorithms. In 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), volume 2, pages 19– 26.
- Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, page 707.
- Lev Pevzner and Marti A. Hearst. 2002. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36.
- Martin Scaiano and Diana Inkpen. 2012. Getting more from segmentation evaluation. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 362–366, Montréal, Canada. Association for Computational Linguistics.

A Human Judgement Test

The following text was presented to 6 graduate NLP students to verify their sensibility to crossboundary transposition, constant cost transposition, and vanishing transposition errors. For clarity, we have added error type labels to the questions, but the students were not shown these labels during the evaluation.

⁸Our implementation of A, along with relevant materials, can be found at https://github.com/sierra98x/resources.

Intro====

A segmentation splits a sequence of elements into meaningful, non-overlapping segments.

Ex. Split a transcript into utterances:

I did not go to work yesterday, did you? | No, Johnathan filled in for me.

Note: For simplicity, the elements in every segmentation example are masked.

The previous example would look like this: A.A.A.A.A.A.A.A.A|B.B.B.B.B.B.B

Questions===

For each of the following instances, 3 segmentations are presented: Gold, H1, and H2.

Determine whether H1 or H2 is closer to gold, or if they are the same, according to your interpretation.

-AT [Cross-Boundary Transposition (LABEL NOT SHOWN)]

A|B|C.C.C.C.C.C.C.C.C.D.D.D.D.D.D.D.D.D.D. - Gold A|B|C.C.C.C.C.C.C.C.C.D.D|D.D.D.D.D.D.D.D - H2

-RTC [Constant Cost Transposition (LABEL NOT SHOWN)]

A|B.B|C.C.C.C.C.C.C.C|D.D.D.D.D.D.D.D - Gold A|B.B|C.C.C.C.C.C.C.C.D|D.D.D.D.D.D.D - H1

A|B.B|C.C.C.C.C.C.C.C|D.D.D.D.D.D.D.D - Gold A.B|B|C.C.C.C.C.C.C.C|D.D.D.D.D.D.D.D - H2

-VT [Vanishing Transposition (LABEL NOT SHOWN)]

A.A.A.A.A.A.A.A|B.B.B.B.B.B.B.B.B - Gold A.A.A.A.A.A.A.A.B.B|B.B.B.B.B.B.B - H1

A.A.A.A.A.A.A.A|B.B.B.B.B.B.B.B.B - Gold A.A.A.A.A.A.A.A.B.B.B|B.B.B.B - H2

The students achieved perfect agreement on the evaluation and judged as more similar the candidate segmentation with the smallest impact on the gold segments.

Instance	H1 Votes	H2 Votes	Same Votes
AT	0	6	0
RTC	6	0	0
VT	6	0	0

Table 1: Human evaluation results