Notes on Bioinformatics

Riccardo Lo Iacono

January 19, 2026

Contents.

- 1 - Introduction to Bioinformatics.

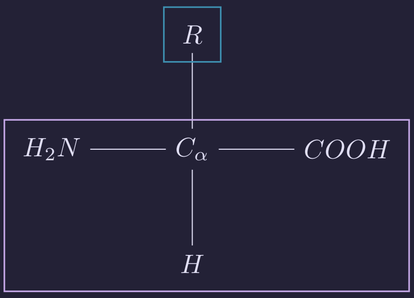
When we talk about bioinformatics, we refer to some sort of bridge between life sciences and informatics. The reason this kind of relationship is needed is due to the fact that biology, with Deoxyribonucleic Acid (DNA) sequencing, etc, is a source of big data; additionally, biology can benefit from informatics to ease some of its tasks, i.e., protein sequencing. In this notes we focus on various algorithms to menage both DNA and protein sequencing. Before doing so, in the remainder of this section we provide to the reader some notions of molecular biology; that is, we overview what aminoacids are, how these are related to proteins, the structure of DNA and RNAs and how these leadto proteins synthetisis.

- 1.1 - A brief introduction to molecular biology.

Human bodies, as well as any onther living thing, is constituted of several proteins which differ for their purpose. In this sense, we distinguish:

- **Structural proteins:** serves a building block for cells. An example of these is collagen.
- **Enzymes:** act as "cathalysts" for some chemical reaction. Some reactions are so slow that, in the case enzymes did not exist, life itself would not exist.
- **Transport proteins:** carry vital substunces through the body, i.e., hemoglobin.
- **Antibodies.**

Considering proteins in general, meaning without caring what's their task, these end up being a chain of smaller molecules: the *amino acids*. With reference the figure below, all amino acids share a common part (the one boxed in 🟪) and differ for the corresponding side chain (a.k.a the R-group, the one boxed in 🟦). Though one could argue the existence of



exceptions, we condider the classic 20 side chains, and consequentially the 20 classic amino acids (See Appendix A for the whole list).

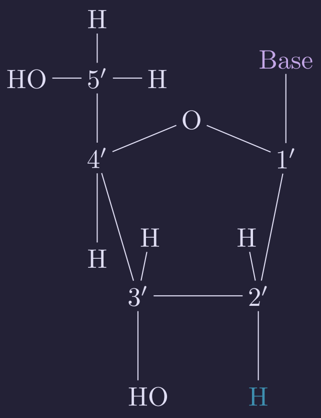
In truth proteins are not chains of amino acids, rather their residue: within a proteins, amino acids are bound to each other by petptide bonds, in which the COOH of an amino acid A_i binds with the H_2N of the amino acid A_{i+1}. This bond leads to the release of a water molecule, thus neither of the amino acids has its original molecular structure.

But how do we get proteins? To answer this question we need to understand what DNA and RNA are.

- 1.1.1 - From nucleic acids to proteins.

In nature there exist two types of nucleic acids: DNA and RNA. We present each one briefly.

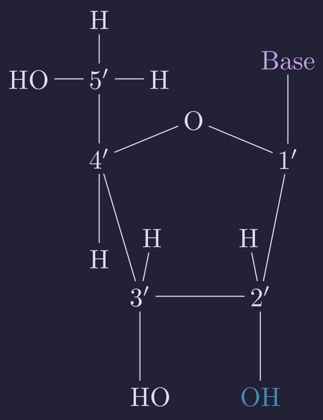
Starting with DNA, as for proteins, this is a chain of simpler molecules: the nucleotides. Structurally, it presents as a double strand chain in an helix. Chemically speaking (see the figure below), DNA it's a repetion of very similar units.



We distinguish four distinct bases: Adenine (A), Guanine (G), Cytosine (C) and Thymine (T). In DNA adenine bonds with thymine while guanine bonds with cytosine.

Each DNA strand, though bound in pair, follows an orientation: either from 5' to 3' or viceversa. In the following we use the former. This mechanism is what allows DNA replication. Essentially, since hydrogen bonds are not covalent (strong) they can be easily broken; in addition, since each nucleotide bonds only with a different one, from a single DNA strand we can get a copy of it. More precisely, given a DNA and a *primer*, that is a segment of DNA that triggers the replication, and the four nucleotides, we can synthesize a new complementary strand.

RNA is very similar to DNA (see figure below), and differs from it for the following reasons:



- Structure: RNA is single-stranded;
- Thymine is replaced with Uracile (U);
- We have different types of RNA.

RNA shows its importance in DNA duplication, as explained in the next section.

- 1.1.2 - Synthesis.

We have seen that proteins are residues of amino acids, thus, to identify each proteins we can consider the amino acids in its primiry structure. This is exactly the role of DNA. Precisely, each triplet of nucleotide (codone) encodes an amino acid. Let us observe that 4 nucleotides should produce 64 distinct amino acids, but as we have said, nature only provides us only 20. We can conclude that some codones identify the same acid. Additionally, some codones (UAG, UAA, UGA) do not encode any amino acid, rather they are used to signal the end of a gene.

The process that leads to proteins synthesis begins with a phase know as *transcription*. In this phase, a codone AUG identifies the begin of a gene which is copied onto an RNA molecule. We obtain this way the messanger RNA, or mRNA. Now, the process just described works only for procaryotes; for eucaryotes things are a bit more complex. Essentially, eucaryotes genes are composed of aternating parts: the *introns* and the *exons*. We care only of exons. To solve this issue, once we get the mRNA the parts corresponding to introns are removed.

Synthesis is done in a special cellular structure: the ribosome. Ribosomes are composed by proteins and a new type of RNA, the ribosomal

RNA. Here mRNA is read sequentially and a special type of RNA, the tRNA, carries the amino acid associated with the codone being read.

To summarize: within DNA we produce copies of genes, these are then transcibed via RNA and within ribosomes we synthesize the proteins.

What we just described assumes no error occurs during DNA replication. As easily understood this is rarely the case. In fact, if errors did not occure, evolution would impossible. More in general, when dealing with DNA sequences we have to consider mutations, i.e, nucleotide X becomes nucleotide Y, and/or gaps, some nucleotide could get added/removed from the original sequence.

- 2 - Pair-wise alignment.

One of the most important problems in biology is that of similarity; that is, given two strings encoding either DNA or proteins sequences, find the best alignment to make them look similar. We explain the concept of similarity in the next section.

There are essentially two approches to solve the problem, either via alignment based algorithms or via alignment-free algorithms. The former will be covered in the present section, in which we focus on pair-wise alignment, i.e., alignment of two strings; and in the one following, in which we discuss multiple alignment. We cover alignment-free algorithms in *Section 4*.

- 2.1 - Alignment and similarity measures.

Sequence alignment is the process trough which we search for patterns within the given sequences. The process can be either applied locally or globally, depending on the needs. We discuss the Needleman-Wunsh algorithm for the global alignment case and the Smith-Waterman algorithm for the local case, (see [3, 6] for major details).

Before we proceed further, let us consider the naïve approach; that is, we just take in consideration mismatches and matches.

Considering two sequences, essentially, we consider all the possible alignments and choose the one with the highest number of matches.

Example

Let $s_1 = ACCAC$ and $s_2 = ACGGCC$. Consider all possible alignments:

$$\begin{array}{l}ACCAC\\ ACGGCC\end{array} \qquad \begin{array}{l}ACCAC\\ ACGGCC\end{array}$$

$$\begin{array}{l}ACCAC\\ ACGGCC\end{array} \qquad \begin{array}{l}ACCAC\\ ACGGCC\end{array}$$

$$\begin{array}{l}ACCAC\\ ACGGCC\end{array} \qquad \begin{array}{l}ACCAC\\ ACGGCC\end{array}$$

$$\begin{array}{l}ACCAC\\ ACGGCC\end{array} \qquad \begin{array}{l}ACCAC\\ ACGGCC\end{array}$$

$$\begin{array}{l}ACCAC\\ ACGGCC\end{array} \qquad \begin{array}{l}ACCAC\\ ACGGCC\end{array}$$

It's easy to observe that the best one is the top-right one.

This approach has a severe issue: it does not take in account gaps, which, as we have said, are extremely frequent in nature. Therefore, though easy to inplement, we should consider something different.

The necessity to consider gaps lead to a question: how do we define "the best alignment" when gaps are involved? To solve the problem we define the so called *similarity measure*; i.e., a measure that takes in account both matches/mismatches and gaps. The one we use is the following

$$S_{AB} = \sum_{i=1}^{L} s(a_i, b_i) - \sum_{k=1}^{NG} \delta + \gamma[l(k) - 1]$$

where δ is said to be the *opening penalty*, $\gamma \leq \delta$ is called *extention penalty* and $l(k)$ is the length of the k-th gap.

Let us note that the concept of match/mismatch holds just for DNA and RNA sequences; in the context of amino acids, and therefore proteins, a different criterion is needed. Throuhgout this notes we use the *amino acid interconvertibility* criterion. In this case we make use of two special matrices (more correctly, substitution matrices) which tells us the value to assign to any given pair of amino acid. The ones we consider are PAM-1 and BLOSOM-62.

- 2.2 - Needleman-Wunsh algorithm.

Proposed in 1970, the Needleman-Wunsh algorithm allows one to compute the best global alignment in $\mathcal{O}(n^2)$ time (we assume the two sequences to have approximately the same length). The version we discuss is a slight variant of the algorithm proposed in [3].

Let s_1 and s_2 be the sequences to align, also let n, m be the respective lengths. The algorithm proceeds as follow:

1. Create an $n \times m$ matrix M: each entry represents the intersection of the i-th symbol of s_1 with the j-th symbol of s_2.
2. Set the inital value of each entry of the matrix accordingly to the type of sequences in analisis:
   - <u>nucleotides:</u> use the match/mismatch score; or
   - <u>amino acids:</u> use the value provided by the chosen substitution matrix.
3. Update the values of each entry as follow:

   $$M(i,j) = \max \begin{cases} M(i-1, j-1) + M(i,j), \\ M(i-1, j) - \delta, \\ M(i, j-1) - \delta. \end{cases}$$

   If any of the index is negative, that entry has value equal to zero.
4. Within the last row, or column, dind the cell with the maximum score/s; from it track the best path/s proceeding backwards.

- 2.3 - Smith-Waterman algorithm.

Proposed in [6], the Smith-Waterman algorithm allows one to find the best local alignment in $\mathcal{O}(n^2)$ time (we assume the two sequences to have approximately the same length).

Let s_1 and s_2 be the sequences to align, and let n, m be their lengths. We proceed as follow:

1. Create an $n \times m$ matrix: each entry represents the intersection of the i-th symbol of s_1 with the j-th symbol of s_2.

2. Set the inital value of each entry of the chosen substitution matrix; either PAM or BLOSUM.

3. Update the value of each entry using the following relation:

$$M(i,j) = \max \begin{cases} 0, \\ M(i-1,j-1) + M(i,j), \\ M(i-1,j) - \delta \text{ or } \gamma, \\ M(i,j-1) - \delta \text{ or } \gamma. \end{cases}$$

   If $M(k,l)$ does not exists, i.e., either k or l are negative, assume that entry to be zero.

4. Within the whole matrix, find the maximum[1]score/s and from it track back the best path/s.

In the above relation, the choice between δ and γ depends on the gap; i.e., if the gap just opened use δ, use γ otherwise.

See that if we limit the search of the maximum score to the last row and column, we get the Needleman-Wunsh algorithm discussed previously; that is, the Needleman-Wunsh algorithm we described is a particular case of the Smith-Waterman algorithm.

- 2.4 - Heuristic algorithms.

Consider the following: assume a new protein was just discovered; hence, we would like to know which other known protein shares same similarities with it. To complete such task, we have to run a similarity test with all the proteins stored in a biological database, (we discuss databases in some later section).

Observe that the use of SW or NW algorithms is not feasible. In fact, we would have to run a $\mathcal{O}(n^2)$ algorithm for each sequences in the database. Since these store thousands or hundred of thousands of

[1] Alternatively, one could set a threshold and consider the path starting at each entry above said threshold.

sequences, such algorithms are clearly not suitable for the task. In this context the use of heuristic algorithms is preferred. We recall that a heuristic algorithms do not garantee the correct/best solution, but they provide a good trade-off between efficiency and correctness.

In this section we focus on pair-wise alignment by means of heuristic algorithms; we discuss FASTA and BLAST.

- 2.4.1 - FASTA.

FASTA (or FAST-All, see [2] for mojor details) is a heuristic local alignment algorithm; that is, it finds the best possible alignment, in the heuristic sense, in a relatively short amount of time.

Assume we have a sequence s and we want to find the most similar sequence in a database[2]. We proceed by pre-computing an index of positions as follow: fix an integer k and compute all factors of s of lenght k; for each of these factors we store the possion where this occur in the database sequences. For instance, let $s = ACGGTYFF$ and let

$$w_1 = ACCGFFNFG$$
$$w_2 = DFCGFFNFG$$
$$w_3 = LACGFFNFG$$

be the database sequences. Fix $k = 2$, the factors of s then are: $AC, CG, GG, GY, TY, YF, FF$. Considering for example the factor AC, we have that it occurs at position 0 in w_1, at position 1 in w_3 and does not occur in w_2. We do the same for each of the factors.

Once the index is computed, we try to identify the diagonals, as done in the SW-algorithm; note that in this case the diagonals is given by the diffence of the positions in the factors of s and their occurence in each of the w_i. We then recalculate the score of those regions of the longest segments by means of a substitution matrix, e.g. BLOSUM-62. For each of the databases sequences, consider the ten regions with highest score and compute their sum *Init1*. Sort the sequences by this value. It could happen that some of the initial regions can be merged in the same alignment if gaps are added. If this happens, we try adding such gaps and then sum the score of these region minus some penalty for the gaps. Call such score *InitN*. Lastly, consider the sequences with the highest InitN, apply a variant of SW to them.

It's easy to see that FASTA is really fast, up to 10 times faster then SW; but some severe drowbacks are also evident: the heuristic nature of the algorithm makes it unsuitable for high precision alignments, also, the efficiency depends on the value of k, the lower this is, the faster is the algorithm with the caveat that in the first phase an indenty criterion is used.

- 2.4.2 - BLAST.

BLAST, (see [1] for all the details), is a tool to search the best local alignment of two strings.

As done for the FASTA algorithm, assume that s is our query sequence and w_1, w_2, w_3 are our database sequences. We begin by computing all factors of length k, as done for FASTA. For each factor, by means of a substitution matrix, create a list of similar factors up to some threshold value t. Then, within each w_i look for one of the factors in the list created. We then try to extend in both direction our alignment. To do so we consider a new parameter, call it x, that is used to tell the algorithm to persist in the extention even when a negative value is found.

Again, the use of this algorithm do not garantees the best possible alignment, but with a high probability the result is close enough.

[2] By database we refer to biological databases, which we describe in *Section:4.3*.

- 3 - Multiple alignment.

Multiple alignment is somewhat a generalization of the pair-wise alignment; in fact what it does is align three or more sequences and give the best possible alignment. Here the use of the word best is misleading, as we are about to explain.

Consider the following: can we use dynamic programming to compute the best alignment, as done in the case of two sequences? Assume we want to align three sequences. If we proceed as done for the Needleman-Wunsh algorithm, we must consider the n^3 matrix these form. Also, how shall we compute the scores?

The acute reader might have seen how the problem is isomorphic to the problem of computing the best path in an n-dimension Manhattan Cube.

Since the dimensions of the matrix increase with the number of sequences, a dynamic programming based approach should be avoided, as these require $\mathcal{O}\left(n^k\right)$ time, with k the number of sequences.

Remark. Despite the complexity, dynamic programming based algorithms exist, but are only used in the case of few sequences with a relatively short length.

Hence, unless extreme precision is needed, (e.g. medical analysis), heuristic algorithms suffice our needs. We use the remainder of this section to discuss a couple of such algorithms.

- 3.1 - The greedy approach.

As we saw, the use of dynamic programming is inconcivable for multiple alignment. In this section we focus on heuristic algorithms that compute the best alignment in by means of a greedy approach. Roughly, algorithms procced as follow: let $w_1, w_2, w_3, \ldots, w_n$ be strings to align. By some heuristical method choose $w_i, w_j, i \neq j$ such that these are the most similar; combine them to reduce the alignment from n sequences to $n-1$. For instance, let

$$\begin{aligned} w_1 &= ABBBCALLTA \\ w_2 &= GCGAVBGHAK \\ w_3 &= LBBBJALLTA \end{aligned}$$

be the sequences to align, then we can combine w_1, w_3 in the string $w_{1,3} = a/lBBBcjALLTA$, therefore reducing the alignment to the sequences $w_2, w_{1,3}$.

algorithms that use this approach are known as progressive alignment algorithms.

- 3.1.1 - ClustalW.

Introduced in [8] by Thompson et al., is a three step process that computes the best alignment. Let n be the number of sequences to align, the algorithm proceeds to compute all $\binom{n}{2}$ pair-wise alignment. From the latters, it defines a similarity matrix by some scoring function, (we discuss these in the next section). From the similarity matrix, by some tree-building procedure (eg. neighbor join, see *Section* **??**) construct a guide tree; us this to define the alignment.

- 3.1.2 - Scoring functions for multiple alignment.

Among all possible scoring functions, the ones we care about are:

1. the *sum of pairs*,
2. the *entropy*, and
3. the *multiple lsc*.

We briefly overview each one. Let v_i, v_j be two sequences in the multiple alignment, for each pair of i, j we compute $s^*(v_i, v_j)$; hence, the overall score $s(v_1, \ldots, v_n)$ is given by

$$s(v_1, \ldots, v_n) = \sum_{i \neq j} s^*(v_i, v_j)$$

One can also extend the computation of the score to the single column, and subsequentially sum these values.

In the case of entropy, we make use of Shannon's entropy (see [5] for all the details). Essentially we compute the probability of each nucleotide/acid in each column, then we compute the entropy of each column, lastly we proceed to sum all the computed entropies.

As for the multiple lsc, given k sequences we count as match a column in which all symbols coincide, that is, the same nucleotide/acid appear in each sequence in that column. The overall multiple lsc is the number of mathes. Note that, by the way it's defined, this score is useful in the case of very similar sequences.

- 4 - Molecular evolution.

In *Section 3.1.1* we introduced the idea of using trees to build the multiple alignment of a set of sequences. In the present section we begin with an introdution on molecular evolution, i.e., how genes mutate through time; then we describe the two kinds of trees used to produce the multiple alignment.

- 4.1 - Genes mutations.

As stated multiple times throughout these notes, a gene mutate as consequence of nucleotide substitutions, deletions, etc. We'd like to note that not all variations of a gene are significant, as it could just concern a really small portion of the species. On the other hand if a mutation iterests a large of the population, where by large, in practice, we consider anything greater than the 1%, it makes sense to study such a gene.

Now, the mutation itself may occur either due to *natural selection* or *neutral genetic drift*. The former refers to the capacity for reproduction of genetically distinc individual within the population; the latter on the other hand is a purely random process. On the practical side there are two distinc approaches to study/estimate substitution: the deterministic one, based purely on observations, and the stochastic one, which make use of time-dependent variables. Despite we don't describe any of them, since several exist in the literature, we consider stochastical approaches. In some cases we refer to the *molecular clock hypothesis*, that is the idea for which the number of substitutions depends on the time elapsed after their divergence.

- 4.2 - Evolutionary trees.

Multiple alignment is based on the phylogenetic trees. These are trese in which the nodes represent the taxonomic units, while the branches describe the relashionship between them. Each internal node has three branches: one that leads to the ancestor and two to the descendants.

These kind of trees can by distinguished by the meaning of the branches lengths: if a meaning is assigned to the length we talk about *cladograms*, otherwise we talk about *phylogram*. In turns we can make an additional distinction: rooted and unrooed trees, whose meaning is pretty obvious. The former give qualitative informations about the species, the latter quantitative ones.

We consider two methods to build evolutionary trees: UPGMA and Neighbor-Join. Alternatives to the one proposed do exist, but make use of charcter state-base optimizations, which is far from our interests.

- 4.2.1 - UPGMA.

Proposed in [7] by Sokal and Michener, is a phylogenetic tree building procedure based on a distance measure. It is, in it's essence, a clusting algorithm. In more details (see the steps below), given the distance matrix of the taxonomic units (TU), it clusters the two with the minimal distance in a new TU and its distance is recomputed. The process is repeated until one TU is left. That is, do

1. For all i, initialize $C_i = \{s_i\}$, where s_i is a TU.
2. Find the two clusters C_i, C_j with minimal distance $D_{i,j}$. Let n_i, n_j be the sizes of the two clusters.
3. Merge C_i, C_j into C_{ij}; create a node with C_i, C_j as children with a path length of ${}^{D_{i,j}}/_{2}$.
4. Update the distance matrix by the follwing rule:

   $$D_{ij,k} = \frac{n_i}{n_i + n_j} D_{i,k} + \frac{n_j}{n_i + n_j} D_{j,k} \forall k \neq i, j.$$

   Delete rows and columns i, j.
5. Repeat (2) - (4) until a single TU is left.

- 4.2.2 - Neighbor-Join.

Presented in [4] by Saitou and Nei, Neighbor-Join is procedure that build unrooted trees in a similar manner to UPGMA. The procedure is the following:

1. Let $C_i = \{s_i\}$, for all i,
2. Compute

   $$u_i = \sum_{k \neq i} \frac{D_{i,k}}{n - 2}$$

   where n is the number of TU, and $D_{i,k}$ is an entry of the distance matrix D.
3. Choose i, j such that $D_{i,j} - u_i - u_j$ is minimal; group C_i, C_j into C_{ij}.
4. Compute the length of the branches for each cluster as follow:

   $$l_i = \frac{1}{2} D_{i,j} + \frac{1}{2}(u_i - u_j)$$
   $$l_j = \frac{1}{2} D_{i,j} + \frac{1}{2}(u_j - u_i)$$

5. Update the distance matrix with the entries

$$D_{ij,k} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2} \forall k \neq i, j,$$

   and delete rows and columns i, j.

6. Repeat (2) - (5) until two TU remains; connect them by an edge equal to their distance.

- 4.3 - Biological databases.

So far it has been discussed on how we process DNA/RNA sequences; a question arise naturally: how do we store such sequences, along with all the indormations we derived? The obvious solution is by means of databases.

The not so trivial problem now is how to keep the data coherent across all these databases. We note that these are distributed all over the word since the amount of data to be stored doesn't allow a localized solution. As in the case of classic databases, the best approach is that of updating each database once new inforamtions are available. For this reason an agreement was made across all international institutes.

Focusing on the the structure of these databases, each usually focuses on a singular biological element (eg. only nucleotide, only amino acids, etc). Note that the informations stored in different databases is the same, the only diffence are in how these are stored (flat-files, fasta-format, etc).

For the sake of completeness, a nucleotide centerd databases is often refferd to as a primary database, as it contains the essential informations of a given sequence; whilest amino-acid centerd ones are refferd to as secondary databases.

- 5 - Hidden Markov Models.

Let us consider one of the most notorious problems in biology: that of CG-islands. The problem is the following: is well known that each neucleotide has probabily $^1/_4$ of occurring within a gene; hence, any dineucleotide should appear with a frequency of $^1/_{16}$. It turns out that this is not the case, in fact the CG dineucleotide is underrepresented.

The problem can be seen as an analogous to the *Fair Bet Casino* one. Therefore, to solve the problem we consider the so called *Hidden Markov Models (hmm)*.

Formally, a HMM is defined by the 4-tuple (Q, Σ, A, E) where Q is a set of states with some unknown probabily distribution, Σ is the output alphabet, $A \subseteq Q \times Q$ stores the probabily of going from state i to state $j, i, j \in Q$ and $E \subset Q \times E$ stores the probabily of emmitting a symbol σ while in state k.

- 5.1 - The decoding problem.

Given a sequence of observations x, i.e., a sequence of output symbols, is it possible to find a path π, that is, a sequence of hidden states, such that $\Pr(x|\pi)$ is maximized? It turns out that there exists an algorithm that does exactly so: the Viterbi algorithm.

Briefly, we compute $s_{l,i+1}$, which represents the probability of reaching state l and produce the prefix $x_1 \cdots x_{i+1}$, as

$$\max_{k \in Q} \{ s_{k,i} \cdot a_{kl} \cdot e_l(x_{i+1}) \}$$

where $s_{start,0} = 1$ and $s_{k,0} = 0, \forall k \neq start$.

Since the values obtained by the algorithm are very small, in most cases we consider the logarithm of such quantities.

As an alternative approach one can conpute the besst path as follow: consider the forward probability $f_{k,i}$ of producing $x_1 \cdots x_i$ reaching state k, defined by the recurrence

$$f_{k,i} = e_k(x_i) \sum_{l \in Q} f_{l,i-1} a_{lk}$$

and the backwards probability $b_{k,i}$ of emitting $x_{i+1} \cdots x_n$ starting from state k, in turns defined as

$$b_{k,i} = \sum_{l \in Q} e_l(x_{i+1}) b_{l,i+1} a_{kl}.$$

At any moment i we have that

$$\Pr(\pi_i = k, x) = \frac{\Pr(x, \pi_i = k)}{\Pr(x)} = \frac{f_{k,i} b_{k,i}}{\Pr(x)}.$$

References.

[1] Stephen F. Altschul et al. “Basic local alignment search tool”. In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. DOI: 10.1016/S0022-2836(05)80360-2.

[2] David J. Lipman and William R. Pearson. “Rapid and Sensitive Protein Similarity Searches”. In: *Science* 227.4693 (1985), pp. 1435–1441. DOI: 10.1126/science.2983426.

[3] Saul B. Needleman and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453. DOI: 10.1016/0022-2836(70)90057-4.

[4] N Saitou and M Nei. “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” In: *Molecular Biology and Evolution* 4.4 (July 1987), pp. 406–425. DOI: 10.1093/oxfordjournals.molbev.a040454.

[5] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. DOI: 10.1002/J.1538-7305.1948.TB01338.X.

[6] T.F. Smith and M.S. Waterman. “Identification of common molecular subsequences”. In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5.

[7] Robert R. Sokal and Charles Duncan Michener. “A statistical method for evaluating systematic relationships”. In: *University of Kansas science bulletin* 38 (1958), pp. 1409–1438.

[8] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”. In: *Nucleic Acids Research* 22.22 (Nov. 1994), pp. 4673–4680. DOI: 10.1093/nar/22.22.4673.

Acronyms.

A Adenine. 2

C Cytosine. 2

DNA Deoxyribonucleic Acid. 1

G Guanine. 2

hmm Hidden Markov Models. 15

T Thymine. 2

TU taxonomic units. 13

U Uracile. 3

Appendix A.

Amino Acid	3-letter	1-letter
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V