Algorithms in Bioinformatic

Julian Löffler

University of Freiburg, Germany

Advisor: Omer Alkhnbashi

January 28, 2019

Contents

1 Pairwise Sequence-Alignment

Pairwise sequence alignment methods are used to find the best-matching piecewise (local or global) alignments of two query sequences. Pairwise alignments can only be used between two sequences at a time, but they are efficient to calculate and are often used for methods that do not require extreme precision (such as searching a database for sequences with high similarity to a query). The technique of dynamic programming can be applied to produce global alignments via the Needleman-Wunsch algorithm, and local alignments via the Smith-Waterman algorithm. In typical usage, protein alignments use a substitution matrix to assign scores to amino-acid matches or mismatches, and a gap penalty for matching an amino acid in one sequence to a gap in the other. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty. (In standard dynamic programming, the score of each amino acid position is independent of the identity of its neighbors, and therefore base stacking effects are not taken into account.(https://en.wikipedia.org/wiki/Sequence_alignment#Pairwise_alignment)

1.1 Needleman-Wunsch-Algorithm

Needleman-Wunsch is used for to calculate the pairwise aligments of sequences. The idea is to use dynamic programming, to calculate the optimal alignment of two sequences seq_1, seq_2 (or more general for two strings), and store information, which sequence of operations we have to perform, to convert seq_1 to seq_2. Operations are insertions ($\uparrow$), deletions ($\leftarrow$) and matches ($\nwarrow$). An insertion means, we align a gap in seq_1 to a character in seq_2. A deletion means, we align a character in seq_1 to a gap in seq_2. A match means, we match the current characters in seq_1 and seq_2. Let S,T be matrices of size $|seq_1|+1 \times |seq_2|+1$, in S we store scores and in T we store tracebacks cells, traceback cells are lists of tuples of predecessors of a cell, of the form (operation, predecessor) The algorithm works as follows:

1. Input: 2 sequences of length m,n.
2. Initialize S, T:
   - $S[0][0] = 0, S[0][j] = gapcost * j, S[i][0] = gapcost * i,$ for $1 \leq i \leq |seq_1|+1$ and $1 \leq j \leq |seq_2|+1$
   - $T[0][0] = [], T[0][j] = [(\leftarrow, T[0][j-1])], T[i][0] = [(\uparrow, T[i-1][0])],$ for $1 \leq i \leq |seq_1|+1$ and $1 \leq j \leq |seq_2|+1$
3. Calculate the rest of the matrix.:
   - $S[i][j] = max((\nwarrow, S[i-1][j-1]+score(seq_1[i], seq_1[j])), (\leftarrow, S[i][j-1]), (\uparrow, S[i-1][j]))$
   - $T[i][j]$ is the list of predecessors, which contains everything that is a maximum in the above case.

4. The bottom right element in S, i.e. $S[-1][-1]$, stores the score of the optimal alignment,
5. The bottom right element in T, i.e. $T[-1][-1]$, stores the traceback cell of the optimal alignment.
   To obtain all paths back to the root $T[0][0]$, we traverse through all predecessors, until we reach $T[0][0]$.

1.2 example:

- Consider the sequences AAT and AAC,
- Scoring: match = 1, mismatch = -1, gap penalty = -1.
- 1. Initialize matrices S,T:
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 0. & 0. & 0. \\ -2. & 0. & 0. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & 0 & 0 & 0 \\ \uparrow & 0 & 0 & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$
- 2. calc S[1][1], T[1][1]
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & 0. \\ -2. & 0. & 0. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & 0 \\ \uparrow & 0 & 0 & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$
- 2. calc S[1][2], T[1][2]
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & 0. \\ -2. & 0. & 0. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & 0 \\ \uparrow & 0 & 0 & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$
- 3. calc S[1][3], T[1][3]
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 0. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & 0 & 0 & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$
- 4. calc S[2][1], T[2][1]
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 0. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & 0 & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$
- 5. calc S[2][2], T[2][2]
  $$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 2. & 0. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & \nwarrow & 0 \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$

- 6. calc S[2][3], T[2][3]

$$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 2. & 1. \\ -3. & 0. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & \nwarrow & \leftarrow \\ \uparrow & 0 & 0 & 0 \end{bmatrix}$$

- 7. calc S[3][1], T[3][1]

$$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 2. & 1. \\ -3. & -1. & 0. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & 0 & 0 \end{bmatrix}$$

- 8. calc S[3][2], T[3][2]

$$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 2. & 1. \\ -3. & -1. & 1. & 0. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & 0 \end{bmatrix}$$

- 9. calc S[3][3], T[3][3]

$$S = \begin{bmatrix} 0. & -1. & -2. & -3. \\ -1. & 1. & 0. & -1. \\ -2. & 0. & 2. & 1. \\ -3. & -1. & 1. & 1. \end{bmatrix} \quad T = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow \mid \leftarrow & \leftarrow \\ \uparrow & \nwarrow \mid \uparrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \nwarrow \end{bmatrix}$$

- Our traceback matrix stored all predecessors of each cell, so we simply track back and obtain the set of tracebacks

$$tracebacks = \{[\nwarrow, \nwarrow, \nwarrow]\}$$

  In this case there is only one traceback.

- The optimal aligment is obtained by iterating over the traceback. We start at the start of the sequences. And handle each traceback element as follows.
  - If we read $\nwarrow$, we align the current characters and move forward 1 position in both sequences.
  - If we read $\leftarrow$, we align a gap "-" to the current character in seq_2 align and move forward 1 position in seq_2.
  - If we read $\uparrow$, we align a gap "-" to the current character in seq_1 align and move forward 1 position in seq_1.

  If we do that for our example, we obtain the alignment:

$$\begin{matrix} A & A & T \\ A & A & C \end{matrix}$$

1.3 Gotoh-Algorithm

Gotohs algorithm is similar to needleman wunsch. Just the recursion changes, we are now using 3 matrices D,Q,P, where the recursions are as follows:

$$D[i][j] = max(D[i-1][j-1] + w(a,b))$$

$$P[i][j] = max(D[i-1][j] + g(1), P[i-1][j] + \beta)$$

$$Q[i][j] = max(D[i][j-1] + g(1), Q[i][j-1] + \beta)$$

The traceback now goes through those 3 matrices. We handle it as before, and store the operations in a single traceback matrix.

2 Phylogeny/Clustering

2.1 UPGMA/WPGMA

We now explain UPGMA and WPGMA, based on [1, 2]. Consider some set of sequences, which is shown in Figure 1.

A: TCAACTAC
B: ACTGCAAA
C: GGCTGTAA
D: AGTTGCAA
E: TTTGAACT

Figure 1: Example sequences

The aim is grouping the most similar sequences, regardless of their evolutionary rate or phylogenetic affinities. Unweighted / Weighted Pair Group Method with Arithmetic Mean (UPGMA/WPGMA) is a hierarchical clustering method used for the creation of guide trees. The algorithm works as follows:

1. Compute the distance between each pair of sequences.
2. Treat each sequence as a cluster C by itself.
3. Merge the two closest clusters. The distance between two clusters is the average distance between all their sequences:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{r \in C_i, s \in C_j} d(r,s)$$

   where C_i, C_j are clusters and $i \neq j$.
4. Repeat 2. and 3. until only one cluster remains.

In WPGMA the distance between clusters is calculated as a simple average:

$$d(C_i \cup C_j, C_k) = \frac{d(C_i, C_k) + d(C_j, C_k)}{2}$$

WPGMA is computationally easier than UPGMA. Unequal numbers of taxa in the clusters cause problems, this means the distances in the original matrix do not contribute equally to the intermediate calculations. The branches do not preserve the original distances, the final result is therefore said to be weighted.

Clustering works only if the data are ultrametric. Ultrametric distances are defined by the satisfaction of the three-point condition:

- For any three taxa it holds:

$$d(A, C) \leq max(d(A, B), d(B, C))$$

So we assume that all taxa evolve with the same constant rate The runtime is $O(n^3)$ for the trivial approach, $O(n^2 log(n))$, when using a heap for each cluster and $O(k3^k n^2)/O(n^2)$ implementations for special cases. (by Fionn Murtagh, by Day and Edelsbrunner)

Example 1 (UPGMA example)
Let A, B, C, D, E be sequences. Say we have calculated a distance matrix D using a pairwise alignment algorithm, which were explained in Section 1. D is shown in Figure 2

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

Figure 2: Distance matrix D of A,B,C,D,E

We now want to create the guide tree for sequences A,B,C,D,E. The sequences are from now on treated as clusters. $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}$

1. *Figure 3 illustrates the initialization of the algorithm. We start with the whole distance matrix and create a guide tree, which consists just of leave nodes that represent the clusters.*

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

Figure 3: Initialization. On the left side, we see the distance matrix D. On the right side we see the guide tree.

2. *Figure 4 illustrates this step of the algorithm. {A} and {C} are the closest clusters, so the algorithm merges {A} and {C} into {A,C}. In the guide tree a new node is added, and nodes A and C connected to it. Each branch gets a total distance of* $\frac{d(\{A\},\{C\})}{2} = \frac{4}{2}$.

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

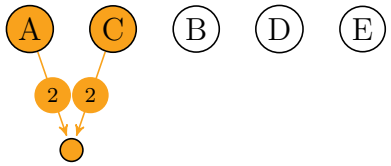

Figure 4: Merge of {A} and {C}. On the left side, we see the distance matrix D. On the right side we see the guide tree, where the orange colored nodes branches represent the changes made to the guide tree.

3. *Figure 5 illustrates the distance matrices, before and after the merge of* $\{A\}$ *and* $\{C\}$. *The distance between* $\{A, C\}$ *and the other clusters is the average distance between all their sequences. Example: the new distance between* $\{A, C\}$ *and* $\{B\}$ *is:*

$$d(\{A,C\},\{B\}) = \frac{1}{|\{A,C\}||\{B\}|}(d(\{A\},\{B\}) + d(\{C\},\{B\}))$$
$$= \frac{1}{2}(8+8) = 8$$

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

→

	A,C	B	D	E
A,C	0	8	6	8
B	8	0	8	4
D	6	8	0	8
E	8	4	8	0

Figure 5: Recalulation of distances, after the merge of {A} and {C}. On the left side, we see the old distance matrix. On the right side we see the new distance matrix, where {A} and {C} are merged.

4. *Figure 6 illustrates this step of the algorithm. {B} and {E} are the closest clusters, such that the algorithm merges {B} and {E} into {B,E}. In the guide tree a new node is added, and nodes B, E are connected to it. It assigns* $\frac{d(\{B\},\{E\})}{2} = \frac{4}{2} = 2$ *to each branch.*

	A,C	B	D	E
A,C	0	8	6	8
B	8	0	8	4
D	6	8	0	8
E	8	4	8	0

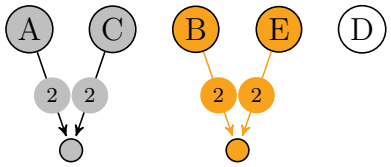

Figure 6: Merge of {B} and {E}. On the left side, we see the distance matrix. On the right side we see the guide tree, where the orange colored nodes branches represent the changes made to the guide tree.

5. *Figure 7 illustrates the distance matrices, before and after the merge of* $\{B\}$ *and* $\{E\}$*. The distance between* $\{B, E\}$ *and the other clusters is the average distance between all their sequences.*

	A,C	B	D	E
A,C	0	8	6	8
B	8	0	8	4
D	6	8	0	8
E	8	4	8	0

→

	A,C	B,E	D
A,C	0	8	6
B,E	8	0	8
D	6	8	0

Figure 7: Recalulation of distances, after the merge of {B} and {E}. On the left side, we see the old distance matrix. On the right side we see the new distance matrix, where {B} and {E} are merged.

6. *Figure 8 illustrates this step of the algorithm. {A,C} and {D} are the closest clusters, the algorithm merges {A,C} and {D} into {A,C,D}. Again a new node is introduced in the guide tree, to which we connect D and the common node of A and C. It assigns* $\frac{d(\{A,C\},\{D\})}{2} = \frac{6}{2} = 3$ *total length to each branch. Observe, that the 1 is obtained calculating* $3 - 2 = 1$.

	A,C	B,E	D
A,C	0	8	6
B,E	8	0	8
D	6	8	0

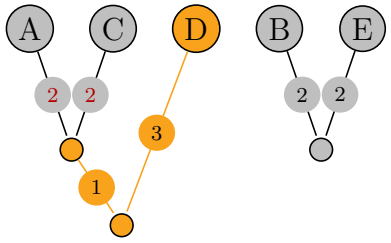

Figure 8: Merge of {A,C} and {D}. On the left side, we see the distance matrix. On the right side we see the guide tree, where the orange colored nodes branches represent the changes made to the guide tree.

7. *Figure 9 illustrates the distance matrices, before and after the merge of* $\{A, C\}$ *and* $\{D\}$. *The distance between* $\{A, C, D\}$ *and* $\{B, E\}$ *is the average distance between all their sequences.*

	A,C	B,E	D
A,C	0	8	6
B,E	8	0	8
D	6	8	0

$\longrightarrow$

	A,C,D	B,E
A,C,D	0	8
B,E	8	0

Figure 9: Recalulation of distances, after the merge of {A,C} and {D}. On the left side, we see the old distance matrix. On the right side we see the new distance matrix, where {A,C} and {D} are merged.

8. *Figure 10 illustrates this step of the algorithm. {A,C, D} and {B,E} are remaining. The algorithms merges {A,C,D} and {B,E} into {A,C,D, B, E}. In the guide tree a new node is introduced, the node which connects* A, C *and* D *and the node which connects* B *and* E, *are connected to it. The algorithm assigns* $\frac{d(\{A,C,D\},\{B,E\})}{2} = \frac{8}{2} = 4$ *total length to each branch. Observe, that the 1 is obtained calculating* $4 - 3 = 1$. *Observe, that the 2 is obtained calculating* $4 - 2 = 2$.

	A,C,D	B,E
A,C,D	0	8
B,E	8	0

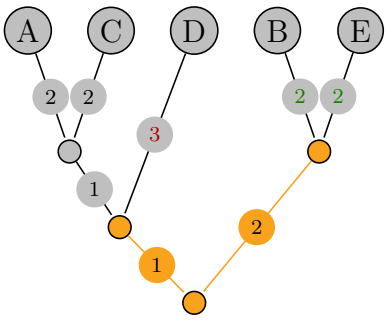

Figure 10: Merge of {A,C,D} and {B,E}. On the left side, we see the distance matrix. On the right side we see the guide tree, where the orange colored nodes branches represent the changes made to the guide tree.

9. *There are no more clusters to merge, the algorithm terminates. Figure 11 illustrates the final state of the distance matrix and guide tree.*

	A,C,D,B,E
A,C,D,B,E	0

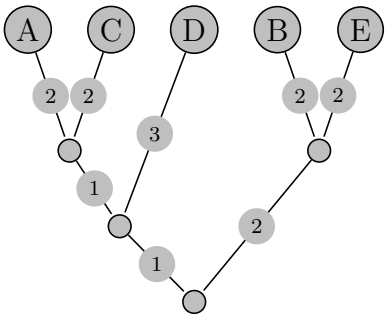

Figure 11: On the left side, we see the final distance matrix. On the right side we see the final guide tree

3 Multiple Sequence-Alignment

3.1 Feng-Doolittle-Algorithm

The Feng-Doolittle-Algorithm is used for progressive alignment and the workflow is as follows:

1. INPUT: a set of sequences S.
2. Perfom all pairwise sequence alignments using Needleman Wunsch or another alignment algorithm.
3. Convert the scores of all alignments to evolutionary distances.
4. Contruct a guide Tree.

5. Compute the MSA by traversing the guide tree node root to the bottom and obtain a MSA. There are three cases which are handled:

   Case 1: Apply operation 1 on the sequences returned by the leafs and return the alignment.

   Case 2: Apply operation 2 on the sequence and the alignment and return the alignment.

   Case 3: Both Children are inner nodes, Apply operation 3 on the alignments and return the alignment.

   Operation 1 computes best pairwise alignment of two nodes and changes occurences of gap symbol to X.

   Operation 2 aligns a sequence S to an alignment A where we 1. Compute pairwise alignment score between S and all sequences S' in A, 2. Align S according to the sequence in the best pairwise alignment, 3. Change occurences of gap symbol to X.

   Operation 3 aligns an alignment A1 to an alignment A2. For each pair of sequences S1 in A1 and S2 in A2 compute pairwise alignment score Align A1 and A2 according to the pairwise alignment with minimal distance. Change occurences of gap symbol to X

4 Nussinov

We now introduce the algorithm of nussinov, which is a basic Dynamic Programming algorithm which was one of the first attempts at predicting the structure of RNA molecules computationally. The nussinov algorithm get a RNA sequence as input. The aim of the algorithm is to find out the pairings of the nucleotides in the sequence and compute a optimal structure. We can break down the problem into many subproblems which are easier to compute than the full problem. Once we have obtained the solution for the subproblems, we can retrieve the solution to the full problem by doing a backtrace. In order to know how good our structure is, we assign a score to it. One possible scoring scheme could be adding 1 to the score per paired set of nucleotides, and 0 otherwise. We want a pairing that will give us the highest possible score. This quantity we call $score(i, j)$, where i, j are the indices of the sequence between which we obtain the pairing score. The nussinov algorithm computes a folding score for all substrings bound by i and j and store the value in what is known as a dynamic programming matrix. The matrix has shape $N \times N$, N is the length of the input sequence. We need a way to evaluate scores given a subsequence. To do this, we set some rules on the structure of an RNA sequence: If i and j form a pair: (1) The pair i and j must form a valid watson-crick pair. (2) $i < j - 4$. This ensures that bonding is not happening between positions that are too close to each other, which would produce steric clashes. (3) If pair (i, j) and

(k, l) are in the structure, then $i < k < j < l$. This ensures that there is no crossing over of pairs which would result in pseudoknots. (4) No base appears in more than one pair.

The workflow of the algorithm is a s follows:

1. Initialize the matrix D.
2. fill the matrix D recursively by following the following recursion:

$$matrix[i][j] = \begin{cases} 0 \text{ if } i >= j - min_loop_length \\ max(unpaired, paired) \text{ else} \end{cases}$$

   where *unpaired* is the score of letters i,j-1,

$$unpaired = score(i, j-1, sequence)$$

   paired is the score of pairing letter at j, $paired = [1 + score(i, k-1) + score(k+1, j-1)|$ for k in range $(i, j-4)$, if $sequence[k]$ and $sequence[j]$ is a basepair]
3. traceback recursively starting at $i = 0, j = N$. if j is unpaired, there will be no change in score when we take it out, so we just recurse to the next index. if the score at i, j is the result of adding 1 from pairing (j, k) and whatever score comes from the substructure to its left $(i, k-1)$ and to its right $(k+1, j-1)$. if $k - 1 < 0$ and $matrix[i][j] == matrix[k+1][j-1] + 1$, we add k, j to our structure, and traceback from $k+1$, $j-1$. If $k - 1 >= 0$, and $matrix[i][j] == self.matrix[i][k-1] + matrix[k+1][j-1] + 1$, we append k, j to our structure, and explore from $i, k-1$ and $k+1, j-1$

References

[1] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[2] A. Apostolico, M. Comin, A. W. M. Dress, and L. Parida, "Ultrametric networks: a new tool for phylogenetic analysis," *Algorithms for Molecular Biology*, vol. 8, p. 7, 2013.