Sequence Alignment

6

6.1 Introduction

Sequence alignment is the process of comparing two or more sequences by searching for a series of characters that appear in the same order in these sequences. In DNA sequence alignment, we would be searching for an alignment of nucleotides, whereas amino acid sequences are aligned in proteins. Using sequence alignment, similar segments of DNA, RNA, or proteins can be identified which may indicate functional, structural, or evolutionary relationships between these sequences [23].

The general aim of any sequence comparison method in bioinformatics is to determine whether the similarities between two or more sequences is incidental or they are derived from a common ancestral sequence in which case they are *homologous*. Homology indicating a common ancestor may reveal a common function or structure. Homologous genes, for example, are derived from the same ancestral gene which is altered due to a number of mutations. Homology is displayed more easily in protein amino acid sequences than the DNA nucleotide sequences. This is due to the smaller alphabet of 4 nucleotides (A, C, G, T) in DNA sequences compared to 20 amino acids in proteins. That is, the chances of finding matches in DNA sequences are greater than finding amino acid matches in protein sequences. Also, different codons in DNA encode for the same amino acid and the 3D structure of a protein is determined by its amino acid sequence. In order to have the same functionality of a protein which is based on its 3D shape, the evolutionary process in a protein sequence is slower. In general, we would be interested in both the alignment of DNA/RNA nucleotide sequences and protein amino acid sequences.

Sequence comparison also allows finding evolutionary relationships among organisms. This association is commonly used to construct *phylogenetic trees* and *networks* which display ancestor–descendant affinities and these structures can be used for a wide range of applications including disease analysis as we will see in Chap. 15. Also, the distances between the sequences as computed by sequence alignment algorithms are frequently input to sequence clustering algorithms which groups biological sequences based on their similarities as we review in Chap. 7. A *sequence*

K. Erciyes, *Distributed and Sequential Algorithms for Bioinformatics*,
Computational Biology 23, DOI 10.1007/978-3-319-24966-7_6

motif is a repeating DNA nucleotide or protein amino acid sequence which has a biological significance. Sequence comparison methods are used to discover these motifs as we will analyze in Chap. 8. In summary, the distances and similarities between biological sequences is required in various sequence analysis methods and the alignment methods are usually the first step that provides the needed input to all of these methods.

Two sequences are aligned in *pairwise alignment* and multiple sequences are aligned in *multiple alignment*. In *global alignment*, two homologous sequences of similar lengths are compared over their entire sequence. This method is used to find similarities of two closely related sequences. In many cases, however, only certain segments of two sequences may be similar but the rest of the sequences may be completely unrelated. For example, two proteins may consist of a number of domains and only one or two of these domains may be similar. The global alignment will not display a high similarity between these two proteins in this case. *Local alignment* refers to the method of finding similar regions in two sequences which may have very different lengths. Multiple sequence alignment can be performed by global alignment if the input sequences are closely related and we are searching for the similarity of these sequences as a whole; or local alignment in which case we are interested to find similar subsequences of otherwise not related sequences. Different types of alignment methods need different algorithms; however, they can be coarsely classified as dynamic programming based, heuristic or a combination of both in general.

In this chapter, we first state the sequence alignment problem, describe ways of evaluating goodness of any alignment method and then analyze representative sequential global, local and multiple sequence alignment algorithms in detail. We then provide parallel/distributed algorithms aimed to solve these problems and review current research in this area.

6.2 Problem Statement

Sequence alignment is the basic and most fundamental method of comparing two biological sequences. In the very common application of such alignment, we have an input sequence called the *query* that needs to be identified since it is newly discovered or not aligned before; and this query is typically aligned with each sequence in a database of sequences. The sequences in the database that have the highest scores are then identified as the ones having highest similarity and therefore relatedness to the query sequence. This affinity in base structures may imply phylogenetic relationships and also similar functionality to aid the analysis of the newly discovered sequence.

6.2.1 The Objective Function

We need to asses the quality of an alignment which reflects its goodness. The cost-benefit approach identifies three scores during alignment:

- The benefit of aligning two identical characters (*match*)
- The cost of aligning two different characters (*mismatch* or *substitution*)
- The cost of aligning a character in a sequence with a gap in the other sequence (insertion or deletion-*indel*). The first sequence has a gap in the related column in insertion, and the second sequence has a gap in the corresponding column in deletion.

Insertion and deletion of gaps refer to the operations on the first sequence, that is, insertion/deletion means inserting/deleting a gap to/from the first sequence. An alignment between two DNA sequences X and Y is shown below with matches, mismatches, insertions, and deletions.

```
position        1  2  3  4  5  6  7  8  9  10  11
X:              G  A  G  T  A  -  C  -  G   C   T
Y:              A  A  G  -  A  G  C  T  -   G   T

positions of comparison :
matches     : 2, 3, 5, 7 and 11
mismatches  : 1 and 10
insertions  : 6 and 8
deletions   : 4 and 9
```

A positive score is associated with a match and negative scores are used to penalize a mismatch and an indel. The negative scores or penalties are based on observed statistical occurrences of an indel and a mismatch and typically, the indels are penalized more, reflecting their relatively less prevalence in the genome alignment. As an example, let us use the scores +2 for a match, –1 for a mismatch and –2 for an indel. Given the two DNA sequences X = ATGGCTACAC and Y = GTGTACTAC, we can have various alignments four of which are shown with mismatches (m) and indels (i) marked. Among these four options, the alignment in (a) or (b) should be chosen as they both have the highest scores.

```
A T G G C T A C - A C      A T G G C T A C -  A C
G T - G - T A C T A C      G T G - - T A C T  A C
m   i   i       i          m     i i       i
   (a) Score = 7              (b) Score = 7

A T G - G C T A C - A C     A T G - G C T A G A C
- - G T G - T A C T A C     G T G T A C T - - A C
i i   i   i       i        m     i m     i  i
   (c) Score = 4              (d) Score = 4
```

The aim of any alignment method is to maximize the total score. However, there are exponential number of combinations to check and if we can find an alignment that has a higher score than others, then using it should be preferred. Formally, alignment of two sequences can be defined as follows:

Definition 6.1 (*sequence alignment*) Let $\sum_{\text{org}}$ be an alphabet and $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$ be two sequences over this alphabet and let $\sum \leftarrow \sum_{\text{org}} \cup \{-\}$, that is, the space character added to the original alphabet. An alignment of these two sequences is a two-row matrix where the first row are the elements of X and the second row are the elements of Y and each row contains at least one element of Σ_{org}.

A related parameter between two sequences is the *edit distance* between them which is defined as follows:

Definition 6.2 (*edit distance*) Edit distance, or the Levenshtein distance, is the minimum number of substitutions, insertions, and deletions between two sequences. Hamming distance is an upper bound on edit distance.

For the above example, the edit distance between the two sequences is 4 which occurs in (a). The procedure for sequence alignment is very similar to finding LCS between them; however, we now have costs associated with matches, mismatches, and indels and search for the highest scoring alignment.

6.2.2 Scoring Matrices for Proteins

Proteins consist of a sequence of amino acids from a 20-letter alphabet. Some mismatches are more likely to occur in proteins and they are more frequently encountered than other substitutions. This fact necessitates the use of a weighting scheme for each amino acid substitution. Scoring matrices for mismatches in protein amino acid sequences define the scores for each substitution in these sequences. The two widely used matrices for this purpose are the point accepted mutation (PAM) matrix [11] and the blocks substitution matrix (BLOSUM) [17]. They both use statistical methods and are based on counting the observed substitution frequency and comparison of this value with the expected substitution frequency.

A positive score in the entry m_{ij} of a PAM matrix M means that the probability of the substitution between i and j is more than its expected value; therefore, it bears some significance. The entry m_{ij} is formed by considering the expected frequencies of i and j, and the frequency of alignment between i and j in the global alignment of homologous sequences [4]. The nth power of M is then taken to form the PAM-n matrix such as PAM-80, PAM-120, or PAM-250. A large value of n should be used to align proteins that are not closely related.

PAM may not provide realistic values for remotely related protein sequences as it uses extrapolation of values. BLOSUM matrix structure proposed by Henikoff and Henikoff [17] overcomes this difficulty by analyzing segments of proteins rather than the whole. If two segments of proteins under consideration have similarity over a threshold value, they are clustered. The threshold value t is specified as BLOSUM-p which means the matrix is generated by combining sequences which have at least t % similarity. The BLOSUM62 matrix which which is formed by clustering sequences that have at least 62 % identity level is shown below. A small value of t is used for distantly related protein sequences and more closely related ones can be aligned using a larger value.

```
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  4 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1  0 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -2 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -2 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0  0 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1  0 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -2 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
X  0 -1 -1 -1 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2  0  0 -2 -1 -1 -1 -1 -1 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```

6.3 Pairwise Alignment

Pairwise alignment is the comparison of two distinct DNA or protein sequences. It can be performed globally to compare two sequences as a whole or locally to detect similar subsequences in the two sequences as outlined below.

6.3.1 Global Alignment

Global alignment assumes that the sequences to be compared are sequentially homologous and attempts to align all of the sites optimally within the sequences.

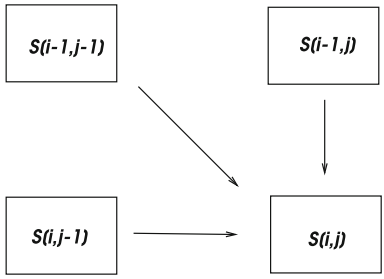


Fig. 6.1 An entry $S[i, j]$ of the alignment matrix

An *alignment matrix* S is a convenient way of displaying the alignment between two sequences. In order to represent two sequences X and Y of lengths n and m, S contains n rows and m columns. When aligning two sequences, we can have four options. The characters match; they do not match; a gap is inserted in the first sequence; or a gap is inserted in the second sequence. The filling of the alignment matrix is based on selecting the option which gives the highest score. An entry $S[i, j]$ of an alignment matrix depends on the values of the entries just before it in the preceding column, row, and diagonal as shown in Fig. 6.1. The first row and column of this graph are initialized with the gap penalties when these occur at the beginning of sequences X and Y.

We can therefore compute the value of the element (i, j) by checking the three previous entries at $(i - 1, j)$, $(i - 1, j - 1)$, and $(i, j - 1)$. The first value of the array is initialized to 0. Finding all other entries can be done using the dynamic programming approach where subsolutions are used to find the solution. Needleman and Wunsch provided the first dynamic algorithm for this purpose as described next.

6.3.1.1 Needleman–Wunsch Algorithm

Let us specify the scores for each character comparison. A match is given a score α, a mismatch a β and an indel γ when two characters a_i and b_i are compared, as shown below. The match score is commonly positive as this is what we require and the other scores are negative.

$$\text{score}(a_i, b_i) = \begin{cases} \alpha, a_i = b_i \\ \beta, a_i \neq b_i \\ \gamma, a_i =' -' \text{ or } b_i =' -' \end{cases}$$

Let us assume an optimal alignment A between two sequences X and Y, and $A' \subset A$ as an alignment of $X' \subset X$ and $Y' \subset Y$. If A is optimal, A' is also an optimal alignment which leads to the following dynamic algorithm solution. We consider the prefixes $X_i = x_1 \ldots x_i$ and $Y_j = y_1 \ldots y_j$ of two sequences. In order to find the optimal alignment, we need to select the one that gives the highest score from the following:

1. Align x_i with y_i and X_{i-1} with B_{j-1}
2. Align x_i with a gap and X_{i-1} with B_j
3. Align y_j with a gap and X_i with B_{j-1}.

The first case shows whether there is a match or a mismatch and δ is equal to α when there is a match, and β when a mismatch occurs. Aligning a prefix X_i of X with no element of Y is the product of the length of X_i with the gap penalty and the same is valid for Y. Therefore, $M[i, 0] = \gamma i$ and $M[0, j] = \gamma j$. Needleman–Wunsch algorithm brings together all of the concepts we have discussed until now in the dynamic programming based algorithm as shown in Algorithm 6.1.

Algorithm 6.1 *NW_Alg*

Input : Sequences $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m\}$
Output : Array $S[n, m]$
$S[0, 0] \leftarrow 0$
for $j = 1$ to n **do**
    $S[0, j] \leftarrow \gamma \times j$
end for
for $i = 1$ to m **do**
    $S[i, 0] \leftarrow \gamma \times i$
end for
for $j = 1$ to n **do**

$$M[i, j] = max \begin{cases} M[i-1, j] + \gamma \\ M[i, j-1] + \gamma \\ M[i-1, j-1] + \delta(a_i, b_j) \end{cases}$$

end for

Let us form the alignment array M for two given DNA sequences $X =$ ACCGT and $Y =$ AGCCTC with $n = 5$ and $m = 6$; and we select $\alpha = 2$, $\beta = -1$ and $\gamma = -1$ for simplicity. We first fill the first row and column of M with gap penalties and assign 0 value for the first entry. We then form each entry using the dynamic programming relation in the NW algorithm to obtain the final array as shown below.

		A	G	C	C	T	C
	0	-1	-2	-3	-4	-5	-6
A	-1	2	1	0	-1	-2	-3
C	-2	1	1	3	2	1	0
C	-3	0	0	3	5	4	3
G	-4	-1	2	2	4	4	3
T	-5	-2	1	1	3	6	5

The global alignment problem can now be reduced to finding the best scoring path between vertices $M[0, 0]$ and $M[n, m]$. We can now start from the lowest right corner of M and work our way upwards until we reach $M[0, 0]$ by following in reverse direction of the path we have chosen while filling the array. An up arrow means a gap in the top sequence, a left arrow represents a gap in the second sequence on the lefthandside, and a diagonal arrow shows a match or a mismatch without any gap in that position. The alternative paths result in alternative alignments with the same score. Implementing this procedure for the above example yields the following alignment with a score of 5 (4 matches and 3 indels).

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

The time to fill the array is $O(nm)$ which is the size of the array and hence the space requirement is the same. At the end of the algorithm, best alignment score is stored in $M[n, m]$.

6.3.2 Local Alignment

The global alignment may not provide the correct results because of the genome shuffling and rearrangements. A segment of a sequence inversion may happen in a sequence causing a subsequence to look radically different. Local alignment provides us information about the conserved subsequences within organisms. A local alignment between two sequences with four matches and a mismatch is shown in bold below.

A T G **C T A G T** G C C
G C A **C T T G T** A A T

As a general rule, subsequences of the sequences are aligned separately without considering the general order of the global sequences in local alignment. Given two sequences $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$, some parts of X and Y may be aligned with high scores by local alignment but the remaining subsequences may be very different. One way of tackling this problem is to identify all possible subsequences of X and Y and then compute all global alignments between every pair of these subsequences and select the one with the highest score. Unfortunately, this brute force algorithm can find subsequences of X and Y in $O(n^2)$ and $O(m^2)$ times, and performing global alignment results in complexity of $O(n^2m^2nm)$ or $O(n^3m^3)$ which is unacceptable for large sequences. A commonly used algorithm for local alignment was proposed by Smith and Waterman [31] which is an adaptation of the NW algorithm for local alignment. The main differences between these two

algorithms are as follows. First, a fourth value of zero is allowed in addition to the three possible values in the NW algorithm to prevent negative values in the alignment graph. The first row and column of the array M contain zeros now to discard gaps occurring in the beginning of sequences. We still attempt to find a path with maximum value but we do not have to start from the beginning to allow for local alignments. Instead, we start with the maximum value of the array and stop when a zero is encountered which signals the end of the regional alignment. Algorithm 6.2 shows the pseudocode of this algorithm.

Algorithm 6.2 *SW_Alg*

1: **Input** : Sequences $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m\}$
2: **Output** : Array $S[n, m]$
3: $S[0, 0] \leftarrow 0$
4: **for** $j = 1$ to n **do**
5: $\quad S[0, j] \leftarrow 0$
6: **end for**
7: **for** $i = 1$ to n **do**
8: $\quad S[i, 0] \leftarrow 0$
9: **end for**
10: **for** $j = 1$ to n **do**

$$M[i, j] = max \begin{cases} M[i-1, j] + \gamma \\ M[i, j-1] + \gamma \\ M[i-1, j-1] + \delta(a_i, b_j) \\ 0 \end{cases}$$

11: **end for**

As an example, given two DNA sequences X = GGATACGTA and Y = TCATACT with $n = 9$ and $m = 7$, and scoring as before with $\alpha = 2$, $\beta = -1$ and $\gamma = -1$; we proceed similar to the global alignment algorithm by first initializing the first row and column of the similarity matrix M. We then form each entry using the SW algorithm and fill the entries of M, keeping the track of the selected path as in NW algorithm. Backtracking starts from the highest value element of M this time, stopping whenever a zero is encountered as shown below.

We start with the maximum value entry 9 and backtrack the path until a 0 is encountered. The obtained maximum local alignment between these two sequences is shown below in bold.

G G **A T A C G T** A
T C **A T A C - T**

	G	G	A	T	A	C	G	T	A
	0	0	0	0	0	0	0	0	0
T	0	0	0	2	← 1	0	0	2	1
C	0	0	0	1	1	3	2	1	1
A	0	0	2	1	3	2	2	1	3
T	0	0	1	4	3	2	1	4	3
A	0	0	2	3	6	5	4	3	6
C	0	0	1	2	5	8	← 7	6	5
T	0	0	0	3	4	7	6	9	8

If we start with 6 shown in the upper path, a local alignment with one mismatch and one gap, and a score of 6 is obtained as follows.

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

We can have various local alignments between these two sequences using alternative paths, starting with the next largest value and continuing until a 0. The time and space complexities of this algorithm are $O(nm)$ as in NW algorithm since we need to fill the alignment matrix as before.

6.4 Multiple Sequence Alignment

Multiple sequence alignment (MSA) aims at aligning more than two biological sequences. We have a set of k input sequences $\mathcal{S} = S_1, \ldots, S_k$ and our aim is to provide alignment of these k sequences. We can have global and local multiple alignment in MSA. In theory, we can use NW algorithm for the global alignment of k sequences, for each possible pairs of a set of k sequences $\mathcal{S} = S_1, \ldots, S_k$ each with a length of n, invoking this algorithm $k(k-1)/2$ times. For large k, this method is inefficient due to its increased time complexity of $O(n^2k^2)$. For this reason, heuristics are widely used for MSA. The main methods of MSA can be stated as follows:

- *Exact methods*: These algorithms typically use dynamic programming outlined and have high running times and can be effective for only 3 or 4 sequences.
- *Approximation algorithms*: They have polynomial run times but only approximate the solution. However, performance is guaranteed to be within the approximation ratio.

- *Heuristic methods*: The algorithms based on heuristics typically search only a subset of the possible alignments and find an alignment that is suboptimal. There is no performance guarantee but they are widely used in practice.
- *Probabilistic methods*: They assume a probabilistic model and search alignment that best fits this model.

We will now take a closer look at representative algorithms for these methods starting with an approximation algorithm.

6.4.1 Center Star Method

The center star method is an approximation algorithm with a ratio of 2. The main idea of this algorithm is to identify a sequence which is closest to all others as the center and then work out the alignments of all sequences with respect to this center. There are various methods to measure the distances between the sequences. As a simple approach, we can find the consensus sequence S_{cs} of $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n input sequences which is the sequence containing the most frequent symbols ofeach column to be matched in each sequence. We can then work out the distance of each sequence S_i to S_{cs} and mark the sequence with shortest distance to S_{cs} as the central sequence S_c. Alternatively, we can compute pairwise distances between all pairs of sequences which is called the *sum of pairs distance* which is used in the center star method. The center star algorithm specifically consists of the following steps:

1. *Input*: A set $\mathcal{S} = \{S_1, \ldots, S_k\}$ of k sequences of length n each.
2. *Output*: MSA of sequences in $\mathcal{S}$.
3. Work out the distance matrix D between sequences such that d_{ij} entry of D is equal to the distance between S_i and S_j.
4. Find the center sequence S_c that has the minimum value of sum of pairs, $\sum_{i=1}^{k} d_{ij}$.
5. For each $S_i \in \mathcal{S} \setminus S_c$, find an optimal global alignment between S_i and S_c using Needleman–Wunsch algorithm.
6. Insert gaps in S_c to complete MSA.

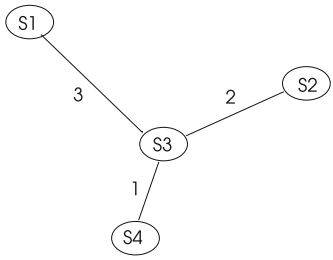
The center sequence is the one that is most similar to all other sequences. For example, given the four sequences below, we can find their pairwise similarities as shown.

```
S1: A C C G T G G C  S1: A C C G T G A T   S1: A C C G T G T T
S2: C G C C T C T T  S3: C A G G T C T G   S4: C G T A A T A G
       d=3                  d=3               d=5

S2: C G C C T C G A  S2: C G C C T C A G   S3: C G C C T C A G
S3: C A G G T C T A  S4: C G T A A T T A   S4: C G T A A T T A
       d=2                  d=4               d=1
```

Fig. 6.2 The star tree formation for four sequences

	S_1	S_2	S_3	S_4
S_1	0	3	3	5
S_2	3	0	2	4
S_3	3	2	0	1
S_4	5	4	1	0



The total number of comparisons is $k(k-1)/2$ times, 6 in this case, resulting in a total time complexity of $O(k^2n^2)$ for this step. We can then form the distance matrix D with these values and find the sequence that has the greatest similarity to all others as shown in Fig. 6.2. This step involves summing rows of the matrix D and detecting the sequence with the lowest sum, which is S_3 in this case, in $O(k^2n^2)$ time.

We now need to align sequences S_1, S_3, S_4 to the central sequence S_2 by the Needleman–Wunsch algorithm in $O(kn^2)$ time. Finally, gaps are inserted in the aligned sequences to complete the multiple sequence alignment $O(k^2n)$ resulting in a total time of $O(k^2n^2)$ since the first step dominates. It can be shown using the triangle inequality between three sequences that the approximation ratio of this algorithm is 2 [33].

6.4.2 Progressive Alignment

Progressive alignment methods employ heuristic algorithms to compute the MSA of a set of sequences. A general approach is to to align two closely related sequences and then progressively align other sequences. Typically in the first step, all possible pairwise alignments of k sequences for a total of $k(k-1)/2$ pairs are computed. A phylogenetic tree (see Chap. 14) that shows the evolutionary relationships based on their distances is then estimated and used as a guide to perform alignment. The most similar sequences that are close to each other in the phylogenetic tree are then pairwise aligned. CLUSTALW is one such widely used global progressive alignment tool that can be used for both DNA and protein sequence analysis [32]. It performs the following steps:

1. Computation of all pairwise alignment scores and forming the distance matrix D based on these scores.
2. Construction of a phylogenetic tree T using D by the neighbor-joining (NJ) method (see Sect. 14.3).
3. Perform MSA with the sequences starting with the closely related ones in the tree.

The NJ algorithm proposed by Saitou and Nei [29] is basically a hierarchical clustering algorithm that iteratively groups closely related input data which are also farthest to the rest of inputs. The CLUSTALW algorithm then iteratively performs

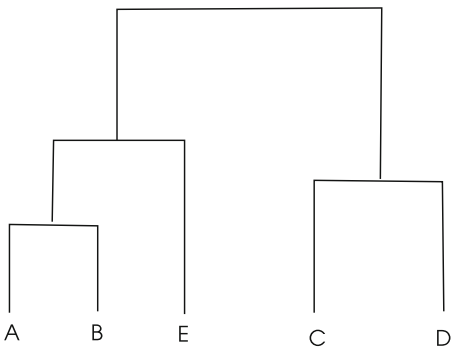


Fig. 6.3 A phylogenetic tree which has the five input sequences *A, B, C, D*, and *E* as its leaves. Sequences *A* and *B* are first aligned as they are closest, and then sequences *C* and *D* are aligned followed by the alignment of the resulting sequence with *E*. Finally, the resulting two sequences are aligned

pairwise sequence alignment with the closest sequences in the tree. Figure 6.3 displays an example phylogenetic tree where the input sequences are the leaves of this tree.

The NJ algorithm in general will produce unrooted trees where the input sequences may not be equidistant to their ancestors. The root in this tree is placed in a location from which the average distances on its both sides are equal. The CLUSTALW algorithm will use this tree to pairwise align the closest sequences as guided by the tree. Starting from the leaves, the closest leaves are aligned iteratively to form larger clusters at each step. Progressive alignment methods introduce significant errors when sequences are distantly related. Also, the guide tree is formed using pairwise alignments which may not reflect the evolutionary process accurately.

6.5 Alignment with Suffix Trees

Suffix trees can be used for global sequence alignment. A fundamental method used for this purpose is called *anchoring* in which similar regions called *anchors* in two sequences are first identified using suffix trees. The segments between the anchors are then aligned using dynamic programming or using the same method recursively or a combination of both approaches.

MUMmer is one such algorithm that uses suffix trees for extracting maximal unique matches (MUMs) that are used for anchoring [12]. Given two sequences X and Y of lengths n and m, a MUM is a subsequence of both X and Y of length greater than a given threshold d. A MUM of X and Y has to be unique in both of them. A brute force algorithm needs to search all possible prefixes of both strings in $O(nm)$ time. However, this problem can be simplified by the aid of a generalized suffix tree. We need to build a generalized suffix tree for the two strings and search for internal nodes that have exactly two leaves, one from each sequence. We then check whether the node representing the substring is maximal. If this condition is satisfied, the prefix starting from the root and ending at the internal node represents a MUM. This algorithm takes $O(n + m)$ steps which is the time to construct the generalized suffix tree and also the time for other steps.

The order of MUMs is also conserved between related genomes, and therefore we can predict that the conserved regions in two biological sequences contain ordered MUMs rather than randomly distributed MUMs. The idea behind the MUMmer algorithm is this observation and it attempts to find these conserved regions by finding the longest common subsequence (LCS) of them [12]. The LCS problem can be solved by the dynamic programming algorithm we reviewed in Sect. 5.4 in $O(n^2)$ time and $O(n^2)$ space or by using generalized suffix trees. However, since each MUM is unique, it can be replaced by a special character allowing a solution in $O(n \log n)$ time [33]. The regions between the anchors are aligned using the Needleman–Wunsch algorithm. Multiple genome aligner (MGA) is a tool for multiple sequence alignment based on suffix trees [18]. The longest nonoverlapping sequence of maximal multiple exact matches (multiMEMs) are computed and then used to guide the multiple alignments in this algorithm. The LAGAN [5] is another tool based on anchoring; however, it uses the CHAOS local alignment algorithm and uses the local alignments produced by CHAOS as anchors limiting the search area of the Needleman–Wunsch algorithm around these anchors [6]. LAGAN provides the visual display of alignment results.

6.6 Database Search

It is of interest to compare a newly discovered biological sequence against many other existing ones in databases to find its affinity to them, and therefore to predict and compare the functionality of the new sequence. The sequences deciphered using modern sequencing techniques have increasingly large sizes making it difficult to align them using the SW or NW algorithms which have $O(nm)$ time complexities. The focus of the research studies have then been the design of algorithms that use heuristics and provide approximate but fast solutions. There are many sequence alignment tools for this purpose two of which are more commonly used than others and we will describe them briefly.

6.6.1 FASTA

FASTA is an early local pairwise sequence alignment tool for database comparison of a biological sequences [27]. Its predecessor was called FASTAP [21] and handled protein sequence alignment only, and since FASTA can search for both protein and DNA sequences, it was called FASTA (Fast-All). It is a heuristic algorithm that compares a given input query sequence against the sequences in a database. Its operation can be summarized as follows:

1. Given a query sequence Q and a set of sequences $\mathcal{S} = S_1, \ldots, S_n$ in the database, it searches for exact matches of length l between the query and a database sequence $S_i \in \mathcal{S}$. These matches are called *hotspots*. Commonly used values of values of

l are 2 for protein amino acid sequences and between 4–6 for DNA sequence comparisons.
2. The hotspots are combined into a long sequence called *initial regions*. These regions are scored using the similarity matrix M. Only a small part of M is aligned and the best scoring 10 alignments are considered for the next step.
3. Using dynamic programming, the ten best partial alignments are combined to give a longer alignment.
4. SW algorithm is used to align these sequences.

The main idea of this program is to find subsequence matches between the query sequence and each of the sequences in the database, enlarge them and compute local alignment in these regions using dynamic programming. There are few efforts on parallelizing FASTA such as [19,30] on a cluster of workstations.

6.6.2 BLAST

Basic local alignment search tool (BLAST) developed at the National Center for Biotechnology Information by Altchul and colleagues [1] is a popular tool for local sequence alignment. BLAST and its derivative algorithms are one of the most widely used tools for sequence alignment. The main idea of BLAST is to search only a subspace of the sequences. In its basic version, gaps are not allowed during alignment which simplifies the alignment procedure greatly. The assumption here is if there is a similarity between two sequences, it will show even if the gaps are not allowed.

A *segment pair* in BLAST is defined as a pair of equal-length subsequences between two sequences S_1 and S_2 which are aligned without gaps. A maximal segment pair (MSP) of S_1 and S_2 is the highest scoring segment between them. As the first step, BLAST searches all sequences with length l in the database that have an MSP score higher than a threshold τ with the input query Q [9]. It searches the short sequences first and then extends them. The found subsequences are called *hits* which are then extended in both directions to find if the score is higher than τ. In detail, BLAST performs the following steps:

1. We are again given a query sequence Q and a set of sequences $\mathcal{S} = S_1, \ldots, S_n$ in the database. BLAST searches hits of length l that have an MSP of score higher than τ between the query and the database sequence $S_i \in \mathcal{S}$. Typical values of l are 3 for protein amino acid sequences and 11 for DNA sequences. The threshold τ is dependent on the scoring matrix used.
2. It searches for pairs of hits which have a maximum distance of d between them.
3. The hit pairs are extended in both directions and the alignment score is checked at each extension. This process is stopped when the score does not change. The pair of hits scoring above a threshold after the extension are called high scoring pairs (HSPs).
4. The consistent HSPs are combined into local alignment that gives the highest score.

The newer versions of BLAST alow gaps [2,26]. The BLAST algorithm also provides an estimate of the statistical significance of the output. This tool is available for free usage at www.ncbi.nlm.nih.gov/blast/.

6.7 Parallel and Distributed Sequence Alignment

We have reviewed basic global and local exact alignment algorithms and the commonly used database alignment tools which use heuristic methods that provide approximate results. The database tools are simpler to parallelize on a distributed memory computer system as we can easily partition the database across the machines or duplicate it if its size is not very large. We will first look at ways of parallelizing the exact algorithms and then review existing methods for distributed alignment using the database tools.

6.7.1 Parallel and Distributed SW Algorithm

The SW algorithm is a dynamic programming method to provide local alignment of two sequences as we have reviewed. We can have fine-grain or coarse-grain parallel running of this algorithm on a number of processors [7]. In fine-grain parallel computing, we have small tasks that cooperate more frequently for small data sizes.

Forming the alignment matrix is the most time-consuming part of the algorithm. We can have a fine-grain parallel mode of SW algorithm by assigning each cell of the alignment matrix to a process. The value of each cell $S[i, j]$ in this matrix is dependent on the values of the preceding row, column, and diagonal values. We can therefore employ a scheme in which every process responsible for the cell $S[i, j]$ that calculates its value sends it to the cells $S[i + 1, j + 1]$, $S[i, j + 1]$, and $S[i + 1, j]$ for further processing as shown in Fig. 6.4. As the computation progresses along waves which increase in size until diagonal and then decrease, this scheme is called the *wavefront* method [4]. We would need nm processes to fill the matrix for two sequences of lengths n and m. Hence, specific architectures such as array processors are suitable for this method. An early attempt that used this approach was reported in [14] which used 12 processors with shared memory, and another implementation was described in [28]. The same technique can be used to find global alignment between two sequences using the NW algorithm.

The coarse-grain distributed sequence alignment is basically based on parallel database operations in which sequences from a database are searched in parallel. In a typical supervisor–worker parallel computation model, the supervisor process sends a number of sequences to each worker to align. The workers send the results to the supervisor which ranks them and keeps the best alignments. As the processing times of workers will be of varying lengths, a dynamic load balancing strategy is usually needed in this mode of operation to keep processes busy at all times [7].

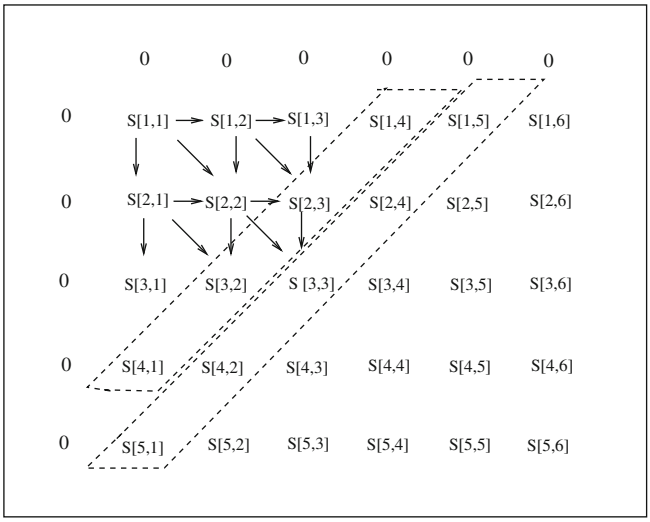


Fig. 6.4 The wavefront method for parallel SW algorithm

6.7.2 Distributed BLAST

When the database to be searched is comparatively small, a simple way to provide parallelism is to replicate it on a number of machines. We can then divide the batched queries to k processors which implement the BLAST algorithm in parallel. The supervisor–worker model can be adopted in this case in which the supervisor process gathers all of the results obtained from the individual workers and outputs the final result. Figure 6.5 shows this process visually. This method has been the subject of various studies including BeoBLAST [15] and Hi-per BLAST [24].

Recent sequencing techniques allow discovery and provision of many biological sequences which constitute large databases. These databases cannot usually be accommodated in a single computer and a convenient way of providing parallelism using such large databases is to partition the data. We can again implement the supervisor–worker model where each process implements BLAST on partial data and sends the partial results to the supervisor which combines them to get the final output. The general approach of a distributed BLAST algorithm employing partitioned database is shown in Fig. 6.6.

The TurboBLAST tool implements database segmentation along with load balancing and scheduling algorithms to run BLAST on a cluster of workstations [3]. This approach has also been applied in mpiBLAST [10] which uses the message passing interface (MPI) parallel programming environment [16]. Its claimed benefits are first the decreased disk I/O operations due to partitioned database and the reduction of interprocess communications between processes as each worker uses data in its partition only. At the start of mpiBLAST, each worker process notifies the supervisor of the database segments it has. The supervisor then inputs the query

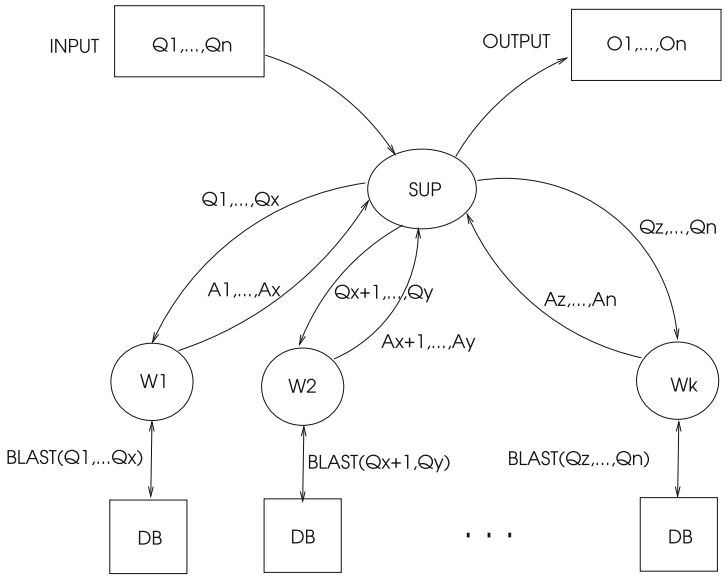


Fig. 6.5 Distributed BLAST using replicated database. The input query batch is $Q_1, \ldots, Q_n$. Each worker W_i has a copy of the database and receives a portion of the input query batch from the supervisor process (SUP). Each worker then runs part of the BLAST query in its local database, obtains the results $(A_p, \ldots, A_q)$ and sends them to the supervisor which combines the partial results and outputs them

batch and broadcasts the batch to all workers. Upon a worker W_i announcing it is idle and can start working, the supervisor assigns W_i a database segment. The worker W_i then performs alignment in the segment it is allocated and reports the result to the supervisor process. The operation of the algorithm is very similar to what is depicted in Fig. 6.6 with the additional enhanced load balancing in which a worker that has finished searching a database can be assigned another search in a different database segment as assigned by the supervisor. The authors report that mpiBLAST achieves super-linear speedup in all tests [10].

The multithreaded versions of BLAST are also available to run on shared memory multiprocessor systems. This mode of operation is similar to partitioned database approach; however, the database is loaded to shared memory now and each thread can work in its partition. Thread and shared memory management may incur overheads and cause scalability issues. NCBI BLAST [25] and WU BLAST [34] are the examples of multithreaded BLAST systems [35]. The UMD–BLAST is an interface that enables to use the most suitable parallel/distributed BLAST algorithm. It inputs the database size, query batch size, and query length and determines which algorithm to use. For large databases which cannot be accommodated in the memory of a single computer, UMD–BLAST uses mpiBLAST; for long query batches with not very large query lengths, BLAST++ which employs replicated database is used. Otherwise, the multithreaded BLAST is employed and the outputs are combined [35].

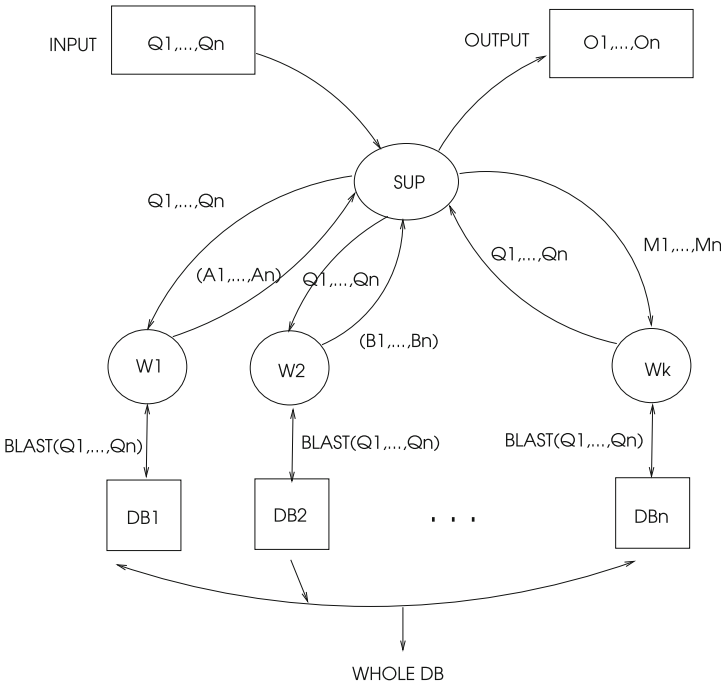


Fig. 6.6 The operation of distributed BLAST using partitioned database. Each worker W_i now has a segment of the database and receives the full input query batch from the supervisor process (SUP). It then runs all of the BLAST query in its database partition, finds the results, and sends them to the supervisor which combines the partial results and outputs them

6.7.3 Parallel/Distributed CLUSTALW

Let us review the main three steps of the CLUSTALW algorithm [32]. In the first step, it computes pairwise distances followed by the construction of the guide tree using the neighbor-joining algorithm in the second step. This tree is used as a guide to perform alignment in the last step where the leaves are first aligned followed by the alignment of close nodes in the tree in sequence.

Parallelization whether using shared memory or distributed memory computers involves implementing these three steps in parallel. The first step requires calculation of distance between k sequences with $k(k-1)/2$ comparisons. This step is trivial to parallelize again using the supervisor–worker model of parallel computation. We can have the supervisor send groups of sequences to each worker process which compute the distances and the results are then gathered at the supervisor. The parallel implementation of the second and third steps is not so straightforward due to the data dependencies involved.

ClustalW-MPI [20] is the distributed implementation of the CLUSTALW method using the MPI parallel programming environment based on the described approach

above. The distance matrix is first formed by allocating *chunks* of independent tasks to processes. Large batches result in decreased interprocess communication times but may have poor load balancing. On the other hand, small size of batches provides balanced process loads at the expense of increased communication overheads.

The guided tree is formed by the neighbor-joining method as in CLUSTALW; however, few modifications to the original algorithm resulted in the complexity of $O(n^2)$ time for constructing the guided tree. As this algorithm searches sequences that are closer to each other but also have the highest distance to all other clusters, a parallel search method was designed to search for such sequence clusters. However, details of this method are not described in the paper. In the final progressive alignment step, a mixture of fine and coarse-grained parallelism methods is used. Coarse-grain parallelism involves aligning external nodes of the guided tree and the speedup obtained is reported as $n/\log n$ where n is the number of nodes of the tree. The authors also implemented recursive parallelism and calculated the forward and backward steps of the dynamic programming in parallel. They showed experimentally the speedup achieved for aligning 500-sequence test data as 15.8 using 16 processors.

Another parallel version of CLUSTALW, called pCLUSTAL, which can run on various hardware from parallel multiprocessors to distributed memory parallel computers using MPI was described in [8]. This study also uses supervisor–worker paradigm in which the supervisor process p_0 maps the sequence-pairs to processes and each process then performs sequential CLUSTALW algorithm on its own data set. The results are then gathered at p_0 which builds the guided tree T. It then examines T for independently executable alignments and assigns these to processes. The final step involves gathering of all the alignment results at p_0. The experiments were carried on protein sequences of average length of 300 amino acids. They showed the time-consuming pairwise alignment step takes time proportional to $1/k$ where k is the number of processors.

A shared memory implementation CLUSTALW in SGI multicomputers was described in [22] using OpenMP and speedups of 10 on 16 processors was reported, and a comparison of various implementations is presented in [13].

6.8 Chapter Notes

Comparison of biological sequences using alignment is needed as the first step of various analysis methods in bioinformatics, for example, alignment of sequences provides their affinities which can be used to infer pyhlogenetic relationships between them. Global alignment refers to comparing two or more sequences as a whole, and local alignment methods attempt to align subsequences of the sequences under consideration. The alignment methods, whether global or local, can be broadly classified as exact and heuristic approaches. Exact methods typically use dynamic programming and have favorable performances as we have seen in SW and NW algorithms. However, even these linear times become problematic when the size of the sequences is very large. Heuristic methods do not search exact solutions and typically narrow

the search space by sampling of the data which results in favorable run times for large sequences. FASTA and widely used BLAST are two commonly used tools which adopt heuristic tools.

However, even the heuristic methods in sequential form are increasingly becoming more inadequate as the sizes of databases increase due to the expansion in the number of discovered sequences as a result of high volume efficient sequencing technologies. A possible way to speedup the heuristic alignment methods is to employ parallel and distributed processing. This can be achieved typically either by replicating the database if this is not relatively large, or partitioning it. We described these two approaches as implemented in various BLAST versions.

Sequence alignment is probably one of the most investigated and studied topic in bioinformatics and the tools for this purpose are among the mostly publicly used software in bioinformatics. There are books devoted solely to this topic as general alignment or multiple sequence alignment, and this topic is treated in detail in many contemporary bioinformatics books. Our approach in this chapter was to briefly review the fundamental methods of alignment only, with emphasis on distributed alignment.

Exercises

1. Work out the global alignment between the two DNA sequences below using the dynamic programming approach of NW algorithm. Show all matrix iterations.

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

2. Find the local alignment between the two protein sequences below using the SW algorithm. Show all matrix iterations.

```
B N Q R S T U R V Y A C K
A N Q T T V T U R X E A C
```

3. For the following four DNA sequences, implement center star method of multiple sequence alignment by first finding the distances between them and forming the distance matrix D. Find the central sequence and align all of the sequences to the central sequence using NW algorithm and finally insert gaps in sequences to complete the alignment.

```
S1: A C C G A A C
S2: A G C G C T G
S3: C C C T A T G
S4: A T C G A T G
```

4. Compare FASTA and BLAST in terms of method used and the accuracy achieved.

5. Given the following two DNA sequences, draw the general suffix tree for them and find the LCS of these two sequences using this generalized suffix tree.

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

6. Provide the pseudocode of a distributed BLAST algorithm based on supervisor–worker model. Assume the input queries are distributed to k processes and each worker returns the results to the supervisor.

References

1. Altschul SF, Gish W, Miller W, Myers EW et al (1990) Basic local alignment search tool. J Mol Biol 215(3):403–410
2. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 25(17):3389–3402
3. Bjornson R, Sherman A, Weston S, Willard N, Wing J (2002) Turboblast: a parallel implementation of blast based on the turbohub process integration architecture. In: IPDPS 2002 Workshops
4. Boukerche A (2006) Computational molecular biology. In: Albert YZ (ed) Parallel computing for bioinformatics and computational biology models, enabling technologies, and case studies. Wiley series on parallel and distributed computing, Chap, 6
5. Brudno M, Do C, Cooper G, Kim M, Davydov E, Green ED, Sidow A, Batzoglou S (2003) LAGAN and multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. Genome Res 13:721–731
6. Brudno M, Chapman M, Gottgens B, Batzoglou S, Morgenstern B (2003) Fast and sensitive multiple alignment of large genomic sequences. BMC Bioinform 4:66
7. Chaudhary V, Liu F, Matta V, Yang LT (2006) Parallel implementations of local sequence alignment: hardware and software. In: Albert YZ (ed) Parallel computing for bioinformatics and computational biology models, enabling technologies, and case studies. Wiley series on parallel and distributed computing, Chap, 10
8. Cheetham JJ et al (2003) Parallel CLUSTALW for PC clusters, computational science and its applications. In: ICCSA 2003, pp 300–309. LNCS. Springer
9. Cristianini N, Hahn MW (2006) Introduction to computational genomics, a case studies approach. Cambridge University Press, Cambridge
10. Darling A, Carey L, Feng W-C (2003) The design, implementation, and evaluation of mpiBLAST. In: Cluster world conference and expo and the 4th international conference on linux clusters: the HPC revolution, San Jose, CA
11. Dayhoff MO, Schwartz RM, Orcutt BC (1978) A model of evolutionary change in proteins. Atlas Protein Seq Struct 5(3):345–352
12. Delcher AL, Phillippy A, Carlton J, Salzberg SL (2002) Fast algorithms for large-scale Genome alignment and comparison. Nucleic Acids Res 30(11):2478–2483
13. Duzlevski O (2002) SMP version of ClustalW 1.82, unpublished. http://bioinfo.pbi.nrc.ca/clustalw-smp/

14. Galper AR, Brutlag DR (1990) Parallel similarity search and alignment with the dynamic programming method. Technical report KSL 90–74, Stanford University
15. Grant J, Dunbrack R, Manion F, Ochs M (2002) BeoBLAST: distributed BLAST and PSI-BLAST on a Beowulf cluster. Bioinformatics 18(5):765–766
16. Gropp W, Lusk E, Skjellum A (2014) Using MPI: portable parallel programming with the message passing interface, 3rd edn. MIT Press, ISBN: 9780262527392
17. Henikoff S, Henikoff JG (1992) Amino acid substitution matrices from protein blocks. Proc Nat Acad Sci USA 89(22):10915–10919
18. Hohl M, Kurtz S, Ohlebusch E (2002) Efficient multiple genome alignment. Bioinformatics 18:312–320
19. Janaki C, Joshi RR (2003) Accelerating comparative genomics using parallel computing. Silico Biol 3(4):429–440
20. Li Kuo-Ben (2003) ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19(12):1585–1586
21. Lipman DJ, Pearson WR (1985) Rapid and sensitive protein similarity searches. Science 227(4693):1435–1441
22. Mikhailov D, Cofer H, Gomperts R (2001) Performance optimization of Clustal W: parallel Clustal W, HT Clustal, and MULTICLUSTAL. White papers, Silicon Graphics, Mountain View, CA
23. Mount DM (2004) Bioinformatics: sequence and genome analysis, 2nd edn. Cold Spring Harbor Laboratory Press, Cold Spring Harbor. ISBN 0-87969-608-7
24. Naruse A, Nishinomiya N (2002) Hi-per BLAST: high performance BLAST on PC cluster system. Genome Inform 13:254–255
25. National Center for Biotechnology Information, NCBIBLAST. http://www.ncbi.nih.gov/BLAST/
26. Pearson WR (1990) Rapid and sensitive sequence comparison with FASTP and FASTA. Method Enzymol 183:63–98
27. Pearson WR, Lipman DJ (1988) Improved tools for biological sequence comparison. Proc Nat Acad Sci USA 85:2444–2448
28. Rognes T, SeeBerg E (2000) Six-fold speedup of Smith Waterman sequence database searches using parallel processing on common microprocessors. Bioinformatics 16(8):699–706
29. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol BioI Evol 4(4):406–425
30. Sharapov I (2001) Computational applications for life sciences on Sun platforms: performance overview, Whitepaper
31. Smith TF, Waterman MS (1981) Identification of common molecular subsequences. J Mol Biol 147:195–197
32. Thompson JD, Higgins DG, Gibson TJ (1994) CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. Nucleic Acids Res 22:4673–4680
33. Wing-Kin S (2009) Algorithms in bioinformatics: a practical introduction. CRC Press (Taylor & Francis Group), Chap, 5
34. WU blast http://blast.wustl.edu/blast/README.html. Washington University School of Medicine
35. Wu X, Tseng C-W (2006) Searching sequence databases using high-performance BLASTs. In: Albert YZ (ed) Parallel computing for bioinformatics and computational biology models, enabling technologies, and case Studies. Wiley series on parallel and distributed computing, Chap, 9