

Chapter 3

Bio-Sequence Comparison and Applications

Xiaoqiu Huang

Department of Computer Science

Iowa State University

226 Atanasoff Hall

Ames, IA 50011-1040, USA

3.1 Introduction

The structure of a genome is a linear sequence of nucleotides, which encodes genes and regulatory elements. Genes are homologous if they are related by divergence from a common ancestor (Attwood, 2000). Homologous genes perform the same or similar functions. The sequences of homologous genes in related organisms are usually similar. For example, the sequences of homologous genes in humans and mice are 85% similar on an average (Makalowski et al., 1996). If a new genomic DNA sequence is very similar to the sequence of a gene whose function is known, it is very likely that the genomic DNA sequence contains a gene and its function is similar to the function of the known gene. If a new genomic DNA sequence is highly similar to a cDNA sequence, then the genomic DNA sequence contains a gene and the structure of the gene can be found by aligning the two sequences. Thus methods for comparing sequences are very useful for understanding the structures and functions of genes in a genome. This chapter focuses on methods for comparing two sequences, which often serve as a basis for multiple sequence comparison methods, a topic for the next chapter.

In the first part of this chapter, we describe algorithms for comparing two sequences. We present a global alignment algorithm for comparing two sequences that are entirely similar. We give a local alignment algorithm for comparing sequences that contain locally similar regions. We also describe efficient computational techniques for comparing long sequences. In the second part, we consider a general problem of comparing two sets of sequences. Every sequence in one set is compared with every sequence in the other set. We describe an efficient algorithm for this problem. In the third part, we present four applications to illustrate that sequence alignment programs are useful for analysis of DNA and protein sequences. In the last part, we provide two directions for developments of new and improved sequence comparison methods.

3.2 Global Alignment

In this section, we first define a global alignment model. Then we describe a dynamic programming algorithm for computing an optimal global alignment of two sequences. Next we present a linear-space algorithm for computing an optimal global alignment. Finally we look at a way of reducing the time requirements of the algorithms.

3.2.1 An alignment model

A similarity relationship between two sequences A and B can be represented by an alignment of two sequences, an ordered list of pairs of letters of A and B . The alignment consists of substitutions, deletion gaps and insertion gaps. A substitution pairs a letter of A with a letter of B . A substitution is a match if the two letters are identical and a mismatch otherwise. A deletion gap is a gap where letters of A correspond to no letter of B , and an insertion gap is a gap where letters of B correspond to no letter of A . The length of a gap is the number of letters involved. Deletion and insertion gaps are defined with regard to transformation of sequence A into sequence B . An alignment of A and B shows a way to transform A into B , where a letter of A is replaced by a letter of B in every substitution, the letters of A in every deletion gap are deleted, and the letters of B in every insertion gap are inserted.

Below is an alignment of two DNA sequences AGCTACGTACACTACC and AGCTATCGTACTAGC. This alignment contains 13 matches, 1 mismatch, an insertion gap of length 1, and a deletion gap of length 2.

```
AGCTA-CGTACACTACC
AGCTATCGTAC--TAGC
```

The similarity of an alignment is measured by a numerical number. Let $\sigma(a, b)$ be the score of a substitution involving letters a and b . Let numbers q and r be gap-open and gap-extension penalties, respectively. The numbers q and r are nonnegative.

The score of a gap of length k is $-(q + k \times r)$. Values for the parameters σ , q and r are specified by the user. A letter-independent substitution table is usually used for comparison of DNA sequences. For example, each match is given a score of 10 and each mismatch a score of -20 . Possible values for q and r are 40 and 2, respectively, for DNA sequences. A letter-dependent substitution table such as PAM250 (Dayhoff et al., 1978) and BLOSUM62 (Henikoff and Henikoff, 1992) is usually used for comparison of protein sequences. Possible values for q and r are 10 and 2, respectively, for proteins. The similarity score of an alignment is just the sum of scores of each substitution and each gap in the alignment. The score of the example alignment given above is 24 using the given set of values for DNA sequences. An optimal (global) alignment of two sequences A and B is an alignment of A and B with the maximum score.

3.2.2 A dynamic programming algorithm

Let $A = a_1a_2\dots a_m$ and $B = b_1b_2\dots b_n$ be two sequences of lengths m and n . A technique called dynamic programming in computer science is used to compute an optimal global alignment of A and B . Let $A_i = a_1a_2\dots a_i$ and $B_j = b_1b_2\dots b_j$ be initial segments of lengths i and j of A and B . In this technique, a matrix S is introduced: $S(i, j)$ is the maximum score of all alignments of A_i and B_j . Thus $S(m, n)$ is the score of an optimal alignment of A and B . To compute the matrix S efficiently, two additional matrices D and I are introduced. Let $D(i, j)$ (D for deletion) be the maximum score of all alignments of A_i and B_j that end with a deletion gap. Let $I(i, j)$ (I for insertion) be the maximum score of all alignments of A_i and B_j that end with an insertion gap.

First consider how to compute $S(i, j)$. For $i > 0$ and $j > 0$, let $P(A_i, B_j)$ denote an alignment of A_i and B_j with the maximum score $S(i, j)$, that is, an optimal alignment of A_i and B_j . The last aligned pair of $P(A_i, B_j)$ has to be one of the following aligned pairs: a substitution pair (a_i, b_j) , a deletion pair $(a_i, -)$, and an insertion pair $(-, b_j)$. If the last aligned pair of $P(A_i, B_j)$ is a substitution pair (a_i, b_j) , then the portion of

$P(A_i, B_j)$ before the last substitution pair (a_i, b_j) is an alignment of A_{i-1} and B_{j-1} with the maximum score $S(i-1, j-1)$, because $P(A_i, B_j)$ is an alignment of A_i and B_j with the maximum score. In this case, alignment $P(A_i, B_j)$ consists of alignment $P(A_{i-1}, B_{j-1})$ and the substitution pair (a_i, b_j) . Thus the score of $P(A_i, B_j)$ is equal to the score of $P(A_{i-1}, B_{j-1})$ plus $\sigma(a_i, b_j)$, that is,

$$S(i, j) = S(i-1, j-1) + \sigma(a_i, b_j).$$

If the last aligned pair of $P(A_i, B_j)$ is a deletion pair $(a_i, -)$, then $P(A_i, B_j)$ is an alignment of A_i and B_j that ends with a deletion and has the maximum score. By the definition of $D(i, j)$, we have

$$S(i, j) = D(i, j).$$

Similarly, if the last aligned pair of $P(A_i, B_j)$ is an insertion pair $(-, b_j)$, we have

$$S(i, j) = I(i, j).$$

By the definitions of the matrices S , D , and I , the following inequalities are always true.

$$S(i, j) \geq S(i-1, j-1) + \sigma(a_i, b_j),$$

$$S(i, j) \geq D(i, j),$$

$$S(i, j) \geq I(i, j).$$

Thus we conclude that for $i > 0$ and $j > 0$,

$$S(i, j) = \max\{S(i-1, j-1) + \sigma(a_i, b_j), D(i, j), I(i, j)\}.$$

Next consider how to compute $D(i, j)$. For $i > 0$ and $j > 0$, let $X(A_i, B_j)$ denote an alignment of A_i and B_j with the maximum score $D(i, j)$, which ends with a deletion pair $(a_i, -)$. Let $Y(A_{i-1}, B_j)$ denote the portion of $X(A_i, B_j)$ before the last pair. If $Y(A_{i-1}, B_j)$ ends with a deletion pair, then $Y(A_{i-1}, B_j)$ is an alignment of A_{i-1} and B_j with the maximum score $D(i-1, j)$, because $X(A_i, B_j)$ is a largest-scoring alignment of A_i and B_j that ends with a deletion gap. In other words, $X(A_i, B_j)$ consists of $X(A_{i-1}, B_j)$ and the deletion pair $(a_i, -)$. So we have

$$D(i, j) = D(i-1, j) - r.$$

Note that the gap open penalty for the gap that includes the deletion pair $(a_i, -)$ is already included in $D(i-1, j)$. If $Y(A_{i-1}, B_j)$ does not end with a deletion pair, then $Y(A_{i-1}, B_j)$ is an alignment of A_{i-1} and B_j with the maximum score $S(i-1, j)$,

because $X(A_i, B_j)$ is a largest-scoring alignment of A_i and B_j that ends with a deletion gap. In other words, $X(A_i, B_j)$ consists of $P(A_{i-1}, B_j)$ and the deletion pair $(a_i, -)$.

So we have

$$D(i, j) = S(i-1, j) - q - r,$$

where the expression $-q - r$ is the score of the gap that consists only of the deletion pair $(a_i, -)$, and $S(i-1, j)$ is the score of an optimal alignment $P(A_{i-1}, B_j)$, which ends with a substitution pair or an insertion pair.

Appending the deletion pair $(a_i, -)$ to alignment $X(A_{i-1}, B_j)$ yields an alignment of A_i and B_j with score $D(i-1, j) - r$. Similarly, appending the deletion pair $(a_i, -)$ to alignment $P(A_{i-1}, B_j)$ yields an alignment of A_i and B_j with score $S(i-1, j) - q - r$. Because both alignments end with a deletion pair, we have by the definition of $D(i, j)$ that

$$D(i, j) \geq D(i-1, j) - r,$$

$$D(i, j) \geq S(i-1, j) - q - r.$$

Note that if $i = 1$, then $D(i-1, j)$ is undefined. We assume that $D(0, j)$ is given a value of $S(0, j) - q$ so that the inequality involving $D(i-1, j)$ still holds if $i = 1$. Combining all those inequalities together, we conclude that for $i > 0$ and $j > 0$,

$$D(i, j) = \max\{D(i-1, j) - r, S(i-1, j) - q - r\}.$$

The recurrence for computing the matrix I for $i > 0$ and $j > 0$ is developed similarly.

The recurrences for the matrices S , D , and I for $i = 0$ or $j = 0$ can be easily developed. The recurrences for computing the matrices S , D , and I are summarized below.

$$S(0, 0) = 0,$$

$$S(i, 0) = D(i, 0) \text{ for } i > 0,$$

$$S(0, j) = I(0, j) \text{ for } j > 0,$$

$$S(i, j) = \max\{S(i-1, j-1) + \sigma(a_i, b_j), D(i, j), I(i, j)\}$$

for $i > 0$ and $j > 0$.

$$D(0, j) = S(0, j) - q \text{ for } j \geq 0,$$

$$D(i, 0) = D(i-1, 0) - r \text{ for } i > 0,$$

$$D(i, j) = \max\{D(i-1, j) - r, S(i-1, j) - q - r\} \text{ for } i > 0 \text{ and } j > 0.$$

$$I(i, 0) = S(i, 0) - q \text{ for } i \geq 0,$$

$$I(0, j) = I(0, j-1) - r \text{ for } j > 0,$$

$$I(i, j) = \max\{I(i, j-1) - r, S(i, j-1) - q - r\} \text{ for } i > 0 \text{ and } j > 0.$$

We present an alternative way for developing the recurrences for computing the three matrices. The alternative presentation is based on a grid graph of $m + 1$ rows and $n + 1$ columns in Figure 3.1. Each entry in the graph consists of three nodes that correspond to the three matrices, respectively. For $i > 0$, each vertical edge from row $i - 1$ to row i corresponds to a deletion pair $(a_i, -)$. For $j > 0$, each horizontal edge from column $j - 1$ to column j corresponds to an insertion pair $(-, b_j)$. For $i > 0$ and $j > 0$, each diagonal edge from entry $(i - 1, j - 1)$ to entry (i, j) corresponds to a substitution pair (a_i, b_j) . Each directed path from node S of entry $(0, 0)$ to node S of entry (i, j) corresponds to an alignment of A_i and B_j . Assume that for any entry, the edge from node D to node S and the edge from node I to node S have a score of 0. The score of a path from node S of entry $(0, 0)$ to a node of entry (i, j) is the sum of scores of every edge on the path. For any entry (i, j) , define $S(i, j)$ to be the maximum score of paths from node S of entry $(0, 0)$ to node S of entry (i, j) , and define $D(i, j)$ and $I(i, j)$ similarly with respect to nodes D and I of entry (i, j) . If there is no path from node S of entry $(0, 0)$ to node D (or I) of entry (i, j) , then $D(i, j)$ (or $I(i, j)$) can be set to $S(i, j) - q$ or any smaller value. This will simplify the presentation of a recurrence for computing the matrix D (or I) without causing any change to the value $D(i + 1, j)$ (or $I(i, j + 1)$).

Consider how to compute $S(i, j)$ for an internal entry (i, j) with $i > 0$ and $j > 0$. We partition the paths from from node S of entry $(0, 0)$ to node S of entry (i, j) into three groups. One group contains all the paths that end with a diagonal edge of score $\sigma(a_i, b_j)$ and the maximum score of paths in this group is $S(i - 1, j - 1) + \sigma(a_i, b_j)$. Another group contains all the paths that end with a vertical edge and the maximum score of paths in this group is $D(i, j)$. The last group contains all the paths that end with a horizontal edge and the maximum score of paths in this group is $I(i, j)$. Thus $S(i, j)$ is the maximum of the three expressions, which is exactly identical to

the recurrence for $S(i, j)$ with $i > 0$ and $j > 0$ given previously. Next consider how to compute $S(i, j)$ for a border entry (i, j) with $i = 0$ or $j = 0$. It is easy to see that $S(0, 0) = 0$. For $i = 0$ and $j > 0$, all the paths from node S of entry $(0, 0)$ to node S of entry (i, j) end with a horizontal edge and hence $S(i, j)$ is equal to $I(i, j)$. Similarly, for $i > 0$ and $j = 0$, $S(i, j)$ is equal to $D(i, j)$. The recurrences for computing the matrices D and I can be developed in the same way.

The matrices can be computed in order of rows or columns. The value $S(m, n)$ is the score of an optimal alignment of A and B . If only the score $S(m, n)$ is needed, then two linear arrays and a few scalars are sufficient to carry out the computation. This algorithm is the result of a number of developments (Needleman and Wunsch, 1970; Sellers, 1974; Wagner and Fisher, 1974; Waterman et al., 1976; Gotoh, 1982).

An optimal alignment is found by a traceback procedure on the matrices S , D , and I . An optimal alignment corresponds to a path through the grid graph (Figure 3.1) from node S of entry $(0, 0)$ to node S of entry (m, n) . Let the current node be a newly determined node. An optimal path is recovered by repeatedly determining a node that is immediately before the current node on the path. Thus the pairs of an optimal alignment are generated in a reverse order with the last pair produced first. Initially, the current node is node S of entry (m, n) . First consider the case where the current node is node S of entry (i, j) . The recurrences for S are used to determine a new node. If $i = 0$ and $j = 0$, then the traceback procedure terminates. Otherwise, if $j = 0$ or $S(i, j) = D(i, j)$, then the new node is node D of entry (i, j) . Otherwise, if $i = 0$ or $S(i, j) = I(i, j)$, then the new node is node I of entry (i, j) . Otherwise, the new node is node S of entry $(i - 1, j - 1)$ and a new pair for the optimal alignment is a substitution pair (a_i, b_j) . Next consider the case where the current node is node D of entry (i, j) . The recurrences for D are used to determine a new node. If $i = 1$ or $D(i, j) = S(i - 1, j) - q - r$, then the new node is node S of entry $(i - 1, j)$. Otherwise, then the new node is node D of entry $(i - 1, j)$. In each situation, a new pair for the optimal alignment is a deletion pair $(a_i, -)$. The case where the current node is node I of entry (i, j) is similarly handled.

The traceback procedure requires that the complete matrices be saved or addi-

tional information be saved to indicate how the value at each matrix entry is generated, which takes computer memory proportional to the product $m \times n$. Thus for two sequences of length 10,000, the algorithm takes computer memory in the order of 100,000,000 words. Because of the high computer memory requirement, only an optimal alignment of two sequences of at most a few thousand letters can be constructed on an ordinary computer using this algorithm. The time requirement of the algorithm is also proportional to the product $m \times n$. For two sequences of length 10,000, it takes less than a minute to compute the matrix S on an ordinary workstation. Thus the space requirement of this algorithm is much more limiting than the time requirement.

3.2.3 A linear-space algorithm

Hirschberg (1975) developed a linear-space algorithm for computing an optimal alignment of two sequences for the case $q = 0$. The algorithm takes computer memory in the order of $m+n$ and computer time in the order of $m \times n$. Because of the Hirschberg algorithm, computer memory is no longer a limiting factor for long sequences. In practice, the Hirschberg algorithm is even faster than the quadratic-space algorithm because an access to a linear array takes less time than an access to a quadratic array. As computers become faster, longer sequences can be aligned by the Hirschberg algorithm. Currently it takes about one hour on an ordinary workstation to produce an optimal alignment of two sequences of 100,000 letters. Myers and Miller (1988) generalized the algorithm of Hirschberg to handle the case where q is nonnegative.

The main idea of the space-efficient algorithm is to determine a middle pair of positions on an optimal alignment in linear space. Then the portions of the optimal alignment before and after the middle pair of positions are constructed recursively. Let $imid$ be $\lfloor m/2 \rfloor$, where $\lfloor y \rfloor$ is the largest integer less than or equal to y . We develop an algorithm for finding a position $jmid$ such that the pair of positions $imid$ and $jmid$ is on an optimal alignment of A and B . Let $P(A, B)$ denote an optimal alignment of A and B . Partition $P(A, B)$ into two parts immediately after position $imid$ of sequence A such that the first part does not end with any insertion gap. Let

$jmid$ be the largest position of sequence B in the first part. What is the necessary condition on $jmid$? Let A_i^s denote the suffix $a_{i+1}a_{i+2}\dots a_m$ of sequence A . Notation B_j^s is similarly defined. Then the first part of $P(A, B)$ is an alignment, denoted by $P_1(A_{imid}, B_{jmid})$, of A_{imid} and B_{jmid} and the second part is an alignment, denoted by $P_2(A_{imid}^s, B_{jmid}^s)$, of A_{imid}^s and B_{jmid}^s .

If $P_1(A_{imid}, B_{jmid})$ ends with a deletion gap and $P_2(A_{imid}^s, B_{jmid}^s)$ begins with a deletion gap, then we have

$$score(P(A, B)) = score(P_1(A_{imid}, B_{jmid})) + score(P_2(A_{imid}^s, B_{jmid}^s)) + q,$$

where $score(x)$ is the score of an alignment x . Including the term q on the right-hand side ensures that the deletion gap containing both a_{imid} and a_{imid+1} is charged by a gap open penalty exactly once. Since $P(A, B)$ is an optimal alignment of A and B , $P_1(A_{imid}, B_{jmid})$ has to be a largest-scoring alignment of A_{imid} and B_{jmid} that ends with a deletion gap and $P_2(A_{imid}^s, B_{jmid}^s)$ has to be a largest-scoring alignment of A_{imid}^s and B_{jmid}^s that begins with a deletion gap. Define $\bar{D}(i, j)$ to be the maximum score of alignments of A_i^s and B_j^s that begins with a deletion gap. From the definitions of the matrices D and \bar{D} , we obtain

$$S(m, n) = D(imid, jmid) + \bar{D}(imid, jmid) + q.$$

Because $D(imid, j) + \bar{D}(imid, j) + q$ is the score of an alignment of sequences A and B for each j , $0 \leq j \leq n$, we have

$$S(m, n) \geq D(imid, j) + \bar{D}(imid, j) + q \text{ for each } j, 0 \leq j \leq n.$$

Thus $jmid$ is a position j such that $D(imid, j) + \bar{D}(imid, j) + q$ is the maximum. The maximum value is the score of an optimal alignment of A and B .

If $P_1(A_{imid}, B_{jmid})$ does not end with a deletion gap or $P_2(A_{imid}^s, B_{jmid}^s)$ does not begin with a deletion gap, then we have

$$score(P(A, B)) = score(P_1(A_{imid}, B_{jmid})) + score(P_2(A_{imid}^s, B_{jmid}^s)).$$

Note that $P_1(A_{imid}, B_{jmid})$ can not end with any insertion gap because of the way $jmid$ is defined. Since $P(A, B)$ is an optimal alignment of A and B , $P_1(A_{imid}, B_{jmid})$ has to be an alignment of A_{imid} and B_{jmid} with the maximum score and $P_2(A_{imid}^s, B_{jmid}^s)$ has to be an alignment of A_{imid}^s and B_{jmid}^s with the maximum score. Define $\bar{S}(i, j)$ to be the maximum score of alignments of A_i^s and B_j^s . From the definitions of the

matrices S and \bar{S} , we obtain

$$S(m, n) = S(imid, jmid) + \bar{S}(imid, jmid).$$

Because $S(imid, j) + \bar{S}(imid, j)$ is the score of an alignment of sequences A and B for each j , $0 \leq j \leq n$, we have

$$S(m, n) \geq S(imid, j) + \bar{S}(imid, j) \text{ for each } j, 0 \leq j \leq n.$$

Thus $jmid$ is a position j such that $S(imid, j) + \bar{S}(imid, j)$ is the maximum. The maximum value is the score of an optimal alignment of A and B .

Define df to be

$$df = \max\{D(imid, j) + \bar{D}(imid, j) + q \mid 0 \leq j \leq n\}.$$

Define st to be

$$st = \max\{S(imid, j) + \bar{S}(imid, j) \mid 0 \leq j \leq n\}.$$

Then we have

$$S(m, n) = \max\{df, st\}.$$

If $df > st$, then a pair of positions $imid$ and $jmid$ is on an optimal alignment of sequences A and B , where $jmid$ is a position at which the maximum value df is obtained and df is the score of the optimal alignment of A and B . Otherwise, a pair of positions $imid$ and $jmid$ is on an optimal alignment of sequences A and B , where $jmid$ is a position at which the maximum value st is obtained and st is the score of the optimal alignment of A and B .

Define $\bar{I}(i, j)$ to be the maximum score of alignments of A_i^s and B_j^s that begin with an insertion gap. The recurrences for computing the matrices \bar{S} , \bar{D} , and \bar{I} can be developed similarly as those for the matrices S , D , and I . Here we present the recurrences for \bar{S} , \bar{D} , and \bar{I} without justification.

$$\bar{S}(m, n) = 0,$$

$$\bar{S}(i, n) = \bar{D}(i, n) \text{ for } 0 \leq i < m,$$

$$\bar{S}(m, j) = \bar{I}(m, j) \text{ for } 0 \leq j < n,$$

$$\bar{S}(i, j) = \max\{\bar{S}(i+1, j+1) + \sigma(a_{i+1}, b_{j+1}), \bar{D}(i, j), \bar{I}(i, j)\}$$

$$\text{for } 0 \leq i < m \text{ and } 0 \leq j < n.$$

$$\bar{D}(m, j) = \bar{S}(m, j) - q \text{ for } 0 \leq j \leq n,$$

$$\begin{aligned}\bar{D}(i, n) &= \bar{D}(i+1, n) - r \text{ for } 0 \leq i < m, \\ \bar{D}(i, j) &= \max\{\bar{D}(i+1, j) - r, \bar{S}(i+1, j) - q - r\} \\ &\text{for } 0 \leq i < m \text{ and } 0 \leq j < n.\end{aligned}$$

$$\begin{aligned}\bar{I}(i, n) &= \bar{S}(i, n) - q \text{ for } 0 \leq i \leq m, \\ \bar{I}(m, j) &= \bar{I}(m, j+1) - r \text{ for } 0 \leq j < n, \\ \bar{I}(i, j) &= \max\{\bar{I}(i, j+1) - r, \bar{S}(i, j+1) - q - r\} \\ &\text{for } 0 \leq i < m \text{ and } 0 \leq j < n.\end{aligned}$$

An algorithm for computing an optimal alignment of A and B in linear space consists of the following steps. If m is small enough, compute an optimal alignment of A and B using a traceback procedure. Otherwise, determine a pair of positions $imid$ and $jmid$ on an optimal alignment of A and B , and recursively compute the portions of the alignment before and after the pair of positions.

The positions $imid$ and $jmid$ are determined as follows. Set $imid = \lfloor m/2 \rfloor$. Compute the matrices S , D , and I from row 0 to row $imid$, and save $S(imid, j)$ and $D(imid, j)$ for $0 \leq j \leq n$. Compute the matrices \bar{S} , \bar{D} , and \bar{I} from row m down to row $imid$, and save $\bar{S}(imid, j)$ and $\bar{D}(imid, j)$ for $0 \leq j \leq n$. Let jd be a position at which the maximum score df is obtained. Let js be a position at which the maximum score st is obtained. If $df > st$, then set $jmid = jd$. Otherwise, set $jmid = js$. The algorithm is illustrated in Figure 3.2.

We first look at the space requirement of the algorithm. Since it requires only a few linear arrays to carry out the computation of the matrices, the algorithm requires space linear in the lengths of sequences. Next we look at the time requirement of the algorithm. Let $t(m, n)$ be the time required by the algorithm to compute an optimal alignment of two sequences of lengths m and n . If m is less than or equal to a constant c_1 , then a traceback procedure is used to compute an optimal alignment. Choose a constant c_2 such that

$$t(m, n) \leq c_2(m + n) \text{ for } m \leq c_1.$$

If m is greater than the constant c_1 , then an optimal alignment is computed by finding a pair of positions $imid$ and $jmid$ on the alignment and computing the portions before and after the pair recursively. The length of A_{imid} is $imid = \lfloor m/2 \rfloor$ and the length

of A_{imid}^s is $m - imid = \lceil m/2 \rceil$. Thus the time to compute an optimal alignment of A_{imid} and B_{jmid} is $t(\lfloor m/2 \rfloor, jmid)$ and the time to compute an optimal alignment of A_{imid}^s and B_{jmid}^s is $t(\lceil m/2 \rceil, n - jmid)$. Choose a constant c_3 such that the time on the non-recursive part of the algorithm is at most c_3mn . Thus we have

$$t(m, n) \leq c_3mn + t(\lfloor m/2 \rfloor, jmid) + t(\lceil m/2 \rceil, n - jmid) \text{ for } m > c_1.$$

It can be proved by induction that

$$t(m, n) \leq 2c_3mn + 2c_2(m + n).$$

This means that the algorithm takes time in proportion to the product of the sequence lengths.

3.2.4 Performing computation in a band

One approximation for reducing the time of the global alignment algorithm is to restrict the computation to a band of diagonals in each matrix (Sankoff and Kruskal, 1983; Pearson and Lipman, 1988). A diagonal k of a matrix consists of those entries (i, j) with $j - i = k$. A band from diagonals ld to hd consists of those entries with $ld \leq j - i \leq hd$. If sequences A and B are very similar, it is likely that an optimal alignment of A and B is completely within a narrow band of diagonals. To carry out the computation in a band of diagonals, each entry outside the band is given a value of negative infinity and each entry inside the band is computed according to the recurrences. Note that any band that covers an optimal alignment of A and B has to contain entries $(0, 0)$ and (m, n) . Later we describe a fast method to estimate the width of a band so that it is likely to cover an optimal alignment. However, the method does not guarantee that the band always covers an optimal alignment. Chao et al. (1992) developed an efficient algorithm for computing an alignment in a band. A few computational techniques were proposed to compute an optimal alignment in a band or a small matrix area (Fickett, 1984; Ukkonen, 1985; Spouge, 1991).

3.3 Local Alignment

The global alignment algorithm described above is intended for sequences that are similar over their entire lengths. However, there are situations where two sequences are not globally similar, but contain similar regions. For instance, genomic sequences from distantly related organisms contain short similar exons, but long different introns and intergenic regions. A local alignment algorithm should be used to find similar regions between two sequences. A local alignment between two sequences A and B is an alignment of a region of A and a region of B . An optimal local alignment between A and B is a local alignment with the maximum score.

An algorithm for computing an optimal local alignment between A and B is developed using dynamic programming. Define $LS(i, j)$ (L for local) to be the maximum score of local alignments ending at positions i and j of A and B . Similarly, define $LD(i, j)$ for alignments that end with a deletion gap and $LI(i, j)$ for alignments that end with an insertion gap. The recurrences for computing the matrices LS , LD , and LI are developed in a similar way as those for the matrices in the global alignment algorithm.

$$\begin{aligned}
 LS(i, j) &= 0 \text{ for } i = 0 \text{ or } j = 0, \\
 LS(i, j) &= \max\{0, LS(i-1, j-1) + \sigma(a_i, b_j), LD(i, j), LI(i, j)\} \\
 &\quad \text{for } i > 0 \text{ and } j > 0.
 \end{aligned}$$

$$\begin{aligned}
 LD(0, j) &= -q \text{ for } j \geq 0, \\
 LD(i, j) &= \max\{LD(i-1, j) - r, LS(i-1, j) - q - r\} \\
 &\quad \text{for } i > 0 \text{ and } j > 0.
 \end{aligned}$$

$$\begin{aligned}
 LI(i, 0) &= -q \text{ for } i \geq 0, \\
 LI(i, j) &= \max\{LI(i, j-1) - r, LS(i, j-1) - q - r\} \\
 &\quad \text{for } i > 0 \text{ and } j > 0.
 \end{aligned}$$

The zero in the recurrence for LS is the score of the empty local alignment, an alignment of two regions of length 0. The zero in the recurrence serves two purposes. First, an optimal local alignment can start at any positions i and j in sequences A and B . There is no penalty for not including, in the optimal local alignment, the

initial regions of A and B before positions i and j . Second, any local alignment of a negative score is ignored because it can not be an initial portion of any optimal local alignment. The justification for the recurrences is similar to that for the recurrences in the global alignment algorithm and is omitted.

An entry (ie, je) with the maximum value in the matrix LS is the end point of an optimal local alignment between A and B . The optimal local alignment can be found by a traceback procedure starting at the entry (ie, je) , which requires quadratic space. This algorithm is the result of Smith and Waterman (1981) and Gotoh (1982). Selecting an entry with the maximum value serves similar purposes to terminal regions of A and B as including the zero in the recurrence to initial regions of A and B . Those two features in the local alignment algorithm are responsible for generation of an optimal local alignment, instead of an optimal global alignment.

Alternatively, an optimal local alignment is computed in linear space by first determining its start point (is, js) and then applying the linear space global alignment procedure to sequences $a_{is}a_{is+1}\dots a_{ie}$ and $b_{js}b_{js+1}\dots b_{je}$. The end point (ie, je) is an entry with the maximum value in the matrix LS . The start point (is, js) is obtained by computing the matrices \bar{S} , \bar{D} , and \bar{I} with respect to sequences A_{ie} and B_{je} . Then (is, js) is an entry with the maximum value in the matrix \bar{S} .

Waterman and Eggert (1987) generalized the algorithm to compute k best local alignments between two sequences. Two local alignments are independent if they share no substitution from a common pair of positions in the sequences. A first best local alignment is an optimal local alignment between the two sequences. A second best local alignment is a largest-scoring local alignment that is independent of the first best local alignment. A third best local alignment is a largest-scoring local alignment that is independent of the first and second best local alignments. Other best local alignments are similarly defined. A space-efficient algorithm SIM was developed by Huang and Miller (1991). The SIM algorithm computes k best local alignments between two sequences in linear space.

3.4 A Fast Algorithm

The sequence alignment algorithms described in the previous sections take time in proportion to the product of sequence lengths. Thus it is impractical to use those alignment algorithms to compare very long sequences. Fast approximation algorithms are required to compare long sequences. Below we describe a fast algorithm to identify similar regions between two sequences and to produce an alignment for each pair of similar regions.

The fast algorithm consists of three major steps. In step 1, high-scoring segment pairs between the two sequences are computed. A segment pair is an alignment without any gaps. Segment pairs of scores greater than a cutoff are saved for the next step. In step 2, high-scoring chains of segment pairs are computed using dynamic programming and chains that begin with the same segment pair are grouped together. The score of a chain group is the maximum score of chains in the group. In step 3, for each chain group of score greater than a cutoff, a chain with the maximum score in the group is selected. The two sequence regions involved in the chain and a band of diagonals that covers all segment pairs in the chain are determined. Then the linear-space global alignment algorithm is applied to the two regions to compute a largest-scoring alignment of the regions over the band of diagonals. We define chains of segment pairs and describe computation of chains of segment pairs in detail below.

3.4.1 Chains of segment pairs

A segment pair between sequences A and B is a gap-free alignment of two segments of A and B . The score of a segment pair is the sum of scores of each match and mismatch in the segment pair. For a segment pair s , let $astart(s)$ and $aend(s)$ denote the starting and ending positions of the segment in sequence A , let $bstart(s)$ and $bend(s)$ denote the starting and ending positions of the segment in sequence B , and let $score(s)$ denote the score of s . The first antidiagonal of a segment pair s is defined to be $antis(s) = astart(s) + bstart(s)$, and the last antidiagonal of s is defined to be $antid(s) = aend(s) + bend(s)$.

A chain of segment pairs is a list of segment pairs in increasing order of their last antidiagonals such that each segment pair is not far from its predecessor and adjacent segment pairs do not have a large overlap. Specifically, any two adjacent segment pairs s and s' in the list satisfy the requirement

$$\begin{aligned}antis(s') - antid(s) &< d_1, \\ astart(s') - aend(s) &> -d_2, \text{ and} \\ bstart(s') - bend(s) &> -d_2\end{aligned}$$

for some nonnegative integers d_1 and d_2 . Let $close(s, s')$ denote the condition given above. A chain of segment pairs is used as an approximation of a local alignment between sequences A and B with the segment pairs being ungapped portions of the alignment. Note that the use of the d_1 cutoff permits efficient computation of high-scoring chains.

A linear gap penalty is charged for the regions between two adjacent segment pairs. For some nonnegative integers q and r , the penalty for connecting two segment pairs s and s' is

$$gap(s, s') = q + r \times [l(astart(s') - aend(s)) + l(bstart(s') - bend(s))],$$

where $l(x) = x$ if $x > 0$ and 0 otherwise. For two adjacent segment pairs s and s' in a chain, define $tscore(s, s')$ to be the score of the longest portion of s' that has no overlap with s . The score of a chain c of segment pairs s_1, s_2, \dots, s_k is defined to be

$$score(c) = score(s_1) + \sum_{i=2}^k [tscore(s_{i-1}, s_i) - gap(s_{i-1}, s_i)].$$

To ensure that each segment pair contributes to the chain, we require that for any two adjacent segment pairs s and s' in the chain, $tscore(s, s')$ be greater than a cutoff ic . Two segment pairs s and s' are identical if $astart(s) = astart(s')$ and $bstart(s) = bstart(s')$. Two chains of segment pairs are non-intersecting if they don't have any common segment pair (Chao and Miller, 1995).

3.4.2 Fast computation of chains of segment pairs

Segment pairs of scores greater than a cutoff between sequences A and B are approximately computed using a hashing technique as follows. Assume that $m \leq n$. A

lookup table is constructed for sequence A such that for each word of length w , the table provides the positions of each occurrence of the word in sequence A . The word length w is chosen such that the size of the lookup table is close to the length m of A . For each position p of sequence B , a word of length w beginning at position p of B is considered as follows. The lookup table is used to locate each occurrence of the word in sequence A . Each exact match of length w is extended in both directions until the score drops below the maximum score by at least d_3 units (Altschul et al., 1990). If a word match is contained in a segment pair already considered, the match is not extended. The segment pair of the maximum score found during the extension is saved if the score is greater than the cutoff.

After the computation of segment pairs between A and B , non-intersecting chains of segment pairs with scores greater than a cutoff f are computed. Let s_1, s_2, \dots, s_k be a list of all the segment pairs in increasing order of their last antidiagonals. Let $Q(s_i)$ be the maximum score of chains ending with segment pair s_i . The matrix Q is computed using dynamic programming (Wilbur and Lipman, 1983; Pearson and Lipman, 1988; Chao and Miller, 1995; Huang, 1996).

$$Q(s_1) = score(s_1),$$

$$Q(s_i) = \max\{score(s_i), Q(s_j) + tscore(s_j, s_i) - gap(s_j, s_i) \mid 1 \leq j < i, close(s_j, s_i), \text{ and } tscore(s_j, s_i) > ic\} \text{ for } i > 1.$$

For segment pairs s_j and s_i with $j < i$, if the overlap cutoffs d_1 and d_2 are violated or the score of the nonoverlapping portion of s_i is not large enough, then s_j is excluded from consideration as an immediate predecessor to s_i in any chain. To compute $Q(s_i)$, it suffices to use each s_j in decreasing value of j such that $antid(s_j) > antis(s_i) - d_1$. To compute $tscore(s_j, s_i)$ efficiently for each s_j , an array R of size d_2 is computed for s_i before $Q(s_i)$, where for $0 \leq t < d_2$, $R(t)$ is the sum of the scores of the first $t + 1$ aligned pairs in s_i if there are at least $t + 1$ aligned pairs in s_i and $score(s_i)$ otherwise. Let $aover(s_j, s_i)$ denote $astart(s_i) - aend(s_j)$ and let $bover(s_j, s_i)$ denote $bstart(s_i) - bend(s_j)$. Then for each s_j , if $aover(s_j, s_i) > 0$ and $bover(s_j, s_i) > 0$, then $tscore(s_j, s_i)$ is equal to $score(s_i)$. Otherwise, we have

$$tscore(s_j, s_i) = score(s_i) - R(\max\{-aover(s_j, s_i), -bover(s_j, s_i)\}).$$

Largest-scoring chains of segment pairs are partitioned into equivalence classes by the starting segment pair of the chains (Huang and Miller, 1991; Chao and Miller, 1995). Two chains are in the same class if and only if they begin with the same segment pair. The score of an equivalence class is the maximum score of chains in the class.

Chain classes of scores greater than f can be easily computed along with the matrix Q as follows (Huang and Miller, 1991; Chao and Miller, 1995). Let $K(s_i)$ be the first segment pair of a largest-scoring chain ending with segment pair s_i . For each segment pair s_i , $K(s_i)$ is initialized to s_i . When $Q(s_i)$ is set to $Q(s_j) + tscore(s_j, s_i) - gap(s_j, s_i)$, $K(s_i)$ is set to $K(s_j)$. For an equivalence class c , let $start(c)$ be the starting segment pair for the class, let $end(c)$ be the ending segment pair of a largest-scoring chain in the class, and let $score(c)$ be the score of the class. Thus we have $Q(end(c)) = score(c)$. The equivalence classes of scores greater than f are saved. After $Q(s_i)$ and $K(s_i)$ are computed, we perform one of the two tasks below if $Q(s_i)$ is greater than f . If there is an equivalence class c with $start(c) = K(s_i)$, set $end(c)$ to s_i and $score(c)$ to $Q(s_i)$ if $score(c) < Q(s_i)$. If there is no equivalence class c with $start(c) = K(s_i)$, create a new class c with $start(c) = K(s_i)$, $end(c) = s_i$, and $score(c) = Q(s_i)$. After the computation of the equivalence classes is completed, for each saved equivalence class, a largest-scoring chain in the class is obtained by a traceback technique. These largest-scoring chains are non-intersecting. To see this, if two chains were intersecting, i.e., they had a common segment pair s , then the two chains would begin with the same segment pair $K(s)$ and hence would belong to the same equivalence class. This contradicts the fact that the two chains are from different equivalence classes.

3.5 An Algorithm for Comparing Two Sets of Sequences

We consider a general problem of comparing every sequence in one set with every sequence in the other set. The goal is to find pairs of sequences with similar regions between the two sets and to report those similar regions. If one set is a large database of sequences and the other set is a set of query sequences, then the problem is a database searching problem.

We develop an efficient algorithm for this general problem as follows. All sequences in the smaller set are concatenated to form a composite string with a new character inserted at every sequence boundary (Huang and Madan, 1999). One lookup table is made for the composite string. For each sequence in the larger set, the fast algorithm in the previous section is used to compare the sequence with the composite string through the lookup table. Special care is taken to ensure that no word match is extended beyond any sequence boundary indicated by the new character in the composite string and that only segment pairs from the same sequence in the composite string can be combined into chains. Note that the construction of one lookup table for the composite string enables us to find directly pairs of sequences with similar regions without going through every pair of sequences from the two sets, many of which may not contain similar regions.

3.6 Applications

We present four applications of sequence alignment programs to analysis of DNA and protein sequences. First, we give an example of using a global alignment program to compare homologous human and mouse protein sequences. Second, we look at a case of using a global alignment program to compare syntenic human and mouse genomic DNA sequences. Third, we provide an example of using a fast comparison program and a rigorous alignment program to identify the exon-intron boundaries of genes in a genomic DNA sequence. Fourth, we give an instance of using a fast comparison

program to determine the similarity relationships between two large sets of sequences. All the applications were performed on a Sun Ultra 5 workstation with 128 Mb of main memory.

3.6.1 Comparison of two protein sequences

A novel gene named *Usp29* was recently found in a region of mouse chromosome 7 and a homologous region of human chromosome 19 (Kim et al., 2000). The cDNA sequence of mouse gene *Usp29* encodes a protein of 869 amino acids (GenBank accession no. AF229257). Because the sequence of the mouse protein is similar to the sequences of yeast and nematode proteins from the type-2 family of ubiquitin carboxyl-terminal hydrolases, the mouse protein is likely to function as a ubiquitin carboxyl-terminal hydrolase and is therefore named *Usp29* (ubiquitin-specific processing protease 29). (Ubiquitin carboxyl-terminal hydrolase is also known as ubiquitin-specific processing protease.) Proteins in the type-2 family contain two conserved domains named the cys box and the his box, which define the active sites of those proteins. The cDNA sequence of human gene *Usp29* encodes a protein of 922 amino acids (GenBank accession no. AF229438). Two questions could be asked about the mouse and human proteins. What is the level of overall sequence conservation between the mouse and human proteins? Are the two conserved domains that are unique to the type-2 family highly conserved between the mouse and human proteins?

The two questions were addressed by computing an optimal alignment of the mouse and human protein sequences with a program named GAP (global alignment program). The GAP program computes an optimal global alignment of two sequences in quadratic time and linear space, where terminal gaps are not penalized and long internal gaps in the shorter sequence are given a constant penalty (Huang, 1994). An alignment of the two mouse and human *Usp29* sequences was produced by GAP with the following values for its parameters: BLOSUM62 for substitution matrix, 15 for gap open penalty, and 2 for gap extension penalty. The running time of GAP was less than a second. The alignment showed that the two sequences have a low percent

identity of 41%, well below an average percent identity of 85% between human and mouse protein sequences. However, the two conserved domains are highly conserved between the mouse and human proteins (Figure 3.3). The high level of sequence conservation between the two domains of the mouse and human proteins also suggests that the mouse and human proteins belong to the type-2 family.

3.6.2 Comparison of two genomic sequences

We look at an example of comparing two large genomic sequences from syntenic regions of the human and mouse genomes. The number, order, and orientation of genes in syntenic regions of two different species are conserved between the two species. The 223-kb human genomic sequence (GenBank accession no. U47924) is from a gene-rich cluster at the *CD4* locus on human chromosome 12p13 (Ansari-Lari et al., 1996). The 227-kb mouse genomic sequence (GenBank accession no. AC002397) is from the syntenic region on mouse chromosome 6 (Ansari-Lari et al., 1998). The two *CD4* sequences were previously compared with a modified version of SIM program by Ansari-Lari et al. (1998). In this application, we show that coding regions in the two *CD4* sequences can be identified by computing a global alignment of the two sequences.

A program named GAP3 was used to compare the *CD4* genomic sequences. The GAP3 program computes an optimal global alignment of two sequences in quadratic time and linear space, where long, different regions in the two sequences are given a constant penalty (Huang, unpublished results). To align the two *CD4* sequences on the basis of coding regions, instead of repeat elements, the repeat elements in the *CD4* genomic sequences were masked by RepeatMasker (Smit and Green, 1996) and the masked versions of the sequences were used by GAP3 for alignment. The GAP3 program produced a large alignment of the two sequences, which contains 46,019 base matches (20%). Many of the matching regions on the alignment correspond to exons of the sequences. Portions of the alignment corresponding to two exons are shown in Figure 3.4. The following values were used for the parameters of GAP3: 10 for

match score, -15 for mismatch score, 60 for gap open penalty, and 3 for gap extension penalty. The computation took 4 hours and 39 minutes and 6.7 Mb of main memory.

3.6.3 Identification of exon-intron boundaries

In this application, we demonstrate that sequence alignment programs are useful for finding the exon-intron boundaries of a gene in a genomic sequence if the cDNA sequence of the gene is known. The genomic sequence used in this example is the *CD4* mouse genomic sequence from the last subsection. The mouse genomic sequence contains a gene whose cDNA sequence was determined eight years before. The cDNA sequence (GenBank accession no. NM_013509) encodes a protein of 434 amino acids, which functions as a gamma enolase. The mouse genomic sequence was compared with the cDNA sequence by a software tool named AAT (analysis and annotation tool) (Huang et al., 1997).

The AAT tool contains a fast database search program named DDS (DNA-DNA search) and a rigorous alignment program named GAP2 for comparing a genomic sequence with a database of cDNA sequences. The DDS program quickly computes high-scoring chains of segment pairs between the genomic sequence and the database of cDNA sequences. Every high-scoring chain indicates that a region of the genomic sequence is similar to a cDNA sequence. For each pair of a genomic region and a cDNA sequence, the GAP2 program computes an optimal alignment of the genomic region and the cDNA sequence. The GAP2 program is an improvement to the GAP program, where dinucleotides AG and GT are used by GAP2 to identify exon-intron boundaries.

On the *CD4* mouse genomic sequence and the cDNA sequence, DDS reported a high-scoring chain between a region of the *CD4* sequence from bases 129,471 to 137,445 and the cDNA sequence. The GAP2 program produced a 8,424-bp alignment of the genomic region and the cDNA sequence, where eleven exons of the genomic region are aligned with portions of the cDNA sequence. Portions of the alignment are shown in Figure 3.5. The exon-intron boundaries identified by GAP2 in the genomic

region are exactly identical to those reported in the GenBank entry of the CD4 mouse sequence. The DDS program took less than a second and the GAP2 program took 22 seconds on the data. The default values were used for the parameters of DDS and GAP2. Note that in this application, the database just contains one cDNA sequence. In a real situation, the database contains all cDNA sequences that have been produced.

3.6.4 Comparison of two sets of sequences

We describe an application of a program to comparison of two large sets of sequences. We developed a version, named DDS.BTAB, of the DDS program (Huang et al., 1997) for comparing two sets of sequences. The DDS.BTAB program quickly computes high-scoring chains between sequences in one set and sequences in the other set. The DDS.BTAB program was applied to comparison of two sets of sequences produced by two DNA sequence assembly programs. The two assembly programs were used to assemble the same set of raw DNA fragments into long sequences. One program produced a set of 47 sequences of a total of 1.9 megabases and the other program produced a set of 623 sequences of a total of 2.2 megabases. Obviously, the results from the two assembly programs were quite different. We wanted to know major differences between the two assembly results by finding major similarities between the two sets of sequences. The DDS.BTAB program was used to compute major similarities between the two sets of sequences.

The DDS.BTAB program produced 286 chains of scores greater than 2000, where a match was given a score of 2, a mismatch a score of -3 , a gap was penalized with a gap open penalty of 10 and a gap extension penalty of 1, and segment pairs of scores greater than 80 were used. A high value of 2000 was used for the chain score cutoff in order for DDS.BTAB to report only significant matches. The computation took 68 seconds. The word length used in this run was 11. Those major matches between the two sets of sequences allowed us to figure out the relationships between the two sets of sequences.

3.7 Future Developments

We suggest two directions for developments of new and improved sequence comparison methods. One direction is to improve existing methods so that they can distinguish distantly related sequences from unrelated sequences. If two related protein sequences have a percent identity of 30% or higher, then existing methods can determine that the two sequences are related. On the other hand, if two protein sequences have a percent identity of 25%, then existing methods can not determine if the two sequences are related or not. The other direction is to develop new methods for comparing two large genomes such as the human and mouse genomes. The immediate objectives of the genome comparison are to identify conserved coding regions and regulatory elements between the two genomes. Exons are often conserved between the human and mouse genomes, while introns and intergenic regions are often divergent. The new methods must be efficient enough to handle huge sequences and have new features to address various issues.

3.8 Acknowledgments

I would like to thank the reviewers and editors for many helpful suggestions on the presentation of this chapter.

Bibliography

- [1] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J. Basic local alignment search tool. *J. Mol. Biol.*, 215, 403-410, 1990.
- [2] Ansari-Lari, M.A., Muzny, D.M., Lu, J., Lu, F., Lilley, C.E., Spanos, S., Malley, T. and Gibbs, R.A. A gene-rich cluster between the CD4 and triosephosphate isomerase genes at human chromosome 12p13. *Genome Res.*, 6, 314-326, 1996.
- [3] Ansari-Lari, M.A., Oeltjen, J.C., Schwartz, S., Zhang, Z., Muzny, D.M., Lu, J., Gorrell, J.H., Chinault, A.C., Belmont, J.W., Miller, W. and Gibbs, R.A. Comparative sequence analysis of a gene-rich cluster at human chromosome 12p13 and its syntenic region in mouse chromosome 6. *Genome Res.*, 8, 29-40, 1998.
- [4] Attwood, T. K. The babel of bioinformatics. *Science*, 290, 471-473, 2000.
- [5] Chao, K.-M. and Miller, W. Linear-space algorithms that build local alignments from fragments. *Algorithmica*, 13, 106-134, 1995.
- [6] Chao, K.-M., Pearson, W.R. and Miller, W. Aligning two sequences within a specified diagonal band. *Comput. Appl. Biosci.*, 8, 481-487, 1992.
- [7] Dayhoff, M.O., Schwartz, R.M. and Orcutt, B.C. A model of evolutionary change in proteins. In Dayhoff, M. O. (ed.), *Atlas of Protein Sequence and Structure*. Vol. 5, Suppl. 3. National Biomedical Research Foundation. Washington, DC, pp. 345-358, 1978.
- [8] Fickett, J.W. Fast optimal alignment. *Nucleic Acids Res.*, 12, 175-180, 1984.

- [9] Gotoh, O. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162, 705-708, 1982.
- [10] Henikoff, S. and Henikoff, J.G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89, 10915-10919, 1992.
- [11] Hirschberg, D.S. A linear space algorithm for computing maximal common subsequences. *Commun. Assoc. Comput. Mach.*, 18, 341-343, 1975.
- [12] Huang, X. On global sequence alignment. *Comput. Appl. Biosci.*, 10, 227-235, 1994.
- [13] Huang, X. Fast comparison of a DNA sequence with a protein sequence database. *Microbial & Comparative Genomics*, 1, 281-291, 1996.
- [14] Huang, X., Adams, M.D., Zhou, H. and Kerlavage, A.R. A tool for analyzing and annotating genomic sequences. *Genomics*, 46, 37-45, 1997.
- [15] Huang, X. and Madan, A. CAP3: A DNA sequence assembly program. *Genome Res.*, 9, 868-877, 1999.
- [16] Huang, X. and Miller, W. A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math.*, 12, 337-357, 1991.
- [17] Kim, J., Noskov, V.N., Lu, X., Bergmann, A., Ren, X., Warth, T., Richardson, P., Kouprina, N. and Stubbs, L. Discovery of a novel, paternally expressed ubiquitin-specific processing protease gene through comparative analysis of an imprinted region of mouse chromosome 7 and human chromosome 19q13.4. *Genome Res.*, 10, 1138-1147, 2000.
- [18] Makalowski, W., Zhang, J. and Boguski, M.S. Comparative analysis of 1196 orthologous mouse and human full-length mRNA and protein sequences. *Genome Res.*, 6, 846-857, 1996.
- [19] Myers, E.W. and Miller, W. Optimal alignments in linear space. *Comput. Applic. Biosci.*, 4, 11-17, 1988.

- [20] Needleman, S.B. and Wunsch, C.D. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48, 443-453, 1970.
- [21] Pearson, W.R. and Lipman, D. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85, 2444-2448, 1988.
- [22] Sankoff, D. and Kruskal, J.B. (eds) *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparisons*. Addison-Wesley, Reading, Massachusetts, 1983.
- [23] Sellers, P.H. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26, 787-793, 1974.
- [24] Smit, A.F.A. and Green, P. 1996.
<http://ftp.genome.washington.edu/cgi-bin/RepeatMasker>.
- [25] Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. *J. Mol. Biol.*, 147, 195-197, 1981.
- [26] Spouge, J.L. Fast optimal alignment. *Comput. Applic. Biosci.*, 7, 1-7, 1991.
- [27] Ukkonen, E. Algorithms for approximate string matching. *Information and Control*, 64, 100-118, 1985.
- [28] Wagner R.A. and Fischer, M.J. The string-to-string correction problem. *J. Assoc. Comput. Mach.*, 21, 168-173, 1974.
- [29] Waterman, M.S. and Eggert, M. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *J. Mol. Biol.*, 197, 723-728, 1987.
- [30] Waterman, M.S., Smith, T.F. and Beyer, W.A. Some biological sequence metrics. *Adv. Math.*, 20, 367-387, 1976.
- [31] Wilbur, W.J. and Lipman, D.J. Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl. Acad. Sci. USA*, 80, 726-730, 1983.

3.9 Figure Legends

Figure 3.1 A grid graph. (A) An overview of the grid graph. (B) A detailed view of four adjacent entries in the graph. The edges from D to S and from I to S have a score of 0. The score of each remaining edge is shown next to the edge.

Figure 3.2 Three pairs of positions on an optimal alignment and four subsubproblems produced by the alignment algorithm after two levels of division. An optimal alignment is indicated by a line of dots. In an initial call to the algorithm, a pair of positions i_1 and j_1 is determined and the original problem is divided into two subproblems. The time required by the non-recursive portion of the algorithm in the initial call is proportional to $m \times n$. The initial call makes two recursive calls, one for each subproblem. In each recursive call, a pair of positions is determined and the subproblem is further divided into two subsubproblems. The total time required by the non-recursive portion of the algorithm in the two calls is proportional to $(m \times n)/2$.

Figure 3.3 Portions of an alignment of mouse and human *Usp29* protein sequences. Two conserved domains are indicated by asterisks.

Figure 3.4 Portions of a large alignment of mouse and human CD4 genomic sequences. Two mouse exons are correctly aligned with two human exons with respect to exon-intron boundaries. The four exon-intron boundaries are indicated by asterisks under exon bases and plus signs under intron bases.

Figure 3.5 Portions of an alignment of a genomic region and a cDNA sequence. The cDNA sequence is correctly aligned with the genomic region with respect to exon-intron boundaries. The 5' and 3' coordinates of exons 9 and 10 are shown.