Alignment of Low Information Sequences

David R. Powell, Lloyd Allison, Trevor I. Dix, David L. Dowe

Department of Computer Science, Monash University, Clayton,
Vic. 3168, Australia
e-mail: {powell, lloyd, trevor, dld}@cs.monash.edu.au

Abstract. Alignment of two random sequences over a fixed alphabet can be shown to be optimally done by a Dynamic Programming Algorithm (DPA). It is normally assumed that the sequences are random and incompressible and that one sequence is a mutation of the other. However, DNA and many other sequences are not always random and unstructured, and the issue arises as how to best align compressible sequences.
Assuming our sequences to be non-random and to emanate from mutations of a first order Markov model, we note that alignment of high information regions is more important than alignment of low information regions and arrive at a new alignment method for low information sequences which performs better than the standard DPA for data generated from mutations of a first order Markov model.

Keywords: Sequence Alignment, DNA, Biology, Information Theory.

1 Introduction

Sequence alignment shows how two (or more) sequences could be related. Alignments have a number of important applications in biology including sequence assembly [14, 15, 8], construction of evolutionary trees [6] and comparison of protein secondary structure [16, 17, 7]. Aligning a pair of sequences involves matching characters from the two sequences either with each other or the null character '-', to indicate an insertion or deletion. The following is an example of aligning the sequences `ATACTAGA` and `AACTTGGA`.

```
                          ATACTAG-A
                          | ||| | |
                          A-ACTTGGA
Also written as: <A,A> <T,-> <A,A> <C,C> <T,T> <A,T> <G,G> <-,G> <A,A>
```

Typically alignment models try to maximize the number of matches, and minimize the number of mutations (e.g. edit distance). There is generally the tacit assumption that the sequences themselves are random (i.e. incompressible).

The compressibility of non-random sequences should be taken into account when obtaining their alignment. As an example, consider sequences in which runs or repeated letters are common. The figure below shows a low information region (`AAA`) and a high information region (`AGT`) and partial matches that might occur in alignments:

```
...AAA...          ...AGT...
...AAA...          ...AGT...
```

The second partial match is more significant because AGT is more surprising and therefore less likely to have occurred by chance in both sequences. Thus, the alignment incorporating this partial alignment should be given a better 'score' than the first.

It is shown how to make precise the intuition from the above example. There are two distinct parts to the alignment problem - the problem of how to score an alignment, and the problem of searching for the alignment with the best score. Both problems are discussed.

It is worth noting that although this alignment problem is discussed with the application to DNA sequences in mind, it is not limited to them. The method discussed in this paper is applicable to the alignment of sequences that are noisy observations of some true non-random sequence.

In the Section 2, we will first provide some background on a standard sequence alignment technique. In Section 3.1, we state our model that produces the sequences that we wish to align. The following section describes how to assign a score to a given alignment. The next two sections present how to search for the optimal alignment when our sequence model is a zeroth order Markov model, and in a later section when that model is a first order Markov model.

2 Standard Sequence Alignment

The simplest model for scoring alignments is a point mutation model where each mutation costs 1. That is, a mismatch/insert/delete costs 1 and a match costs 0. For example, under this model the alignment below would cost 2, one change and one delete (or insert depending on perspective).

```
AGTGC aligned with ACGC

     A G T G C
     A C - G C
```

Allison [2] showed that these costs can be normalised and then viewed as probabilities. This allows us to view the sequence alignment search as picking the alignment with the maximum likelihood.

A generic dynamic programming algorithm (DPA) for sequence alignment is typically used to find the optimal alignment. This basic algorithm is shown below for aligning two strings 'A' and 'B'.

```
D[0,0] = 0
D[i,0] = i, i=1..|As|
D[0,j] = j, j=1..|Bs|

for j = 1..|Bs|
```

```
for i = 1..|As|
  D[i,j] = min(D[i,j-1] + insertCost,
               D[i-1,j] + deleteCost,
               D[i-1,j-1] + (if (As[i] = Bs[j]) then
                               matchCost
                             else
                               mismatchCost))
```

The cell $D[|A|, |B|]$ contains the cost of the optimal alignment. The actual alignment is obtained by tracing back through the D matrix.

Figure 1 shows an example alignment on the D matrix. In this example, with inserts, deletes and mismatches costing 1, and matches 0, there are many optimal alignments (nine in fact). Only the 'northern' and 'southern' most alignments are shown - all other alignments lie between these two.

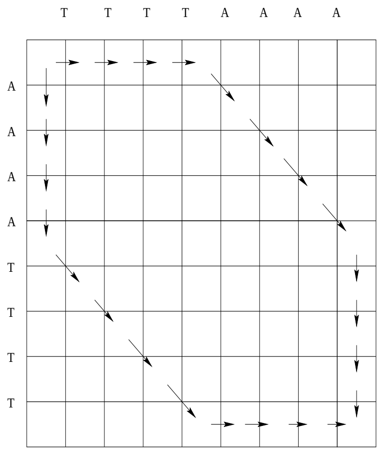

Fig. 1. Northernmost and southernmost optimal alignments for aligning the sequences 'AAAATTTT' and 'TTTTAAAA'.

More complex costs are sometimes used in alignments, such as linear gap costs [5] where a run of n inserts/deletes have a cost of $a \times n + b$ (where a and b are constants).

Although this algorithm (or variations of it) is commonly used, the model under which 'A' and 'B' are assumed to be generated is typically not stated.

3 Costing an Alignment

A family of sequences, F, is *non-random* (or *compressible* or of *low information content*) [1] if there is some model M with a compression algorithm $m()$ such that, on average, $|m(S)| < |S|$ for sequences drawn from F. We want to use $m()$ in costing our alignments because it gives a precise measure of the information content of a sequence. However, $m()$ deals with a single sequence, not an alignment of two sequences, and thus we require a model of alignment that incorporates $m()$.

For our alignment model, we assume that there is one true (unknown) sequence, R, and two noisy observations of it, S1 and S2. The sequence R is assumed to be compressible by $m()$ on average. The problem we look at is inferring R given S1 and S2.

This model is considered to be a good fit to the DNA sequence assembly problem [14, 15, 8], where these is some "true" (non-random [10, 11, 1, 12]) DNA sequence that is sequenced in fragments using techniques that introduce noise. These noisy fragments are then used to infer the true DNA sequence.

3.1 Specifying our Model for Data Generation

We need to define how the sequences R, S1 and S2 come about, and therefore their mutual alignment. Assume we have a sequence, R, generated by our known model $m()$. It is necessary to carefully state the method by which the noisy observations, S1 and S2, are generated from R.

To generate S1 (and likewise S2) from R, do the following:

1. For each character in R do the following :
   (a) Choose n from the geometric distribution [2] $Pr(n) = (P_{insert})^n(1 - P_{insert})$.
   Then insert n characters.
   (b) Copy the character of R correctly (match), copy it incorrectly (mismatch), or do not copy it (delete), with the respective probabilities P_{match}, $P_{mismatch}$, P_{delete}. (Note: $P_{match} + P_{mismatch} + P_{delete} = 1$.)
2. Choose n from the geometric distribution $Pr(n) = (P_{insert})^n(1 - P_{insert})$.
   Then insert n characters.

P_{insert} is typically small (~ 0.05), thus the average number of insertions in a sequence of length $|R|$ is $\sim 0.05 \times |R|$.

The character chosen to mismatch to, and the characters chosen to insert are selected from a uniform distribution over A,T,G and C. This was done for simplicity, although it is easily modified so that the characters chosen depend on the character that is being mismatched from (e.g. an 'A' may be more likely to mismatch to a 'T' than a 'G' or 'C').

[1] in a similar sense to the notions of Solomonoff [13], Kolmogorov [9] and Chaitin [4].
[2] The mean of this distribution is $P_{insert}/(1 - P_{insert})$.

Note that by outlining this generating model explicitly, there is a real distinction between insertions and deletions, thus, they must be handled differently. This is different to the standard DPA alignment that treat insertions and deletions in a very similar manner.

3.2 Model $m()$ of R

We want our sequence model, $m()$, to be a model over all possible sequences. To keep things simple we will begin with $m()$ being a 0th order Markov Model (MM) with a geometric prior on the length (and later we will extend to a 1st order MM).

Our Markov model is used to define the probability of any character in a sequence, dependent only on the previous n characters, where n is the order of the Markov model. So, for a zeroth order Markov model of a DNA sequence, $m()$ is just a fixed distribution over A,T,G and C. Some DNA sequences have a very biased base distribution (such as being GC rich), so a zeroth order MM would be a better model for those sequences than a model that assumes all bases are equally likely.

R is generated by using $m()$ in the following manner:

1. Decide whether to append another character to R with probability P_y.
2. Choose the character to add from the 0th order MM.

The expected length of R is thus $P_y/(1-P_y)$, so for R to be of any substantial expected length, P_y needs to be very close to 1.

4 Encoding R, S1, S2 and their alignment

We have defined how R, S1, S2 and their alignment are generated. We now outline how to assign a probability of how likely a particular instance of R, S1, S2 and their alignment is of being generated. This is obviously very closely related to the generating model of Section 3.1.

Because the probabilities become very small, we will deal in $-\log$ of the probabilities (base e unless otherwise stated). We will talk about encodings, or 'messages' which are encoded efficiently, so the length of a message is equal to $-\log$(probability of the event).

Given R, S1, S2 and their alignment, we wish to encode these efficiently. We will construct a message containing R (encoded using $m()$), S1 and S2 encoded as differences from R. The shorter this message, the more likely this alignment is. The encoding is as follows:

1. First, R is encoded using m().
2. S1 is encoded as a series of operations as follows:
   (a) For each character of R do the following:
      i. Encode run of n inserted characters in S1, by encoding n with $prob(n) = (P_{insert})^n(1 - P_{insert})$ and the characters each with uniform probability=1/4.

ii. State whether this character of R has been matched, mismatched or deleted.
iii. If mismatched, state mismatched character with prob [3]=1/4.

(b) Encode a final run of inserts for S1.

3. S2 is encoded in a similar manner.

Note that this process is the same irrespective of $m()$. The choice of $m()$ only affects the search.

For most applications, sequences S1 and S2 will be available, and from these R and the alignment must be inferred. By using the encoding outlined above, it is possible to compare two (or more) inferred alignments. The next step is searching for the optimal alignment.

5 Search for optimal R and alignment (for 0th order MM)

Given S1 and S2, it is necessary to search for the best R and alignment, where 'best' means the most likely (having the shortest encoding). The search is done by modifying the generic dynamic programming algorithm (DPA) commonly used for sequence alignment. To keep the explanation simple we will first discuss the search disregarding the possibility of inserts (i.e. for every character in S1 and S2 there is a corresponding character in R). Each cell, $D[i,j]$, in the D matrix, will contain the best cost of aligning (and inferring R) for the sequences S1[1..i] and S2[1..j]. To allow for this incremental approach to the costing of the alignment, the encoding scheme described in the previous section is slightly modified in the following manner: 1 character of R is encoded, followed by the corresponding operation for S1 (a match, mismatch or delete) then the operation for S2. The calculation for D matrix cell $D[i,j]$ is shown below.

```
score = infinity;
for each Rchar in (A,T,G,C)
  # First the diagonal move, i.e. a match or mismatch
  score = min(score,
       D[i-1,j-1] +            # Cost so far
       mInc(Rchar) +           # Incremental cost of add another char to R
       mis(Rchar, S1[i]) +     # Cost to encode mis/match for S1
       mis(Rchar, S2[i]))      # Cost to encode mis/match for S2

  # Now the vertical move, i.e. a character from S1 but not from S2
  score = min(score,
```

[3] Need to consider if mismatches are allowed to the same character or not. These can be considered to be equivalent. Given an alignment which is generated with mismatches allowed to the same character, it can be interpreted as being generated with mismatches having to be to a different character by modifying P_{match} and $P_{mismatch}$ (and vice versa).

```
      D[i-1,j] +             # Cost so far
      mInc(Rchar) +          # Incremental cost of add another char to R
      mis(Rchar, S1[i]) +    # Cost to encode mis/match for S1
      del)                   # Cost to encode a delete for S2

  # And the horizontal move, i.e. a character from S2 but not from S1
  score = min(score,
      D[i,j-1] +             # Cost so far
      mInc(Rchar) +          # Incremental cost of add another char to R
      del +                  # Cost to encode a delete for S1
      mis(Rchar, S2[j]) +    # Cost to encode mis/match for S2

D[i,j] = score
```

Finally, the cell $D[|S1|, |S2|]$ will contain the cost for the inferred R and for aligning that R with S1 and S2. R and the alignment can be found by back tracking through the D matrix. The algorithm, like the standard DPA, is $O(n^2)$ (where n is the length of the sequences) but with the multiplicative constant increased by the alphabet size.

5.1 Handling inserts

Performing an optimal search while allowing for insertions is somewhat more complicated. The algorithm resembles that of Gotoh [5] used for linear gap cost alignment.

The encoding from the previous section must be modified to allow for inserts. This encoding is the same as in Section 4, simply rearranged. Thus, before each character or R is encoded, the run of inserts for S1 is encoded followed by a run of inserts for S2.

By allowing inserts, it is possible to have a transition in the D matrix that does not correspond to a character in R. For each cell of the D matrix, there are three possibilities for the current *state* corresponding to whether the last transition wasn't an insertion, was an insertion into S1 or was an insertion into S2. The first two of these possibilities can be combined into a single state. The two states we are left with correspond to whether we are currently inserting into S1 or S2.

State 1 corresponds to inserting in S1, and State 2 inserting into S2. When moving from State 1 to State 2, it is necessary to encode the end of S1 insertions. When moving from State 2 to State 1, the end of S2 insertions must be encoded. This is detailed below. The numbers in the parentheses on the left are a key for what needs to be encoded. It is also shown graphically in Figure 2.

Key for what needs to be encoded:
(1) - Need to encode end of inserts for both S1 and S2
(2) - Need to encode end of inserts for S2
(3) - No end of inserts needs to be encoded (continuing a run of inserts)
(4) - Need to encode end of inserts for S1

State 1 is determined in the following fashion:
(1) Diagonal from State 1 an R character, and a match/mis for S1 and S2
(1) Vertical from State 1 an R character, match/mis for S1, del for S2
(1) Horizontal from State 1 an R character, del for S1, match/mis for S2
(2) Diagonal from State 2 an R character, and a match/mis for S1 and S2
(2) Vertical from State 2 an R character, match/mis for S1, del for S2
(2) Horizontal from State 2 an R character, del for S1, match/mis for S2
(3) Vertical from State 1 no R character, insert for S1

and State 2 as follows:
(3) Horizontal from State 2 no R character, insert for S2
(4) Horizontal from State 1 an R character, del for S1, match/mis for S2

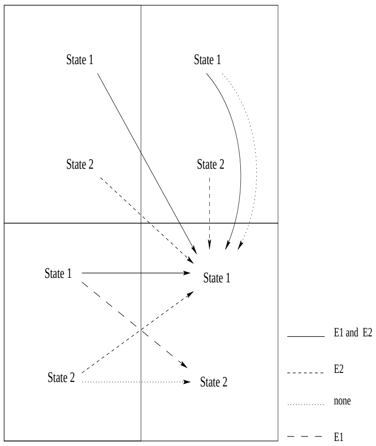

Fig. 2. Diagram for state calculation. E1 - encoding length for end of inserts in S1, E2 - encoding length for end of inserts in S2.

As an example, consider the following:
S1 contains a character 'A', and S2 has no corresponding character. This can be explained either as (1) an insert in S1 (R had no corresponding character), or (2) as a deletion in S2 (R had a character 'A').

Case 1 Probability of saying it is an Insert × prob stating inserted character.
$P_1 = P_{insert} \times 1/4$

Case 2 Probability to end inserts for S1 and S2. Then character for R, a match and a delete.
$P_2 = (1 - P_{insert}) \times (1 - P_{insert}) \times P_y \times m(A) \times P_{match} \times P_{delete}$

Set P_1 equal to P_2 and solve for $m(A)$:
$m(A) = 1/4 \times P_{insert}/((1 - P_{insert})^2 \times P_y \times P_{match} \times P_{delete})$

Typical values:
$P_{insert} = 0.05$
$P_{match} = 0.9$
$P_{delete} = 0.05$
P_y is typically very close to 1.

For these typical values, $m(A) \approx 0.311$. Thus for the above typical values, if $m(A) > 0.31$, R will be inferred to have the character 'A' (case 2), otherwise it will be cheaper to say R had no such character, so 'A' was inserted in the noisy observation (case 1).

6 First order Markov Model

A 0th order Markov model is a special case of and typically an inferior modelling tool to a 1st order Markov model. As previously noted, the encoding does not need to be modified (except as it depends on $m()$), just the search. Since a 1st order MM needs only the previous character as context information, $m()$ can still be used incrementally in the encoding of R.

The search for the optimal R and alignment when $m()$ is a 1st order MM is very similar to that in the previous section. If insertions are not allowed, it is simply a matter of adding a state for each of A,T,G or C to each cell of the D matrix. For each state it is necessary to calculate the encoding given the previous character of R was an A, T, G or C, thus 4 times as much work is done than when $m()$ is a 0th order MM. The search is still $O(n^2)$, with the multiplicative constant increased by the size of the alphabet squared when compared to the standard DPA.

If insertions are allowed, we need two super-states, each containing states for A, T, G and C. These two super-states are analogous to the two states in the 0th order MM search. The calculation of encoding lengths is done by an obvious combination of 1st order MM with no inserts and the 0th order MMs that allows inserts. To simplify the back tracing step (to determine alignment), extra information is carried in each state about where it came from.

7 Null Encoding

For this work, the null model is that S1 and S2 are noisy observations of two *different* true sequences. The length of the null encoding is the sum of the length for encoding S1 and S2 separately. The significance of an alignment can be judged by comparing its encoded length with that of the null encoding. If the alignment is statistically significant, its encoding will be shorter that the null encoding.

The null encoding is also useful in the case where *part* of S1 and S2 correspond to the same true sequence, such as in determining sequence overlap. The overlap region is encoded using the alignment encoded defined earlier, and the two regions of S1 and S2 that don't overlap are encoded using the null encoding. Thus we can judge whether an overlap is significant by comparing with the null model. (This is useful for the sequence assembly problem.)

The null encoding is calculated in a similar manner to the encoding of an alignment. S1 is encoded by first encoding R, then encoding differences from it. When searching for the optimal encoding, it is reasonable to ignore the possibility for a delete (a character not in S1 that was in R). Thus, the null encoding can be calculated in $O(|S1|)$ time.

8 Results

A criterion is needed to compare alignments if one is to claim that some alignment is 'closer' to the true alignment. Our method for measuring the difference between two alignments is to measure the area between them. An alignment defines a path from the top left of the DPA matrix to the bottom right (Figure 1), thus, there is some area between any two different alignments. This area is our measure of how close those two alignments are and it can deal with alignments involving insertions/deletion of two sequences of different length.

A program was implemented to perform the search as described in Section 6, and another to generate sequences by the method described in Section 3.1. The program to generate the data had $m()$ being a first order Markov model (see Table 1 for the Markov model transition probabilities). This Markov model was also included in our low information alignment program.

Table 1. Markov model transition probabilities used to generate test data.

	A	T	G	C
A	0.11	0.09	0.3	0.5
T	0.09	0.11	0.5	0.3
G	0.5	0.3	0.09	0.11
C	0.5	0.11	0.09	0.3

For the mutation probabilities, one parameter p was used so $P_{match} = (1 - 2p)$, $P_{delete} = P_{mismatch} = p$ and $P_{insert} = p$. The lowinfo alignment used the same MM and mutation probabilities as the generating program. In our lowinfo alignment an insertion event is separate from a deletion, mismatch or match thus $P_{match} + P_{delete} + P_{mismatch}$ should equal 1. The standard DPA alignment has all four events being possible at the same time so, $P_{match} + P_{delete} + P_{mismatch} + P_{insert}$ should equal 1. For this reason we set the DPA alignment program's mutation probabilities to $P_{match} = (1 - 3p)$, $P_{delete} = P_{mismatch} = P_{insert} = p$, which is different to that used for the lowinfo alignment program. It is not clear if these are the 'correct' mutation probabilities to use for the DPA alignment, but they seem reasonable. We experimented with other mutation costs for the DPA alignment program, but it had little effect on the results.

Sequences were generated with a $P_y = 100/101$ (see Section 3.1). As an attempt to make our results only be a function of the mutation probability, p, and not the length of the sequences, only sequences whose length fell between 90 and 120 characters were used.

The generated sequences were then aligned using both the standard alignment algorithm (DPA alignment), and by our alignment algorithm (lowinfo alignment). Then the area between DPA alignment and the true alignment was calculated, and the area between the lowinfo alignment and the true alignment. Results are plotted versus the mutation probability p in Figure 3. Often there are a number of optimal alignments (for both the DPA and the lowinfo) so the northernmost alignment was chosen arbitrarily.

Choosing the northernmost optimal alignment with the standard DPA algorithm is straightforward, the $min()$ function is simply modified to choose steps in the order horizontal first, diagonal next and finally vertical when the costs of two more steps are equal [4]. Selecting the northernmost optimal alignment from our lowinfo alignment algorithm is more difficult. Essentially the same ordering on steps is done as for the standard DPA, however a complication arises in our alignment algorithm from the fact that the final cell of the D matrix contains a number of states that can correspond to different alignments. So the choice of the final state can affect the chosen optimal alignment. The problem arises when a number of these final states have an equal best score. We choose the state that has its last step in the most northerly direction. This is not guaranteed to be the most northerly optimal alignment, but it is believed to in all but the most pathological cases.

Another feature of alignment algorithms is the number of different optimal alignments they produce. Most applications require a single alignment out of the alignment algorithm, so having a number of very different optimal alignments can be a problem. A arbitrary choice must often be made between optimal alignments (above we arbitrarily chose the northernmost alignment). So an alignment algorithm that produces similar optimal alignments is "better" than one that produces very different alignments (provided the true alignment is close to the optimal alignments). Our way of quantifying this similarity is to calculate the area between the northernmost optimal alignment and the southernmost optimal alignment in the DPA matrix. Figure 1 illustrates the northern and southernmost optimal alignments, the area between these is our measure of how similar the optimal alignments produced by an alignment algorithm are. We call this area the envelope of optimal alignments. Figure 4 plots this envelope area for our lowinfo alignment algorithm, and for the standard DPA alignment algorithm.

Table 2 shows a summary of the data plotted in Figures 3 and 4, except the table is for data collected from 1000 runs. From both plots, it is obvious the DPA

---

[4] We are dealing with floating point numbers, so values a and b are considered equal if $|a - b| < \epsilon$ where we used $\epsilon = 0.0000001$.

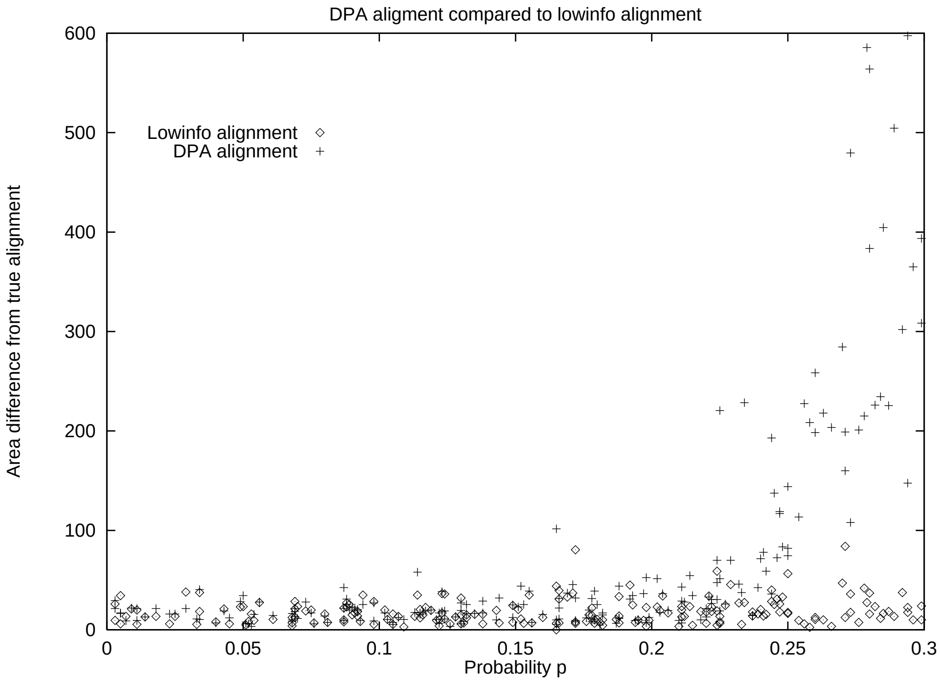

Fig. 3. Comparison of the DPA alignment and the lowinfo alignment with the true alignment for 200 runs.

alignment algorithm deteriorates very rapidly when the mutation probability, p, is greater than about 0.2. For this reason Table 2 also shows a summary of the data when all data points with $p > 0.2$ are removed.

Table 2. Summary of results shown in Figure 3 and Figure 4.

	All data		Probability $p < 0.2$	
	DPA	lowinfo	DPA	lowinfo
Total runs	1000	1000	652	652
Avg area between optimal and true alignments	70.7	18.6	19.7	16.7
Avg area in optimal alignment envelope	21.8	5.6	14.4	4.9
Number of times the better alignment was found	241	722	210	408

This data indicates that for sequences that fit our model, our lowinfo alignment algorithm produces an alignment closer to the true alignments on average than the standard DPA alignment algorithm. This is seen both in terms of the

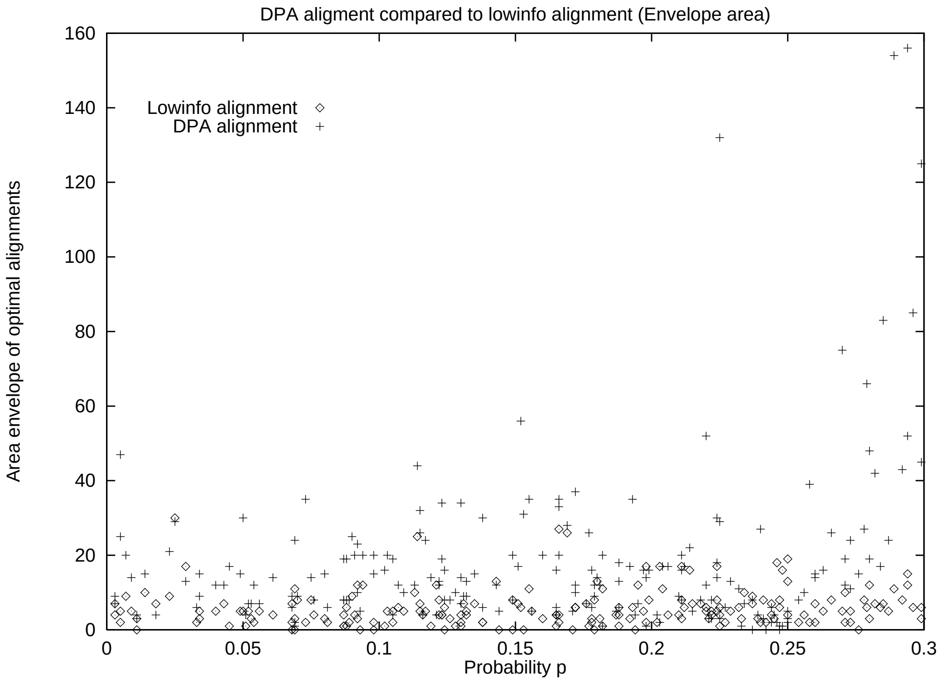

Fig. 4. Comparison of the DPA alignment with lowinfo alignment for envelope of optimal alignments.

average area between the true and optimal alignments, and by the number of times our algorithm finds a closer optimal alignment (last row of Table 2). It is also evident that the optimal alignments found by our alignment algorithm are 'closer' (by our envelope are measure) than those found by the standard DPA algorithm.

9 Future Work

Above we have described how to find the most likely true sequence R, and the best alignment. There are (at least) two other things we may want to infer given S1 and S2:

1. If we are interested in the most likely alignment (and not the true sequence), then we should sum over all possible true sequences.
2. If we want to know how related the sequences S1 and S2 are then we should sum over all possible true sequences and alignments [3].

It is not possible to sum over all possible true sequences because there are an infinite amount of them. But we may be able to make a good approximation to this by only summing over 'likely' true sequences. By likely we mean those sequences that are similar to S1 and S2.

There is also the obvious extension of making $m()$ more complicated. This will almost certainly increase the difficulty of the search for the optimal alignment. If $m()$ can not be used incrementally (i.e. encode the sequence $S[1..i]$ in $O(1)$ time given the encoding of $S[1..i-1]$), an optimal search is unlikely to remain $O(n^2)$. Some stochastic type search would probably be necessary for the more complex models.

Throughout this paper we have assumed that the model, $m()$, and the mutation probabilities are known to the alignment algorithm. This may not be the case in practice, so it may be necessary to estimate $m()$ and the mutation probabilities from the sequences that are to be aligned. This may be possible by beginning with an estimate of $m()$ and the mutation probabilities, determining an alignment for them, re-estimating from that alignment, then iterating with the new estimate of $m()$ and the mutation probabilities.

There exist a number of algorithms that are an improvement of the standard DPA. The algorithm by Ukkonen [18] for aligning similar sequences runs in $O(n+d^2)$ time, where d is the edit distance of the two sequences. This algorithm requires mutation costs to be fixed integer values, which they may not be. But it may be possible to choose different costs that still maintain the same ordering on alignments [2]. This algorithm may be able to be used to find the optimal alignment for the alignment model described in this paper, possibly with some restrictions.

10 Conclusion

Standard alignment algorithms do not take into account the information content (or non-randomness) of the data. The high information regions of sequences should be more carefully aligned than low information regions because they are less likely to occur by chance. In this paper we have outlined a method for performing sequence alignment (in quadratic time) that takes into account the randomness of the sequence as given by a model $m()$. Our method can also infer a 'true sequence' that the two observed sequences are noisy observations of. We have shown that our alignment algorithm find a better alignment that the standard DPA when our model of the sequences hold. We have also shown that the optimal alignments found by our alignment algorithm are 'closer' (more consistent) than those found by the standard DPA.

11 Acknowledgements

David Dowe was supported by Australian Research Council (ARC) Large Grant No. A49602504.

References

1. P. Agarwal and D. J. States. The repeat pattern toolkit (RPT): Analyzing the structure and evolution of the C. elegans genome. *Proceedings Second International Conference on Intelligent Systems for Molecular Biology*, pages 1–9, 1994.
2. L. Allison. Normalization of affine gap costs used in optimal sequence alignment. *Journal of Theoretical Biology*, 161:263–269, 1993.
3. L. Allison, C. S. Wallace, and C. N. Yee. Finite-state models in the alignment of macromolecules. *Journal of Molecular Evolution*, 35:77–89, 1992.
4. G. Chaitin. On the length of programs for computing finite sequences. *Journal of the Association for Computing Machinery*, 13:547–549, 1966.
5. O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162, 1982.
6. P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *J. Mol. Evol.*, 20:175–186, 1984.
7. L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Theoretical Biology*, 233:123–138, 1993.
8. X. Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14:18–25, 1992.
9. A. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:4–7, 1965.
10. A. Milosavljevic and J. Jurka. Discovering simple DNA sequences by the algorithmic significance method. *CABIOS*, 9(4):407–411, 1993.
11. D. R. Powell, D. L. Dowe, L. Allison, and T. I. Dix. Discovering simple DNA sequences by compression. *Pacific Symposium on Biocomputing*, to appear, 1998.
12. E. Rivals, O. Delgrange, J.-P. Delahaye, M. Dauchet, M.-O. Delorme, M.-O. Delorme, A. Henaut, and E. Ollivier. Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences. *Comp. Appl. BioSci*, 13(2):131–136, 1997.
13. R. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22,224–254, 1964.
14. R. Staden. A new computer method for the storage and maipulation of DNA gel reading data. *Nucleic Acids Research*, 8(16):3673–3694, 1980.
15. R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucleic Acids Research*, 10:4731–4751, 1982.
16. W. R. Taylor and C. A. Orengo. A holistic approach to protein structure alignment. *Protein Engineering*, 2(7):505–519, 1989.
17. W. R. Taylor and C. A. Orengo. Protein structure alignment. *Journal of Theoretical Biology*, 208:1–22, 1989.
18. E. Ukkonen. On approximate string matching. *Foundations of Computation Theory*, 158:487–495, 1983.