Laboratory: Algorithms in Bioinformatics

Lab Report

Ashwath Sampath
MSc. Informatik
Matrikelnummer: 4368430

Chair for Bioinformatics
Department of Informatik
Albert-Ludwigs-Universität Freiburg
12 February 2018

Contents

Chapter 1

NEEDLEMAN-WUNSCH ALGORITHM

1.1 Motivation und Background

- For 2 DNA or protein sequences, there are often many possible global alignemnts. The problem of selecting the best of these alignments is not a trivial one. The best alignment is the one which implies the fewest number of mutations in the two sequences.
- Comparing all alignments is not practical because of the sheer number of possible alignments, even for small sequences. There was therefore a need for an efficient algorithm that could solve this problem of finding the optimal global alignment for 2 sequences.
- The Needleman-Wunsch algorithm was put forward by Saul Needleman and Christian Wunsch in 1970 to solve this problem.

1.2 The Needleman-Wunsch Algorithm

- The Needleman-Wunsch algorithm is an algorithm which is used to find the optimal global alignment between two sequences. It is a dynamic programming solution with the following inputs and outputs.
- Input: 2 DNA or Protein Sequences a and b
- Output: Optimal alignment between a and b with insertions, deletions and substitutions.
- The Needleman-Wunsch matrix for prefix alignments of a and b is defined as:

$$(D_{ij})_{0 \leq i \leq |a| \; and \; 0 \leq j \leq |b} \tag{1.1}$$

with

$D_{ij} := min\{w(u^*, v^*) | (u^*, v^*) is \; an \; alignment \; of \; prefixes \; (a_1, ..., a_n, b_1, ..., b_n)$.
Note that w is the cost of aligning u^* with v^*

- Coarse description of the method: The Needleman-Wunsch algorithm involves using dynamic programming to find the final alignment. Here, the two sequences a and b

are initially lined up against each other in a matrix (along with an extra row and column for the empty sequence). This extra row and column are initialized to the previous row/column score + the gap cost (the cost to create a gap, which is created by lining up a base/amino acid against nothing). Then, the matrix is filled row-wise: each cell is calculated from the neighbouring 3 cells based on whether it is an insertion, deletion or substitution. The optimal score is obtained at cell D(m+1,n+1). From here, a traceback to the initial cell gives the optimal path(s). The optimal sequence alignments are obtained from the tracebacks.

Initialization:
$D_{0,0} = opt.\ alignment\ of\ (a_1, ..., a_0, b_1, ..., b_0) = 0$
$D_{0,j} = opt.\ alignment\ of\ (a_1, ..., a_0, b_1, ..., b_j) = \sum_{k=1}^{j} w(-, b_k)$
where $w(-, b_k)$ gives the score obtained for aligning the k^{th} character of sequence b with a gap in sequence a.
$D_{i,0} = opt.\ alignment\ of\ (a_1, ..., a_i, b_1, ..., b_0) = \sum_{k=1}^{i} w(a_k, -)$
where $w(a_i, -)$ gives the score obtained for aligning the k^{th} character of sequence a with a gap in sequence b.

Recursion: $\forall i, j > 0$:

$$D_{ij} = min \begin{cases} D_{i,j-1} + w(-, b_j), \\ D_{i-1,j-1} + w(a_i, b_j), \\ D_{i-1,j} + w(a_i, -) \end{cases}$$

Traceback: The traceback matrix $tr_{i,j}$ for $D_{i,j}$ is defined as the matrix of elements $tr_{i,j} \subseteq \{\leftarrow, \uparrow, \nwarrow\}$ where
$tr_{0,0} = \phi$
$tr_{0,j} = \leftarrow$
$tr_{i,0} = \uparrow$
$\forall i, j > 0$,
$\nwarrow \in tr_{i,j} \Leftrightarrow D_{i,j} = D_{i-1,j-1} + w(a_i, b_j)$,
$\uparrow \in tr_{i,j} \Leftrightarrow D_{i,j} = D_{i-1,j} + w(a_i, -)$,
$\leftarrow \in tr_{i,j} \Leftrightarrow D_{i,j} = Di, j - 1 + w(-, b_j)$.

1.2.1 Procedure

- **Defintions:**
Dynamic Programming is a method by which the smallest sub-problems are solved first, the results of these are used to solve increasingly larger problems, until the final solution is obtained. All dynamic programming problems need a scoring function, a recursion and an initialization. There are generally 2 steps:
1. fill in a matrix D using the recursion.
2. do a traceback using the filled-in matrix D to find the best alignment(s).

A *sequence* S can be either a string of Nucelotide bases (DNA and RNA), or a string of amino acids (proteins).

Alignment: If σ is the alphabet of amino acid/DNA characters and if 2 words $s, b \in \sigma^*$, an *alignment* of (a,b) consists of two sequences $a^*, b^* \in (\sigma U\{-\})^*$ such that (i) $|a^*| = |b^*|$ (ii) $\forall 1 \leq i \leq |a^*| :!(a_i^* = - = b_i^*)$ (iii) $a^*|\sigma = aandb^*|\sigma = b$

Edit operation: An *edit operation* is an insertion, deletion or substitution applied on a sequence. An edit operation modifies the sequence. When there are 2 sequences, an insertion in a sequence S1 is the same as a deletion in a sequence S2 with a '-' symbol being inserted in S2's alignment. Conversely, a deletion from S1 corresponds to an insertion into S2, and results in a *(*gap) symbol '-' being inserted in S1's alignment.

Substitution matrix: A substitution matrix is a scoring matrix which is used to get the score between all pairs of amino acids (it also includes the score of aligning an amino acid with itself). 2 popular substitution matrices which are commonly used are PAM250 and BLOSUM62.

Scoring function: A scoring function is used to assign costs to insertions, deletions and substitutions. Substituation matrices can be used to get the substitutions scores.
Gap Penalty: A gap penalty is the cost to be paid while inserting a gap into one of the sequences. It is a function that is subadditive.
$g(k+l) \leq g(k) + g(l)$

- **Detailed procedure description:** The Needleman Wunsch is a dynamic programming method by which the best alignment of 2 sequences S1 (of length m) and S2 (of length n) is obtained.
Step 1: To fill in the matrix, take one sequence (say S1) on top, the other (say S2) on the left. Insert 1 extra row (which corresponds to aligning S2 with the empty sequence) and 1 extra column (which corresponds to aligning S1 with the empty sequence).
Step 2: Initialize the first row and column of the matrix as per the initialization rule given above.
Step 3: Calculate the score for each cell of the matrix from the scores of 3 cells: it is the maximum of
(i) the cell on top : cost of inserting a gap (gap is inserted into S2)
(ii) the cell on the left : cost of inserting a gap (gap is inserted into S1)
(iii) the cost of substituting the character in S2 for the character in S1. This can be either a match (where the same character appears in S1 and S2 at that cell), or a mismatch (where one character is replaced by a different character).
The scores for substitution are generally obtained from a scoring matrix like PAM or BLOSUM. The gap cost needs to be chosen.
Step 4: Fill in the matrix row-wise from the $D_{1,1}$ till $D_{m+1,n+1}$. Here each cell corresponds to a short alignment, and gives its score. For example, $D_{1,1}$ corresponds to the alignment of of just the first character of S1 with the first character of S2, $D_{3,4}$ gives the score for aligning the first 3 characters of S1 with the first 4 characters of S2, and $D_{m+1,n+1}$ gives the cost of aligning the complete sequences S1 and S2.
Step 5: The score in $D_{m+1,n+1}$ is the optimal score for the alignment of S1 and S2.

The pseudocode is shown in Figure 1.1.

Step 6: Traceback: The next step is the traceback step. We start at the $D_{m+1,n+1}$ cell and compute which surrounding cell it was calculated from: the left one, the top one, or the northwest diagonal one. An appropriate arrow is inserted in the traceback matrix. This is done from the bottom until we reach the top-left cell ($D_{0,0}$ These arrows are used to get the final traceback (there can be multiple tracebacks), and hence the optimal sequence alignment(s).

$D_{0,0} := 0$
for $i := 1$ to n **do**
  $D_{i,0} := D_{i-1,0} + w(a_i, -)$
end for
for $j := 1$ to m **do**
  $D_{0,j} := D_{0,j-1} + w(-, b_j)$
end for
for $i := 1$ to n **do**
  for $j := 1$ to m **do**

$$D_{i,j} := \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) \\ D_{i-1,j} + w(a_i, -) \\ D_{i,j-1} + w(-, b_j) \end{cases}$$

  end for
end for

Figure 1.1: Pseudocode of traceback

(a) Initialization of Needleman-Wunsch matrix (b) Filling out the Needleman-Wunsch matrix

Figure 1.2: Filling the Needleman-Wunsch matrix

- The time and space complexity of the Needleman-Wunsch algorithm are both $O(n^2)$.

1.2.2 Example

- An example of the Needleman-Wunsch algorithm is shown in Figures 2-5. Figures 2 and 3 show the initialization and filling of the Needleman-Wunsch matrix, and figures 4 and 5 show the traceback process. The final optimal score obtained is 2, and there are 2 possible alignments (from 2 tracebacks). They are:
  A– –T
  AAGT

  –A–T
  AAGT

(a) Start of traceback

(b) Complete traceback

Figure 1.3: Traceback

1.3 Summary

- The Needleman-Wunsch algorithm is the simplest global pairwise sequence alignment algorithm. It is reasonably efficient, but has some drawbacks. The space complexity for computation of similarity calculation can be reduced by storing only the current and last row (Hirschberg algorithm). Also, linear gap costs are not biologically realistic as it is often cheaper to extend an existing gap than to create a new gap.

1.4 Literature

- [1] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2017_WS/V_Bioinformatik_1/sequence-align.pdf
- [2] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2017_WS/V_Bioinformatik_1/needleman-wunsch.pdf
- [3] http://math.mit.edu/classes/18.417/Slides/alignment.pdf

Chapter 2

Gotoh Algorithm

- One problem with the Needleman-Wunsch algorithm is that it uses linear gap penalties. This is biologically unrealistic . In reality, we generally find many more large gaps than small gaps. This is why an algorithm like the Gotoh algorithm is necessary. It uses affine gap penalties: where creation of gaps is more expensive (has a higher penalty) than extension of existing gaps.
- Another algorithm, the Waterman Smith Beyer algorithm, considers gap penalties of different sizes, but its time efficiency is $O(n^3)$.
- The Gotoh algorithm is more efficient as it works in $O(n^2)$ time, while keeping the benefit of different gap opening and extension costs.

2.1 The Gotoh Algorithm

- The Gotoh algorithm, like the Needleman-Wunsch algorithm, is a pairwise sequence algorithm used to align 2 sequences. In addition to the alignment matrix D, it defines 2 further matrices P and Q.
- Input: 2 DNA or Protein Sequences a and b
- Output: Optimal alignment between a and b with insertions, deletions and substitutions.
- The dimensions of the 3 matrices used in the Gotoh algorithm are given below. The recursion is defined in the next section.

$$(D_{ij})_{0\leq i\leq |a| \text{ and } 0\leq j\leq |b} \tag{2.1}$$

$$(P_{ij})_{0\leq i\leq |a| \text{ and } 0\leq j\leq |b} \tag{2.2}$$

$$(Q_{ij})_{0\leq i\leq |a| \text{ and } 0\leq j\leq |b} \tag{2.3}$$

- **Coarse description of the algorithm:** The Gotoh algorithm is a dynamic programming algorithm which uses 3 different matrices D, P and Q. The 2 sequences are lined up against each other in the 3 matrices. The matrices all have an extra column and row for aligning with the empty sequence. The D matrix is used to specify substitutions or to traverse to the other 2 matrices. The P matrix is used to add gaps in sequence 2 and the Q matrix is used to add gaps in Sequence 1. The matrices are filled in the order P, Q and then D for each cell. To get the final alignment, a traceback is done from $D_{m+1,n+1}$, where m and n are the lengths of the

first and second sequences respectively. The traceback invloves all 3 matrices and the final alignment is obtained when the algorithm traverses back to $D_{0,0}$.

Initialization:
$D_{0,0} = opt.\ alignment\ of\ (a_1, ..., a_0, b_1, ..., b_0) = 0$
$D_{0,j} = opt.\ alignment\ of\ (a_1, ..., a_0, b_1, ..., b_j) = g(j)$
where g(j) is the cost of adding a new gap in successive columns.
$D_{i,0} = opt.\ alignment\ of\ (a_1, ..., a_i, b_1, ..., b_0) = g(i)$
where g(i) is the cost of adding a new gap in successive rows.

We recurse only on the first index for P. So we initialize only for $P_{0,j}$
$P_{0,j} = \infty$
$P_{i,0}$ = Not used

We recurse only on the second index for Q. So we initialize only for $Q_{i,0}$
$Q_{i,0} = \infty$
$Q_{0,j}$ = Not used

Recursion: $\forall i, j > 0$:

$$D_{ij} = min \begin{cases} D_{i-1,j-1} + w(a_i, b_j), \\ P_{i,j}, \\ Q_{i,j} \end{cases}$$

where for $1 \leq i \leq |a|$ *and* $1 \leq j \leq |b|$,

$$P_{i,j} = min \begin{cases} D_{i-1,j} + g(1) \\ P_{i-1,j} + \beta \end{cases}$$

$$Q_{i,j} = min \begin{cases} D_{i,j-1} + g(1) \\ Q_{i,j-1} + \beta \end{cases}$$

Order of calculation:
initialization
for i=1 to n
    for j=1 to n
        Calculate $P_{i,j}$
        Calculate $Q_{i,j}$
        Calculate $D_{i,j}$
    end
end

Traceback:
$tr^D \in \{^D\nwarrow, ^Q\bullet, ^P\bullet\}$
$\forall i, j > 0 : \ ^D\nwarrow \in tr^D_{i,j} \Leftrightarrow D_{i,j} = D_{i-1,j-1} + w(a_i, b_j),$
$^Q\bullet \in tr^D_{i,j} \Leftrightarrow D_{i,j} = Q_{i,j},$
$^P\bullet \in tr^D_{i,j} \Leftrightarrow D_{i,j} = P_{i,j},$

$tr^P \in \{^D\uparrow, ^Q\uparrow\}$
$\forall i, j > 0 : \ ^P\uparrow \in tr^P_{i,j} \Leftrightarrow P_{i,j} = P_{i-1,j} + \beta,$

$$^{D}\uparrow \in tr^{P}_{i,j} \Leftrightarrow P_{i,j} = D_{i-1,j} + g(1),$$

$tr^{Q} \in \{^{D}\leftarrow, ^{Q}\leftarrow\}$
$\forall i,j > 0: \; ^{Q}\leftarrow \in tr^{Q}_{i,j} \Leftrightarrow Q_{i,j} = Q_{i,j-1} + \beta,$
$^{D}\uparrow \in tr^{P}_{i,j} \Leftrightarrow Q_{i,j} = D_{i,j-1} + g(1)$

Finding Alignments:
a: $\nwarrow, \uparrow \Leftrightarrow$ *symbol* a_i
$\leftarrow \Leftrightarrow -$
b :$\nwarrow, \leftarrow \Leftrightarrow$ *symbol* b_j
$\uparrow \Leftrightarrow -$
$\bullet$ = change of matrix

- Advantages: The Gotoh algorithm is an efficient algorithm which uses affine penalty gaps. Linear gaps are not biologically-realistic, and sub-additive gap penalty algorithms are inefficient even though they consider gaps of different lengths. The Gotoh algorithm overcomes both of these problems.

2.1.1 Procedure

- *Definitions:*
  Affine gap cost/penalty: A gap penalty is called affine, iff there are real constants α and β such that $\forall k \in \mathbb{N}$:

$$g(k) = \alpha + (\beta \times k) \tag{2.4}$$

- **Detailed Procedure Description:** Consider 2 sequences S1 and S2. We use the Gotoh algorithm to find the optimal alignment(s) between them, considering affine gap costs.
  Step 1: To fill the D, P and Q matrices, take one sequence (say S1) on top, the other (say S2) on the left. Insert 1 extra row (which corresponds to aligning S2 with the empty sequence) and 1 extra column (which corresponds to aligning S1 with the empty sequence).
  Step 2: Initialize the first row and column of all 3 matrices as per the initialization rules given above.
  Step 3:
  (i) Calculate each $P_{i,j}$ cell from the P cell in the previous row ($P_{i-1,j}$) and the D cell in the previous row ($D_{i-1,j}$).
  (ii) Calculate each $Q_{i,j}$ cell from the Q cell in the previous column ($Q_{i,j-1}$) and the D cell in the previous column ($D_{i,j-1}$).
  (iii) Calculate each $D_{i,j}$ cell from $D_{i-1,j-1}$, $P_{i,j}$ and $Q_{i,j}$

  Step 4: The filling of the matrix should be done row-wise, P and Q values are calculated first, and then the corresponding D value is calculated.
  Step 5: The score in $D_{m+1,n+1}$ gives the optimal score for the alignment of S1 and S2.
  Step 6: Traceback: The next step is the traceback step. We start at the $D_{m+1,n+1}$ cell and find which cell it was calculated from, and assign the suitable arrows, as explained in the previous section. These arrows are used to get the final traceback (there can be multiple tracebacks), and hence the optimal sequence alignment(s).

- The Gotoh algorithm runs in $O(n^2)$ space and $O(n^2)$ time.

$(D_{i,j}) =$

		A	C	C	T
	0	5	6	7	8
C	5	1			
C	6				

$(Q_{i,j}) =$

		A	C	C	T
	0	–	–	–	–
C	∞	10			
C	∞				

$(P_{i,j}) =$

		A	C	C	T
	0	∞	∞	∞	∞
C	–	10			
C	–				

(a) Initialization of the matrices

$(D_{i,j}) =$

		A	C	C	T
	0	5	6	7	8
C	5	1	5	6	8
C	6	6	1	5	7

$(Q_{i,j}) =$

		A	C	C	T
	0	–	–	–	–
C	∞	10	6	7	8
C	∞	11	11	6	7

$(P_{i,j}) =$

		A	C	C	T
	0	∞	∞	∞	∞
C	–	10	11	12	13
C	–	6	10	11	13

(b) Filled out Gotoh matrices

Figure 2.1: Filling the matrices

$(D_{i,j}) =$

		A	C	C	T
	0	5	6	7	8
C	5	1	5	6	8
C	6	6	1	5	7

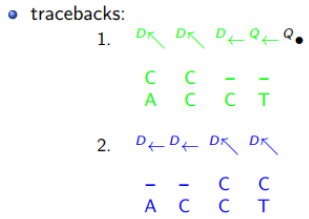
$(Q_{i,j}) =$

		A	C	C	T
	0	–	–	–	–
C	∞	10	6	7	8
C	∞	11	11	6	7

$(P_{i,j}) =$

		A	C	C	T
	0	∞	∞	∞	∞
C	–	10	11	12	13
C	–	6	10	11	13

(a) One possible traceback



(b) Both the possible tracebacks found

Figure 2.2: Traceback

2.1.2 Example

- Consider the following 2 sequences:
  a = CC
  b= ACCT

  The steps of the Gotoh algorithm (taken from [2]) are shown in the Figure 2.1 and Figure 2.2.
  The alignemnts obtained are:
  CC– –
  ACCT

  – –CC
  ACCT

2.2 Summary

- The Gotoh algorithm is an efficient dynamic programming algorithm which is used to find (pairwise) alignments between 2 sequences. It uses affine gap penalties instead of linear gap penalties. There are different costs for opening new gaps (higher cost) and extension of existing gaps (lower cost). The algorithm makes use of 3 matrices to get the final alignment, but still runs in $O(n^2)$ space and time.

2.3 Literature

- [1] http://math.mit.edu/classes/18.417/Slides/alignment.pdf
- [2] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2016_WS/V_Bioinformatik_1/W-S-B_and_Gotoh.pdf

Chapter 3

Feng-Doolittle Algorithm

- The Feng Doolittle Algorithm is an algorithm used to find multiple sequence alignments, i.e. alignments between 3 or more sequences. It is a progressive alignment algorithm in which a guide tree is first created using phylogenetic tree reconstruction, and a multiple alignment is built bottom-up along the guide tree. It finally returns the alignment of all sequences that is produced at the root of the tree.

3.1 The Feng-Doolittle Algorithm

- The Feng-Doolittle algorithm, unlike the algorithms seen so far, is a progressive alignment algorithm used to find multiple sequence alignments. The Needleman-Wunsch and Gotoh algorithms, on the other hand, were pairwise sequence alignment algorithms.
- The Needleman-Wunsch algorithm can be used to do pairwise alignment of every pair of sequences and to thus build a multiple sequence alignment, but this is very inefficient: $O(n^n)$
- Input: 3 or more sequences
- Output: Optimal alignment between all the sequences with insertions, deletions and substitutions.
- **Coarse description of the algorithm:** The Feng-Doolittle algorithm is used to align 3 or more sequences together. It calculates the similarities between each pair of sequences, converts the similarities into distances. The obtained distance scores are used to generate a UPGMA or a WPGMA guide tree. Finally, progressive alignments are generated along the guide tree by combining sub-alignments. An important feature of the Feng-Doolittle algorithm is that once a gap is created, it stays a gap forever.

3.1.1 Procedure

- *Definitions:*
  Multiple sequence alignment: Let $a^1, ..., a^n$ be N sequences. A multiple sequence alignment (MSA) of $a^1, ..., a^n$ is a matrix

$$A = (A_{i,j})_1 \leq i \leq N, 1 \leq j \leq K \tag{3.1}$$

where the rows correspond to the sequences, and the columns correspond to the MSA columns.

Here,
1. $\forall i, j : A_{i,j} \in \sum U\{-\}$, where $\sum$ is the alphabet of all sequences.
2. $\forall i : (A_{i,1}...A_{i,k})|\sigma = a^i$ (sequence i is in row i)
3. $\forall j : not(\forall i : A_{i,j} = -)$, i.e. there exists no column with only gaps

Sum of pairs scoring: $S_c(A_{1j}, ..., A_{Nj}) = \sum_{1 \leq k < l \leq N} s_p(A_{k,j}, A_{l,j})$
where s_p is obtained using a substiution matrix: PAM or BLOSUM.
The sum of pairs score is simply the sum of all pairwise alignments.

- **Detailed Procedure Description:**
  Step 1: Compute the pairwise sequence alignment scores (similarity) between every pairs of sequences.
  Step 2: Convert the obtained simiilarity scores into distances using the following formula:

$$D = -log\ S_{eff} = -log \frac{S_{obs} - S_{rand}}{S_{max} - S_{rand}} \tag{3.2}$$

  where S_{obs} is the observed score for a pair of sequences,
  S_{max} is the maximum possible score,
  S_{rand} is the expected score of an alignment of two random sequences of the same length and composition (i.e. with the same amino acids).
  The effective score S_{eff} can be viewed as a normalized percentage similarity with values between 0 and 1. It converges exponentially slowly to 0 for increasing evolutionary distance.
  Step 3: The distances, D, are stored in a matrix. This matrix is used to construct a rooted guide tree using either UPGMA or WPGMA.
  Step 4: A progressive alignment is generated bottom-up from the guide tree. Alignments of 2 child sequences are obtained at a parent node, and the alignment of all the sequences is obtained at the root of the tree.
  Step 5: The way this works is that the most similar two alignments are put in a 'group' (group 1) and aligned together (say S1 and S2 are in group 1). After this, the next sequence S3 is aligned with the best sequence in group 1 (the one which is most similar to S3, say this is S1). This sequence S3 now goes into group 1, and corresponding gaps are added in the sequence which isn't aligned in group 1 (S2). This process continues for all the other sequences as per the distribution of nodes in the phylogenetic tree.

- One disadvantage of this algorithm is that conservation of columns is not used. Another disadvantage is that all alignments are determined by pairwise sequence alignments.

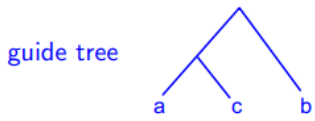
- The Feng-Doolittle algorithm runs in $O(n^2)$ space and $O(n^2)$ time.

3.1.2 Example

- Consider the following 3 sequences:
  a = ACCAT
  b = ACGGAT

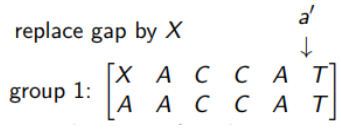
$a \leftrightarrow b = 2$	A C – C A T A C G G A T
$b \leftrightarrow c = 0$	A C G G A T A A C C A T
$a \leftrightarrow c = 4$	– A C C A T A A C C A T

(a) Pairwise similarities of the 3 sequences



(b) Guide tree generated from alignments

Figure 3.1: Steps 1 and 2: generating guide tree from pairwise alignments



(a) Replace gaps by X

$b \leftrightarrow a' = 2$	X	A	C	–	C	A	T
	–	A	C	G	G	A	T
$b \leftrightarrow c = 0$	A	C	G	G	A	T	
	A	A	C	C	A	T	

(b) Join b to one of the sequences in group 1

$$\text{group2: } \begin{bmatrix} X & A & C & - & C & A & T \\ A & A & C & - & C & A & T \\ - & A & C & G & G & A & T \end{bmatrix} \rightarrow \begin{bmatrix} X & A & C & X & C & A & T \\ A & A & C & X & C & A & T \\ X & A & C & G & G & A & T \end{bmatrix}$$

(c) Getting the final alignment

Figure 3.2: Steps 3-5 of the Feng-Doolittle algorithm

c = AACCAT

- Scoring function:

$$S(x, y) = \begin{cases} 1 \; if \; x = y \\ -1 \; otherwise \end{cases} \tag{3.3}$$

- The steps of the Feng-Doolittle algorithm (taken from [2]) are shown in figures 3.1 and 3.2.
- After creating the 2 groups (in fig.3.2(a) and fig. 3.2(c), we align the groups together to get the multiple sequence alignment.
- We do this by considering all alignments for each sequence of group 1 with a sequence of group 2. Note that after aligning one sequence in group 1 with the 3rd sequence (in group 2), we have to add corresponding gaps in the other sequence in group 1.

3.2 Summary

- The Feng-Doolittle algorithm is a progressive alignment algorithm for aligning more than 2 sequences. It is reasonably efficient: $O(n^2)$ space and time. It uses a pairwise

alignment algorithm to get similarity scores, converts them into distance scores, and then builds a guide tree (usually UPGMA) which it uses to get the final multiple alignment.

3.3 Literature

- [1] https://ab.inf.uni-tuebingen.de/teaching/ss08/gbi/script/chapter06_multialign.pdf
- [2] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2016_WS/V_Bioinformatik_1/multiple-alignment.pdf

Chapter 4

UPGMA and WPGMA

- UPGMA and WPGMA are hierarchical bottom-up clustering algorithms which are used to construct phylogenetic trees. WPGMA uses simple averaging in intermediate clustering of 2 nodes/clusters while UPGMA takes weighted means while clustering 2 nodes/clusters.

4.1 The UPGMA and WPGMA Algorithms

- Coarse description: The WPGMA and UPGMA algorithms are simple 'agglomerative' or bottom-up clustering methods which are used to create phylogenetic trees. In both algorithms, the two nearest clusters are combined into a higher-level cluster. The distance between any 2 clusters A and B is the average of all distances between pairs of objects 'x' in A and 'y' in B. In other words, the distance is the mean distance between elements of each cluster. WPGMA just takes a simple mean while UPGMA assigns weights to each clusters to get a weighted mean.
- Input: n x n distance matrix D
- Output: A rooted phylogenetic tree (called a dendogram) which corresponds to the pairwise distance matrix

4.1.1 Procedure

- *Definitions:*
  Phylogenetic tree: A phylogenetic tree is an undirected, connected, aclyclic binary graph
  Graph:

$$G = (V, E), V \text{ are nodes or vertices and } E \subseteq V \times V \text{ are edges} \tag{4.1}$$

  where the rows correspond to the sequences, and the columns correspond to the MSA columns.

  Undirected graph: An undirected graph is a graph in which $(i, j) = (j, i)$
  Acyclic graph: In an acyclic graph, there exists a unique path between any two nodes
  Binary graph/tree: In a binary graph, each node has at most two children.
  Rooted tree: A rooted tree is a tree which has a distinct 'root' node.
  Additive Tree: A distance matrix is called an additive tree metric if $\exists$ a tree τ such that:

$$\forall i, j : D(i, j) = \sum weights \text{ on path } i \to j \tag{4.2}$$

Ultrametric tree: D(i,j) is called an ultrametric tree if distances from the root to each leaf are the same. UPGMA and WPGMA are ultrametric trees.

- **Detailed Procedure Description for UPGMA:**
  INPUT: n x n distance matrix D
  Step 1: Initialize set C to consist of n singleton clusters 1,...,n.
  Step 2: Initialize dist(A,B) on C by defining for all i and j in C:

$$dist(i,j) = D(i,j) \tag{4.3}$$

  Step 3: Repeat until $|C| = 2$, add root node following step 3.3.
  Step 3.1 Determine pair A,B of clusters in C such that dist(A,B) is minimal.

$$d_{min} = dist(A,B), A \neq B \tag{4.4}$$

  Step 3.2 Define new cluster E = A U B and update the cluster set.

$$C = C - \{A,B\}U\{E\} \tag{4.5}$$

  Step 3.3 Define node e with children a,b, where e has distance $d_{min}/2$ to all child leaves of e. Calculate branch weights with reference to the additivie tree metric.
  Step 3.4 Define for all $F \in C$ with $F \neq E$

$$dist(E,F) = dist(AUB,F) = \frac{|A|dist(A,F) + |B|dist(B,F)}{|A| + |B|} \tag{4.6}$$

- **Detailed Procedure Description for WPGMA:**
  INPUT: n x n distance matrix D
  Step 1: Initialize set C to consist of n singleton clusters 1,...,n.
  Step 2: Initialize dist(A,B) on C by defining for all i and j in C:

$$dist(i,j) = D(i,j) \tag{4.7}$$

  Step 3: Repeat until $|C| = 2$, add root node following step 3.3.
  Step 3.1 Determine pair A,B of clusters in C such that dist(A,B) is minimal.

$$d_{min} = dist(A,B), A \neq B \tag{4.8}$$

  Step 3.2 Define new cluster E = A U B and update the cluster set.

$$C = C - \{A,B\}U\{E\} \tag{4.9}$$

  Step 3.3 Define node e with children a,b, where e has distance $d_{min}/2$ to all child leaves of e. Calculate branch weights with reference to the additivie tree metric.
  Step 3.4 Define for all $F \in C$ with $F \neq E$

$$dist(E,F) = dist(F,E) = \frac{dist(A,F) + dist(B,F)}{2} \tag{4.10}$$

- WPGMA is in most cases an inferior method to UPGMA, and is therorfore almost never used.
- While UPGMA is a fast, simple approximation, it is not used any more for proper phylogeny studies because trees may have wrong distances/topology.
- The UPGMA and WPGMA algorithms run in $O(n^2)$ space and $O(n^2)$ time.

4.1.2 Example

An example for UPGMA is shown in Figure 4.2.
. WPGMA is almost the same except that it does not take weighted means.

- Fig. 4.2(a): First, the distance matrix is displayed. This is created from multiple sequences. A singleton cluster is initialized for {a}, {b}, {c}, {d} and {e}. These are the leaves of the tree.

- Fig. 4.2 (b): dmin =2. So, c and d are selected and combined into a new cluster. The new node f=c U d is inserted into the tree.
  dist(i,f) = (dist(i,c) + dist(i,d)) / 2

- Fig. 4.2 (c): dmin = 6. So, a and b are selected and combined into a new cluster. The new node g = a U b is inserted into the tree.

- Fig. 4.2 (d): dmin = 6. So, f and e are selected and combined into a new cluster. The new node h = f U e is inserted into the tree.

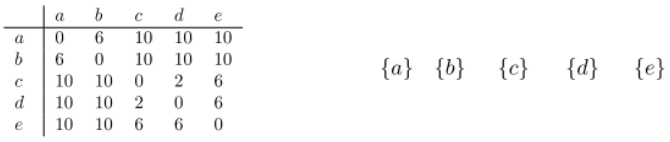
- Fig. 4.2 (e): dmin = 10. So, g and h are selected and combined into a new cluster. The new node is the root of the tree.

4.2 Summary

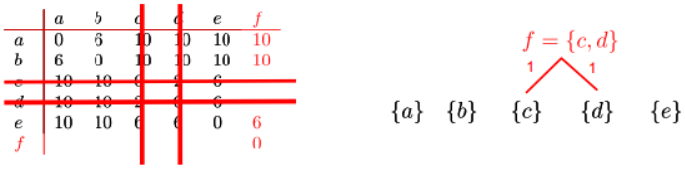
- UPGMA and WPGMA are 2 hierarchical clustering algorithms which are used to create phylogenetic trees. While they are simple and fast, they are not used very much for phylogeny any more as they don't offer the advantages of other methods such as Felenstein's method. Among the 2 algorithms, UPGMA is more often used than WPGMA because it uses weighted means in its calculations, and is hence more accurate.

4.3 Literature

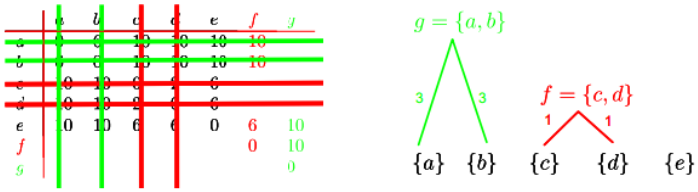
- [1] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2016_WS/V_Bioinformatik_1/phylo-UPGMA-sita.pdf
- [2] https://en.wikipedia.org/wiki/UPGMA
- [3] https://en.wikipedia.org/wiki/WPGMA
- [4] http://www.bioinf.uni-freiburg.de/Lehre/Courses/2016_WS/V_Bioinformatik_1/multiple-alignment.pdf
- [5] http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Feng-Doolittle



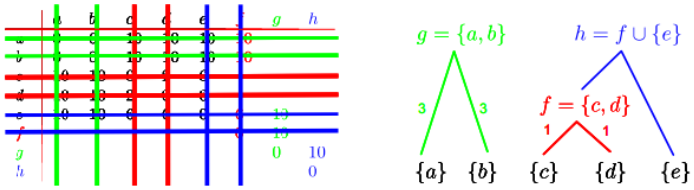
(a) Initial distance matrix and tree of singleton clusters



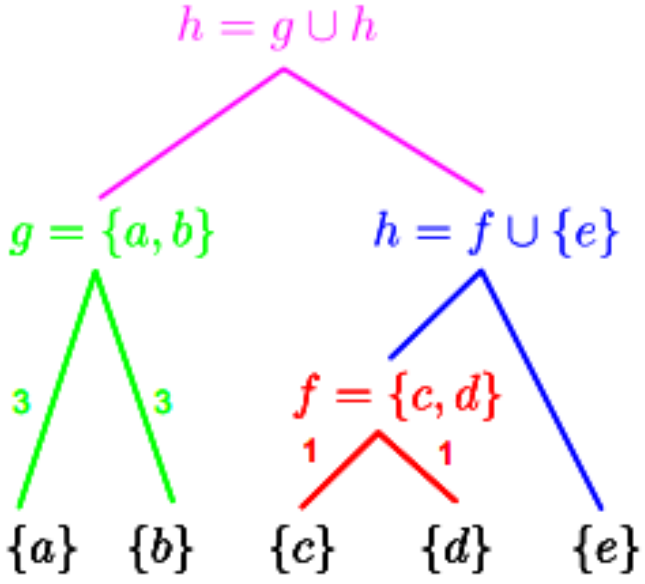
(b) Step 1: dmin = 2. Select c,d



(c) Step 2: dmin =6. Select a,b



(d) Step 4: dmin=6. Select f,e



(e) Step 5: dmin=10. Select g,h. Root created

Figure 4.1: Steps in constructing a UPGMA tree from a distance matrix

Chapter 5

Nussinov Algorithm

5.1 Motivation und Background

- In an RNA single strand, we want the number of bonds/base pairs to be maximized so that the molecule is more stable. Therefore, the RNA structure should be folded in such a way that the hydrogen bonds is maximized.
- However, it is NP-hard to find a crossing RNA-structure (a structure which contains pseudoknots) which maximizes the number of base pairs.
- To solve this problem, we instead search for all non-crossing structures. We look for an algorithm which can find non-crossing structures. The simplest algorithm for this is called the Nussinov algorithm.

5.2 The Nussinov Algorithm

The Nussinov algorithm is a recursive algorithm which is used to find the non-crossing RNA structure with maximum number of base pairs.

- Input an RNA Sequence S
- Output: Non-crossing RNA structure P of S that maximizes $|P|$
  where $|P|$ is the no. of base pairs in p.
- A Nussinov matrix N for sequence S is defined as:

$$N = (N_{ij})_{1 \leq i \leq n \, and \, i \leq j \leq n} \tag{5.1}$$

  where

  $N_{ij} := max\{|P||P \; is \; a$ non-crossing $ij - substructure \; of \; S\}$

  An RNA structure P of S is called an ij-substructure of S iff $P \subseteq \{i \dots j\}$

- Coarse description: The solution involves dynamic programming, where we recursively fill the matrix diagonal-wise. We line the sequence up against itself, initialize the principal diagonal (and the diagonal to its left) elements to be 0. Then we fill the matrix diagonal-by-diagonal till we get the maximum number of base pairs in cell N_{1n}. We finally do a traceback to find which bases bond to each other.

- Initialization and Recursion:

  Init: (for $1 \leq i \leq n$)
  $N_{ii} = 0$ and $N_{ii-1} = 0$

  Recursion: (for $1 \leq i < j \leq n$)

$$N_{ij} = max \begin{cases} N_{ij-1} \\ max_{S_k, S_j complementary} \quad i \leq k \leq j \quad N_{ik-1} + N_{k+1j-1} + 1 \end{cases}$$

- Case 1: if S_j is unpaired, the following new subproblem has to be solved.
  - *Maximize no. of base pairs between i and j-1*
- Case 2: if S_j maps to S_k, the problem is split into 2 subproblems :
  - *Maximize no. of base pairs between i and k-1.*
  - *Maximize no. of base pairs between k+1 and j-1 .*

5.2.1 Procedure

- *Defintions:*
  Let $S \in \{A, C, G, U\}^*$ be an **RNA sequence** of length $n = |S|$.
  An **RNA structure** of S is a set of base pairs:
  $P \subseteq \{(i, j) | 1 \leq i < j \leq n, S_i \; and S_j \; are \; complementary\}$
  such that the degree of P is at most one
  i.e. $for \; all (i, j), (i', j') \in P : (i = i' \Leftrightarrow j = j') \; and \; i' \neq j'$
  Two base pairs (i, j) and (i', j') are **crossing** iff
  $i < i' < j < j'$ *or* $i' < i < j < j'$
  Conversely, two base pairs (i, j) and (i', j') are **non-crossing** iff
  $i' < i < j < j'$ *or* $i < i' < j' < j$

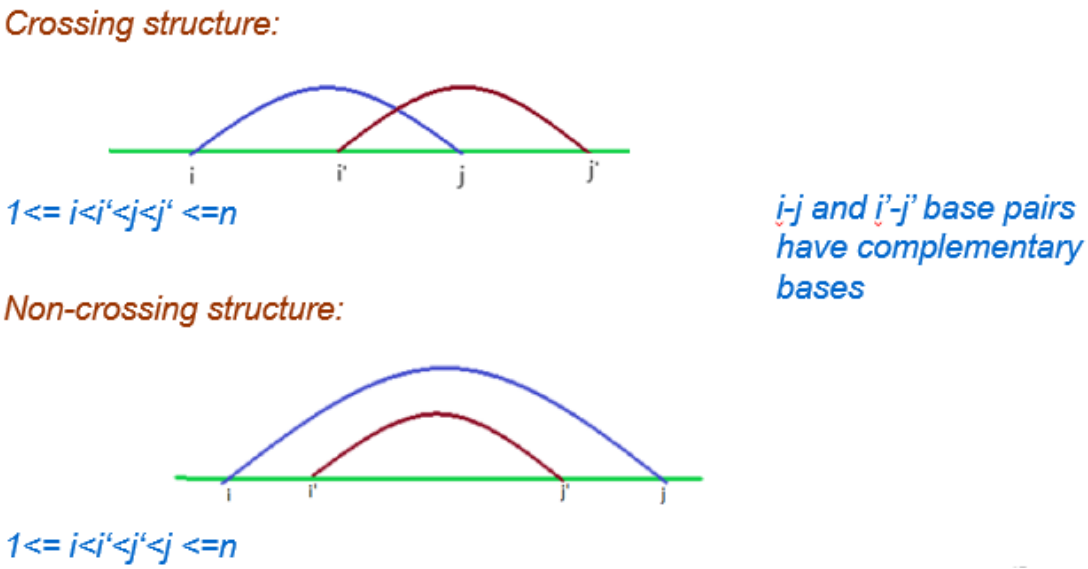


Figure 5.1: RNA Structures

A crossing structure is said to have 'pseudoknots', while a non-crossing structure does not have pseudoknots and is said to be pseudoknot-free.

- *Algorithm explanation:*
  The solution involves dynamic programming, where we recursively fill the matrix diagonal-wise. We line the sequence up against itself, and initialize the principal diagonal elements to be 0. Also, we set all the cells in the diagonal to the left of the principal diagonal to be zero.

  Then we start filling the matrix to the right of the principal diagonal, one diagonal at a time. We first fill the diagonal which corresponds to a distance of 1 between bases, then the diagonal which correspond to a distance 2 between bases and so on. The last diagonal is a single cell, N_{1n}, which gives the maximum no. of base pairs between base no. 1 and base no. n.

  Now we have to perform the traceback. This is done by starting at the cell in the top right corner (N_{1n}), and determining the recursion case (and the entries in N) that yielded the value for the cell. This is essentially a reverse-process of filling the matrix. The recursion pseudocode is shown in Figure 2.

CALL: traceback$(1, n)$ with time complexity $O(n^2)$

Procedure traceback(i, j)
  if $j \leq i$ **then**
    return
  else if $N_{ij} = N_{ij-1}$ **then**
    traceback$(i, j-1)$;
    return
  else
    for all $k : i \leq k < j$, S_k and S_j complementary **do**
      if $N_{ij} = N_{i\,k-1} + N_{k+1\,j-1} + 1$ **then**
        print (k,j);
        traceback$(i, k-1)$; traceback$(k+1, j-1)$;
        return
      end if
    end for
  end if=0

Figure 5.2: Pseudocode of traceback

- Theoretical analysis and details: The recursion takes $O(n^2)$ space and $O(n^3)$ time. The traceback takes $O(n^2)$ time

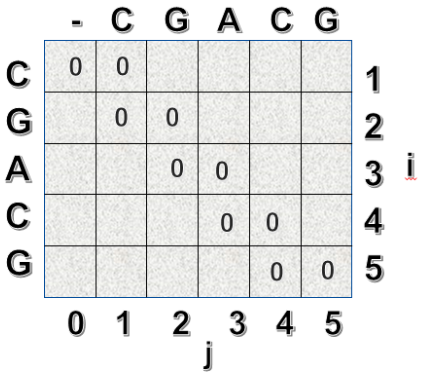
5.2.2 Example

A detailed example of the filling of the Nussinov Matrix for the RNA sequence CGACG is shown in Figure 3 (a)-(e). The final traceback is shown if Figure 3(f) with one of the optimal structures. Note that there is another optimal structure which is not found by the algorithm. The final solution can also be written in dot-bracket notation as:

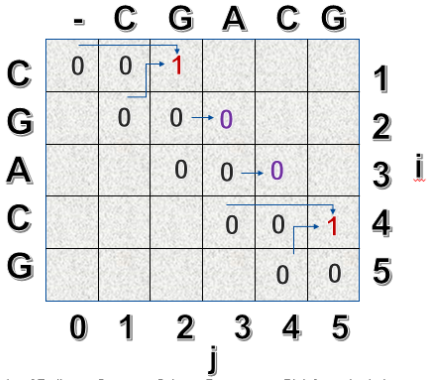
$$((.)) \tag{5.2}$$

5.3 Summary

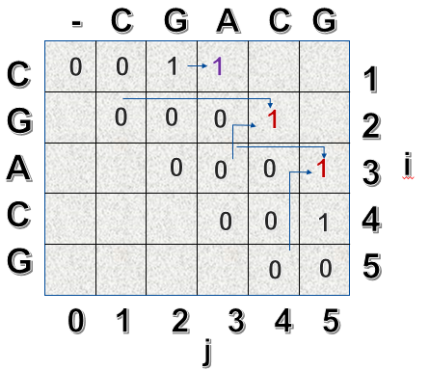
The Nussinov algorithm is an efficient algorithm to find the non-crossing RNA structure with maximum number of base pairs. It is a very simple algorithm which is easy to interpret. However, it has certain limitations. It cannot find crossing structures. It



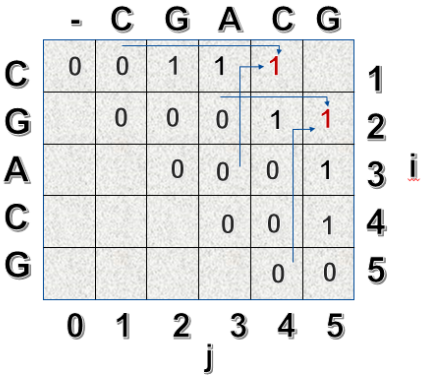
(a) Initialization of matrix



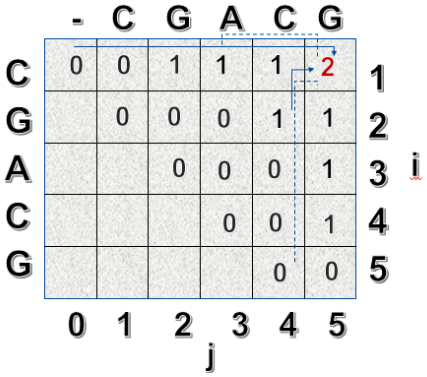
(b) Fill cells with dist(i,j) = 1



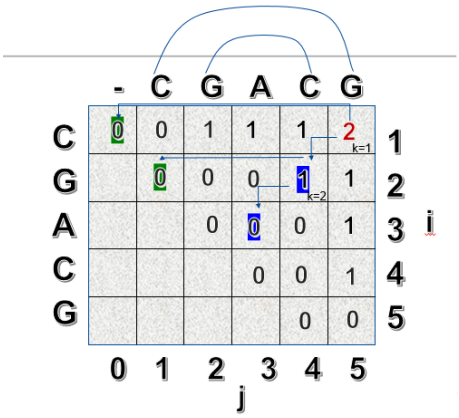
(c) Fill cells with dist(i,j) = 2



(d) Fill cells with dist(i,j) = 3



(e) Fill cells with dist(i,j) = 4



(f) Traceback and final structure

Figure 5.3: Example: Applying the Nussinov Algorithm

misses some optimal structures when there are multiple optima, and doesn't find sub-optimal structures. Finally, it does not consider common biological concepts like base pair stacking.

5.4 Literature

- http://math.mit.edu/classes/18.417/Slides/rna-prediction-nussinov.pdf
- http://www.bioinf.uni-freiburg.de/Lehre/Courses/2017_SS/V_BioinfoII/Nussinov.pdf
- http://www.bioinf.uni-freiburg.de/Lehre/Courses/2017_SS/V_BioinfoII/Nussinov-traceback-example.pdf
- http://web.cs.ucdavis.edu/~gusfield/cs122f10/ECS122A_11-01-10.wmv