AT&T Bell Laboratories

subject: **A Program for Aligning Sentences in Bilingual Corpora**

date: **October 4, 1990**

Work Project No. 311401-2399, 311404-2099
File Case 20878, 60011

from: **William A. Gale**
Org. 11215
MH 2C-278
x2149

Kenneth W. Church
Org. 11222
MH 2D-454
x5325

11215-901004-11TM
11222-901004-11TM

TECHNICAL MEMORANDUM

ABSTRACT

Researchers in both machine translation and bilingual lexicography have recently become interested in studying bilingual corpora, bodies of text such as the Canadian Hansards (parliamentary proceedings) which are available in multiple languages (such as French and English). One useful step is to align the sentences, that is, to identify correspondences between sentences in one language and sentences in the other language.

This paper will describe a program (*align*) for aligning sentences based on a very simple statistical model of character lengths. The program uses the fact that longer sentences in one language tend to be translated into longer sentences in the other language, and that shorter sentences tend to be translated into shorter sentences. A probabilistic score is assigned to each proposed correspondence of sentences, based on the ratio of lengths of the two sentences (in characters) and the variance of this ratio. This probabilistic score is used in a dynamic programming framework to find the maximum likelihood alignment of sentences.

This simple approach works remarkably. An evaluation was performed based on a trilingual corpus of economic reports issued by the Union Bank of Switzerland (UBS) in English, French and German. The method correctly aligned all but 4% of the sentences. Moreover, it is possible to extract a large subcorpus which has a much smaller error rate. By selecting the best scoring 80% of the alignments, the error rate is reduced from 4% to 0.7%. There were roughly the same number of errors in each of the English-French and English-German alignments, suggesting that the method may be fairly language independent.

1. Introduction

Researchers in both machine translation (e.g., Brown et al, 1990) and bilingual lexicography (e.g., Klavans and Tzoukermann, 1990) have recently become interested in studying bilingual corpora, bodies of text such as the Canadian Hansards (parliamentary debates) which are available in multiple languages (such as French and English). The sentence alignment task is to identify correspondances between sentences in one

language and sentences in the other language. This task is a first step towards the more ambitious task of aligning words which correspond to each other.

The input is a pair of texts such as:

English	**French**
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates. The higher turnover was largely due to an increase in the sales volume. Employment and investment levels also climbed. Following a two-year transitional period, the new Foodstuffs Ordinance for Mineral Water came into effect on April 1, 1988. Specifically, it contains more stringent requirements regarding quality consistency and purity guarantees.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment. La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes. L'emploi et les investissements ont également augmenté. La nouvelle ordonnance fédérale sur les denrées alimentaires concernant entre autres les eaux minérales, entrée en vigueur le 1er avril 1988 après une période transitoire de deux ans, exige surtout une plus grande constance dans la qualité et une garantie de la pureté.

The output identifies the correspondance between sentences. Most English sentences correspond to exactly one French sentence, but it is possible for an English sentence to correspond to two or more French sentences. The first two English sentences (below) illustrate a particularly hard case where two English sentences correspond to two French sentences. No smaller alignments are possible because the clause ''... sales ... were higher ...'' in the first English sentence aligns with (part of) the second French sentence. The next two alignments below illustrate the more typical case where one English sentence aligns with exactly one French sentence. The final alignment matches two English sentences to one French sentence. These alignments agreed with the results produced by a human judge.

English	**French**
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment.
The higher turnover was largely due to an increase in the sales volume.	La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes.
Employment and investment levels also climbed.	L'emploi et les investissements ont également augmenté.
Following a two-year transitional period, the new Foodstuffs Ordinance for Mineral Water came into effect on April 1, 1988. Specifically, it contains more stringent requirements regarding quality consistency and purity guarantees.	La nouvelle ordonnance fédérale sur les denrées alimentaires concernant entre autres les eaux minérales, entrée en vigueur le 1er avril 1988 après une période transitoire de deux ans, exige surtout une plus grande constance dans la qualité et une garantie de la pureté.

Aligning sentences is just a first step toward constructing a probabilistic dictionary for use in aligning words in machine translation, or for constructing a bilingual concordance for use in lexicography.

Although there has been some previous work on the sentence alignment, e.g., (Brown et al, 1990), (Kay, personal communication), (Warwick, personal communication), the alignment task remains a significant obstacle preventing many potential users from reaping many of the benefits of bilingual corpora, because the proposed solutions are often unavailable, unreliable, and/or inefficient (Zampolli, personal communication), (Warwick, personal communication).

Almost all of the previous work on sentence alignment has yet to be published. Kay's approach (Kay, personal communication) is said to be very ''heavy''; it ought to be possible to achieve fairly reasonable results with much less computation (Warwick, personal communication). In (Brown et al., 1990), there is a very brief discussion of sentence alignment. In its entirity, it reads:

> ... we have been able to extract about 3 million pairs of sentences by using a statistical algorithm based on length. Approximately 99% of these pairs are made up of sentences that are actually translations of one another.

Brown et al.'s discussion was justifiably brief because sentence alignment was not the main topic of their paper; they were trying to argue for a revival of statistical methods for machine translation.[1] Nevertheless, it would be helpful to the research community to provide a more detailed discussion of a practical sentence alignment algorithm and its evaluation.

Our paper will describe a program for aligning sentences based on a very simple statistical model of character lengths. The model makes use of the fact that longer sentences in one language tend to be translated into longer sentences in the other language, and that shorter sentences tend to be translated into shorter sentences. A probabilistic score is assigned to each pair of proposed sentence pairs, based on the ratio of lengths of the two sentences (in characters) and the variance of this ratio. This probabilistic score is used in a dynamic programming framework in order to find the maximum likelihood alignment of sentences.

It is remarkable that such a simple approach can work as well as it does. An evaluation was performed based on a trilingual corpus of 15 economic reports issued by the Union Bank of Switzerland (UBS) in English, French and German (N = 14,680 words, 725 sentences, and 188 paragraphs in English and corresponding numbers in the other two languages). The method correctly aligned all but 4% of the sentences. Moreover, it is possible to extract a large subcorpus which has a much smaller error rate. By selecting the best scoring 80% of the alignments, the error rate is reduced from 4% to 0.7%. There were roughly the same number of errors in each of the English-French and English-German alignments, suggesting that the method may be fairly language independent.

To further research in bilingual corpora, the Appendix contains C-code for the core of *align*. Also, sentence aligned versions of the the Hansards from 350 days in 1986, 1987, and 1988 may be requested from the authors. These include about 700,000 sentences with about 20 million words in each language.

2. Paragraph Alignment

The sentence alignment program is a two step process. First paragraphs are aligned, and then sentences within a paragraph are aligned. It is fairly easy to align paragraphs in our trilingual corpus of Swiss banking reports since the boundaries are usually clearly marked. However, there are some short headings and signatures that can be confused with paragraphs. Moreover, these short ''pseudo-paragraphs'' are not always translated into all languages. Fortunately, it is very easy to distinguish the short ''pseudo-

1. Statistical methods were popular in the 1950s when machine translation was first unsuccessfully tried, but they are being revived with the availability of extensive machine readable bilingual corpora.

paragraphs'' from real ones by a simple threshold on length. The following histogram shows very clearly a bimodal distribution. It turns out that ''pseudo-paragraphs'' usually have less than 50 characters and that real paragraphs usually have more than 100 characters.

Bimodal Distribution of Paragraph Lengths

Figure 1. This histogram shows the frequency of English paragraphs whose lengths were less than 200 characters. There is a clear gap around 50, with headings and signatures in the short group and content paragraphs in the long group.

Since pseudo-paragraphs can be distinguished on the basis of length, the paragraph aligning algorithm is very simple. It begins by considering the first two paragraphs in a pair of documents. When considering two paragraphs, if both have length over 100 characters, or both have lengths under 50 characters, then align them, and continue. However if one of the paragraphs is short, while the other is long, then align the short paragraph with null, and continue.

For English and German, this simple algorithm made no mistakes. The algorithm was unable to align one of the French documents with either English or German. It turned out that the French translation of the document had omitted one long paragraph and had duplicated a heading and another long paragraph. (This document was excluded for the purposes of the remainder of this experiment.) If this frequency of paragraph errors were found in a larger corpus, it might be worthwhile building a more elaborate paragraph alignment mechanism.

3. A Dynamic Programming Framework

Now, let us consider how sentences can be aligned within a paragraph. The program makes use of the fact that longer sentences in one language tend to be translated into longer sentences in the other language, and that shorter sentences tend to be translated into shorter sentences. A probabilistic score is assigned to each proposed pair of sentences, based on the ratio of lengths of the two sentences (in characters) and the variance of this ratio. This probabilistic score is used in a dynamic programming framework in order to find the maximum likelihood alignment of sentences.

We were led to this approach after noting that the lengths (in characters) of English and German paragraphs are highly correlated (.991), as illustrated in the following figure. The high correlation between the lengths of the aligned paragraphs suggested using sentence lengths as a basis for aligning sentences within a paragraph.

Paragraph Lengths are Highly Correlated

Figure 2. The horizontal axis shows the length of English paragraphs, while the vertical scale shows the lengths of the corresponding German paragraphs. The correlation in their lengths is .991.

Statistical techniques for alignment of sequences have been developed for use in a variety of settings, such as the matching of genetic code sequences from different species, speech sequences from different speakers, gas chromatograph sequences from different compounds, and geologic sequences from different locations (Kruskal, 1983). To the extent that the order of the sentences does not differ between the two languages of a bilingual corpus, we could expect these sequence matching techniques to be useful. Details of the alignment techniques differ considerably from one application to another, but all use a distance measure to compare two individual elements within the sequences, and a dynamic programming algorithm to minimize the total distances between aligned elements within two sequences. Placing the sentence alignment problem within this setting, we find we need to develop a new distance measure, but that standard dynamic programming techniques suffice.

Kruskal and Liberman (1983) describe distance measures as belonging to one of two classes, which they call trace and time-warp. The difference between the two lies in how they treat a distance between one

element of one sequence and several elements from the other sequence. In trace applications, such as genetic code matching, the one element must match just one of the several elements well. In time-warp applications, such as speech template matching, the one element must match each of the several elements well. A distance measure for sentence matching must lie outside these two groups, because it needs to compare the aggregate of the the several elements with the one element. Thus we need a new distance measure.

The ideal distance measure is based on probability, because new information can then be combined with old information in a consistent way. Since a distance measure must be additive to use the dynamic programming techniques, the logarithm of the probability is used. Thus we seek an estimate of the probability that two sentences or groups of sentences match.

We use the following simple model. Each character in one language gives rise to a random number of characters in the other language. These random variables are independent and identically distributed with a normal distribution. The model is then specified by the mean and standard deviation of the the distribution. Let c be the expected number of characters in language 2 per character in language 1, and s^2 be the variance of the number of characters in language 2 per character in language 1. Then the expected number of characters in the sentences translating a group of sentences of length l_1 in language 1 is $l_1 c$ with variance $l_1 s^2$. Let $\delta = (l_2 - l_1 c)/\sqrt{l_1 s^2}$. Then if sentences of length l_1 and l_2 in the two languages do match, δ has a normal distribution with mean zero and variance one.

To estimate Prob(match$|\delta$), we appeal to Bayes Theorem, which says that this conditional probability is proportional to Prob($\delta|$match) P(match). We take Prob($\delta|$match) to be the probability that a random variable, z, with a standardized (mean zero, variance one) normal distribution, has magnitude at least as large as $|\delta|$ That is,

$$Prob(\delta|match) = 2\ (1 - P(|\delta|))$$

where

$$P(\delta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\delta} e^{-t^2/2}\, dt$$

We take the prior probability of a match, P(match), to depend only on the number of sentences being matched in the two languages, as given in Table 1 below. The distance measure used thus depends on c, s, and the probabilities of matching given numbers of sentences.

We can estimate c, the ratio of lengths from the aggregate lengths of the matched paragraphs. This gives German characters per English character = 81105/73481 = 1.1, and French characters per English character = 72302/68450 = 1.06. As explained later, using these precise and language dependent quantities does not improve the performance over $c = 1$, which gives a symmetric and simpler model. If *align* is used with languages for which the character lengths differ by more than ten percent, this parameter should be reviewed.

Our model assumes that variance s^2 is proportional to length. The follow figure doesn't contradict this assumption, and allows us to estimate the constant of proportionality. The line shown in the figure is the result of a robust regression. The result for English-German is $s^2 = 7.3$, and for English-French is $s^2 = 5.6$. Again, the sensitivity study described later showed that the differences between these two slopes were not important, so we combined the data across the languages, and used $s^2 = 6.8$ in *align*.

Variance Is Modeled Proportional to Length

Figure 3. The horizontal axis plots the length of English paragraphs, while the vertical axis shows the square of the difference of English and German lengths, an estimate of variance. The variance increases with length. We model the increase as proportional to length, as shown by the line. Five extreme points lying above the top of this figure have been suppressed since they contributed nothing to the robust regression.

A sentence in one language normally matches exactly one sentence in the other language (1-1), but we allow three other possibilities as well: 1-0, 2-1, and 2-2. That is, there are four categories for matching groups of sentences: one to zero (1-0), one to one (1-1), two to one (2-1), and two to two (2-2). The 1-0 category includes all sentences in either language aligned with nothing in the other. Likewise the 2-1 category includes cases with a pair of sentences in either language aligned with one in the other. The probabilities of the categories were determined from the data as shown in the following table.

Table 1
Categories of Matching

category	frequency	fraction
1-0	13	.0099
1-1	1167	.89
2-1	117	.089
2-2	15	.011

The fractions shown in the right hand column times the probability $P(\delta|\text{match})$, or actually the logarithm of the product, gives the distance measure for a proposed alignment. The distance measure thus has six parameters: c, s, f_{01}, f_{11}, f_{21} and f_{22}. We also refer to this distance measure as the *probability score*. The distance measure was then used in a dynamic programming subroutine written by Mike Riley to align the sentences.

4. Evaluation

A primary judge did all of the alignments, and two corroborative judges independently aligned selected hard paragraphs. In doing so, they used materials prepared with the three languages in columns, each paragraph begun on a new page, and having numerically corresponding sentences beginning at the same level. They did the alignment by drawing lines between sentences that shared a clause. If a sentence was not translated, they wrote in a zero and drew a line to that.

All of the sentences were aligned by the primary judge. The primary judge is a native speaker of English with a reading knowledge of French and of German. Each of the other two judges aligned one of the languages in 43 paragraphs with 230 sentences, chosen as particularly difficult. One of the other judges is a native speaker of French and fluent in English; the other is a native speaker of German and fluent in English. The primary judge was found to be wrong on one alignment in each of French and German in the hard group of paragraphs.

The 43 hard paragraphs were selected by making all three pairs of alignments, then attempting to trace each sentence from English to German, from there to French, and from there back to English. The 43 paragraphs included all sentences in which this process could not be completed around the loop. This group of paragraphs, selected through a trilingual criterion, had 82 percent of the errors made by the program, another justification for calling them hard. Since the primary judge had an error rate of only one half of one percent on these hard paragraphs, we believe that one judge is sufficient for defining the correct alignment of the sentences.

In evaluating the program, we considered only the English-German and English-French alignments, because the independence of the French-German alignments given the other two is questionable. Errors are

reported with respect to the alignments judged correct. That is, each correct alignment that is not called for by the program is scored as an error. This convention allows us to compare the performances of different algorithms. There were 36 errors out of 621 total alignments for English-French and 19 errors out of 695 alignments for English-German. Overall, there were 55 errors on 1316 alignments, or 4.2 percent errors.

The following table shows the errors by match category.

Table 2
Complex Matches are More Difficult

category	English-French			English-German			total		
	N	err	%	N	err	%	N	err	%
1-0	8	8	100	5	5	100	13	13	100
1-1	542	14	2.6	625	9	1.4	1167	23	2.0
2-1	59	8	14	58	2	3.4	117	10	9
2-2	9	3	33	6	2	33	15	5	33
3-1	1	1	100	1	1	100	2	2	100
3-2	1	1	100	0	0	0	1	1	100

The table shows that 1-1 alignments are by far the easiest. The 2-1 alignments, which come next, have four times the error rate for 1-1. The 2-2 alignments are harder still, but a majority of the alignments are found. The 3-1 and 3-2 alignments were not possible for the algorithm, so naturally it missed all three of these. The most surprising category is 1-0, in which all cases were missed. Furthermore, of the sentences that the algorithm is most inclined to assign to the 1-0 category, none belong there, so adding inducements of various kinds to make 1-0 assignments introduces new errors without reducing the errors reported on here. Apparently when the assumption that the two documents are translations of each other fails, which is the case for the 1-0 category, then we have considerable trouble finding the correct alignment without knowledge of the languages.

We investigated the possible dependence of the error rate on four variables. First was length of the alignment: the average of the total lengths of the sentences grouped for each language. Second was the paragraph length: the average length of the paragraphs in which the alignment occurred. Third was the complexity: whether the alignment was of the 2-1 or 2-2 category. Fourth was the probability score: the score assigned to the alignment if the program was correct, or the maximum score for any sentence involved in the true alignment if it was not correct.

Since each alignment was judged either right or wrong, we used logistic regression of the judgement on the four variables. The coefficients and their standard deviations are shown in the following table.

Table 3
Probability Score Predicts Errors

variable	coefficient	std. dev.
score	.071	.011
complexity	.52	.47
sentence length	.0013	.0029
paragraph length	.0003	.0005

The table shows that the probability score assigned to an alignment is a good predictor of whether it is in error. No other factor considered was significantly different from no effect after considering the probability score. Thus we could not improve the probability score by including any more information from these sources.

The fact that the probability score predicts the error rate may be useful, as suggested by the following figure.

Probability Score Allows Selecting Low Error Subset

Figure 4. By retaining the alignments with the lowest p percent of the probability scores, as shown on the horizontal axis, the error rate among the retained alignments is as shown on the vertical axis. An error rate of about two thirds of one percent can be obtained with 80 percent of the alignments retained.

Less formal tests of the error rate in the Hansards suggests that the overall error rate is about two percent, while the error rate for the easy 80 percent of the sentences is about 0.4%. Apparently the Hansards have better translations than these UBS reports.

5. Evaluation of Algorithm Variations

We considered several variations of the algorithm used in *align*. There are also several possible extensions of the algorithm.

5.1 Variations

One variation considered was the use of words instead of characters to measure the lengths of the sentences. Words might be expected to be worse, since fewer words per sentence suggests more uncertainty. In fact, doing so raised the English-French errors from 36 to 50, and the English-German errors from 19 to 35. The total errors were thereby increased from 55 to 85, or from 4.2 percent to 6.5 percent.

Lengths measured in characters are better because they have a lower ratio of standard deviation to length at the mean sentence length. We have modeled variance as proportional to sentence length, $V = s^2 l$. Thus the standard deviation is $\sigma(l) = s\sqrt{l}$. At the mean sentence length, m, the ratio of standard deviation to mean is thus $\sigma(m)/m = s\sqrt{m}/m = s/\sqrt{m}$. Using the true alignments, we find for character lengths $s^2 = 6.5$, and for word lengths $s^2 = 1.9$, while the mean character length is 117 and the mean word length is 17. Thus the $\sigma(m)/m$ ratios are .22 for characters and .33 for words. This means that the character lengths are less noisy than are the word lengths, and thus are better for comparison.

We tried the algorithm ignoring paragraph boundaries. The English-French errors were increased from 36 to 84, and the English-German errors from 19 to 86. The overall errors were increased from 55 to 170. Thus a top-down approach reduces errors by a factor of three. This suggests that alignments by clause or phrase within a sentence would be valuable, if parsers were available for each language.

Originally we did not allow for 2-2 alignments. Table 2 shows that the program was right on 10 of 15 actual 2-2 alignments. This was achieved at the cost of introducing 2 spurious 2-2 alignments. Thus in 12 tries, the program was right 10 times, wrong 2 times. This is significantly different from random, since there is less than .01 chance of getting 10 or more heads out of 12 flips of a fair coin. Thus it is worth while to include the 2-2 alignment possibility.

We showed that the best estimates of the model parameters, c and s^2, are somewhat different from the values $c = 1$ and $s^2 = 6.8$ that we used. When we used the exact estimates for each individual language pair, then we found exactly the same total number of errors. The models were not exactly the same, however, as there were four changes (two right and two wrong) for French and two changes (one right, one wrong) for German. Since the parameters we use are somewhat less dependent on a particular language, and since their use makes little difference for the two language pairs we have studied, their use seems desirable.

5.2 Extensions

We rejected one document in which one paragraph was omitted and two paragraphs were duplicated. A more powerful paragraph alignment could be built that might handle this case. The distance measure for paragraphs would be the average distance between their sentences under the best sentence alignment. We expect this measure would discriminate strongly between paragraphs that should be aligned and others, so that a paragraph aligning very poorly with any available paragraph could be called inserted. This measure essentially pushes the paragraph measure down a level to sentences.

For sentence alignment, it would undoubtedly be useful to use a probabilistic dictionary to augment the probability based on lengths. That is, we could push the sentence match down to the word level. This would be possible because our distance measure is a probability; it is thus possible to combine other sources of information with it in a principled fashion. We expect that this would give as good a sentence matching job as could be done by hand. However, it is not clear if it is necessary, since alignments with less than one percent errors are likely to suffice for many purposes, such as building a probabilistic dictionary.

6. Conclusions

This paper has proposed a method for aligning sentences in a bilingual corpus. We want to emphasize that the method is based on a probabilistic model, which was described in Section 3. The model is based on the observation that the lengths of aligned paragraphs have a .991 correlation. This suggests that aligned sentences should also have nearly the same lengths. The model quantifies this observation.

The method is also accurate. Overall, there was a 4.2 percent error rate on 1316 alignments. The alignments included both English-French and English-German corpora. Since the probability score assigned to each alignment is predictive of errors, it is also possible to select out 80 percent of the alignments that have an overall error rate of 0.7 percent.

The method is not strongly dependent on the language pair. Both English-French and English-German corpora were processed using the same parameters. The model does have six parameters that could vary with the language pair. They each have operational interpretations, and can be estimated for a particular language pair if necessary.

It is better to count the sentence lengths in characters than in words, because there is less variability in the ratios of sentence lengths so measured. Using words as units increases the error rate by half, from 4.2 percent to 6.5 percent. The algorithm works by aligning paragraphs first. Omitting this step increases errors by a factor of three.

Using a probability based model allows extensions that include other sources of information. After the preliminary alignments are made, a probabilistic dictionary can be built. The information from this dictionary could be fed back to the sentence alignment task to improve it. Also, if a more powerful paragraph alignment is needed, the total scores for sentence alignment within a pair of paragraphs would be a useful distance measure for the paragraphs.

Acknowledgements

We thank Susanne Wolff and and Evelyne Tzoukermann for their pains in aligning sentences. Susan Warwick provided us with the UBS trilingual corpus and posed the problem addressed here.

William A. Gale

MH-11215-WAG/KWC **Kenneth W. Church**

References

Brown, P., J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roossin, (1990) ‘‘A Statistical Approach to Machine Translation,’’ *Computational Linguistics*, v 16, pp 79-85.

Klavans, J., and E. Tzoukermann, (1990), ‘‘The BICORD System,’’ *COLING-90*, pp 174-179.

Kruskal, J., (1983), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Mass.

Kruskal, J., and M. Liberman, (1983), ‘‘The Symmetric Time-Warping Problem: From Continuous to Discrete,’’ in (Kruskal, 1983).

Appendix: Program

with Michael D. Riley

The following code is the core of *align*. It is a C language program which takes as input texts in two files with two embedded special markers, one for ‘‘hard’’ regions (such as paragraphs), and one for ‘‘soft’’ regions (such as sentences). The hard regions may not be changed, and there must be equal numbers of them in the two languages, although there may be just one hard region. The soft regions may be deleted or inserted (1-0 and 0-1), substituted (1-1), contracted or expanded (2-1 and 1-2), or merged (2-2). The output is two files with equal numbers of soft regions, each on a line, with the hard region markers separating groups of lines. If the -v command line option is included, each soft region is preceded by its probability score.

```
/*Subject: align_regions.c */

#include <fcntl.h>
#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <values.h>
#include <sys/stat.h>

/*

  usage

  align_regions -D '.PARA' -d '.End of Sentence'  file1 file2

  outputs two files: file1.al & file2.al

  regions are delimited by the -D and -d args

  the program is allowed to delete -d delimiters as necessary in order
  align the files, but it cannot change -D delimiters.

  */

#define dist(x,y) distances[(x) * ((ny) + 1) + (y)]
#define pathx(x,y) path_x[(x) * ((ny) + 1) + (y)]
#define pathy(x,y) path_y[(x) * ((ny) + 1) + (y)]

#define BIG_DISTANCE 2500


struct alignment {
  int x1;
  int y1;
  int x2;
```

```
  int y2;
  int d;
};

char *hard_delimiter = NULL;      /* -D arg */
char *soft_delimiter = NULL;      /* -d arg */
int verbose = 0;           /* -v arg */
int debug = 0;                  /* -b arg */

/*

  seq_align by Mike Riley
  Sequence alignment routine.
  This version allows for contraction/expansions.

  x and y are sequences of objects, represented as non-zero ints, to be aligned.

  dist_funct(x1, y1, x2, y2) is a distance function of 4 args:

    dist_funct(x1, y1, 0, 0) gives cost of substitution of x1 by y1.
    dist_funct(x1, 0, 0, 0) gives cost of deletion of x1.
    dist_funct(0, y1, 0, 0) gives cost of insertion of y1.
    dist_funct(x1, y1, x2, 0) gives cost of contraction of (x1,x2) to y1.
    dist_funct(x1, y1, 0, y2) gives cost of expansion of x1 to (y1,y2).
    dist_funct(x1, y1, x2, y2) gives cost to match (x1,x2) to (y1,y2).

  align is the alignment, with (align[i].x1, align[i].x2) aligned
    with (align[i].y1, align[i].y2).  Zero in align[].x1 and align[].y1
    correspond to insertion and deletion, respectively.  Non-zero in
    align[].x2 and align[].y2 correspond to contraction and expansion,
    respectively.  align[].d gives the distance for that pairing.

  The function returns the length of the alignment.

*/

int
seq_align(x, y, nx, ny, dist_funct, align)
     int *x, *y, nx, ny;
     int (*dist_funct)();
     struct alignment **align;
{
  int *distances, *path_x, *path_y, n;
  int i, j, oi, oj, di, dj, d1, d2, d3, d4, d5, d6, dmin;
  struct alignment *ralign;

  distances = (int *) malloc((nx + 1) * (ny + 1) * sizeof(int));
  path_x = (int *) malloc((nx + 1) * (ny + 1) * sizeof(int));
  path_y = (int *) malloc((nx + 1) * (ny + 1) * sizeof(int));
  ralign = (struct alignment *) malloc((nx + ny) * sizeof(struct alignment));

  for(j = 0; j <= ny; j++) {
    for(i = 0; i <= nx; i++) {
      d1 = i>0 && j>0 ?          /* substitution */
        dist(i-1, j-1) + (*dist_funct)(x[i-1], y[j-1], 0, 0)
          : MAXINT;
      d2 = i>0 ?          /* deletion */
        dist(i-1, j) + (*dist_funct)(x[i-1], 0, 0, 0)
          : MAXINT;
      d3 = j>0 ?          /* insertion */
```

```
      dist(i, j-1) + (*dist_funct)(0, y[j-1], 0, 0)
        : MAXINT;
    d4 = i>1 && j>0 ?          /* contraction */
      dist(i-2, j-1) + (*dist_funct)(x[i-2], y[j-1], x[i-1], 0)
        : MAXINT;
    d5 = i>0 && j>1 ?          /* expansion */
      dist(i-1, j-2) + (*dist_funct)(x[i-1], y[j-2], 0, y[j-1])
        : MAXINT;
    d6 = i>1 && j>1 ?          /* melding */
     dist(i-2, j-2) + (*dist_funct)(x[i-2], y[j-2], x[i-1], y[j-1])
        : MAXINT;

    dmin = d1;
    if(d2<dmin) dmin=d2;
    if(d3<dmin) dmin=d3;
    if(d4<dmin) dmin=d4;
    if(d5<dmin) dmin=d5;
    if(d6<dmin) dmin=d6;

    if(dmin == MAXINT) {
      dist(i,j) = 0;
    }
    else if(dmin == d1) {
      dist(i,j) = d1;
      pathx(i,j) = i-1;
      pathy(i,j) = j-1;
    }
    else if(dmin == d2) {
      dist(i,j) = d2;
      pathx(i,j) = i-1;
      pathy(i,j) = j;
    }
    else if(dmin == d3) {
      dist(i,j) = d3;
      pathx(i,j) = i;
      pathy(i,j) = j-1;
    }
    else if(dmin == d4) {
      dist(i,j) = d4;
      pathx(i,j) = i-2;
      pathy(i,j) = j-1;
    }
    else if(dmin == d5){
      dist(i,j) = d5;
      pathx(i,j) = i-1;
      pathy(i,j) = j-2;
    }
    else                 /* dmin == d6 */ {
     dist(i,j) = d6;
     pathx(i,j) = i-2;
     pathy(i,j) = j-2;
    }
  }
}

n = 0;
for(i=nx, j=ny ; i>0 || j>0 ; i = oi, j = oj) {
  oi = pathx(i, j);
  oj = pathy(i, j);
  di = i - oi;
```

```
  dj = j - oj;

  if(di == 1 && dj == 1) {  /* substitution */
    ralign[n].x1 = x[i-1];
    ralign[n].y1 = y[j-1];
    ralign[n].x2 = 0;
    ralign[n].y2 = 0;
    ralign[n++].d = dist(i, j) - dist(i-1, j-1);
  }

  else if(di == 1 && dj == 0) {  /* deletion */
    ralign[n].x1 = x[i-1];
    ralign[n].y1 = 0;
    ralign[n].x2 = 0;
    ralign[n].y2 = 0;
    ralign[n++].d = dist(i, j) - dist(i-1, j);
  }

  else if(di == 0 && dj == 1) {  /* insertion */
    ralign[n].x1 = 0;
    ralign[n].y1 = y[j-1];
    ralign[n].x2 = 0;
    ralign[n].y2 = 0;
    ralign[n++].d = dist(i, j) - dist(i, j-1);
  }

  else if(dj == 1) {  /* contraction */
    ralign[n].x1 = x[i-2];
    ralign[n].y1 = y[j-1];
    ralign[n].x2 = x[i-1];
    ralign[n].y2 = 0;
    ralign[n++].d = dist(i, j) - dist(i-2, j-1);
  }

  else if(di == 1) {    /* expansion */
    ralign[n].x1 = x[i-1];
    ralign[n].y1 = y[j-2];
    ralign[n].x2 = 0;
    ralign[n].y2 = y[j-1];
    ralign[n++].d = dist(i, j) - dist(i-1, j-2);
  }
  else /* di == 2 && dj == 2 */ { /* melding */
     ralign[n].x1 = x[i-2];
     ralign[n].y1 = y[j-2];
     ralign[n].x2 = x[i-1];
     ralign[n].y2 = y[j-1];
     ralign[n++].d = dist(i, j) - dist(i-2, j-2);
  }
}

*align = (struct alignment *) malloc(n * sizeof(struct alignment));

for(i=0; i<n; i++)
  bcopy(ralign + i, (*align) + (n-i-1), sizeof(struct alignment));

free(distances);
free(path_x);
free(path_y);
free(ralign);
return(n);
```

```
}

/* Returns the area under a normal distribution
   from -inf to z standard deviations */
double
pnorm(z)
     double z;
{
  double t, pd;
  t = 1/(1 + 0.2316419 * z);
  pd = 1 - 0.3989423 *
    exp(-z * z/2) *
      ((((1.330274429 * t - 1.821255978) * t
        + 1.781477937) * t - 0.356563782) * t + 0.319381530) * t;
  /* see Gradsteyn & Rhyzik, 26.2.17 p932 */
  return(pd);
}


/* Return -100 * log probability that an English sentence of length
   len1 is a translation of a foreign sentence of length len2.  The
   probability is based on two parameters, the mean and variance of
   number of foreign characters per English character.
*/

int
match(len1, len2)
     int len1, len2;
{
  double z, pd, mean;

  /* foreign characters per english character */
  double foreign_chars_per_eng_char = 1;

  /* variance per english character */
  double var_per_eng_char = 6.8 ;

  if(len1==0 && len2==0) return(0);
  mean = (len1 + len2/foreign_chars_per_eng_char)/2;
  z = (foreign_chars_per_eng_char * len1 - len2)/sqrt(var_per_eng_char * mean);

  /* Need to deal with both sides of the normal distribution */
  if(z < 0) z = -z;
  pd = 2 * (1 - pnorm(z));

  if(pd > 0) return((int)(-100 * log(pd)));
  else return(BIG_DISTANCE);
}

int
two_side_distance(x1, y1, x2, y2)
     int x1, y1, x2, y2;
{
  int penalty21 = 230;           /* -100 * log([prob of 2-1 match] / [prob of 1-1 match]) */
  int penalty22 = 440;           /* -100 * log([prob of 2-2 match] / [prob of 1-1 match]) */
  int penalty01 = 450;           /* -100 * log([prob of 0-1 match] / [prob of 1-1 match]) */

  if(x2 == 0 && y2 == 0)

    if(x1 == 0)                  /* insertion */
```

```
      return(match(x1, y1) + penalty01);

    else if(y1 == 0)             /* deletion */
      return(match(x1, y1) + penalty01);

    else return (match(x1, y1)); /* substitution */

  else if(x2 == 0)        /* expansion */
    return (match(x1, y1 + y2) + penalty21);

  else if(y2 == 0)        /* contraction */
    return(match(x1 + x2, y1) + penalty21);

  else                    /* melding */
    return(match(x1 + x2, y1 + y2) + penalty22);
}

void
err(msg)
     char *msg;
{
  fprintf(stderr, "**ERROR**: %s0, msg);
  exit(2);
}

/* return the contents of the file as a string
   and stuff the length of this string into len_ptr */
char *
readchars(filename, len_ptr)
     char *filename;
     int *len_ptr;
{
  FILE *fd;
  char *result;
  struct stat stat_buf;

  fd = fopen(filename, "r");
  if(fd == NULL) err("open failed");

  if(fstat(fileno(fd), &stat_buf) == -1)
    err("stat failed");

  *len_ptr = stat_buf.st_size;

  result = malloc(*len_ptr);
  if(result == NULL) err("malloc failed0);

  if(fread(result, sizeof(char), *len_ptr, fd) != *len_ptr)
    err("fread failed");

  if(fclose(fd) == -1)
    err("fclose failed");

  return(result);
}

/* split string into a number of substrings delimited by a delimiter character
   return an array of substrings
   stuff the length of this array into len_ptr */
char **
```

```
substrings(string, end, delimiter, len_ptr)
     char *string, *end, delimiter;
     int *len_ptr;
{
  char *s, **result;
  int i = 0;

  while(string < end && *string == delimiter) string++;

  for(s = string; s < end; s++)
    if(*s == delimiter) i++;
  *len_ptr = i;

  result = (char **)malloc(sizeof(char *) * (i+1));
  if(result == NULL) err("malloc failed");

  i = 0;
  result[i++] = string;
  for(s = string; s < end; s++)
    if(*s == delimiter) {
      result[i++] = s+1;
      *s = 0;
    }
  i--; /*the last entry is beyond the end*/
  if(i != *len_ptr) {
    fprintf(stderr, "align_regions: confusion; i= %d; *len_ptr = %d0, i, *len_ptr);
    exit(2);
  }

  return(result);
}

/* return an array of strings, one string for each line of the file
  set len_ptr to the number of lines in the file */
char **
readlines(filename, len_ptr)
     char *filename;
     int *len_ptr;
{
  char *chars;
  int number_of_chars;
  chars = readchars(filename, &number_of_chars);
  return(substrings(chars, chars + number_of_chars, '0, len_ptr));
}

struct region {
  char **lines;
  int length;
};

void
print_region(fd, region, score)
     int score;
     FILE *fd;
     struct region *region;
{
  char **lines, **end;

  lines = region->lines;
  end = lines + region->length;
```

```
  for( ; lines < end ; lines++)
    fprintf(fd, "%s0, *lines);
}

int
length_of_a_region(region)
     struct region *region;
{
  int result;
  char **lines, **end;

  lines = region->lines;
  end = lines + region->length;
  result = end - lines;

  for( ; lines < end; lines++)
    result += strlen(*lines);
  return(result);
}

int *
region_lengths(regions, n)
     struct region *regions;
     int n;
{
  int i;
  int *result;

  result = (int *)malloc(n * sizeof(int));
  if(result == NULL) err("malloc failed");

  for(i = 0; i < n; i++)
    result[i] = length_of_a_region(regions[i]);
  return(result);
}

struct region *
find_sub_regions(region, delimiter, len_ptr)
     struct region *region;
     char *delimiter;
     int *len_ptr;
{
  struct region *result;
  char **l, **lines, **end;
  int n = 0;

  lines = region->lines;
  end = lines + region->length;

  for(l = lines; l < end; l++)
    if(delimiter && strcmp(*l, delimiter) == 0) n++;

  result = (struct region *)calloc(n+1, sizeof(struct region));
  if(result == NULL) err("malloc failed");
  *len_ptr = n;
  n = 0;
  result[0].lines = lines;
  for(l = lines; l < end; l++)
    if(delimiter && strcmp(*l, delimiter) == 0) {
      result[n].length = l - result[n].lines;
```

```
      result[n+1].lines = l+1;
      n++;
    }
  result[n].length = l - result[n].lines;
  if(n != *len_ptr) {
    fprintf(stderr, "find_sub_regions: n = %d, *len_ptr = %d0, n, *len_ptr);
    exit(2);
  }
  return(result);
}

#define MAX_FILENAME 256

int
main(argc, argv)
     int argc;
     char **argv;
{
  char **lines1, **lines2;
  int number_of_lines1, number_of_lines2;

  struct region *hard_regions1, *hard_regions2, *soft_regions1, *soft_regions2;
  struct region *hard_end1, *hard_end2, tmp;
  int number_of_hard_regions1;
  int number_of_hard_regions2;
  int number_of_soft_regions1;
  int number_of_soft_regions2;

  int *len1, *len2;

  int c, n, i, ix, iy, prevx, prevy;
  struct alignment *align, *a;

  FILE *out1, *out2;
  char filename[MAX_FILENAME];

  extern char *optarg;
  extern int optind;

  while((c = getopt(argc, argv, "vVd:D:")) != EOF)
    switch(c) {
    case 'v':
      verbose = 1;
      break;
    case 'V':
      debug = 1;
      verbose = 1;
      break;
    case 'd':
      soft_delimiter = strdup(optarg);
      break;
    case 'D':
      hard_delimiter = strdup(optarg);
      break;
    default:
      fprintf(stderr, "usage: align_regions [d (soft delimiter)] [D (hard delimiter)]0);
      exit(2);
    }

  if(argc != optind + 2) err("wrong number of arguments");
```

```
sprintf(filename, "%s.al", argv[optind]);
out1 = fopen(filename, "w");
if(out1 == NULL) {
  fprintf(stderr, "can’t open %s0, filename);
  exit(2);
}

sprintf(filename, "%s.al", argv[optind+1]);
out2 = fopen(filename, "w");
if(out2 == NULL) {
  fprintf(stderr, "can’t open %s0, filename);
  exit(2);
}

lines1 = readlines(argv[optind], &number_of_lines1);
lines2 = readlines(argv[optind+1], &number_of_lines2);

tmp.lines = lines1;
tmp.length = number_of_lines1;
hard_regions1 = find_sub_regions(&tmp, hard_delimiter, &number_of_hard_regions1);

tmp.lines = lines2;
tmp.length = number_of_lines2;
hard_regions2 = find_sub_regions(&tmp, hard_delimiter, &number_of_hard_regions2);

if(number_of_hard_regions1 != number_of_hard_regions2) {
  fprintf(stderr, "align_regions: input files do not contain the same number of hard regions (%S).
         hard_delimiter,
         argv[optind], number_of_hard_regions1,
         argv[optind+1], number_of_hard_regions2);
  exit(2);
}

hard_end1 = hard_regions1 + number_of_hard_regions1;
hard_end2 = hard_regions2 + number_of_hard_regions2;

for( ; hard_regions1 < hard_end1 ; hard_regions1++, hard_regions2++) {

  soft_regions1 = find_sub_regions(hard_regions1[0], soft_delimiter, &number_of_soft_regions1);
  soft_regions2 = find_sub_regions(hard_regions2[0], soft_delimiter, &number_of_soft_regions2);

  if(debug){fprintf(out1,"number of soft regions=%d0,number_of_soft_regions1);
           fprintf(out2,"number of soft regions=%d0,number_of_soft_regions2);}

  len1 = region_lengths(soft_regions1, number_of_soft_regions1);
  len2 = region_lengths(soft_regions2, number_of_soft_regions2);

  n = seq_align(len1, len2, number_of_soft_regions1, number_of_soft_regions2, two_side_distance, &

  prevx = prevy = ix = iy = 0;
  for(i = 0; i < n; i++) {
    a = &align[i];
    if(a->x2 > 0) ix++; else if(a->x1 == 0) ix--;
    if(a->y2 > 0) iy++; else if(a->y1 == 0) iy--;
    if(a->x1 == 0 && a->y1 == 0 && a->x2 == 0 && a->y2 == 0) {ix++; iy++;}
    ix++;
    iy++;

    if(debug) {fprintf(out1,"n=%d i=%d x1=%d y1=%d x2=%d y2=%d0,n,i,a->x1,a->y1,a->x2,a->y2);
            fprintf(out2,"n=%d i=%d x1=%d y1=%d x2=%d y2=%d0,n,i,a->x1,a->y1,a->x2,a->y2);}
```

```
      if(verbose) {fprintf(out1, ".Score %d0, a->d);
                fprintf(out2, ".Score %d0, a->d);}

      for( ; prevx < ix; prevx++)
       {if(debug)  fprintf(out1,"ix=%d prevx=%d ",ix,prevx);
       print_region(out1, soft_regions1[prevx], a->d);}
      fprintf(out1, "%s0, soft_delimiter);

      for( ; prevy < iy; prevy++)
       {if(debug) fprintf(out2,"iy=%d prevy=%d ",iy,prevy);
       print_region(out2, soft_regions2[prevy], a->d);}
      fprintf(out2, "%s0, soft_delimiter);
    }
    fprintf(out1, "%s0, hard_delimiter);
    fprintf(out2, "%s0, hard_delimiter);
    free(align);
    free(soft_regions1);
    free(soft_regions2);
    free(len1);
    free(len2);
  }
}
```