Hidden Markov Models for Genome Analysis

Leonardo Gori

May 2022

1 Introduction

Hidden Markov Models (HMMs) are statistical models capable of capturing hidden information from observable sequential symbols (e.g., a nucleotidic sequence). These models have many applications in sequence analysis, in particular, to predict exons and introns in genomic DNA, identify functional motifs (domains) in proteins (profile HMM), align two sequences (pair HMM). In the context of a hidden Markov model (HMM), the forward algorithm is used to calculate a **belief state**: the probability of a state at a certain time, given the history of evidence. The goal of the project is to implement the hidden Markov model algorithm. The history of the whole project is preserved on GitHub. The activity performed during the project is presented in section 2, while in section 3 describes the proposed algorithm. Finally, the actual implementation of the project is described in section 4.

2 Summary of the performed activities

Before the implementation of the program, an analysis of the problem has been performed. In particular, the attempt made was understanding the concepts and the components used for the correct execution of the algorithm.

In the very beginning, the concepts of pairwise alignment, Markov chains and hidden Markov models, the Viterbi algorithm and the forward algorithm have been gathered from chapters 2 and 3 of [DEKM98].

Once gained confidence with the theory behind these models and algorithms, as a second stage of the analysis, the attempt of understanding the Pairwise alignment using HMMs in section 4 of [DEKM98] was made with less difficulties.

Subsequently, following the extract by Benjamin ([Ben17]) and the precious information provided by [RBAA18], the actual implementation of the algorithm has been made.

Finally, the implementation of the algorithm has been made and computational performances improved through the introduction of parallel computation.

3 Algorithm's description

The Pair HMM forward algorithm is a dynamic programming algorithm that is based on the pair HMM theory for the computation of the overall alignment quality between a candidate haplotype and an input read sequences. For what concerns pair HMMs, they can be informally described by two main properties:

- a well set of states S and, for each of such states, its **transition probability** from one state to another (including itself);
- For each state in S, its **emission distribution** of all the values that are contained in a well defined set of values V.

For the purpose of sequence alignment, a special type of HMM, called **pair HMMs** can be defined by:

- a set of states $S = \{M, I, D\}$ and their transition probabilities;

- a set of nucleotide basis $N = \{A, C, G, T\}$, from which V is derived as the set containing all the nucleotide pairs $(a, b), \forall\ a, b \in N$ and, for each of them, its emission probability $\mathcal{P}_s(a, b)$ by each state $s \in S$.

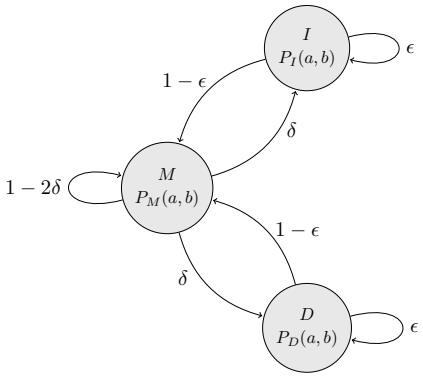

Figure 1: Naive representation of a Pair HMM, composed by its transmission states and emission probabilities

The hidden states are represented as M, D and I. When in state M, the pair-HMM emits symbols from both sequences, implying that the symbols may align to each other. When in state I, it emits one symbol from sequence X, and a blank symbol - meaning no symbol from sequence Y, indicating an insertion in sequence X. Similarly D state represents a deletion in sequence X (or an insertion in sequence Y)

The probability of the particular alignment is the product of the corresponding state transition probabilities. To find the overall alignment probability, we need to find the sum of the probabilities of all such alignments between the two sequences. However, if we follow a bruteforce approach to evaluate each possible sequence alignment, it can be computationally expensive as there can be a large number of possible alignments between two sequences.

In order to improve the performances, the **forward algorithm** is used to efficiently compute the overall alignment probability, through a dynamic programming approach. We define the **state transition matrix** T, containing the transition probabilities of switching from one state to another (i.e.: T_{MI} is the transition probability of switching from a matching state M to an insertion state I). Furthermore, we define $\mathcal{R}$ and $\mathcal{H}$ as the **read** and **haplotype** sequences, and we refer to $\mathcal{R}_i$ and $\mathcal{H}_j$ respectively as the $i-th$ and $j-th$ elements of the read and haplotype nucleotidic sequences. Finally, we represent states M, I and D through matrices of dimension $(|\mathcal{R}| \times |\mathcal{H}|)$, and we refer to elements $M_{i,j}$, $I_{i,j}$ and $D_{i,j}$ as the combined probability of all alignments up to pair $(\mathcal{R}_i, \mathcal{H}_j)$ that end respectively in state M, I and D.

Algorithm 1 Pseudo-code of the Pair HMM forward algorithm ([Ben17])

Initialize:

- $M_{i,0} = I_{i,0} = D_{i,0} = 0, \qquad \forall\ 0 \leq i \leq |\mathcal{R}|$
- $M_{0,j} = I_{0,j} = 0, \qquad \forall\ 0 \leq j \leq |\mathcal{H}|$
- $D_{0,j} = 1/n, \qquad \forall\ 0 \leq j \leq |\mathcal{H}|$

for $1 \leq i \leq |\mathcal{R}|$ **do**
  for $1 \leq j \leq |\mathcal{H}|$ **do**
    $M_{ij} = \mathcal{P}_M(\mathcal{R}_i, \mathcal{H}_j) \cdot (M_{i-1,j-1}T_{MM} + I_{i-1,j-1}T_{IM} + D_{i-1,j-1}T_{DM})$
    $I_{ij} = M_{i-1,j}T_{MI} + I_{i-1,j}T_{II}$
    $D_{ij} = M_{i,j-1}T_{MD} + D_{i,j-1}T_{DD}$
  end for
end for
Total likelihood $P(\mathcal{R}|\mathcal{H})$ is $\sum_j (M_{\mathcal{R},j} + I_{\mathcal{R},j})$.

Due to the dynamic programming's nature of the algorithm, the computation of an element of the matrices is dependent on the previous computation of its neighbours placed on its left, up and left-up positions along the matrices.

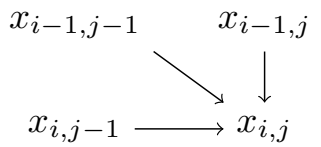

Figure 2: Graphical representation of dependencies that hold in the computation of an element of the matrices: element $x_{i,j}$ is dependent by its neighbours that are placed on the left, up and left-up positions.

The pseudo-code of the PHMM forward algorithm, reported above, performs a row-by-row computation of the elements of matrices M, I, and D.

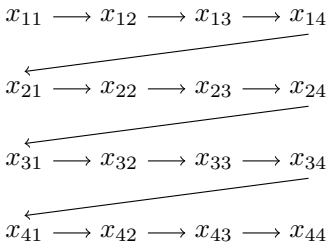

Figure 3: Graphical representation of the flow of computation of the PHMM forward algorithm reported by [Ben17]

For these reasons, the computation of each element of the matrix is dependent on the computation of the previous element in the path described in figure 3, making the execution of the algorithm less performant.

The execution can be improved by computing the elements of the matrices following the order described by their anti-diagonal paths (an intuition is proposed in figure 4). In fact, the computation of each element that is placed on the anti-diagonal line of the matrices is independent by the computation of the neighbours that lie on the same line, and is only dependent by the neighbours that lie on the previous anti-diagonal.

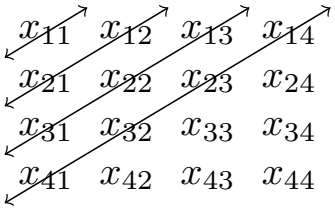

Figure 4: Graphical representation of the flow of computation of the proposed in the implementation. Here, with the double arrow notation, we want to highlight the fact that the order of computation of the elements that lie on the same line is independent one another, and therefore can be executed in a parallel fashion.

Therefore, assumed that the elements of the previous anti-diagonal line have already been computed, a **parallel computation** of all the elements that lie on the same current anti-diagonal can be performed for each loop in the algorithm. Furthermore, the same execution can be performed in an **out-of-order** fashion. This is the goal that has been chased in the project.

4 Implementation

The code is entirely written in C++ programming language, with the use of the following libraries and APIs (omitting the standard ones):

- **random**: used for the random generation of sequences and the random definition of state transition probabilities

- **algorithm**: used for the shuffling of sequences, used for randomization purposes

The parallel execution of the algorithm has been implemented by making use of the **OpenMP** APIs.

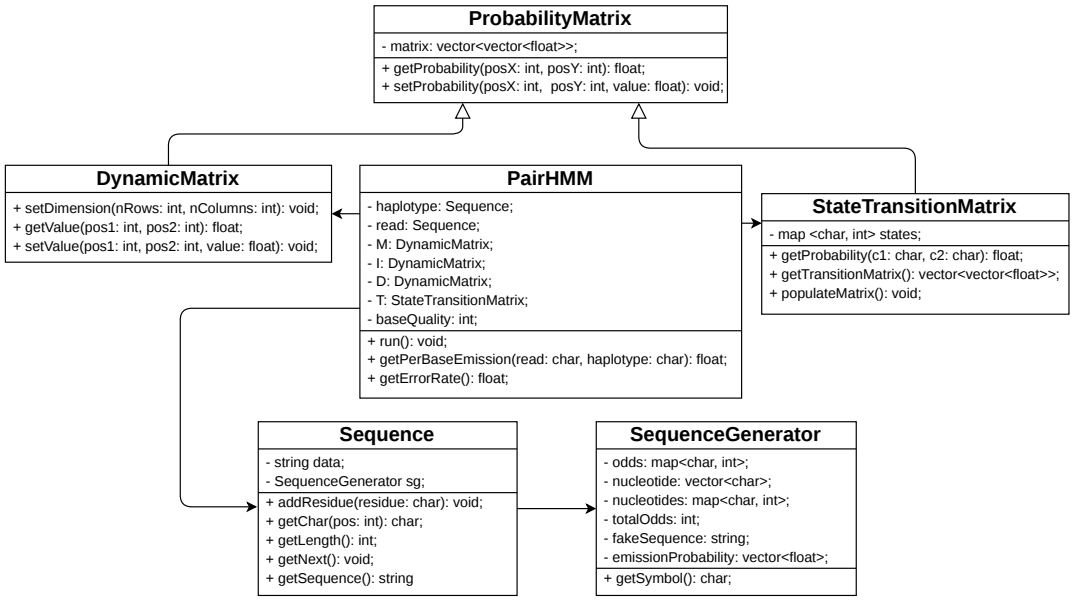

Figure 5: High level UML class diagram reporting the classes that have been defined for the implementation of the algorithm for the analysis of genomic sequences.

Further details about the description of the classes can be retrieved on the GitHub repository.

5 Conclusions

Actually, the implementation that is proposed on GitHub performs the parallel execution of the algorithm along the elements placed on the anti-diagonal line of the matrices. The computation has been improved w.r.t. the previous commit of the repository. In the last version, none of the loop cycles is used for checking the validity of the accessed element of the matrices, letting the execution to focus on the sole execution of the algorithm. Further work may be done as next step of the project; among the possible paths, it would be interesting comparing the execution's performances between the sequential execution of the algorithm described in [Ben17] and the implementation proposed in this project.

References

[Ben17] David Benjamin. Pair HMM probabilistic realignment in HaplotypeCaller and Mutect. `https://github.com/broadinstitute/gatk/blob/master/docs/pair_hmm.pdf`, August 2017.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[HMR+17] Sitao Huang, Gowthami Jayashri Manikandan, Anand Ramachandran, Kyle Rupnow, Wenmei W. Hwu, and Deming Chen. Hardware acceleration of the pair-hmm algorithm for dna variant calling. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 275–284, New York, NY, USA, 2017. Association for Computing Machinery.

[RBAA18] Shanshan Ren, Koen Bertels, and Zaid Al-Ars. Efficient acceleration of the pair-hmms forward algorithm for gatk haplotypecaller on graphics processing units. *Evolutionary Bioinformatics*, 14:1176934318760543, 2018. PMID: 29568218.