J. theor. Biol. (2000) **207,** 325–336
doi:10.1006/jtbi.2000.2177, available online at http://www.idealibrary.com on IDEAL®

Fast, Optimal Alignment of Three Sequences Using Linear Gap Costs

David R. Powell*, Lloyd Allison and Trevor I. Dix

School of Computer Science and Software Engineering, Monash University, 3168 *Australia*

(*Received on* 13 *May* 1999, *Accepted in revised form on* 23 *August* 2000)

Alignment algorithms can be used to infer a relationship between sequences when the true relationship is unknown. Simple alignment algorithms use a cost function that gives a fixed cost to each possible point mutation—mismatch, deletion, insertion. These algorithms tend to find optimal alignments that have many small gaps. It is more biologically plausible to have fewer longer gaps rather than many small gaps in an alignment. To address this issue, linear gap cost algorithms are in common use for aligning biological sequence data. More reliable inferences are obtained by aligning more than two sequences at a time. The obvious dynamic programming algorithm for optimally aligning k sequences of length n runs in $O(n^k)$ time. This is impractical if $k \geqslant 3$ and n is of any reasonable length. Thus, for this problem there are many heuristics for aligning k sequences, however, they are not guaranteed to find an optimal alignment. In this paper, we present a new algorithm guaranteed to find the optimal alignment for three sequences using linear gap costs. This gives the same results as the dynamic programming algorithm for three sequences, but typically does so much more quickly. It is particularly fast when the (three-way) edit distance is small. Our algorithm uses a speed-up technique based on Ukkonen's greedy algorithm (Ukkonen, 1983) which he presented for two sequences and simple costs.

1. Introduction

A new algorithm is presented to align optimally three sequences using linear gap costs. This algorithm is an extension of the two sequences, simple cost algorithm by Ukkonen (1983). For sequences of length n that have an optimal alignment with edit cost d our new algorithm has a time complexity of $O(d^3 + n)$ on average, which is nearly always attained, and $O(nd^2)$ in the worst case. We make explicit a finite-state model (FSM) for the generation of sequences from a parent sequence when using a linear function to cost a run of insertions or deletions. For this model, we show as how the probabilities of matches, changes, insertions and deletions in the finite-state machine relate to the costs in the alignment algorithm.

Sequence alignment algorithms are used in a number of different areas. Currently, one of these important areas is the aligning of macromolecules such as DNA sequences and protein sequences. An alignment shows as how two, or more, sequences may be related. For example, a good alignment between the sequences ATCGCA and TTCGA might be

```
A T C G C A
  | | |   | .
T T C G - A
```

The "–" indicates a character that has been deleted from that sequence, or alternatively

* Author for correspondence.

a character that has been inserted in the other sequence. A version of alignment is to assign a fixed cost to each possible point mutation. For example, simple costs give a cost of 0 to a match, and a cost of 1 to a change, insertion or deletion. Thus, if simple costs were used, the above alignment would have a cost of 2. An alignment with minimal cost is said to be optimal. An alignment can also be interpreted as showing how to edit one sequence into another, thus the cost is sometimes referred to as edit cost or edit distance (Levenshtein, 1966; Sellers, 1974).

Linear gap costs are often used instead of simple costs because linear gap costs better model the underlying process. For example, in some biological processes it is considered more likely to have a small number of long gaps than a large number of small gaps. Linear gap costs favour a small number of longer gaps by having a start-up cost for each gap. Linear gap costs give a run of l insertions or deletions a cost of $w(l) = a + b \times l$, for some fixed values of a and b (Gotoh, 1982). Altschul & Erickson (1986) suggest a linear gap cost function of $w(l) = 2.5 + l/2$, while $w(l) = 3 + l$ is in common use.

In many circumstances it is important to align optimally more than two sequences. The obvious dynamic programming algorithm for k sequences requires $O(n^k)$ time to run which is infeasible for $k \geqslant 3$ with n of any reasonable length. Improved algorithms for k sequences have been studied and algorithms such as those by Carrillo & Lipman (1988) and later by Altschul & Lipman (1989) have been developed. These algorithms use the optimal alignment of pairs of sequences as a heuristic to limit the k-dimensional volume used to find an optimal alignment.

For many applications, such as building evolutionary trees (Sankoff & Morel, 1973; Sankoff & Cedergren, 1983) from sequences or in sequence assembly, it is desirable to align three sequences at a time. There are many multiple sequence alignment algorithms that use pairwise sequence alignment in an iterative method to build up a multiple alignment (Notredame *et al.*, 1998; Thompson *et al.*, 1994; Taylor, 1988; Higgins & Sharp, 1988). Some algorithms such as MASCOT (Hirosawa *et al.*, 1993) use the extra information obtained from three-way alignment to improve the multiple alignment. Three-way alignment is useful in evolutionary tree algorithms because every internal node of the tree has three neighbours. The three-way alignment can be used to make an inference for the sequence at the internal node, this can then be used in an iterative method to improve the evolutionary tree.

Gotoh (1986) presented an algorithm for aligning three sequences with linear gap costs based on the simple alignment DPA. We present an algorithm to which Gotoh's is an approximation. We also develop a fast algorithm for three sequences with linear gap costs which is based on Ukkonen's (1983) algorithm.

The alignment algorithms discussed in this paper can be classified by the following attributes: alignment of two strings vs. three strings; simple costs vs. linear gap costs; and the standard DPA vs. Ukkonen's faster algorithm. For each of these three attributes, the first mentioned is typically the simpler case and the second the more complex and often more desirable. Ukkonen's algorithm is generally preferred over the DPA because the time complexity is reduced. Simple costs are a special case of linear gap costs; thus linear gap costs are more versatile, and often provide a better model. Optimal alignment of three strings is often better than alignment of two strings because it provides more information on how the strings are related.

From these three independent attributes there are $2^3 = 8$ different algorithms possible. Table 1 shows how the two new algorithms presented in this paper relate to previously known alignment algorithms in terms of the three independent attributes. The simplest of the eight algorithms is for two strings with the standard DPA and simple costs, this is discussed briefly in Section 3.1. The next simplest algorithm uses linear gap costs instead of simple costs (Gotoh, 1982) and is summarized in Section 3.1.1. In Section 3.2, we discuss the algorithm by Ukkonen (1983) which is for two strings and simple costs. The final algorithm for two strings again uses the Ukkonen speed-up with two strings but extends it to linear gap costs (Yee & Allison, 1992).

The simplest three string alignment algorithm is an extension of the DPA for two strings and uses simple costs. Ukkonen's speed-up can also be applied to three strings (Allison, 1993b). Three-string alignment with linear gap costs is

Table 1
Summary of the relation of this paper's contributions with previous alignment algorithms

	DPA-based		Ukkonen-based	
	2 sequences	3 sequences	2 sequences	3 sequences
Simple costs	(Levenshtein, 1966) (Sellers, 1974)	Various	(Ukkonen, 1983)	(Allison, 1993b)
Linear costs	(Gotoh, 1982)	(Gotoh, 1986)*	(Yee & Allison, 1992)	This paper

*For tree-costs Gotoh's algorithm is an approximation to our new algorithm.

also possible; an algorithm for this problem based on the DPA was shown by Gotoh (1986). However, Gotoh's algorithm is somewhat limited. We point out these limitations and present an algorithm of which Gotoh's is a special case.

Of course, it is desirable to have an alignment algorithm for three strings with linear gap costs using the Ukkonen speed-up. Such an algorithm is the main contribution of this paper. The average time complexity of the algorithm for strings of length n, is shown to behave as $O(d^3 + n)$ where d is the edit cost.

To illustrate the usefulness of our new Ukkonen-based algorithm for three sequence over the DPA-based algorithm we ran our programs implementing both algorithms on some real biological sequences. We selected clipped DNA sequences from the Transthyretin gene for a human, a mouse and a rat. The Genbank ids of the sequences used are HUMPALA(27-470), MMALBR(27-467) and RATPALTA(10-453), respectively. The costs used are as follows: 0 for a match, 1 for a change, 3 to start a gap, and 1 to continue a gap. Under these costs, the edit distance of the three sequences is 109. Aligning these sequences shows that the Ukkonen-based algorithm runs quickly and has modest memory requirements while the DPA-based algorithm is almost impractical. The details of this example run are given in Section 5.

Ukkonen's algorithm requires the cost of a match to be 0, and the other mutation costs to be small integers. This makes it useful for aligning nucleotide data because such costs are commonly used. It is standard to use mutation costs based on a substitution matrix while aligning protein sequences (Dayhoff *et al.*, 1978; Henikiff & Henikoff, 1992). These matrices do not use small integer costs, thus the Ukkonen algorithm is not typically useful for aligning protein sequences. The new algorithm we present in this paper uses a speed-up based on Ukkonen's algorithm, and correspondingly it typically will not be useful for aligning protein sequences.

It may occur to the reader that if the alignment of three strings is an improvement on two strings, then why not four strings, or indeed k strings. Optimal alignment of k strings with a DPA-type algorithm has a time complexity of $O(n^k)$ for strings of length n. This becomes prohibitive for long strings with k around 4 or above. The next improvement to consider would be Ukkonen's algorithm for k strings, which would be expected to have an average time complexity of $O(d^k + n)$ where the edit cost of the k strings is d. The problem here is that as the number of strings increases, the edit cost also tends to increase. Thus, it is more likely that d will become larger than n, therefore Ukkonen's algorithm will be slower than the DPA version at some point. Although for sequences with a large number of matches a Ukkonen style algorithm will be fast compared to the corresponding DPA style algorithm.

2. Linear Gap Costs

The simple costs used in the basic DPA are not as biologically plausible as linear gap costs. Linear, also called affine, gap costs use a fixed mutation cost, and a linear function for a run of insertions or deletions in the gap. The linear function gives a start-up cost to a gap then a lower cost for each character in the gap, thus a few long gaps are favoured over many short ones. This provides a better model for biological

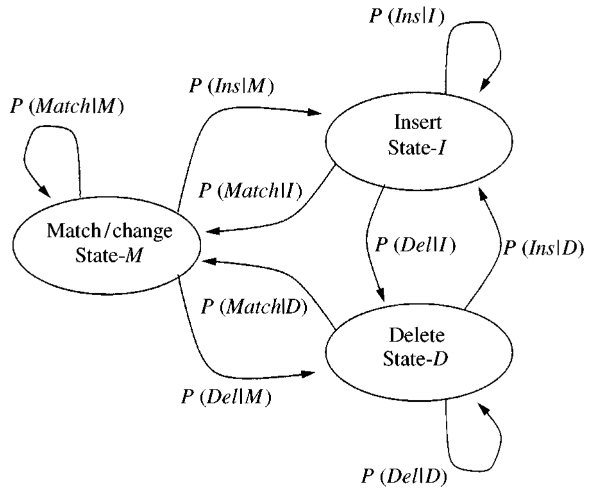

FIG. 1. A three-state finite-state machine to produce one sequence from another.

mutations than simple costs does and thus has been used in many algorithms for aligning biological sequence data (Myers & Miller, 1988; Gotoh, 1990).

An interpretation (Allison *et al.*, 1992) of linear gap costs is to think of one sequence generated from another via a FSM with three states as in Fig. 1. Note that we will use M to denote the *match/change* state, I to denote the *insert* state, and D to denote the *delete* state. The FSM can be seen as processing an input sequence a character at a time, and producing an output sequence a character at a time. Every transition into the *delete* state uses a character from the input sequence without producing any character in the output sequence. A transition into the *insert* state produces a character on the output (chosen from some distribution) without affecting the input. The "match/change" state takes a character from the input sequence and generates a character in the output sequence. Whether the character is copied correctly or mutated is determined with probability, P_{change}.

It is normal when using linear gap costs to have the probability of continuing a gap higher than the starting one. That is, $P(Ins|I) > P(Ins|M)$ and $P(Del|D) > P(Del|M)$.

It is often more convenient to use *costs* instead of probabilities in alignment algorithms. If the cost of a match (or copy) is set to zero, the other costs can be chosen from the probabilities in such a way as to leave the rank order of alignments unchanged (Allison, 1993a). It is common in the literature to have linear gap costs of the form $w(l) = a + b \times l$ for a gap length of $l(l > 0)$. The cost a is the cost of starting a gap, and b is the cost of continuing that gap (typical values, $a = 3$, $b = 1$). In this paper, we refer to the costs in the FSMs in the form $C(Match|M)$. Using this notation, $C(Ins|M) = C(Ins|D) = a + b$ and $C(Ins|I) = b; C(Del|M) = C(Del|I) = a + b$ and $C(Del|D) = b$. It is necessary when using Ukkonen-based algorithms for matches to cost 0, thus: $C(Match|M) = C(Match|D) = C(Match|I) = 0$. And the mutation cost for copying a character incorrectly, $C(Change)$, is often set to 1. Modifying the costs to leave the rank ordering of alignments unchanged can be accomplished by multiplying all costs by a constant factor, or by adding (or subtracting) a constant value for each alignment character. This allows the costs to be modified to convenient small integers as required by Ukkonen's algorithm. This correspondence between probabilities and costs can be used to choose costs that match some known probabilities, or the FSM probabilities can be calculated for commonly used alignment costs.

3. Alignment of Two Sequences

3.1. THE BASIC DPA

The basic DPA to align two strings, *As* and *Bs*, uses an edit distance matrix where each entry $D[x, y]$ contains the edit distance between strings $As[1..x]$ and $Bs[1..y]$. The algorithm is given in Fig. 2. To obtain an alignment using this algorithm it is necessary to trace back through

```
D[0, 0] = 0
D[i, 0] = i * deleteCost, i = 1..|As|
D[i, j] = j * insertCost, 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
                                        changeCost))
```

FIG. 2. The DPA to determine minimum edit distance.

the matrix from $D[|As|, |Bs|]$ to $D[0, 0]$ following which choices were made by the *min*() function.

3.1.1. *DPA with Linear Gap Costs*

The basic DPA can be modified to compute an optimal alignment using a linear gap cost function (Gotoh, 1982). The DPA for linear gap costs has three matrices instead of one—one each for the three possible *states* corresponding to the last state of the FSM, match/change, insert or delete. The linear gap cost DPA has time complexity $O(n^2)$ and space complexity $O(n^2)$, the same as for the basic DPA.

3.2. UKKONEN'S ALGORITHM

Ukkonen (1983), and independently Myers (1986), presented an alignment algorithm that runs in $O(nd)$ time in the worst case and $O(n + d^2)$ on average, where n is the length of the strings, which are assumed to be of the same order, and d is the edit cost. This algorithm uses $O(d^2)$ space or if no alignment is required $O(d)$ space. A necessary condition for this algorithm is that a match costs 0, and all other mutation costs are small positive integers. The smaller the edit cost, d, the faster the algorithm runs. Thus, it is desirable to choose small costs, which may be able to be done without affecting the rank order of the alignments (as detailed at the end of Section 2).

A recent algorithm, Calign (Chao *et al.*, 1997), uses a Ukkonen style method for aligning cDNA and genomic DNA. This algorithm uses restricted affine gap costs, which are essentially linear gap costs with a maximum cost for an insertion. This allows for large gaps as expected when aligning cDNA and genomic DNA by having a fixed cost for large gaps over a pre-defined size.

Ukkonen's algorithm speeds up the basic DPA by recognizing a number of facts about the DPA matrix: not all the entries of D are needed, the diagonals of D are non-decreasing, and only the end point of a run of matches is important. An alternative matrix U is used in Ukkonen's algorithm. Entry $U[ab, d]$ contains the maximum distance obtainable along string *As* for cost d on diagonal *ab*. A row of the U matrix corresponds to a diagonal of the D matrix, and a column of the U matrix to a "contour" of fixed cost in the D matrix. As an example, assume DPA matrix cell $D[i, j]$ is on the optimal alignment, then in terms of the U matrix this cell will be on the diagonal $ab = i - j$, and thus $U[i - j, D[i, j]] \geqslant i$. Given this correspondence between the D matrix and the U matrix, we will sometimes discuss the operation of the Ukkonen algorithm in terms of the D matrix. Ukkonen's algorithm for two sequences with fixed mutation costs is given in Fig. 3.

```
{$U[ab, d]$ = max $a$ s.t. $D[a, b] = d$ where $ab = a - b$
        = − infinity if no such $a$ exists}

$U[0, 0]$ = **max** $a$ s.t. $As[1..a] = Bs[1..a]$
$U[ab, d]$ = − **infinity, if** $|ab| > d$

{*Outer loop, iterated until* $U[|As| - |Bs|, d] = |As|$}
 $U[ab, d]$ = **max**($U[ab + 1, d -$ insertCost$]$,
                $U[ab,\quad d -$ changeCost$] + 1$,
                $U[ab - 1, d -$ deleteCost$] + 1$)

 {*Inner Loop, extends diagonal on a run of matches*}
 **while** ($As[U|ab, d] + 1] = Bs[U[ab, d] - ab + 1]$)
    $U[ab, d] += 1$
```

FIG. 3. Ukkonen's algorithm for simple mutation costs.

The outer loop of Ukkonen's algorithm loops over each entry in the U matrix to determine for diagonal *ab* and cost d how far along string *As* can be reached. This is determined by looking at U with a cost $d - 1$, for point mutation costs of 1, on the same diagonal *ab*, and the two neighbouring diagonals $ab + 1$ and $ab - 1$. The inner loop then extends this distance while strings *As* and *Bs* match, corresponding to a run of matches down a diagonal of the D matrix. As with the basic DPA, the alignment is obtained by tracing back through the choices made in the max( ) function when calculating the U matrix.

Thus, in terms of the D matrix, Ukkonen's algorithm calculates the entries in a region around the final diagonal that has a width equal to the edit distance of the two strings. So, regions in the upper right and the lower left of the D matrix are able to be omitted from the calculation. There is a further saving when a run of exact matches occurs in both strings because the diagonals alongside the run of matches need not be calculated.

The worst case time complexity of Ukkonen's algorithm is easily seen to be $O(nd)$, although in most cases the average complexity of $O(n + d^2)$ is achieved. It is not immediately obvious as for what sequences cause the worst-case performance of the algorithm. An example of such sequences is $As = a^k b^l$ and $Bs = b^l a^k$ where $k < l$. For simple costs these sequences have an edit distance of $d = 2k$. When aligning these sequences with Ukkonen's algorithm and simple costs, the inner loop (Fig. 3) is iterated $l + (l-1) + (l-2) + \cdots + (l-k) = O(lk)$ times. The outer loop is iterated $O(d^2) = O(k^2)$. Thus, the time complexity is $O(k^2 + kl)$ and since $k < l$ and $d = 2k$ this is $O(ld)$.

3.2.1. *Ukkonen's Algorithm with Linear Gap Costs*

Ukkonen's algorithm has been applied to aligning strings using linear gap costs (Yee & Allison, 1992). As with the DPA for linear costs, there are three matrices instead of one, one for each of the possible states of the FSM. The extension of Ukkonen's algorithm to linear gap costs is similar to the extension of the basic DPA to the DPA for linear gap costs.

4. Alignment of Three Sequences

An algorithm for three-way alignment with simple costs as an extension of the two sequence DPA is straightforward. This three-way DPA calculates a three-dimensional "cube" of volume $|A| \cdot |B| \cdot |C|$ where $|A|$, $|B|$ and $|C|$ are the lengths of the three sequence to be aligned. This algorithm has a time complexity of $O(n^3)$ which can be prohibitively large for sequences of any reasonable length. Thus, a more efficient algorithm, such as Allison (1993b) or Altschul & Lipman (1989), Carrillo & Lipman (1988) is required if realistic sequence data is to be aligned.

Another complication that arises while aligning three sequences is the type of costs to use. Three commonly used cost types are all-pairs costs, star costs and tree costs, although for three sequences tree costs and star costs are the same. For three sequences, all-pairs costs is the total cost obtained by summing the three pairwise costs. Star costs implies a common unknown parent sequence. The parent sequence is determined by a consensus vote, and the pairwise cost between each sequence and the parent are summed together. Star costs are the main focus in this paper though modification to all-pairs costs would be straightforward.

4.1. ALIGNMENT OF THREE SEQUENCES WITH LINEAR GAP COSTS

Alignment of three sequences with linear gap costs is a complicated extension of alignment for two sequences. One flavour of three-way alignment is to assume that each of the three sequences has been generated independently from a common parent by a three-state FSM. The alignment problem is to infer from the three sequences how they were generated, and thus the common parent sequence. This three-way alignment problem matches the problem of inferring an evolutionary tree from sequence data. To infer how the sequences were generated, the state each of the three FSMs, at every point of the alignment must also be determined. Note that if sequences to be aligned are assumed to come from a common ancestor, then three-way alignment, unlike two-way alignment, can distinguish insertions from deletions and different costs can be given to them if desired.

As in Section 2 we use M, I and D to denote the match/change, insert and delete states, respectively. Consider the three-way alignment of the sequences $xyyxz$, xxz, and zz:

$$\begin{array}{ccccc@{\quad}ccccc}
x & y & y & x & z & M & I & I & M & M \\
x & - & - & x & z & M & M & M & M & M \\
z & - & - & - & z & M & M & M & D & M
\end{array}$$

(i)

or

$$\begin{array}{ccccc}
M & M & M & M & M \\
M & D & D & M & M \\
M & D & D & D & M
\end{array} .$$

(ii)

The first interpretation (i) infers the common parent sequence is xxz while (ii) infers $xyyxz$.

Which is the more likely of these two depends on the probabilities in the FSMs. Recall that when one of the FSMs is in the insert state the other FSMs are idle. Thus, above in the first alignment the FSMs for the second and third sequence are idle while the two "*y*" characters are inserted in the first sequence.

At every point along the alignment the states of the three FSMs must be inferred, there are $3^3 = 27$ possible combinations of these states, though some will never be inferred. For example, if a character in the parent sequence has been deleted from all three child sequences the corresponding states would be *DDD* and it could not be reasonably inferred. In fact, it is necessary for at least one of the FSMs to be in the *match* state. The *insert* state is special because it generates characters without using the parent sequence, thus if a FSM is in the *insert* state, the other two FSMs can be considered frozen. Therefore, it is sufficient to allow only one FSM to be in the *insert* at a time (e.g. *IMI* is invalid). Ignoring the invalid, or useless combinations of states there are 16 remaining possible combinations of states.

4.2. USING A DPA

Gotoh (1986) presented an algorithm for the alignment of three sequences with linear gap costs; however, this algorithm only allowed combinations of states that contain at least two match states, which gives seven different combinations. Seven different matrices were used, one for each of these combinations. The computation performed on these 3D matrices is analogous to the 2D matrices for two sequences.

Our DPA for three sequences with linear gap costs is similar to Gotoh's, but allows for all plausible combinations of the FSMs' states and thus better matches the model for the sequences. As an example, consider the three sequences TGGTATGCTAGCT, TGGTCGATGCTAG and TGGTCTGATGCTAGCT optimally aligned using the following costs: match cost 0, change cost 1, gap start-up cost 3, and continue gap cost 1. The optimal alignments of these sequences obtained by Gotoh's algorithm and by our algorithms are shown first and second, respectively.

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

Gotoh's algorithm

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

New algorithms

The optimal alignment from Gotoh's algorithm has an edit cost of 15, and the alignment from our algorithm an edit cost of 14. The reason for these different alignments are achieved is because our algorithm allows a run of inserts in the middle of a run of deletes, thus the third T in the third sequence is considered an insert. This is not possible with Gotoh's algorithm.

An optimal alignment for three sequences with linear gap costs can be found by using a DPA similar to that for two sequences. The main difference is that instead of three matrices, as used for two sequences, there is a matrix for each of the possible combinations of states of the three FSMs.

The heart of the algorithm is shown in Fig. 4, where the sequences to be aligned are *As*, *Bs* and *Cs*. This procedure calculates one cell in one matrix. The function transitionCost calculates the cost for changing FSM states, for example to go from the matrix for *MID* to the matrix for *MDD* would have a cost $C(Match|M) + C(Del|I) + C(Del|D)$, whereas the transition $MMD \rightarrow MID$ would have a cost $C(Ins|M)$. The changeCost function simply calculates the cost for any change mutations, for example, if the states of the three FSMs were *MDM* and the corresponding characters were $x - y$, the cost would be $C(Change)$. If the FSMs' states were *MMM* and the characters *xyz* the cost would be $2 \times C(Change)$.

```
procedure DPAcalcCell($i, j, k$, matrix)
    set $i2, j2, k2$ to index of neighbour
    bestCost = infinity

    for fromMatrix in AllMatrices

        cost = cell[$i2, j2, k2$, fromMatrix) +
                transitionCost (matrix, fromMatrix) +
                changeCost(matrix, $A[i]$, $B[j]$, $C[k]$)

        bestCost = min(cost, bestCost)

    endfor
    cell [$i, j, k$, matrix] = bestCost
endproc.
```

FIG. 4. Calculation of a cell for a DPA to find an optimal three-way alignment with linear gap costs.

A given cell $[i, j, k]$ in a given 3D matrix toMatrix is completely determined by the cells with index $[i2, j2, k2]$ from all matrices. The index $[i2, j2, k2]$ of this *neighbouring* cell is determined by the states of the FSMs for the matrix toMatrix. That is, a combination of states has one and only one neighbour in the 3D matrices. The following table lists the neighbours for some given FSMs' states if the cell to be calculated it at $[i, j, k]$. Note that if one of the FSMs is in the *insert* state, the states of the other FSMs are irrelevant in determining the neighbour.

MMM	$[i-1, j-1, k-1]$
MDM	$[i-1, j, k-1]$
DDM	$[i, j, k-1]$
MIM	$[i, j-1, k]$
MID	$[i, j-1, k]$

To find the optimal alignment for three sequences with linear gap costs, the routine DPAcalcCell is called for each cell as shown in Fig. 5. The edit cost is the cheapest cost in any cell from the 3D matrices at $[|As|, |Bs|, |Cs|]$. The optimal alignment is found by backtracking from here through choices made by the min() function.

4.3. USING UKKONEN'S ALGORITHM

The DPA for three sequences with linear gap costs, runs in $O(n^3)$ time, where the three sequences are of length approximately n. By employing Ukkonen's algorithm this can be reduced to $O(nd^2)$ in the worst case, and $O(d^3 + n)$ on average for sequences that have an edit distance of d.

```
for $i = 0..|As|$
  for $j = 0..|Bs|$
    for $k = 0..|Cs|$
      for matrix in AllMatrices
        DPAcalcCell ($i, j, k$, matrix)
```

FIG. 5. Calculating all cells for a DPA for three-way alignment with linear gap costs.

As with the DPA base algorithm, there are 27 (reduced to 16 in practice) matrices, one for each of the possible combinations of FSM states.

The heart of the algorithm is a function UKKcalcCell (analogous to DPAcalcCell in Section 4.2) to calculate the contents of a single cell (see Fig. 6). This function is called by the function Ukk (not shown) which is simply a wrapper that puts the result from UKKcalcCell in a memo array for any subsequent calls. The function transitionCost is as before, and the function countUnique returns the number of unique characters among its three parameters. To make sure the ends of the sequences are not overrun, the function pastEnd is used as to make sure every step is valid.

The cell to be calculated has one neighbour, analogous to the one neighbour for the DPA version, the index of this neighbour is put in $ab1, ac1$. A loop iterates over all possible matrices at the cell $[ab1, ac1]$.

The possible changes between the three sequences are dealt with in the if then else statement as follows: if the MMM matrix is being calculated, then there are two possible costs depending on the characters of the sequences being all different, or two being the same. If the step has a null character, "–", then the cost of the step can again take two values depending on whether the non-null characters are equal. The final possibility is that there is only one non-null character, in which case there can be only one possible cost.

If the cell to be calculated corresponds to the MMM matrix, then there is the possibility of

```
function UKKcalcCell($ab$, $ac$, $d$, matrix)
  set $da$, $db$, $dc$ to direction of neighbour (0 or 1)

  $ab1 = ab - da + dc$
  $ac1 = ac - da + dc$
  bestDist = − infinity

  for fromMatrix in AllMatrices
    cost = $d$ − transitionCost (matrix, fromMatrix)

    if (matrix equals '$MMM$') then
      cost = cost − changeCost

    $a1$ = Ukk($ab1$, $ac1$, cost, fromMatrix)

    if (pastEnd ($a1$, $a1 - ab1$, $a1 - ac1$)) continue

    if (countUnique(if ($da$) then $As[a1]$ else '–',
                    if ($db$) then $Bs[a1 - ab1]$ else '–',
                    if ($dc$) then $Cs[a1 - ac1]$ else '–')
       = 2) then
      {$x$– – –$x$– – –$x$ $xx$– $x$–$x$ –$xx$ $xxy$ $xyx$ $yxx$}

      fromCost = cost
      dist = $a1 + da$
    else                    {$unique = 3$}
      {$xy$– $x$–$y$ –$xy$ $xyz$}

      $a1$ = Ukk($ab1$, $ac1$, cost − changeCost, fromMatrix)
      fromCost = cost − changeCost
      dist = $a1 + da$
    endif

    bestDist = max(bestDist, dist)

  endfor

  {Make sure it is an improvement}
  bestDist = max(bestDist, Ukk($ab$, $ac$, $d - 1$, matrix))

  if (matrix equals '$MMM$') then
    bestDist = extendDiagonal ($ab$, $ac$, $d$, matrix, bestDist)

  return bestDist
endfunc.
```

FIG. 6. Calculation of a cell for Ukkonen's algorithm with three sequences and linear gap costs.

extending the diagonal* along a run of matches. This is done in the extendDiagonal (not shown) function. The extendDiagonal function checks all matrices at $[ab, ac, d]$ to find which reaches the furthest along the As sequence. If from this point there is an exact run of matches the end point of the run of matches is returned. If there is no run of matches, the distance reached in the MMM matrix is returned.

To extract the optimal alignment, the U matrix must be back-traced through the choices made in the max( ) function. The whole U

* The term diagonal here is used in reference to the diagonals in the D matrix of the DPA algorithm.

matrix is required for this, thus $O(d^3)$ space is needed. If only the edit distance is required then the algorithms needs only $O(d^2)$ space.

5. Results

The program implementing the three-sequence alignment with Ukkonen's algorithm and linear gap costs is fairly complex. To ensure the correctness of the program we performed extensive testing. Some of the more rigorous testing procedures are explained below.

The first test was to compare the results of the program with the results of a program implementing the DPA version of the algorithm. The three input sequences were independent random sequences of random length over an alphabet of four characters. This test was performed for several thousand different alignments with sequence length varying from 0 to about 2000, while the edit distance ranged from 0 to about 150. The edit costs produced by both algorithms were the same.

The second test was to take the alignment produced by the Ukkonen version of the program, and from that determine the inferred parent sequence. This parent sequence was then aligned one at a time with each of the three input sequences. The pairwise alignment was performed with a program implementing two-way alignment with linear gap costs. As before, this test was performed several thousand times with similar ranges of lengths and edit distances as for the previous test. The sum of the three edit costs from the pairwise alignments was equal to the edit cost from the three-way alignment.

To illustrate the usefulness of our new Ukkonen-based algorithm for three sequences over the DPA-based algorithm, we ran our programs implementing both algorithms on some real biological sequences. We selected clipped DNA sequences from the Transthyretin gene for a human, a mouse and a rat. The Genbank ids of the sequences used are HUMPALA(27-470), MMALBR(27-467) and RATPALTA(10-453), respectively. The costs used are as follows: 0 for a match, 1 for a change, 3 to start a gap, 1 to continue a gap. Under these costs, the edit distance of the three sequences is 109. These sequences were aligned with the Ukkonen-based algorithm for three sequences with linear gap costs in 6 CPU-min using about 90 Mbytes of memory on a Cyrix 686 with 192 Mbytes of memory. We were unable to align these sequences using the DPA-based algorithm since around 3 Gbytes of memory would be needed and a projected runtime of over 1 CPU-h.

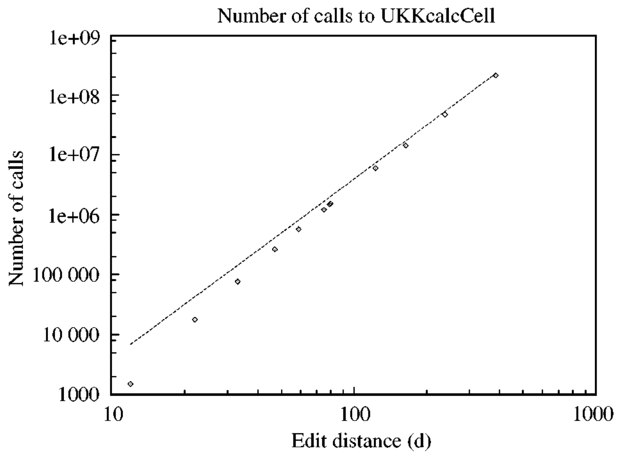

FIG. 7. Log–log plot of the number of calls to UKK calcCell against edit distance. (◇) Run1; (----) d^3.

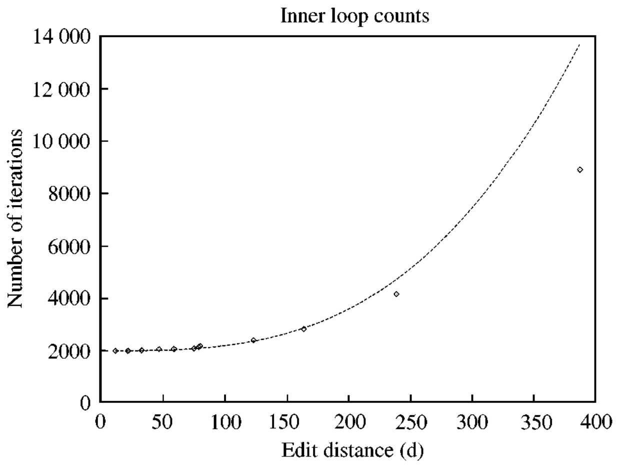

FIG. 8. Plot of the inner loop iterations against edit distance for sequences of approximately 2000 characters. (◇) Run1; (----) $n + d^3/C$.

The time complexity of the algorithm presented in Section 4.3 is not obvious. The algorithm exhibits an average time complexity of $O(d^3 + n)$. Figure 7 shows the number of calls to the UKKcalcCell function which asymptotes to d^3. The inner loop iterations are along runs of matches; Fig. 8 shows this to be less than $O(d^3)$.

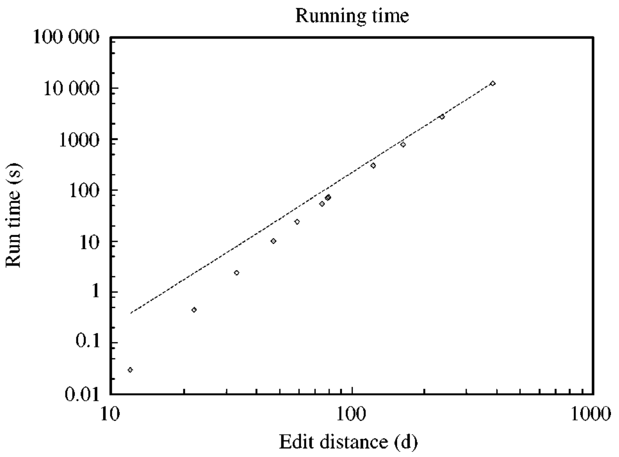

FIG. 9. Log–log plot of running time vs. edit distance. (◇) Run1; (----) d^3.

The "n" comes from the fact that for similar sequences of length n there are $O(n)$ matches along the optimal alignment. The rest of the iteration in the inner loop comes from matches off the optimal alignment.

A log–log plot of the actual running time against edit distance is shown in Fig. 9. From this it can be seen that the running time also asymptotes to d^3.

6. Conclusion

The algorithms discussed in this paper can be classified by three different attributes: alignment of two or three strings; fixed costs or linear gap costs; standard DPA or the faster Ukkonen algorithm. Previously known algorithms were discussed to show how they fit into this classification, and to provide a framework to present our contributions.

The first new algorithm we presented was for the problem of aligning three sequences optimally using linear gap costs and was based on the standard DPA. This algorithm has complexity $O(n^3)$ in both time and space. We made clear the model of generating the three sequences from a common parent sequence, and showed how the costs used in the alignment algorithms relate to the probabilities for a match, mismatch, insertion and deletion for this model.

Using previously known algorithms as explanatory stepping-stones we presented our major contribution, an algorithm for three-string alignment using linear gap costs with the efficiency of the Ukkonen (1983) algorithm. Our algorithm produces the same results as the above-mentioned DPA-based algorithm, however, the Ukkonen-based algorithm is significantly faster for similar sequences. The average time complexity exhibited was $O(d^3 + n)$ with a space complexity of $O(d^3)$ or $O(d^2)$ if only the edit distance was desired. This makes it feasible to use our new Ukkonen-based algorithm to align biological sequences of realistic lengths where the DPA-based algorithms such as Gotoh (1986) are not.

REFERENCES

ALLISON, L. (1993a). Normalization of affine gap costs used in optimal sequence alignment. *J. theor. Biol.* **161,** 263–269.
ALLISON, L. (1993b). A fast algorithm for the optimal alignment of three strings. *J. theor. Biol.* **164,** 261–269.
ALLISON, L., WALLACE, C. S. & YEE, C. N. (1992). Finite-state models in the alignment of macro-molecules. *J. Mol. Evol.* **35,** 77–89.
ALTSCHUL, S. & ERICKSON, B. (1986). Optimal sequence alignments using affine gap costs. *Bull. Math. Biol.* **48** (5–6), 603–616.
ALTSCHUL, S. F. & LIPMAN, D. J. (1989). Trees, stars and multiple biological sequence alignment. *SIAM J. Appl. Math.* **49,** 197–209.
CARRILLO, H. & LIPMAN, D. (1988). The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.* **48,** 1073–1082.
CHAO, K.-M., JAMES OSTELL, J. Z. & MILLER, W. (1997). A tool for aligning very similar DNA sequences. *CABIOS* **13,** 75–80.
DAYHOFF, M. O., SCHWARTZ, R. M. & ORCUTT, B. C. (1978). A model of evolutionary change in proteins. *Atlas Protein Sequence Struct.* **5,** 345–352.
GOTOH, O. (1982). An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162,** 705–708.
GOTOH, O. (1986). Alignment of three biological sequences with an efficient traceback procedure. *J. theor. Biol.* **121,** 327–337.
GOTOH, O. (1990). Optimal sequence alignment allowing for long gaps. *Bull. Math. Biol.* **52,** 359–373.
HENIKIFF, S. & HENIKOFF, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U.S.A.* **89,** 915–919.
HIGGINS, D. G. & SHARP, P. M. (1988). CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene* **73,** 237–244.
HIROSAWA, M., HOSHIDA, M., ISHIKAWA, M. & TOYA, T. (1993). MASCOT: multiple alignment system for protein sequences based on three-way dynamic programming. *CABIOS* **9,** 161–167.
LEVENSHTEIN, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Doklady.* **10,** 707–710.

Myers, E. W. (1986). An O(nd) difference algorithm and its variations. *Algorithmica* **1,** 251–266.

Myers, E. W. & Miller, W. (1988). Optimal alignments in linear space. *CABIOS* **4,** 11–17.

Notredame, C., Holm, L. & Higgins, D. G. (1998). COFFEE: an objective function for multiple sequence alignments. *Bioinformatics* **14,** 407–422.

Sankoff, D. & Cedergren, R. J. (1983). Simultaneous comparison of three or more sequences related by a tree. In: *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison* (Sankoff, D. & Kruskall, J. B., eds), pp. 253–263. New York: Addison Wesley.

Sankoff, D. & Morel, C. (1973). Evolution of 5S RNA and the non-randomness of base replacement. *Nature New Biol.* **245,** 232–234.

Sellers, P. H. (1974). On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.* **26,** 787–793.

Taylor, W. R. (1988). A flexible method to align large numbers of biological sequences. *J. Mol. Evol.* **28,** 161–169.

Thompson, J., Higgins, D. & Gibson, T. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucl. Acids Res.* **22,** 4673–4690.

Ukkonen, E. (1983). On approximate string matching. *Found. Comput. Theory* **158,** 487–495.

Yee, C. N. & Allison, L. (1992). Fast string alignment with linear indel costs. Technical Report 92/165, Department of Computer Science, Monash University.