The **Smith–Waterman algorithm** performs local sequence alignment; that is, for determining similar regions between two strings or nucleotide or protein sequences. Instead of looking at the total sequence, the Smith–Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure.

The algorithm was first proposed by Temple F. Smith and Michael S. Waterman in 1981.Like the Needleman–Wunsch algorithm, of which it is a variation, Smith–Waterman is a dynamic programming algorithm. As such, it has the desirable property that it is guaranteed to find the optimal local alignment with respect to the scoring system being used (which includes the substitution matrix and the gap-scoring scheme). The main difference to the Needleman–Wunsch algorithm is that negative scoring matrix cells are set to zero, which renders the (thus positively scoring) local alignments visible. Backtracking starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment. One does not actually implement the algorithm as described because improved alternatives are now available that have better scaling and are more accurate.

**Working of Smith-Waterman Algorithm :**

**Intialization of Matrix**

The basic steps for the algorithm are similar to that of Needleman-Wunsch algorithm. The steps are:
1.Initialization of a matrix.
2.Matrix Filling with the appropriate scores.
3. Trace back the sequences for a suitable alignment.
To study the Local sequence alignment consider the given below sequences.
CGTGAATTCAT (sequence#1 or A)
GACTTAC (sequence #2 or B)

The two sequences are arranged in a matrix form with A+1columns and B+1rows. The values in the first row and first column are set to zero as shown in Figure 1.

| | - | C | G | T | G | A | A | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | | | | | | | | | | | |
| A | 0 | | | | | | | | | | | |
| C | 0 | | | | | | | | | | | |
| T | 0 | | | | | | | | | | | |
| T | 0 | | | | | | | | | | | |
| A | 0 | | | | | | | | | | | |
| C | 0 | | | | | | | | | | | |

Figure 1: Initialization of Matrix

Variables used:

i,j describes row and columns.
M is the matrix value of the required cell (stated as $M_{i,j}$)
S is the score of the required cell ($S_{i,j}$)
W is the gap alignment

**Matrix Filling**

The second and crucial step of the algorithm is filling the entire matrix, so it is more important to know the neighbor values (diagonal, upper and left) of the current cell to fill each and every cell.

$$M_{i,j} = Maximum\ [M_{i-1,j-1} + S_{i,j},\ M_{i,j-1} + W,\ M_{i-1,j} + W,\ 0]$$

As per the assumptions stated earlier, fill the entire matrix using the assumed scoring schema and initial values. One can fill the 1st row and 1st column with the scoring matrix as follows.

The first residue (nucleotides or amino acids) in both sequences is 'C' and 'G', the matching score or the mismatching score is going to be added the neighboring value which is diagonally located i.e. 0. The upper and left values are added to the gap penalty score from the matrix. So the scoring schema equation can be shown as follows.

$$\begin{aligned} M_{1,1} &= Maximum\ [M_{0,0} + S_{1,1},\ M_{1,0} + W,\ M_{0,1} + W,\ 0] \\ &= Maximum\ [0(-3), 0 + (-4), 0 + (-4), 0] \\ &= Maximum\ [-3, -4, -4, 0] \\ &= 0 \end{aligned}$$

From the above calculations the maximum value obtained is 0. Finding the maximum value for $M_{i,j}$ position, one can notice that there is no chance to see any negative values in the matrix, since we are taking 0 as lowest value.

After filling the matrix, keep the pointer back to the cell from where the maximum score has been determined. In the similar fashion fill all the values of the matrix of the cell.

For the example the matrix can be filled is shown in Figure 2.

| | - | C | G | T | G | A | A | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 5 | 1 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 1 | 2 | 1 | 10 | 6 | 2 | 0 | 0 | 5 | 1 |
| C | 0 | 5 | 1 | 0 | 0 | 6 | 7 | 3 | 0 | 5 | 1 | 2 |
| T | 0 | 1 | 2 | 6 | 2 | 2 | 3 | 12 | 8 | 4 | 2 | 6 |
| T | 0 | 0 | 0 | 7 | 3 | 0 | 0 | 8 | 17 | 13 | 9 | 7 |
| A | 0 | 0 | 0 | 3 | 4 | 8 | 5 | 4 | 13 | 14 | 18 | 14 |
| C | 0 | 5 | 1 | 0 | 0 | 4 | 5 | 2 | 9 | 18 | 14 | 15 |

Figure 2: Matrix filling with back pointers

Each cell is back pointed by one or more pointers from where the maximum score has been obtained.

**Trace backing the sequences for an optimal alignment:**

The final step for the appropriate alignment is trace backing, prior to that one needs to find out the maximum score obtained in the entire matrix for the local alignment of the sequences.It is possible that the maximum scores can be present in more than one cell, in that case there may be possibility of two or more alignments, and the best alignment by scoring it.

In this example we can see the maximum score in the matrix as 18, which is found in two positions that lead to multiple alignments, so the best alignment has to be found.

So the trace back begins from the position which has the highest value, pointing back with the pointers, thus find out the possible predecessor, then move to next predecessor and continue until we reach the score 0 (Figure 3).

| | - | C | G | T | G | A | A | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 5 | 1 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 1 | 2 | 1 | 10 | 6 | 2 | 0 | 0 | 5 | 1 |
| C | 0 | 5 | 1 | 0 | 0 | 6 | 7 | 3 | 0 | 5 | 1 | 2 |
| T | 0 | 1 | 2 | 6 | 2 | 2 | 3 | 12 | 8 | 4 | 2 | 6 |
| T | 0 | 0 | 0 | 7 | 3 | 0 | 0 | 8 | 17 | 13 | 9 | 7 |
| A | 0 | 0 | 0 | 3 | 4 | 8 | 5 | 4 | 13 | 14 | 18 | 14 |
| C | 0 | 5 | 1 | 0 | 0 | 4 | 5 | 2 | 9 | 18 | 14 | 15 |

Figure 3: Trace back of first possible alignment

It is possible to find two pointers pointing out from one cell, where both ways(alignments) can be considered, best one is found by scoring and finding maximum score among them.

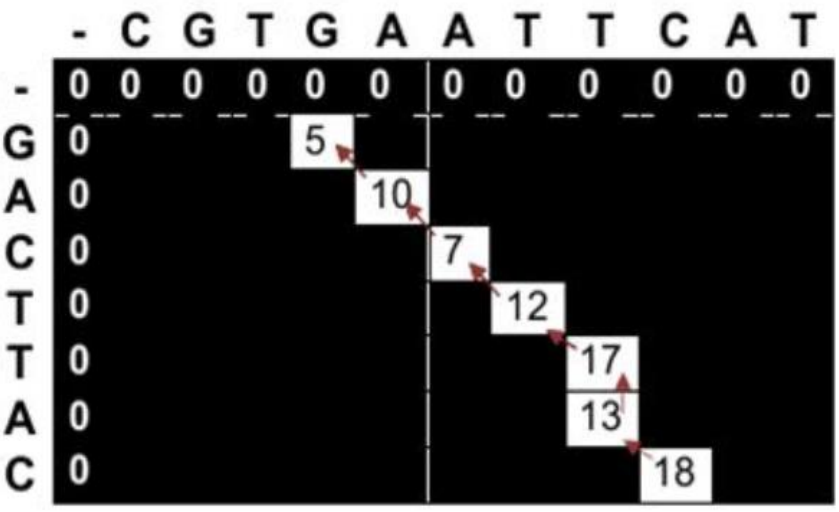


Figure 4: Trace back of second possible alignment

Thus a local alignment is obtained and one can see the possible alignments as in Figure 5.

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

+ + - + + - +   + + - + + - +
5 5 3 5 5 4 5   5 5 3 5 5 4 5
```

Figure 5: Scoring for best alignment

The two alignments can be given with a score, for matching as +5 , mismatch as -3 and gap penalty as -4, sum up all the individual scores and the alignment which has maximum score after this can be taken as the best alignment.

By summing up the scores both of the alignments are giving the same as 18, so one can predict both alignments are the best.

The **Needleman–Wunsch algorithm** is an algorithm used in bioinformatics to align protein or nucleotide sequences. It was one of the first applications of dynamic programming to compare biological sequences. The algorithm was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970. The algorithm essentially

divides a large problem (e.g. the full sequence) into a series of smaller problems and uses the solutions to the smaller problems to reconstruct a solution to the larger problem. It is also sometimes referred to as the optimal matching algorithm and the global alignment technique. The Needleman–Wunsch algorithm is still widely used for optimal global alignment, particularly when the quality of the global alignment is of the utmost importance.

**Working of Needleman -Wunsch Algorithm**

To study the algorithm, consider the two given sequences.

CGTGAATTCAT (sequence #1) , GACTTAC (sequence #2)

The length (count of the nucleotides or amino acids) of the sequence 1 and sequence 2 are 11 and 7 respectively. The initial matrix is created with A+1 column's and B+1 row's (where A and B corresponds to length of the sequences). Extra row and column is given, so as to align with gap, at the starting of the matrix as shown in Figure 1.

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

Figure 1: Initial matrix

After creating the initial matrix, scoring schema has to be introduced which can be user defined with specific scores. The simple basic scoring schema can be assumed as, if two residues (nucleotide or amino acid) at $i^{th}$ and $j^{th}$ position are same, matching score is 1 (S(i,j)= 1) or if the two residues at $i^{th}$ and $j^{th}$ position are not same, mismatch score is assumed as -1 (S(i,j)= -1 ). The gap score(w) or gap penalty is assumed as -1 .

*Note: The scores of match, mismatch and gap can be user defined, provided the gap penalty should be negative or zero.

Gap score is defined as penalty given to alignment, when we have insertion or deletion.

The dynamic programming matrix is defined with three different steps.
1.Initialization of the matrix with the scores possible.
2.Matrix filling with maximum scores.
3.Trace back the residues for appropriate alignment.

**1.Initialization Step**

This example assumes that there is gap penalty. First row and first column of the matrix can be initially filled with 0. If the gap score is assumed, the gap score can be added to the previous cell of the row or column (Figure 2).

| | - | C | G | T | G | A | A | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 | -11 |
| G | -1 | | | | | | | | | | | |
| A | -2 | | | | | | | | | | | |
| C | -3 | | | | | | | | | | | |
| T | -4 | | | | | | | | | | | |
| T | -5 | | | | | | | | | | | |
| A | -6 | | | | | | | | | | | |
| C | -7 | | | | | | | | | | | |

Figure 2: Initialization of matrix

**2.Matrix Fill Step**

The second and crucial step of the algorithm is matrix filling starting from the upper left hand corner of the matrix. To find the maximum score of each cell, it is required to know the neighbouring scores (diagonal, left and right) of the current position. From the assumed values, add the match or mismatch (assumed) score to the diagonal value. Similarly add the gap score to the other neighbouring values. Thus, we can obtain three different values, from that take the maximum among them and fill the $i^{th}$ and $j^{th}$ position with the score obtained.

In terms of matrix positions, it is important to know $[M_{(i-1,j-1)}+S_{(i,j)},M_{(i,j-1)}+w,M_{(i-1,j)}+w]$

Overall the equation can be showed in the following manner

$$M_{i,j} = Maximum\ [M_{i-1,j-1} + S_{i,j},\ M_{i,j-1} + W,\ M_{i-1,j} + W]$$

To score the matrix of the current position (the first position $M_{1,1}$) the above stated formulae can be used. The first residue (nucleotides or amino acids) in the 2 sequences are 'G' and 'C'. Since they are mismatching residues, the score would ($S_{i,j}$=- 1) be -1.

$$M_{1,1} = Max\ [M_{0,0} + S_{1,1},\ M_{1,0} + W,\ M_{0,1} + W]$$
$$= Max\ [0+(-1),\ 0+(-1),0+(-1)]$$
$$= Max\,[-1,-1,-1]$$
$$= -1$$

The obtained score -1 is placed in position i,j (1,1) of the scoring matrix. Similarly using the above equation and method, fill all the remaining rows and columns. Place the back pointers to the cell from where the maximum score is obtained, which are predecessors of the current cell (Figure 3).

| | - | C | G | T | G | A | A | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 | -11 |
| G | -1 | -1 | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 |
| A | -2 | -2 | -1 | -1 | -2 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| C | -3 | -1 | -2 | -2 | -2 | -2 | -2 | -3 | -4 | -3 | -4 | -5 |
| T | -4 | -2 | -2 | -1 | -2 | -3 | -3 | -1 | -2 | -3 | -4 | -3 |
| T | -5 | -3 | -3 | -1 | -2 | -3 | -4 | -2 | 0 | -1 | -2 | -3 |
| A | -6 | -4 | -4 | -2 | -2 | -1 | -2 | -3 | -1 | -1 | 0 | -1 |
| C | -7 | -5 | -5 | -3 | -3 | -2 | -2 | -3 | -2 | 0 | -1 | -1 |

Figure 3: Matrix filling with back pointers

**3.Trace back Step**

The final step in the algorithm is the trace back for the best alignment. In the above mentioned example, one can see the bottom right hand corner score as -1. The important point to be noted here is that there may be two or more alignments possible between the two example sequences. The current cell with value -1 has immediate predecessor, where the maximum score obtained is diagonally located and its value is 0. If there are two or more values which points back, suggests that there can be two or more possible alignments.

By continuing the trace back step by the above defined method, one would reach to the $0^{th}$ row, $0^{th}$ column. Following the above described steps, alignment of two sample sequences can be found. The best alignment among the alignments can be identified by using the maximum alignment score (match =5, mismatch=-1, gap=-2) which may be user defined (Figure 4).

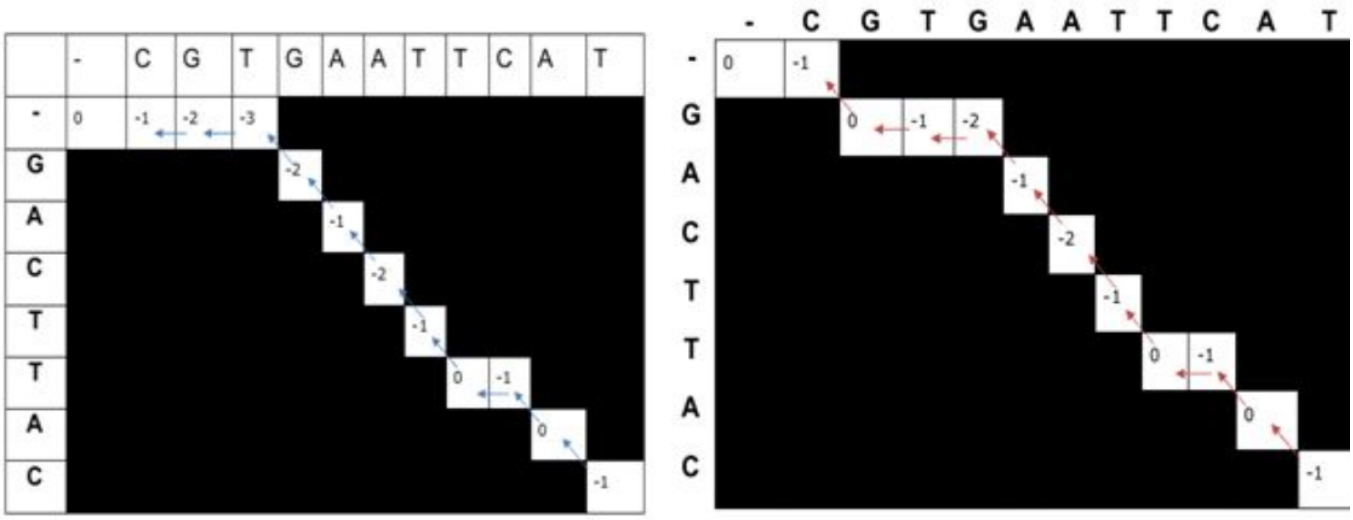


Figure 4: The possible Alignments with trace backing

**Scoring Systems**

Basic scoring schemes

The simplest scoring schemes simply give a value for each match, mismatch and indel. The step-by-step guide above uses match = 1, mismatch = -1, indel = -1. Thus the lower the alignment score the larger the edit distance, for this scoring system we want a high score. Another scoring system might be:

Match = 0

Indel = 1

Mismatch = 1

For this system the alignment score will represent the edit distance between the two strings. Different scoring systems can be devised for different situations, for example if gaps are considered very bad for your alignment you may use a scoring system that penalises gaps heavily, such as:

Match = 0

Mismatch = 1

Indel = 10

**Similarity Matrix**

More complicated scoring systems attribute values not only for the type of alteration, but also for the letters that are involved. For example a match between A and A may be given 1, but a match between T and T may be given 4. Here (assuming the first scoring system) more importance is given to the Ts matching than the As, i.e. we think the Ts matching is more significant to our alignment. This weighting based on letters also applies to mismatches. In order to represent all the possible combinations of letters and their resulting scores we use a similarity matrix. The similarity matrix for the most basic system is represented as:

| | A | G | C | T |
|---|---|---|---|---|
| A | 1 | -1 | -1 | -1 |
| G | -1 | 1 | -1 | -1 |
| C | -1 | -1 | 1 | -1 |
| T | -1 | -1 | -1 | 1 |

Each score represents a switch from one of the letters the cell matches to the other. Hence this represents all possible matches and deletions (for an alphabet of ACGT). Note all the matches go along the diagonal, also not all the table needs to be filled, only this triangle because the scores are reciprocal.= (Score for A → C = Score for C → A). If we implement our T-T = 4 from above we get:

| | A | G | C | T |
|---|---|---|---|---|
| A | 1 | -1 | -1 | -1 |
| G | -1 | 1 | -1 | -1 |
| C | -1 | -1 | 1 | -1 |
| T | -1 | -1 | -1 | 4 |

Different scoring matrices have been statistically constructed which give weight to different actions appropriate to a particular scenario. Having weighted scoring matrices is particularly important in protein sequence alignment due to the varying frequency of the different amino acids. There are two broad families of scoring matrices, each with further alterations for specific scenarios:

PAM

BLOSUM

**Gap penalty**

When aligning sequences there are often gaps (i.e. indels), sometimes large ones. Biologically, a large gap is more likely to occur as one large deletion as opposed to multiple single deletions. Hence we should score two small indels to be worse than one large one. The simple and common way to do this is via a large gap-start score for a new indel and a smaller gap-extension score for every letter which extends the indel. For example, new-indel may cost -5 and extend-indel may cost -1. In this way an alignment such as:

GAAAAAAT

G--A-A-T

which has multiple equal alignments, some with multiple small alignments will now align as:

GAAAAAAT

GAA----T

or any alignment with a 4 long gap in preference over multiple small gaps.