A Program for Aligning Sentences in Bilingual Corpora

William A. Gale*
AT&T Bell Laboratories

Kenneth W. Church*
AT&T Bell Laboratories

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 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 method and a program (align) for aligning sentences based on a 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 scaled difference of lengths of the two sentences (in characters) and the variance of this difference. This probabilistic score is used in a dynamic programming framework to find the maximum likelihood alignment of sentences.

It is remarkable that such a simple approach works as well as it does. 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 that 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 more errors on the English–French subcorpus than on the English–German subcorpus, showing that error rates will depend on the corpus considered; however, both were small enough to hope that the method will be useful for many language pairs.

To further research on bilingual corpora, a much larger sample of Canadian Hansards (approximately 90 million words, half in English and and half in French) has been aligned with the align *program and will be available through the Data Collection Initiative of the Association for Computational Linguistics (ACL/DCI). In addition, in order to facilitate replication of the* align *program, an appendix is provided with detailed c-code of the more difficult core of the* align *program.*

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 correspondences between sentences in one

* AT&T Bell Laboratories 600 Mountain Avenue Murray Hill, NJ, 07974

Table 1
Input to alignment program.

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.

language and sentences in the other language. This task is a first step toward the more ambitious task finding correspondences among words.[1]

The input is a pair of texts such as Table 1. The output identifies the alignment between sentences. Most English sentences match exactly one French sentence, but it is possible for an English sentence to match two or more French sentences. The first two English sentences in Table 2 illustrate a particularly hard case where two English sentences align to two French sentences. No smaller alignments are possible because the clause "...sales ...were higher..." in the first English sentence corresponds to (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 a single French sentence. These alignments agreed with the results produced by a human judge.

Aligning sentences is just a first step toward constructing a probabilistic dictionary (Table 3) for use in aligning words in machine translation (Brown et al. 1990), or for constructing a bilingual concordance (Table 4) for use in lexicography (Klavans and Tzoukermann 1990).

Although there has been some previous work on the sentence alignment (e.g., Brown, Lai, and Mercer 1991 [at IBM], Kay and Röscheisen [this issue; at Xerox], and Catizone, Russell, and Warwick, in press [at ISSCO], 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 computationally prohibitive.

Most of the previous work on sentence alignment has yet to be published. Kay's draft (Kay and Röscheisen; this issue), for example, was written more than two years ago and is still unpublished. Similarly the IBM work is also several years old, but not

1 In statistics, string-matching problems are divided into two classes: *alignment* problems and *correspondence* problems. Crossing dependencies are possible in the latter, but not in the former.

Table 2
Output from alignment program.

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.

Table 3
An entry in a probabilistic dictionary.

(from Brown et al. 1990)		
English	French	Prob (French|English)
the	le	0.610
the	la	0.178
the	l'	0.083
the	les	0.023
the	ce	0.013
the	il	0.012
the	de	0.009
the	à	0.007
the	que	0.007

very well documented in the published literature; consequently, there has been a lot of unnecessary subsequent work at ISSCO and elsewhere.[2]

The method we describe has the same sentence-length basis as does that of Brown, Lai, and Mercer, while the two differ considerably from the lexical approaches tried by Kay and Röscheisen and by Catizone, Russell, and Warwick.

The feasibility of other methods has varied greatly. Kay's approach is apparently quite slow. At least, with the currently inefficient implementation, it might take hours

2 After we finished most of this work, it came to our attention that the IBM MT group has at least four papers that mention sentence alignment. (Brown et al. 1988a,b) start from a set of aligned sentences, suggesting that they had a solution to the sentence alignment problem back in 1988. Brown et al. (1990) mention that sentence lengths formed the basis of their method. The draft by Brown, Lai, and Mercer (1991) describes their process without giving equations.

Table 4
A bilingual concordance.

bank/banque ("money" sense)	
¿ it could also be a place where we would have a ftre le lieu où se retrouverait une espèce de	bank of experts. SENT i know several people who a banque d' experts. SENT je connais plusieurs pers
f finance (mr. wilson) and the governor of the es finances (m . wilson) et le gouverneur de la	bank of canada have frequently on behalf of the ca banque du canada ont fréquemment utilisé au co
reduced by over 800 per cent in one week through us de 800 p. 100 en une semaine à cause d'une	bank action. SENT there was a haberdasher who wou banque. SENT voilà un chemisier qui aurait appr
bank/banc ("place" sense)	
h a forum. SENT such was the case in the georges entre les états-unis et le canada à propos du	bank issue which was settled between canada and th banc de george. SENT c'est dans le but de ré
han i did. SENT he said the nose and tail of the gouvernement avait cédé les extrémités du	bank were surrendered by this government. SENT th banc. SENT en fait, lors des négociations de l
he fishing privileges on the nose and tail of the les privilèges de pêche aux extrémités du	bank went down the tube before we even negotiated banc ont été liquidés avant même qu' on ai

to align a single Scientific American article (Kay, personal communication). It ought to be possible to achieve fairly reasonable results with much less computation. The IBM algorithm is much more efficient since they were able to extract nearly 3 million pairs of sentences from Hansard materials in 10 days of running time on an IBM Model 3090 mainframe computer with access to 16 megabytes of virtual memory (Brown, Lai, and Mercer 1991).

The evaluation of results has been absent or rudimentary. Kay gives positive examples of the alignment process, but no counts of error rates. Brown, Lai, and Mercer (1991) report that they achieve a 0.6% error rate when the algorithm suggests aligning one sentence with one sentence. However, they do not characterize its performance overall or on the more difficult cases.

Since the research community has not had access to a practical sentence alignment program, we thought that it would be helpful to describe such a program (*align*) and to evaluate its results. In addition, a large sample of Canadian Hansards (approximately 90 million words, half in French and half in English) has been aligned with the *align* program and has been made available to the general research community through the Data Collection Initiative of the Association for Computational Linguistics (ACL/DCI). In order to facilitate replication of the *align* program, an appendix is provided with detailed c-code of the more difficult core of the *align* program.

The *align* program is 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 (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 that 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 more errors on the English–French subcorpus than on the English–German subcorpus, showing that error rates will depend on the corpus considered; however, both were small enough for us to hope that the method will be useful for many language pairs. We believe that the error rate is considerably lower in the Canadian Hansards because the translations are more literal.

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. On a corpus this small the paragraphs could have been aligned by hand. It turns out that "pseudo-paragraphs" usually have fewer than 50 characters and that real paragraphs usually have more than 100 characters. We used this fact to align the paragraphs automatically, checking the result by hand.

The procedure correctly aligned all of the English and German paragraphs. However, one of the French documents was badly translated and could not be aligned because of the omission of one long paragraph and the duplication of a short one. This document was excluded for the purposes of the remainder of this experiment.

We will show below that paragraph alignment is an important step, so it is fortunate that it is not particularly difficult. In aligning the Hansards, we found that paragraphs were often already aligned. For robustness, we decided to align paragraphs within certain fairly reliable regions (denoted by certain Hansard-specific formatting conventions) using the same method as that described below for aligning sentences within each paragraph.

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.[3] 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. The fol-

3 We will have little to say about how sentence boundaries are identified. Identifying sentence boundaries is not always as easy as it might appear for reasons described in Liberman and Church (in press). It would be much easier if periods were always used to mark sentence boundaries; but unfortunately, many periods have other purposes. In the Brown Corpus, for example, only 90% of the periods are used to mark sentence boundaries; the remaining 10% appear in numerical expressions, abbreviations, and so forth. In the *Wall Street Journal*, there is even more discussion of dollar amounts and percentages, as well as more use of abbreviated titles such as *Mr.*; consequently, only 53% of the periods in the *Wall Street Journal* are used to identify sentence boundaries. For the UBS data, a simple set of heuristics were used to identify sentences boundaries. The dataset was sufficiently small that it was possible to correct the remaining mistakes by hand. For a larger dataset, such as the Canadian Hansards, it was not possible to check the results by hand. We used the same procedure that is used in Church (1988). This procedure was developed by Kathryn Baker (unpublished).

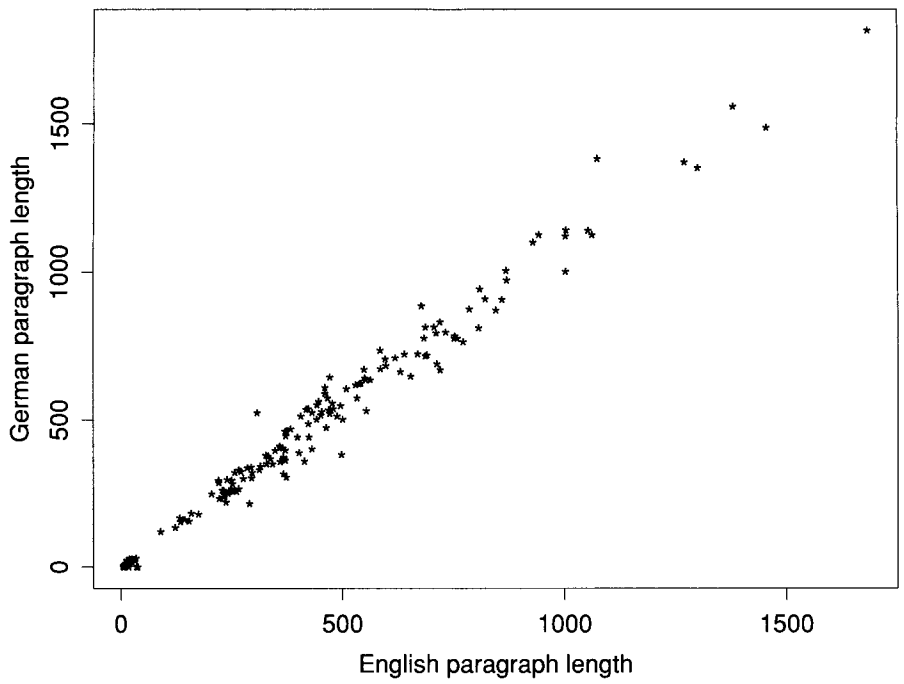

Figure 1
Paragraph lengths are highly correlated. The horizontal axis shows the length of English paragraphs, while the vertical scale shows the lengths of the corresponding German paragraphs. Note that the correlation is quite large (.991).

lowing striking figure could easily lead one to this approach. Figure 1 shows that the lengths (in characters) of English and German paragraphs are highly correlated (.991).

Dynamic programming is often used to align two sequences of symbols in a variety of settings, such as genetic code sequences from different species, speech sequences from different speakers, gas chromatograph sequences from different compounds, and geologic sequences from different locations (Sankoff and Kruskal 1983). We could expect these matching techniques to be useful, as long as the order of the sentences does not differ too radically between the two languages. 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. We have found that the sentence alignment problem fits fairly well into this framework, though it is necessary to introduce a fairly interesting innovation into the structure of the distance measure.

Kruskal and Liberman (1983) describe distance measures as belonging to one of two classes: *trace* and *time-warp*. The difference becomes important when a single element of one sequence is being matched with multiple elements from the other. In trace applications, such as genetic code matching, the single element is matched with just one of the multiple elements, and all of the others will be ignored. In contrast, in time-warp applications such as speech template matching, the single element is matched with each of the multiple elements, and the single element will be used in multiple matches. Interestingly enough, our application does not fit into either of

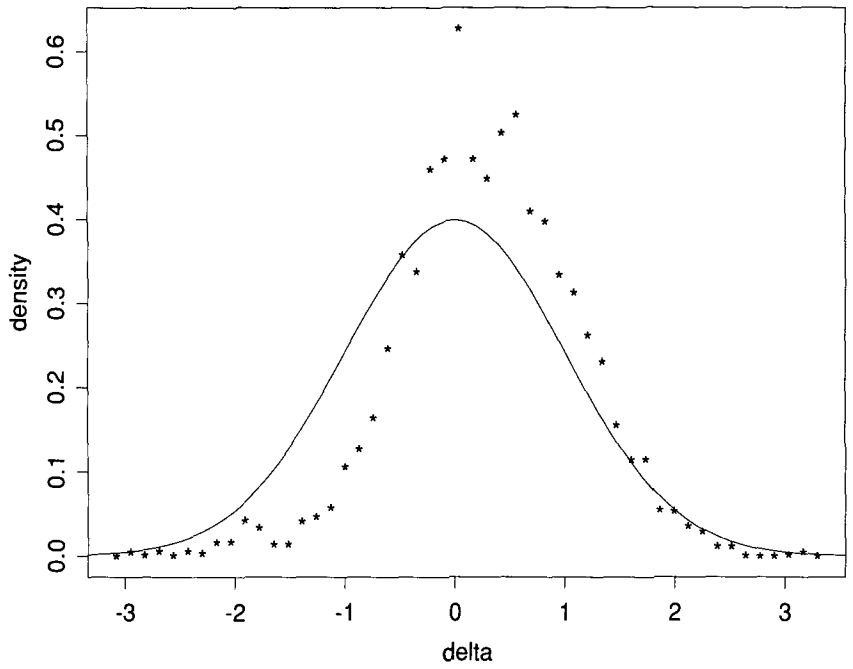

Figure 2
Delta is approximately normal. The horizontal axis shows δ, while the vertical scale shows the empirical density of delta for the hand-aligned regions as points, and a normal $(0, 1)$ density plot (lines) for comparison. The empirical density is slightly more peaked than normal (and its mean is not quite zero), but the differences are small enough for the purposes of the algorithm.

Kruskal and Liberman's classes because our distance measure needs to compare the single element with an aggregate of the multiple elements.

4. The Distance Measure

It is convenient for the distance measure to be based on a probabilistic model so that information can be combined in a consistent way. Our distance measure is an estimate of $-\log Prob(match \mid \delta)$, where δ depends on l_1 and l_2, the lengths of the two portions of text under consideration. The log is introduced here so that adding distances will produce desirable results.

This distance measure is based on the assumption that each character in one language, L_1, gives rise to a random number of characters in the other language, L_2. We assume these random variables are independent and identically distributed with a normal distribution. The model is then specified by the mean, c, and variance, s^2, of this distribution. c is the expected number of characters in L_2 per character in L_1, and s^2 is the variance of the number of characters in L_2 per character in L_1. We define δ to be $(l_2 - l_1 c)/\sqrt{l_1 s^2}$ so that it has a normal distribution with mean zero and variance one (at least when the two portions of text under consideration actually do happen to be translations of one another).

Figure 2 is a check on the assumption that δ is normally distributed. The figure is constructed using the parameters c and s^2 estimated for the program.

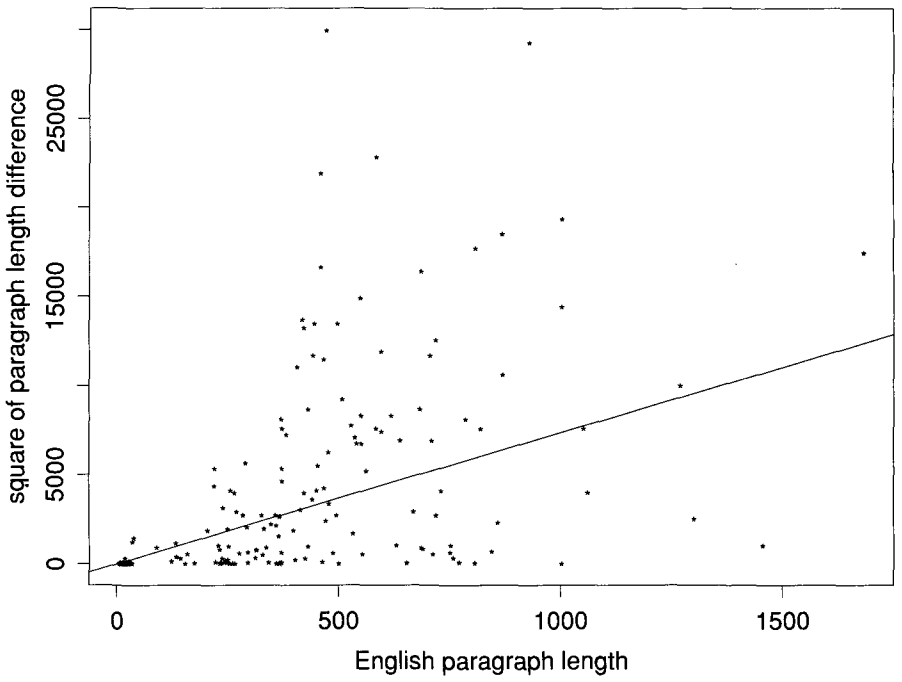

Figure 3
Variance is modeled proportional to length. The horizontal axis plots the length of English paragraphs, while the vertical axis shows the square of the difference of English and German lengths, an estimate of variance. The plot indicates that variance increases with length, as predicted by the model. The line shows the result of a robust regression analysis. Five extreme points lying above the top of this figure have been suppressed since they did not contribute to the robust regression.

The parameters c and s^2 are determined empirically from the UBS data. We could estimate c by counting the number of characters in German paragraphs then dividing by the number of characters in corresponding English paragraphs. We obtain $81105/73481 \approx 1.1$. The same calculation on French and English paragraphs yields $c \approx 72302/68450 \approx 1.06$ as the expected number of French characters per English character. As will be explained later, performance does not seem to be very sensitive to these precise language-dependent quantities, and therefore we simply assume the language-independent value $c \approx 1$, which simplifies the program considerably. This value would clearly be inappropriate for English–Chinese alignment, but it seems likely to be useful for most pairs of European languages.

s^2 is estimated from Figure 3. The model assumes that s^2 is proportional to length. The constant of proportionality is determined by the slope of the robust regression line shown in the figure. The result for English–German is $s^2 = 7.3$, and for English–French is $s^2 = 5.6$. Again, we will see that the difference in the two slopes is not too important. Therefore, we can combine the data across languages, and adopt the simpler language-independent estimate $s^2 \approx 6.8$, which is what is actually used in the program.

We now appeal to Bayes Theorem to estimate $Prob(match \mid \delta)$ as a constant times $Prob(\delta \mid match)\ Prob(match)$. The constant can be ignored since it will be the same for

Table 5
Prob(match)

Category	Frequency	Prob(match)
1-1	1167	0.89
1-0 or 0-1	13	0.0099
2-1 or 1-2	117	0.089
2-2	15	0.011
	1312	1.00

all proposed matches. The conditional probability $Prob(\delta \mid match)$ can be estimated by

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

where $Prob(|\delta|)$ is 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,

$$Prob(\delta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\delta} e^{-z^2/2} dz.$$

The program computes δ directly from the lengths of the two portions of text, l_1 and l_2, and the two parameters, c and s^2. That is, $\delta = (l_2 - l_1 c)/\sqrt{l_1 s^2}$. Then, $Prob(|\delta|)$ is computed by integrating a standard normal distribution (with mean zero and variance one). Many statistics textbooks include a table for computing this. The code in the appendix uses the *pnorm* function, which is based on an approximation described by Abramowitz and Stegun (1964; p. 932, equation 26.2.17).

The prior probability of a match, $Prob(match)$, is fit with the values in Table 5, which were determined from the hand-marked UBS data. We have found that a sentence in one language normally matches exactly one sentence in the other language (1-1). Three additional possibilities are also considered: 1-0 (including 0-1), 2-1 (including 1-2), and 2-2. Table 5 shows all four possibilities.

This completes the discussion of the distance measure. $Prob(match \mid \delta)$ is computed as an (irrelevant) constant times $Prob(\delta \mid match)Prob(match)$. $Prob(match)$ is computed using the values in Table 5. $Prob(\delta \mid match)$ is computed by assuming that $Prob(\delta \mid match) = 2(1 - Prob(|\delta|))$, where $Prob(|\delta|)$ has a standard normal distribution. We first calculate δ as $(l_2 - l_1 c)/\sqrt{l_1 s^2}$ and then $Prob(|\delta|)$ is computed by integrating a standard normal distribution. See the c-function *two_side_distance* in the appendix for an example of a c-code implementation of these calculations.

The distance function d, represented in the program as *two_side_distance*, is defined in a general way to allow for insertions, deletion, substitution, etc. The function takes four arguments: x_1, y_1, x_2, y_2.

1. Let $d(x_1, y_1; 0, 0)$ be the cost of substituting x_1 with y_1,
2. $d(x_1, 0; 0, 0)$ be the cost of deleting x_1,
3. $d(0, y_1; 0, 0)$ be the cost of insertion of y_1,
4. $d(x_1, y_1; x_2, 0)$ be the cost of contracting x_1 and x_2 to y_1,

5. $d(x_1, y_1; 0, y_2)$ be the cost of expanding x_1 to y_1 and y_2, and
6. $d(x_1, y_1; x_2, y_2)$ be the cost of merging x_1 and x_2 and matching with y_1 and y_2.

5. The Dynamic Programming Algorithm

The algorithm is summarized in the following recursion equation. Let s_i, $i = 1 \cdots I$, be the sentences of one language, and t_j, $j = 1 \cdots J$, be the translations of those sentences in the other language. Let d be the distance function described in the previous section, and let $D(i,j)$ be the minimum distance between sentences $s_1, \cdots s_i$ and their translations $t_1, \cdots t_j$, under the maximum likelihood alignment. $D(i,j)$ is computed by minimizing over six cases (substitution, deletion, insertion, contraction, expansion, and merger) which, in effect, impose a set of slope constraints. That is, $D(i,j)$ is defined by the following recurrence with the initial condition $D(i,j) = 0$.

$$D(i,j) = \min \begin{cases} D(i,j-1) + d(0, t_j; 0, 0) \\ D(i-1,j) + d(s_i, 0; 0, 0) \\ D(i-1,j-1) + d(s_i, t_j; 0, 0) \\ D(i-1,j-2) + d(s_i, t_j; 0, t_{j-1}) \\ D(i-2,j-1) + d(s_i, t_j; s_{i-1}, 0) \\ D(i-2,j-2) + d(s_i, t_j; s_{i-1}, t_{j-1}) \end{cases}$$

6. Evaluation

To evaluate *align*, its results were compared with a human alignment. All of the UBS sentences were aligned by a primary judge, a native speaker of English with a reading knowledge of French and German. Two additional judges, a native speaker of French and a native speaker of German, respectively, were used to check the primary judge on 43 of the more difficult paragraphs having 230 sentences (out of 118 total paragraphs with 725 sentences). Both of the additional judges were also fluent in English, having spent the last few years living and working in the United States, though they were both more comfortable with their native language than with English.

The materials were prepared in order to make the task somewhat less tedious for the judges. Each paragraph was printed in three columns, one for each of the three languages: English, French, and German. Blank lines were inserted between sentences. The judges were asked to draw lines between matching sentences. The judges were also permitted to draw a line between a sentence and "null" if they thought that the sentence was not translated. For the purposes of this evaluation, two sentences were defined to "match" if they shared a common clause. (In a few cases, a pair of sentences shared only a phrase or a word, rather than a clause; these sentences did not count as a "match" for the purposes of this experiment.)

After checking the primary judge with the other two judges, it was decided that the primary judge's results were sufficiently reliable that they could be used as a standard for evaluating the program. The primary judge made only two mistakes on the 43 hard paragraphs (one French mistake and one German mistake), whereas the program made 44 errors on the same materials. Since the primary judge's error rate is so much lower than that of the program, it was decided that we needn't be concerned with the primary judge's error rate. If the program and the judge disagree, we can assume that the program is probably wrong.

The 43 "hard" paragraphs were selected by looking for sentences that mapped to something other than themselves after going through both German and French.

Table 6
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	—	1	1	100

Specifically, for each English sentence, we attempted to find the corresponding German sentences, and then for each of them, we attempted to find the corresponding French sentences, and then we attempted to find the corresponding English sentences, which should hopefully get us back to where we started. The 43 paragraphs included all sentences in which this process could not be completed around the loop. This relatively small group of paragraphs (23% of all paragraphs) contained a relatively large fraction of the program's errors (82%). Thus, there seems to be some verification that this trilingual criterion does in fact succeed in distinguishing more difficult paragraphs from less difficult ones.

There are three pairs of languages: English–German, English–French, and French–German. We will report on just the first two. (The third pair is probably dependent on the first two.) Errors are reported with respect to the judge's responses. That is, for each of the "matches" that the primary judge found, we report the program as correct if it found the "match" and incorrect if it didn't. This procedure is better than comparing on the basis of alignments proposed by the algorithm for two reasons. First, it makes the trial "blind," that is, the judge does not know the algorithm's result when judging. Second, it allows comparison of results for different algorithms on a common basis.

The program made 36 errors out of 621 total alignments (5.8%) for English–French and 19 errors out of 695 (2.7%) alignments for English–German. Overall, there were 55 errors out of a total of 1316 alignments (4.2%). The higher error rate for English–French alignments may result from the German being the original, so that the English and German differ by one translation, while the English and French differ by two translations.

Table 6 breaks down the errors by category, illustrating that complex matches are more difficult. 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 are not even considered by the algorithm, so naturally all three instances of these are counted as errors. The most embarrassing category is 1-0, which was never handled correctly. In addition, when the algorithm assigns a sentence to the 1-0 category, it is also always wrong. Clearly, more work is needed to deal with the 1-0 category. It may be necessary to consider language-specific methods in order to deal adequately with this case.

Since the algorithm achieves substantially better performance on the 1-1 regions, one interpretation of these results is that the overall low error rate is due to the high frequency of 1-1 alignments in English–French and English–German translations.

Table 7
The distance measure is the best predictor of errors.

Variable	Coef.	Std. Dev.	Coef./Std. Dev.
Distance Measure	.071	.011	6.5
Category Type	.52	.47	1.1
Paragraph Length	.0003	.0005	0.6
Sentence Length	.0013	.0029	0.5

Translations to linguistically more different languages, such as Hebrew or Japanese, might encounter a higher proportion of hard matches.

We investigated the possible dependence of the error rate on four variables:

1. Sentence Length
2. Paragraph Length
3. Category Type
4. Distance Measure.

We used logistic regression (Hosmer and Lemeshow 1989) to see how well each of the four variables predicted the errors. The coefficients and their standard deviations are shown in Table 7. Apparently, the distance measure is the most useful predictor, as indicated by the last column. In fact, none of the other three factors was found to contribute significantly beyond the effect of the distance measure, indicating that the distance measure is already doing an excellent job, and we should not expect much improvement if we were to try to augment the measure to take these additional factors into account.

The fact that the score is such a good predictor of performance can be used to extract a large subcorpus that has a much smaller error rate. By selecting the best scoring 80% of the alignments, the error rate can be reduced from 4% to 0.7%. In general, we can trade off the size of the subcorpus and the accuracy by setting a threshold, and rejecting alignments with a score above this threshold. Figure 4 examines this trade-off in more detail.

Less formal tests of the error rate in the Hansards suggest that the overall error rate is about 2%, while the error rate for the easy 80% of the sentences is about 0.4%. Apparently the Hansard translations are more literal than the UBS reports. It took 20 hours of real time on a sun 4 to align 367 days of Hansards, or 3.3 minutes per Hansard-day. The 367 days of Hansards contained about 890,000 sentences or about 37 million "words" (tokens). About half of the computer time is spent identifying tokens, sentences, and paragraphs, and about half of the time is spent in the *align* program itself.

The overall error, 4.2%, that we get on the UBS corpus is considerably higher than the 0.6% error reported by Brown, Lai, and Mercer (1991). However, a direct comparison is misleading because of the differences in corpora and the differences in sampling. We have observed that the Hansards are much easier than the UBS. Our error rate drops by about 50% in that case. Aligning the UBS French and English texts is more difficult than aligning the English and German, because the French and English

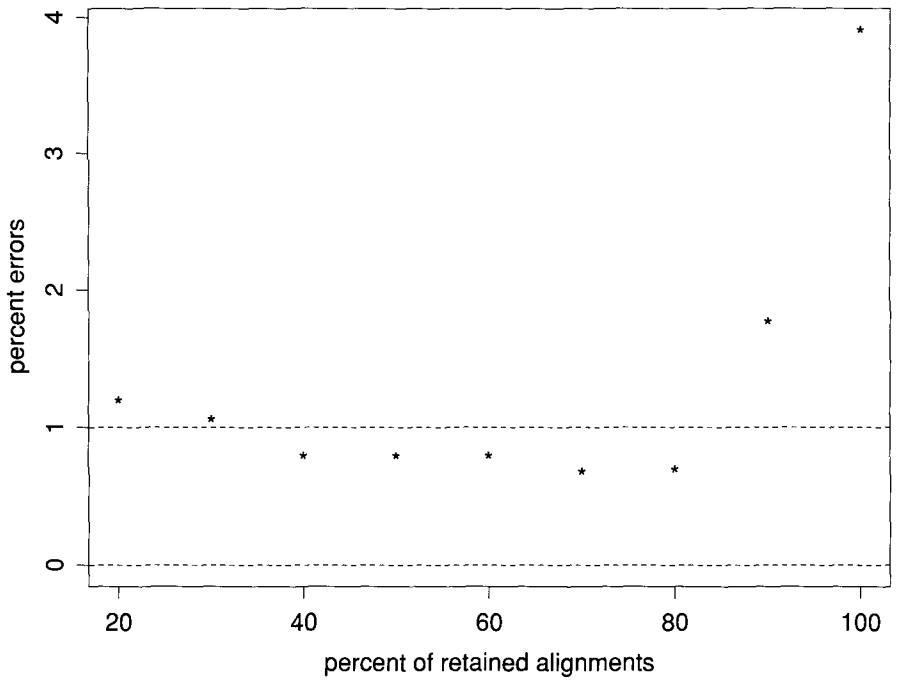

Figure 4
Extracting a subcorpus with lower error rate. The fact that the score is such a good predictor of performance can be used to extract a large subcorpus that has a much smaller error rate. In general, we can trade off the size of the subcorpus and the accuracy by setting a threshold and rejecting alignments with a score above this threshold. The horizontal axis shows the size of the subcorpus, and the vertical axis shows the corresponding error rate. An error rate of about 2/3% can be obtained by selecting a threshold that would retain approximately 80% of the corpus.

versions are separated by two translations, both being translations of the German original. In addition, IBM samples only the 1-1 alignments, which are much easier than any other category, as one can see from Table 6.

Given these differences in testing methodology as well as the differences in the algorithms, we find the methods giving broadly similar results. Both methods give results with sufficient accuracy to use the resulting alignments, or selected portions thereof, for acquisition of lexical information. And neither method achieves human accuracy on the task. (Note that one difference between their method and ours is that they never find 2-2 alignments. This would give their method a minimum overall error rate of 1.4% on the UBS corpus, three times the human error rate on hard paragraphs.) We conclude that a sentence alignment method that achieves human accuracy will need to have lexical information available to it.

7. Variations and Extensions

7.1 Measuring Length in Terms Of Words Rather than Characters

It is interesting to consider what happens if we change our definition of length to count words rather than characters. It might seem that a word is a more natural linguistic unit than a character. However, we have found that words do not perform as well as

characters. In fact, the "words" variation increases the number of errors dramatically (from 36 to 50 for English–French and from 19 to 35 for English–German). The total errors were thereby increased from 55 to 85, or from 4.2% to 6.5%.

We believe that characters are better because there are more of them, and therefore there is less uncertainty. On the average, there are 117 characters per sentence (including white space) and only 17 words per sentence. Recall that we have modeled variance as proportional to sentence length, $V(l) = s^2 l$. Using the character data, we found previously that $s^2 \approx 6.5$. The same argument applied to words yields $s^2 \approx 1.9$. For comparison's sake, it is useful to consider the ratio of $\sqrt{V(m)}/m$ (or equivalently, $s/\sqrt{m}$), where m is the mean sentence length. We obtain $\sqrt{V(m)}/m$ ratios of 0.22 for characters and 0.33 for words, indicating that characters are less noisy than words, and are therefore more suitable for use in *align*.

Although Brown, Lai, and Mercer (1991) used lengths measured in words, comparisons of error rates between our work and theirs will not test whether characters or words are more useful. As set out in the previous section, there are numerous differences in testing methodology and materials. Furthermore, there are apparently many differences between the IBM algorithm and ours other than the units of measurement, which could also account for any difference on performance. Appropriate methodology is to compare methods with only one factor varying, as we do here.

7.2 Ignoring Paragraph Boundaries

Recall that *align* is a two-step process. First, paragraph boundaries are identified and then sentences are aligned within paragraphs. We considered eliminating the first step and found a threefold degradation in performance. 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 the two-step approach reduces errors by a factor of three. It is possible that performance might be improved further still by introducing additional alignment steps at the clause and/or phrase levels, but testing this hypothesis would require access to robust parsing technology.

7.3 Adding a 2-2 Category

The original version of the program did not consider the category of 2-2 alignments. Table 6 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 better than chance, since there is less than 1% chance of getting 10 or more heads out of 12 flips of a fair coin. Thus it is worthwhile to include the 2-2 alignment possibility.

7.4 Using More Accurate Parameter Estimates

When we discussed the estimation of the model parameters, c and s^2, we mentioned that it is possible to fit the parameters more accurately if we estimate different values for each language pair, but that doing so did not seem to increase performance by very much. In fact, we found exactly the same total number of errors, although the errors are slightly different. Changing the parameters resulted in four changes to the output for English–French (two right and two wrong), and two changes to the output for English–German (one right and one wrong). Since it is more convenient to use language-independent parameter values, and doing so doesn't seem to hurt performance very much (if at all), we have decided to adopt the language-independent values.

7.5 Extensions

7.5.1 Hard and Soft Boundaries. Recall that we rejected one of the French documents because one paragraph was omitted and two paragraphs were duplicated. We could have handled this case if we had employed a more powerful paragraph alignment algorithm. In fact, in aligning the Canadian Hansards, we found that it was necessary to do something more elaborate than we did for the UBS data. We decided to use more or less the same procedure for aligning paragraphs within a document as the procedure that we used for aligning sentences within a paragraph. Let us introduce the distinction between hard and soft delimiters. The alignment program is defined to move soft delimiters as necessary within the constraints of the hard delimiters. Hard delimiters cannot be modified, and there must be equal numbers of them. When aligning sentences within a paragraph, the program considers paragraph boundaries to be "hard" and sentence boundaries to be "soft." When aligning paragraphs within a document, the program considers document boundaries to be "hard" and paragraph boundaries to be "soft." This entension has been incorporated into the implementation presented in the appendix.

7.5.2 Augmenting the Dictionary Function to Consider Words. Many alternative alignment procedures such as Kay and Röscheisen (unpublished) make use of words. It ought to help to know that the English string "house" and the French string "maison" are likely to correspond. Dates and numbers are perhaps an even more extreme example. It really ought to help to know that the English string "1988" and the French string "1988" are likely to correspond. We are currently exploring ways to integrate these kinds of clues into the framework described above. However, at present, the algorithm does not have access to lexical constraints, which are clearly very important. We expect that once these clues are properly integrated, the program will achieve performance comparable to that of the primary judge. However, we are still not convinced that it is necessary to process these lexical clues, since the current performance is sufficient for many applications, such as building a probabilistic dictionary. It is remarkable just how well we can do without lexical constraints. Adding lexical constraints might slow down the program and make it less useful as a first pass.

8. Conclusions

This paper has proposed a method for aligning sentences in a bilingual corpus, based on a simple probabilistic model, described in Section 3. The model was motivated by the observation that longer regions of text tend to have longer translations, and that shorter regions of text tend to have shorter translations. In particular, we found that the correlation between the length of a paragraph in characters and the length of its translation was extremely high (0.991). This high correlation suggests that length might be a strong clue for sentence alignment.

Although this method is extremely simple, it is also quite accurate. Overall, there was a 4.2% error rate on 1316 alignments, averaged over both English–French and English–German data. In addition, we find that the probability score is a good predictor of accuracy, and consequently, it is possible to select a subset of 80% of the alignments with a much smaller error rate of only 0.7%.

The method is also fairly language-independent. Both English–French and English–German data were processed using the same parameters. If necessary, it is possible to fit the six parameters in the model with language-specific values, though, thus far, we have not found it necessary to do so.

We have examined a number of variations. In particular, we found that it is better to use characters rather than words in counting sentence length. Apparently, the performance is better with characters because there is less variability in the differences of sentence lengths so measured. Using words as units increases the error rate by half, from 4.2% to 6.5%.

In the future, we would hope to extend the method to make use of lexical constraints. However, it is remarkable just how well we can do without such constraints. We might advocate our simple character alignment procedure as a first pass, even to those who advocate the use of lexical constraints. Our procedure would complement a lexical approach quite well. Our method is quick but makes a few percent errors; a lexical approach is probably slower, though possibly more accurate. One might go with our approach when the scores are small, and back off to a lexical-based approach as necessary.

Acknowledgments

We thank Susanne Wolff and Evelyne Tzoukermann for their pains in aligning sentences. Susan Warwick provided us with the UBS trilingual corpus and convinced us to work on the sentence alignment problem.

References

Abramowitz, M., and Stegun, I. (1964). *Handbook of Mathematical Functions.* US Government Printing Office.

Brown, P.; Cocke, J.; Della Pietra, S.; Della Pietra, V.; Jelinek, F.; Mercer, R.; and Roossin, P. (1988a). "A statistical approach to French/English translation." In *Proceedings, RIAO88 Conference.* Cambridge, MA.

Brown, P.; Cocke, J.; Della Pietra, S.; Della Pietra, V.; Jelinek, F.; Mercer, R.; and Roossin, P. (1988b). "A statistical approach to language translation." In *Proceedings, 13th International Conference on Computational Linguistics (COLING-88).* Budapest, Hungary.

Brown, P.; Cocke, J.; Della Pietra, S.; Della Pietra, V.; Jelinek, F.; Lafferty, J.; Mercer, R.; and Roossin, P. (1990). "A statistical approach to machine translation." *Computational Linguistics,* **16**, 79–85.

Brown, P.; Lai, J.; and Mercer, R. (1991). "Aligning sentences in parallel corpora." In *Proceedings, 47th Annual Meeting of the Association for Computational Linguistics.*

Catizone, R.; Russell, G.; and Warwick, S. (in press). "Deriving translation data from bilingual texts." In *Lexical Acquisition: Using on-line Resources to Build a Lexicon,* edited by Zernik. Lawrence Erlbaum.

Church, K. (1988). "A stochastic parts program and noun phrase parser for unrestricted text." In *Proceedings, Second Conference on Applied Natural Language Processing.* Austin, TX.

Hosmer, D., and Lemeshow, S. (1989). *Applied Logistic Regression.* Wiley.

Klavans, J., and Tzoukermann, E. (1990). "The BICORD system." In *Proceedings, 15th International Conference on Computational Linguistics (COLING-90),* 174–179.

Kay, M., and Röscheisen, M. (1988). "Text-translation alignment." Xerox Palo Alto Research Center.

Kruskal, J., and Liberman, M. (1983). "The symmetric time-warping problem: From continuous to discrete." In *Time Warps, String Edits, and Macro Molecules: The Theory and Practice of Sequence Comparison,* edited by D. Sankoff and J. Kruskal. Addison-Wesley.

Liberman, M., and Church, K. (in press). "Text analysis and word pronunciation in text-to-speech synthesis." In *Advances in Speech Signal Processing,* edited by S. Furui and M. Sondhi.

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

Appendix: Program

with Michael D. Riley

The following code is the core of *align*. It is a C language program that inputs two text files, with one token (word) per line. The text files contain a number of delimiter tokens. There are two types of delimiter tokens: "hard" and "soft." The hard regions (e.g., paragraphs) may not be changed, and there must be equal numbers of them in the two input files. The soft regions (e.g., sentences) may be deleted (1-0), inserted (0-1), substituted (1-1), contracted (2-1), expanded (1-2), or merged (2-2) as necessary so that the output ends up with the same number of soft regions. The program generates two output files. The two output files contain an equal number of soft regions, each on a line. If the -v command line option is included, each soft region is preceded by its probability score.

```
#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

  hard regions are delimited by the -D arg
  soft regions are delimited by the -d arg
  */
#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 MAX_FILENAME 256
#define BIG_DISTANCE 2500

/* Dynamic Programming Optimization */
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 */
/* utility functions */
```

```
char *readchars(), **readlines(), **substrings();
void err();

/*
  seq_align by Mike Riley

  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);
}
/* Local Distance Function */

/* 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 Abramowitz, M., and I. Stegun (1964), 26.2.17 p. 932 */
  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;
  double c = 1;
  double s2 = 6.8 ;

  if(len1==0 && len2==0) return(0);
  mean = (len1 + len2/c)/2;
  z = (c * len1 - len2)/sqrt(s2 * 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                          /* merger */
    return(match(x1 + x2, y1 + y2) + penalty22);
}
/* Functions for Manipulating Regions */

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, "%s\n", *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 = %d\n", n,
    *len_ptr);
    exit(2);
  }
  return(result);
}
/* Top Level Main Function */

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;

  /* parse arguments */
  while((c = getopt(argc, argv, "vd:D:")) != EOF)
    switch(c) {
    case 'v':
      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)]\n");
    exit(2);
  }
if(argc != optind + 2) err("wrong number of arguments");
/* open output files */
sprintf(filename, "%s.al", argv[optind]);
out1 = fopen(filename, "w");
if(out1 == NULL) {
  fprintf(stderr, "can't open %s\n", filename);
  exit(2);
}
sprintf(filename, "%s.al", argv[optind+1]);
out2 = fopen(filename, "w");
if(out2 == NULL) {
  fprintf(stderr, "can't open %s\n", 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)
  err("align_regions: input files do not contain the
                                 same number of hard regions");

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);

  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, &align);

    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(verbose) {
 fprintf(out1, ".Score %d\n", a->d);
  fprintf(out2, ".Score %d\n", a->d);
      }
      for( ; prevx < ix; prevx++)
        print_region(out1, soft_regions1[prevx], a->d);
      fprintf(out1, "%s\n", soft_delimiter);

      for( ; prevy < iy; prevy++)
        print_region(out2, soft_regions2[prevy], a->d);
      fprintf(out2, "%s\n", soft_delimiter);
    }
    fprintf(out1, "%s\n", hard_delimiter);
    fprintf(out2, "%s\n", hard_delimiter);
    free(align);
    free(soft_regions1);
    free(soft_regions2);
    free(len1);
    free(len2);
  }
}
/* Utility Functions */

void
err(msg)
     char *msg;
{
  fprintf(stderr, "**ERROR**: %s\n", 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 failed\n");

  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 = %d\n", 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, '\n', len_ptr));
}
```