Report

Algorithms in Bioinformatics

Joachim Wolff

Matrikelnumber: 3106152

January 25, 2015

Albert-Ludwigs-University Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Chair for Bioinformatics

Contents

1 Introduction **4**
- 1.1 Definitions . . . 4
  - 1.1.1 Sequence alignment . . . 4
  - 1.1.2 Score function . . . 4
  - 1.1.3 Gap . . . 5

2 Pairwise sequence alignment algorithms **6**
- 2.1 The Needleman-Wunsch algorithm . . . 6
  - 2.1.1 Procedure . . . 6
  - 2.1.2 Complexity . . . 7
  - 2.1.3 Example . . . 7
- 2.2 Gotoh algorithm . . . 8
  - 2.2.1 Procedure . . . 8
  - 2.2.2 Complexity . . . 10
  - 2.2.3 Example . . . 10

3 Multiple sequence algorithms **12**
- 3.1 Needleman-Wunsch n = 3 . . . 12
  - 3.1.1 Runtime . . . 12
- 3.2 Sum of pairs . . . 13
  - 3.2.1 Procedure . . . 13
- 3.3 Feng-Doolittle algorithm . . . 14
  - 3.3.1 Procedure . . . 14
  - 3.3.2 Complexity . . . 15
  - 3.3.3 Example . . . 15

4 Phylogeny/Clustering **17**
- 4.1 UPGMA / WPGMA . . . 17
  - 4.1.1 Procedure . . . 17
  - 4.1.2 Complexity . . . 18
  - 4.1.3 Example . . . 18

5 Basic structure prediction - RNA folding **20**
5.1 Nussinov algorithm 20
5.1.1 Procedure 20
5.1.2 Complexity 21
5.1.3 Example 21

Bibliography **23**

1 Introduction

An important aspect in biology is the knowledge if and how two different DNA or RNA sequences are related. For example, if there are two versions of a gene and one is resistant to some outer influence and one is not, it would be interesting to know which base pairs are mutated and are the reason to this resistant. An other example could be the knowledge of how close two humans are related. Given a parent and its child, the DNA of the child should be about 50% equal to the parents one. Also an interesting point is to know how close the DNA of animals are related to each other, this is an important method to understand and reconstruct the development of the evolution.
From a computer scientists point of view, the comparing of two DNAs is just a pairwise sequence alignment. The four bases in DNA, adenine (A), cytosine (C), guanine (G) and thymine (T) are reduced to the first letter of the base and than two DNA sequences can be compared pairwise for each character. This simple approach works unfortunately only if the two sequences are of equal length. To compute the alignments of sequences with a different length, gaps are introduced.

1.1 Definitions

1.1.1 Sequence alignment

To detect similarities or differences in two sequences both sequences have to be compared pairwise in each character. This process is called in general *sequence alignment*.

1.1.2 Score function

To compare the similarity or difference of two sequences to other comparisons the degree of similarity or difference has to be measured. The measure is called a *score function* $s(x)$ or *weight function* $w(x)$. The difference between a similarity score and a difference score is in general that in similarity a match between characters is scored with a 1 and a mismatch with 0, for difference otherwise. Also a similarity score is maximized, a difference score minimized.

1.1.3 Gap

The gap symbol *"-"* is introduced to get the ability to compare sequences of different length. The gap symbol stands for a deletion or insertion at a given position in a sequence.

2 Pairwise sequence alignment algorithms

2.1 The Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm was developed by Saul Needleman and Christian Wunsch [1] in 1969. They developed it "because the method of visual comparison is tedious and because the determination of the significance of a given result usually is left to intuitive rationalization[...]".

2.1.1 Procedure

Basis of the Needleman-Wunsch algorithm is a matrix D with $m+1 \times n+1$ as a datastructure, where m is the length of the input sequence a and n the length of input sequence b. The matrix is initialized with:

$$
\begin{aligned}
&D_{0,0} = 0 \\
&D_{i,0} = D_{i-1,0} + w(a_i, -) \\
&D_{0,j} = D_{0,j-1} + w(-, b_j).
\end{aligned}
$$

The cost function w defined as follows:

Definition

$$
w(a,b) = \begin{cases} 0, & \text{if } a = b \\ 1, & a \neq b \end{cases}
$$

$$
w(a,-) = 1
$$

$$
w(-,b) = 1
$$

The computation of the matrix is now defined recursive:

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

If the computation of the matrix is done the traceback to get the optimal alignment have to be computed. Start in cell $D_{m+1,n+1}$ and note every possible way to get into this cell. In general: if the symbol is an left arrow, a gap is inserted in a, if it is an up arrow, a gap is inserted in b. A diagonal arrow stands for the character from the input sequence a and b. Note that it is possible that there are multiple optimal alignments!

Traceback

$\forall i, j > 0 :$
$D_{i,j} = D_{i-1,j-1} + w(a_i, b_j) \Leftrightarrow \nwarrow$
$D_{i,j} = D_{i-1,j} + w(a_i, -) \Leftrightarrow \uparrow$
$D_{i,j} = D_{i,j-1} + w(-b_j) \Leftrightarrow \leftarrow$

2.1.2 Complexity

Needlman-Wunsch has a runtime and space complexity of $O(n^2)$.

2.1.3 Example

Given the two input sequences $a = AGC$ and $b = AC$ and a cost function like the one defined above, the Needleman-Wunsch algorithm computes a matrix as in Figure 2.1. For example, cell $D_{1,1}$ is the minimum of $D_{0,0} + 0 = 0$, $D_{0,1} + 1 = 2$ and $D_{1,0} + 1 = 2$, which is obvious 0. The traceback starts in cell $D_{m+1,n+1}$. As shown in Figure 2.2 left, it is only possible to get from cell $D_{2,1}$ to $D_{3,2}$:

$D_{3,2} = 1 == D_{2,1} + w(C, C) = 1 + 0 = 1$
$D_{3,2} = 1 \neq D_{2,2} + w(C, -) = 1 + 1 = 2$
$D_{3,2} = 1 \neq D_{3,1} + w(-, C) = 2 + 1 = 3$

The full traceback is shown in Figure 2.2 right. To compute the alignment, the arrows for each optimal alignment have to be pushed to a stack to get a reversed ordering. For the given example we pop $\nwarrow$ and note for the alignment the first character from sequence a and b which is for both A. The next pop gives us $\uparrow$, for sequence a the character is noted and for sequence b a gap. The last pop gives us $\nwarrow$, for both sequences the last character is noted. The computed sequence alignment is: $\begin{pmatrix} AGC \\ A-C \end{pmatrix}$.

	ϵ	A	C
ϵ	0	1	2
A	1	0	1
G	2	1	1
C	3	2	1

Figure 2.1: The computed matrix of the Needleman-Wunsch algorithm.

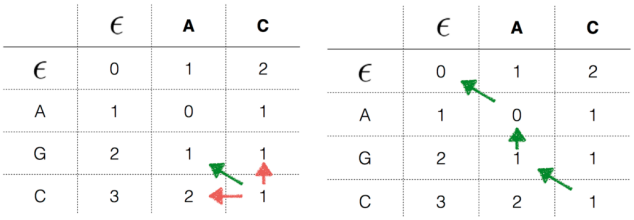

Figure 2.2: The traceback of the Needleman-Wunsch algorithm.

2.2 Gotoh algorithm

As good as the Needleman-Wunsch algorithm works it ignores an important fact in biology: extending a gap is much more likely than inserting a new one [2]. The scoring scheme of Needleman-Wunsch does not consider this. It has to be replaced by a so called *affine gap penalty*. The algorithm of Smith-Waterman-Bayer is considering a new scoring scheme but needs a runtime of $O(nm*(n+m)) \in O(n^3)$ [3]. Osamu Gotoh introduced in 1982 "a new algorithm which allows multiple-sized gaps but runs in essentially MN steps if the gap weight has a special form of $w_k = uk + v(v \geq 0, r \geq 0)$. " [4]

2.2.1 Procedure

***Definition*: Gap penalty**

A function which holds $\forall x, y : g(x+y) \leq g(x) + g(y)$ is called *subadditive*. If it holds $\exists \alpha, \beta : g(x) = \alpha + \beta * x$ the function is called *affine*.

Given this definition Gotoh developed an algorithm to compute an optimal paire-wise sequence alignment. The algorithm uses three matrices D, P and Q. In D all values are stored if there is no gap in the two sequences, P stores all costs if the word ends with a gap in sequence b and Q stores all costs if the word ends with a gap in a. The idea is simple: If you insert a new gap, the costs to this point are

stored in D, add the cost of a new gap and store the result in P or Q. To extend a gap, just add the gap extend costs of β to the value from P or Q and save it there.
The recursion of the algorithm is defined as follows:

$$\forall i, j : i, j > 1 \; and \; 1 \leq i \leq |a| \; and \; 1 \leq j \leq |b|$$

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

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

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

$$w(a, b) = \begin{cases} 0, & \text{if } a = b \\ 1, & a \neq b \end{cases}$$

Data: Sequence a and sequence b
Result: A pairwise alignment of a and b
$D_{0,0} = 0$;
$D_{0,j} = g(j)$;
$D_{i,0} = g(i)$;
$P_{0,j} = \infty$;
$P_{i,0} = not \; used$;
$Q_{0,j} = not \; used$;
$Q_{i,0} = \infty$;
for *i =1; i < m; i++* **do**
  for *j =1; j < n; j++* **do**
    $P_{i,j}$;
    $Q_{i,j}$;
    $D_{i,j}$;
  end
end

Algorithm 1: Gotoh algorithm

For the traceback we use three different stacks, $traceback^D$, $traceback^P$ and $traceback^Q$:

- $d \in traceback^D \mid d \in D = \{^D\nwarrow, ^P\bullet, ^Q\bullet\}$
- $p \in taceback^P \mid p \in P = \{^P\uparrow, ^D\uparrow\}$
- $q \in traceback^Q \mid q \in Q = \{^Q\leftarrow, ^D\leftarrow\}$

To compute the traceback, start in $D_{m,n}$ and push an element to one of the stacks if one of the following conditions is holding:

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

$for\ traceback^P :$
$P_{i,j} = D_{i-1,j} + g(1) \Leftrightarrow\ ^D\uparrow$
$P_{i,j} = P_{i-1,j} + \beta \Leftrightarrow\ ^P\uparrow$

$for\ traceback^Q :$
$Q_{i,j} = D_{i,j-1} + g(1) \Leftrightarrow\ ^D\leftarrow$
$Q_{i,j} = Q_{i,j-1} + \beta \Leftrightarrow\ ^Q\leftarrow$

To get the alignment start poping the elements from $traceback^D$. In general: if the symbol is a left arrow, a gap is included in a, if it is an up arrow, an gap is included in b. If the element is $^P\bullet$ or $^Q\bullet$ from $traceback^D$, switch the stack to $traceback^P$ or $traceback^Q$. If you are in $traceback^P$ and the poped element is $^D\uparrow$ switch the stack back to $traceback^D$, equivalent for $traceback^Q$. In all other cases just take the given character from the input sequence a and b.

2.2.2 Complexity

Gotoh's algorithm needs $O(3 * n^2) \in O(n^2)$ space and $O(3 * n^2 + n) \in O(n^2)$ runtime.

2.2.3 Example

Given are the two input sequences $a = AGC$ and $b = AC$ and the gap penalty $g(x) = 2 + 1k$. The three matrices are computed as in Figure 2.3.

D

	ϵ	A	C
ϵ	0	3	4
A	3	0	3
G	4	3	1
C	5	4	3

P

	ϵ	A	C
ϵ	-	inf	inf
A	-	6	7
G	-	3	6
C	-	4	4

Q

	ϵ	A	C
ϵ	-	-	-
A	inf	6	3
G	inf	7	6
C	inf	8	7

Figure 2.3: The computed matrices of the Gotoh algorithm.

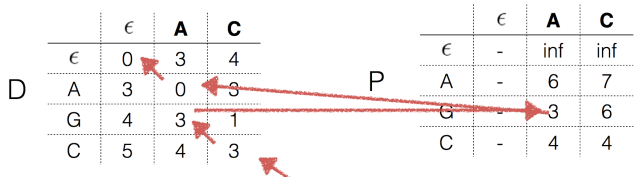

Figure 2.4: The traceback of the Gotoh algorithm.

The computed traceback is ${}^{D}\nwarrow\ {}^{P}\bullet\ {}^{D}\uparrow\ {}^{D}\nwarrow$, it is shown in Figure 2.4. The first element from $traceback^D$ is ${}^{D}\nwarrow$, so in the alignment in both sequences the character is notated, here it is for a and b an A. The next element is ${}^{D}\uparrow$, the character G from sequence a stays and in sequence b a gap is inserted. A pop from $traceback^D$ gives us ${}^{P}\bullet$. Now it is the case to switch to $traceback^P$ and a pop gives us ${}^{D}\nwarrow$. Add to the alignment the character at the actual position, which is $i = 3$ for sequence a and $j = 2$ for b. Again, we have to switch the list because we get a ${}^{D}\nwarrow$ in $traceback^P$. A pop in $traceback^D$ returns the information that the list is empty and the alignment is now finished. The computed alignment is: $\begin{pmatrix} AGC \\ A-C \end{pmatrix}$.

3 Multiple sequence algorithms

3.1 Needleman-Wunsch n = 3

The pairwise Needleman-Wunsch algorithm, seen in chapter 2.1, can be easily extended to a multiple sequence alignment algorithm. Instead of an two dimensional matrix an n-dimensional matrix is used and the recursion has to be extended to get the minimum of all possible neighbour cells of the n-dimensions. For example, the recursion for $n = 3$ would look like as follows:

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

The traceback is also an extension. Needleman-Wunsch with $n = 2$ used three arrows to note the route through the matrix, the extension with $n = 3$ have to use seven arrows to cover all possibilities of the recursion.

3.1.1 Runtime

Needleman-Wunsch with n=3 uses as a data structure a 3-dimensional matrix and for every cell seven computations are necessary. The space complexity is $O(n^3)$ and the runtime is $O(7 * n^3) \in O(n^3)$.
The problems of using Needleman-Wunsch for n-sequences are obvious: The space and runtime complexity is exponential with $O(n^n)$, which makes it impossible to use it on data beyond toy examples.

3.2 Sum of pairs

As it comes to multiple sequence alignments it is difficult to score alignments to compare different solutions. For example in PAM [5] the scoring for two alignments is defined as

$$S_c(x_0, x_1) = \frac{Pr[alignments\ are\ related]}{Pr[unrelated\ random\ sequnces]} = \prod_{i=1}^{n} \frac{p_{x_{0_i}, p_{x_{1_i}}}}{q_{x_{0_i}} q_{x_{1_i}}}$$

and for n-sequences it is extended to

$$S_c(x_0, ..., x_n) = \frac{Pr[alignments\ are\ related]}{Pr[unrelated\ random\ sequnces]} = \prod_{i=1}^{n} \frac{p_{x_{0_i}, ..., p_{x_{n_i}}}}{q_{x_{0_i}} * ... * q_{x_{n_i}}}.$$

The product of probabilities over a large set can get very low and in todays computers we only have a limited precision in floating point operations. Given this fact we use the monotonic logarithm function to compute S_c:

$$S_c(x_0, ..., x_n) = \sum_{i=1}^{n} \log(\frac{p_{x_{0_i}, ..., p_{x_{n_i}}}}{q_{x_{0_i}} * ... * q_{x_{n_i}}}).$$

Of course it is possible to compute these values but it is often the case that not enough data is available to compute the probabilities. A solution is the *sum of pairs* scoring system [6].

3.2.1 Procedure

The idea behind sum of scores is to compute the score for a multiple sequence alignment pairwise between every sequence of the alignment and sum up over theses scores:

$$S_c(x_0, ..., x_n) = \sum_{i=0}^{n} \sum_{j=i+1}^{n} S_c(x_i, x_j)$$

Sum of scores is a fast and good method to score multiple sequence alignments but unfortunately it can be incorrect. If we insert the above definitions the equation should be correct but it is not.

$$S_c(x_0, ..., x_n) = \sum_{i=1}^{n} \log(\frac{p_{x_{0_i}, ..., p_{x_{n_i}}}}{q_{x_{0_i}} * ... * q_{x_{n_i}}}) \neq \sum_{i=0}^{n} \sum_{j=i+1}^{n} \log \frac{p_{x_i}, p_{x_j}}{q_{x_i} q_{x_j}} = S_c(x_0, ..., x_n).$$

3.3 Feng-Doolittle algorithm

The Feng-Doolittle algorithm solves the problem of multiple sequence alignments more efficent than Needleman-Wunsch with $O(n^n)$. It was developed by Da-Fei Feng and Russell F. Doolittle in 1987 "to achieve the multiple alignment of a set of protein sequences and to construct an evolutionary tree depicting their realationship." [7]

3.3.1 Procedure

To do so, they first compute the pairwise alignments of all sequences and score them as follows:

$S(a, b) \Leftrightarrow$ similarity of a and b

$S_{a,b}^{max} = \frac{S(a,a)+S(b,b}{2} \Leftrightarrow$ maximal possible similarity

$S_{rand} \Leftrightarrow$ score for two random sequences with same length as a and b

$S_{a,b}^{eff} = \frac{S(a,b)-S_{rand}}{S_{a,b}^{max}-S_{rand}} \Leftrightarrow$ normalised percentage similarity

$D(a, b) = -\log S_{a,b}^{eff}$.

The scores are stored in a difference matrix and with this information a tree with UPGMA/WPGMA (see Chapter 4) is constructed. Based on the tree the multiple sequence alignment is gernerated:

- Align all nodes with the same parent together.
- Compute the optimal alignment between these sequences and replace all gaps with X

Take a look at the algorithm in Algorithm 2 for details.
A feature of Feng-Doolittle is that an inserted gap is never deleted. Feng and Doolittle introduced this because they "putting more trust in the comparison of recently diverged sequences than in those evolved in the distant past." [7]. This assumption gets problematic if two sequences choose a local optimum e.g. $AAGC$ and $XAGC$ but aligned with an third sequence $AGTT$ a gap in the second position is better. This problem, called *conservation of columns* can not be solved with Feng-Doolittle but there are solutions like *T-Coffee* [8].

Data: Set of sequences
Result: Multiple sequence alignment
Compute pairwise alignments of all sequences and save the score;
Build a tree with these scores;
Use tree to generate alignments;
while *Alignment is not done* **do**
  if *Two leaves* **then**
    Best possible pairwise alignment;
  else if *One leave and one node* **then**
    Best possible pairwise alignment between sequence and every sequence of the group;
  else if *Two nodes* **then**
    Best possible pairwise alignment between every sequence of group0 and every sequence of group1;
end

Algorithm 2: Feng-Doolittle algorithm

3.3.2 Complexity

Feng-Doolittle has a space complexity of $O(n^2)$ and a runtime complexity of $O(n^2)$.

3.3.3 Example

Given are the three sequences $a = ACTG$, $b = AT$ and $c = ACG$ and the scoring with 1 in case of a match and 0 otherwise. As you can see in Figure 3.1 the three sequences have a scoring from $s(A, B) = 0$, $s(A, C) = 2$ and $s(B, C) = -1$ and the clustering results in ((A, C),B) (Figure 3.1 right).

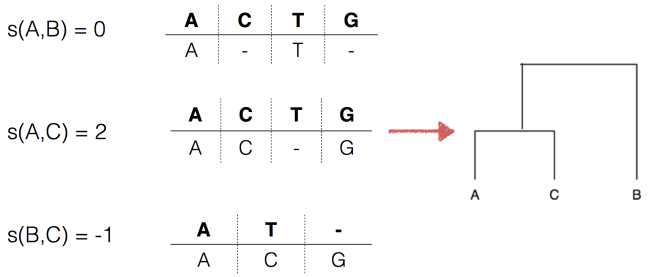

Figure 3.1: Scores of three sequences and clustering

In the next step of the algorithm every two nodes with the same parents are aligned (Figure 3.2). Here it is first sequence A and C and then the group of (AC) together with B. The alignment of (A, B) with a scoring of 0 is better than the alignment of (C', B) with -1 so this alignment is taken.

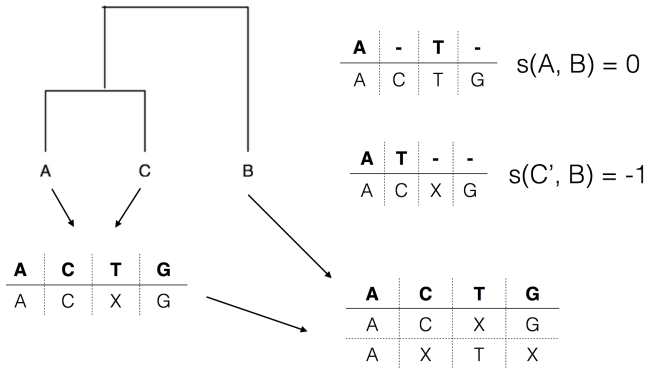

Figure 3.2: Steps of the multiple sequence alignment with Feng-Doolittle

4 Phylogeny/Clustering

Phylogeny is the development of all creatures in the context of evolution. It is an interesting point to know how this development was proceed in time. From a bioinformatics point of view phylogeny is nothing else than a clustering of sequences of DNA. As closer two sequences are under the measure of similarity as close they are clustered.

4.1 UPGMA / WPGMA

4.1.1 Procedure

UPGMA / WPGMA are clustering algorithms which are using a distance matrix as an input and return a binary tree. In every round of the algorithm two clusters are merged to a new cluster, this is represented as a new node in the tree. See Algorithm 3.
The differnce between UPGMA and WPGMA is just the way how the new distances of a new cluster are computed. In UPGMA it is the half of the sum of the distance from both old clusters to the other clusters, and in WPGMA the size of the joined groups is weighted by theire size:

UPGMA: $dist(w,z) = \frac{dist(w,x)+dist(w,y)}{2}$

WPGMA: $dist(w,z) = \frac{|x|*dist(w,x)+|y|*dist(w,y)}{|x|+|y|}$

Data: A matrix with the distances
Result: A tree which shows the distance between the data
Initialization: Every node is a single cluster;
while $i < n$ *-1* **do**
    search minimal distance in matrix d_{min} of two clusters x, y;
    merge clusters x and y to new cluster z;
    define a new node z in the tree where e has distance $d_{min}/2$ to all leaves reachable from x and y;
    compute the distances to all other clusters for this new cluster;
end

Algorithm 3: UPGM/WPGM algorithm

4.1.2 Complexity

UPGMA/WPGMA clustering has a space complexity of $O(n^2)$ and a runtime complexity of $O(n^2)$.

4.1.3 Example

Given as an input is the distance matrix seen in Figure 4.1 right. The shortest distance is between the cluster A and B, they are merged, the new distances to this cluster are computed and the tree with the two leafs A and B and the root node is created. As a weight the paths from the root node to the leaf is half the distanz of A and B. In the next step, Figure 4.2, the shortest distance is between (AB) and C. The tree is extended with the new leaf C and the root node is replaced and is getting a new parent. This node represents the cluster (ABC). The weights on the path from the root node to all leaves have to be equal. Finally (Figure 4.3) the cluster (ABC) is merged with D.

	A	B	C	D
A	0	1	2	3
B	1	0	2	3
C	2	2	0	3
D	3	3	3	0

	(A B)	C	D
(A B)	0	2	3
C	2	0	2
D	3	2	0

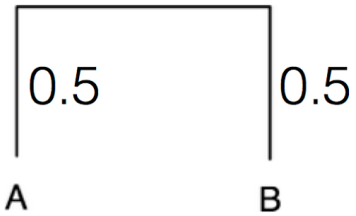

Figure 4.1: Node A and B are having the minimum distance and are clustered.

	(A B)	C	D
(A B)	0	2	3
C	2	0	2
D	3	2	0

	(A B C)	D
(A B C)	0	3
D	3	0

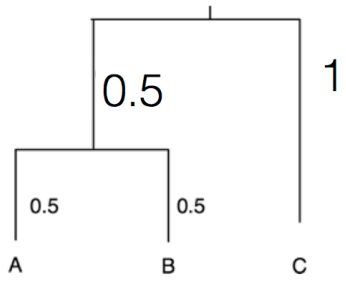

Figure 4.2: Node (A B) and C are having the minimum distance and are clustered.

	(A B C)	D
(A B C)	0	3
D	3	0

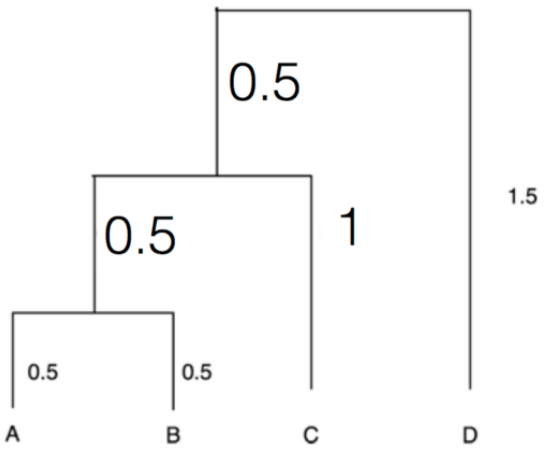

Figure 4.3: Node (A B C) and D are having the minimum distance and are clustered.

5 Basic structure prediction - RNA folding

5.1 Nussinov algorithm

The algorithm of Nussinov [9] [10] predicts the secondary structure of RNA. To do so, it computes which bases have to pair to get a maximum number of base pairs. The problem of secondary structure prediction is for crossing RNA $NP - hard$, because of this fact the Nussinov algorithm is restricted to non-crossing RNA structure output.

5.1.1 Procedure

Nussinov uses as an input a sequence of RNA bases. The used data structure as a $n+1 \times n+1$ matrix. Otherwise than usual, the first row (the "0" row) is empty. For initalization the diagonals $N_{i,i}$ and $N_{i,i-1}$ are filled with 0 and the computation of $N_{i,j}$ is not as usual row by row, the Nussinov algorithm computes it diagonal by diagonal:

$$N_{i,j} = max \begin{cases} N_{i,j-1} \\ \max\limits_{\substack{i \leq k < j \\ S_k S_j complementary}} N_{i,k-1} + N_{k+1,j-1} + 1 \end{cases}$$

The traceback starts in cell $N_{1,n}$ and every possible way to this cell that the recursion defines is checked. If it is a match for a cell $N_{i-1,j}$ than traceback this cell; if it is a match for two complementary bases, the index of the bases are noteted and the traceback splits into two paths, each to the cell which was involved in the recursion. If a cell is a $N_{i,i}$-cell the traceback for this path stops.

Data: Sequence of RNA bases
Result: All opimal base pairs
initalize:;
$N_{i,i} = 0$;
$N_{i,i-1} = 0$;
i =2;
while $i < m$ **do**
  k = i;
  j = 0;
  while $j \leq$ *n-2* **do**
    compute($N_{j,k}$);
    j++;
    k++;
  end
  i++;
end

Algorithm 4: Nussinov algorithm

5.1.2 Complexity

The algorithm of Nussinov has a space complexity of $O(n^2)$ and a runtime of $O(n^3)$.

5.1.3 Example

As an input sequence $AUGCA$ is used and the two diagonals, $N_{i,i}$ and $N_{i,i-1}$ are filled with 0 (Figure 5.1. In the next step cell $N_{1,2}$ is computed by searching the maximum of $N_{1,1} = 0$ and the maximum over all complementary bases $S_1 = A$ and $S_2 = U$. In this case the bases are complemenary and we only have one base pair. The maximum value is 1 (Figure 5.2). In Figure 5.3 cell $N_{1,4}$ is computed. Again, the maximum of $N_{1,3} = 1$ and the maximum of all base pairs $S_1 = A$, $S_2 = U$ and $S_3 = G$ complementary with $S_4 = C$ is searched. Obvious, S_1 and S_2 are not complementary, so the maximum is 2. In Figure 5.4 the traceback splits in $N_{1,4}$ and $N_{1,2}$. The complentary sequences of the maximum are S_3 and S_1. The output of the algorithm is that the bases ($S_1 = A$ and $S_2 = U$) and ($S_3 = G$ and $S_4 = C$) are paired.

	1	2	3	4	5		
	A	U	G	C	A		
0	0					A	1
	0	0				U	2
		0	0			G	3
			0	0		C	4
				0	0	A	5

Figure 5.1: Matrix after the initalization.

	1	2	3	4	5		
	A	U	G	C	A		
0	0					A	1
	0	0				U	2
		0	0			G	3
			0	0		C	4
				0	0	A	5

Figure 5.2: Computation of $N_{1,2}$.

	1	2	3	4	5		
	A	U	G	C	A		
0	0	1	1	2		A	1
	0	0	0	1		U	2
		0	0	1	1	G	3
			0	0	0	C	4
				0	0	A	5

Figure 5.3: Computation of $N_{1,4}$.

	1	2	3	4	5		
	A	U	G	C	A		
0	0	1	1	2	2	A	1
	0	0	0	1	1	U	2
		0	0	1	1	G	3
			0	0	0	C	4
				0	0	A	5

Figure 5.4: Traceback to get matching base pairs.

Bibliography

[1] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[2] R. Durbin, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[3] P. Clote and R. Backofen, *Computational molecular biology: an introduction.* Chichester [etc.]: Wiley, 2000.

[4] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of molecular biology*, vol. 162, no. 3, pp. 705–708, 1982.

[5] R. Backofen, "Stochastic models pam matrices." `http://www.bioinf.uni-freiburg.de//Lehre/Courses/2014_SS/V_Bioinformatik_1/pam-matrices.pdf`. [Online; accessed 23.11.2014].

[6] R. Backofen, "Multiple sequence alignment basics." `http://www.bioinf.uni-freiburg.de//Lehre/Courses/2014_SS/V_Bioinformatik_1/multiple-alignment.pdf`. [Online; accessed 23.11.2014].

[7] D.-F. Feng and R. F. Doolittle, "Progressive sequence alignment as a prerequisitetto correct phylogenetic trees," *Journal of molecular evolution*, vol. 25, no. 4, pp. 351–360, 1987.

[8] C. Notredame, D. G. Higgins, and J. Heringa, "T-coffee: A novel method for fast and accurate multiple sequence alignment," *Journal of molecular biology*, vol. 302, no. 1, pp. 205–217, 2000.

[9] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman, "Algorithms for loop matchings," *SIAM Journal on Applied mathematics*, vol. 35, no. 1, pp. 68–82, 1978.

[10] R. Backofen, "Rna structure and rna structure prediction." `http://www.bioinf.uni-freiburg.de//Lehre/Courses/2013_SS/V_RNA/slides/nussinov.pdf`. [Online; accessed 23.11.2014].