Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

Decoding of Hidden Markov Models with Applications to Sequence Alignment

(Dissertation Thesis)

Mgr. Michal Nánási

Advisor: Mgr. Bronislava Brejová, PhD.

Bratislava, 2014

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Decoding of Hidden Markov Models with Applications to Sequence Alignment

Dissertation Thesis

Study Program: Computer Science

Branch of Study: 9.2.1 Computer Science, Informatics

Supervisor: Mgr. Bronislava Brejová, PhD.

Bratislava, 2014

Mgr. Michal Nánási

85975711

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Mgr. Michal Nánási
Študijný program:	informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	dizertačná
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Decoding of Hidden Markov Models with Applications to Sequence Alignment
Dekódovanie skrytých Markovových modelov a jeho využitie na zarovnávanie sekvencií

Cieľ: Cieľom práce je analyzovať výpočtovú zložitosť niekoľkých optimalizačných kritérií pre inferenciu v skrytých Markovových modeloch a navrhnúť nový model pre zarovnávanie sekvencií s tandemovými opakovaniami.

Školiteľ:	doc. Mgr. Bronislava Brejová, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 18.10.2010

Dátum schválenia: 18.10.2010 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

....

študent školiteľ

85975711

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Michal Nánási
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Dissertation thesis
Language of Thesis:	English
Secondary language:	Slovak

Title:	Decoding of Hidden Markov Models with Applications to Sequence Alignment
Aim:	The goal of the thesis is to analyze computational complexity of several optimization criteria for inference in hidden Markov models and to propose a new model for aligning sequences with tandem repeats.

Tutor:	doc. Mgr. Bronislava Brejová, PhD.	
Department:	FMFI.KI - Department of Computer Science	
Head of department:	doc. RNDr. Daniel Olejár, PhD.	
Assigned:	18.10.2010	
Approved:	18.10.2010	prof. RNDr. Branislav Rovan, PhD. Guarantor of Study Programme

....

Student Tutor

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

. . . .

Acknowledgments

I am deeply grateful to my supervisor Mgr. Broňa Brejová PhD. for her invaluable help and guidance.

Abstrakt

Zaoberáme sa dvomi dôležitými bioinformatickými problémami: anotáciou sekvencií a zarovnávaním sekvencií. V práci sa sústredíme na využitie skrytých Markovových modelov (HMM), dobre známych generatívnych pravdepodobnostných modelov.

V prvej časti študujeme anotáciu sekvencií, konkrétne dvojstupňové dekódovacie algoritmy a výpočtové problémy, ktoré s nimi súvisia. Ukážeme, že dvojstupňové algoritmy môžu zlepšiť presnosť dekódovania a dokážeme, že tri problémy vhodné pre prvý stupeň výpočtu sú NP-ťažké: problém najpravdepodobnejšej množiny, problém najpravdepodobnejšej reštrikcie a problém najpravdepodobnejšej stopy.

Druhá časť sa zaoberá zarovnávaním sekvencií, ktoré obsahujú tandemové opakovania. Tandemové opakovania sú opakujúce sa časti genomických sekvencií, ktoré často spôsobujú chyby v zarovnaniach. Aby sme vyriešili tento problém, vyvinuli sme nový HMM, ktorý modeluje zarovnania obsahujúce tandemové opakovania a skombinovali sme ho s existujúcimi ako aj novými dekódovacími algoritmami. Náš prístup sme vyhodnotili experimentálne.

V oboch problémoch sme používali dekódovacie algoritmy na zlepšenie presnosti predikcií HMM. Dekódovacie algoritmy sú často podceňované a väčšina vývoja ide do vytvárania topológie HMM. Avšak správnym výberom dekódovacej metódy môžeme dosiahnuť významné zlepšenie predikcií.

KĽÚČOVÉ SLOVÁ: skryté Markovove modely, dekódovanie, anotácia, zarovnanie

Abstract

We study two important problems in computational biology: sequence annotation and sequence alignment. In the thesis we concentrate on the use of hidden Markov models (HMMs), well established generative probabilistic models.

In the first part, we study the sequence annotation problem, specifically the two-stage HMM decoding algorithms and the computational complexity of related problems. In particular, we demonstrate that two-stage algorithms can be used to increase the accuracy of decoding, and we prove the NP-hardness for three problems appropriate for the first stage: the most probable set problem, the most probable restriction problem and the most probable footprint problem.

The second part of the thesis focuses on alignment of sequences that contain tandem repeats. Tandem repeats are highly repetitive elements withing genomic sequences that cause biases in alignments. To address this issue, we introduce a new HMM that models alignments containing tandem repeats, combine it with existing and new decoding algorithms, and evaluate our approach experimentally.

In both problems, we use the decoding algorithms to improve the accuracy of HMM predictions. Decoding algorithms are often neglected, and most of the development is focused on the structure of an HMM. However, a proper selection of a decoding method can lead to significant improvements in the predictions.

Keywords: hidden Markov models, decoding, annotation, alignment

Contents

List of Tables

List of Figures

Chapter 1

Introduction

New sequencing technologies are producing more and more biological data, including genomic sequences of many species. Therefore it is important to develop tools for automated analysis of such data. In this thesis we focus on computational methods for sequence annotation and sequence alignment. In the sequence annotation problem, we want to label parts of the sequences according to their function, or meaning. We call such a labeling an *annotation*. For example, we can label each symbol of a genomic sequence base on whether it is part of a gene or not as in the following example (g is a label representing genes and n is a label for non-gene parts).

```
Sequence:     ACGGTGCGTTAGCTGCTCTGATGTCTTCGATCTAGCTAGT
Annotation:   nnnnnnnngggggggggggggggggggnnnnnnnnnnngg
```

The *sequence alignment* is a data structure that characterizes similarity or shared origin of two or more sequences. We insert gap symbols (dashes) so that corresponding parts of the sequence are in the same column as in the following example.

```
Sequence X:   CTGCTAGCTACGT--GTGT
Sequence Y:   ---------ACGTGGAT--
```

Both annotation and alignment are fundamental bioinformatics problems. The first stages of analysis of newly sequences genomes typically include aligning it with the genomes of related species (that is already sequenced), and searching for known structures (like genes) inside new genomes. Many subsequent methods for analysing genomes rely on sequence annotation and alignment. To avoid artefacts in the results of these downstream methods, there is need to develop algorithms for producing sequence annotation and alignment with as low error rate as possible. Tools for both sequence annotation and alignment are often based on hidden Markov models (HMMs) [22, 47, 15, 36, 43, 54, 56, 58, 49, 65, 42, 61, 48, 16, 50]. In this thesis we propose new techniques for use of HMMs in these domains and also give proofs of NP-hardness for several related problems.

We work with generative probabilistic models, hidden Markov Models (HMM) and their variants. In general, an HMM is a state machine that generates a sequence (string) along with a sequence of states (called state path). Since an HMM is a probabilistic model, it also defines the probability of sequences and state paths. The state path contains information about the structure of the generated sequence. In practice we are often given the generated sequence and the state path is hidden. The goal of the decoding algorithm is to reverse the generation process and obtain the original state path or at least its approximation.

When using HMMs for annotation of biological sequences, we construct the HMM so that the structure of the states corresponds to the biological features we are interested in. Each feature can be encoded in one or several states. Then we assume that the genomic sequence of interest was generated by our model and use a decoding algorithm to obtain a state path which is as close as possible to the true state path. The Viterbi algorithm [22] is traditionally used for decoding, but other optimization criteria can be used to obtain more accurate results [16, 31, 58, 67]. Note that accuracy measures may depend on the application domain. In Chapter 3 we study a special type of decoding algorithms: two-stage algorithms. In the first stage, the algorithm infers important aspects of the annotation and in the second stage it fills remaining details in a way consistent with the first-stage results. We show that two-stage algorithms can improve the accuracy of decoding (as far we know, such algorithms were previously used only for reducing the running time). We also study the computational complexity of several decoding criteria appropriate for the first stage and we show NP-hardness results for obtaining the optimal annotations using these criteria. Namely we study the most probable footprint problem, the most probable set problem and the most probable restriction problem.

In sequence alignment, the goal is to search for corresponding parts of the sequences and arrange them into same position in the alignment. To choose the biologically correct alignment, we usually optimize some scoring scheme. We will consider scoring schemes which are defined using pair hidden Markov models (pHMM). A pHMM generates pairs of sequences along with their alignment (an alignment is defined by the state path). This model is an extension of HMM.

In the fourth chapter we propose a tractable method for aligning sequences with tandem repeats. A tandem repeat consists of consecutive copies (not exact) of a certain motif (short genomic sequence). Tandem repeats cause problems with sequence alignments because it is hard to distinguish between individual copies of the motif. We extend a traditional pHMM model for sequence alignment by additional states modeling tandem repeats. We also propose new decoding algorithms tailored to this model. We show on simulated data that our new model and decoding methods decrease the error rate, and with a particular increase of accuracy near the border of tandem repeats.

1.1 Biological Background

In this section we review several biological terms that will be needed. More information about DNA, proteins, and genes can be found in [12, 71]. Practically every cell of living organisms contains one or several *DNA* molecules. DNA is a double stranded molecule consisting of two long sequences (strands) of *nucleotides* (nucleotides are also referred to as bases or *residues*). There are four types of nucleotides in DNA: adenosine, cytosine, guanine, and thymine represented by letters A, C, G and T respectively. In RNA nucleotide T is replaced with uracil, denoted by U. The nucleotides at the same position in the two strands are connected by hydrogen bonds and are complementary: A is always connected with T and C is always connected with G. Therefore we can represent DNA molecule by a sequence over alphabet $\{A, C, G, T\}$, since the complementary strand can be easily computed.

Some parts of DNA encode *proteins*. Proteins play an important role in cell biology since they regulate many processes in the cell and are catalysts to many chemical reactions. Proteins are sequences of *amino acid* molecules. There are 20 amino acids that can be encoded in DNA. Parts of DNA that encode proteins are called *genes* (gene is “substring” of DNA that will be translated into one protein). We refer to the DNA sequences of an organism as to its *genome*.

The rational behind sequence alignment is to represent the evolution of two sequences. According to the evolution theory, currently living organisms evolved from a single common ancestor through small changes in their genomes. There are many types of changes. Substitutions change a nucleotide at some position to another nucleotide. Insertions and deletions add or remove parts of the genomic sequence. Other changes includes duplications (a region of a sequence is copied into a different part of the genome), inversions (a region of the sequence is inverted), and even large genome rearrangements (large regions change their position within the genome).

The speciation is an event, when a new species is created. This happens mostly due to physical separation of populations of the same species. Subsequently, each population evolves differently to a point, that they form different species. We can represent the evolution of a set of species using a binary tree. Each leaf represents a current species, each internal vertex represents a speciation event, and the root represents the common ancestor of all species in the tree. The branch lengths usually correspond to the amount of changes in the genome or to the time.

We say that two parts of biological sequences X and Y are *homologous*, if they originate from the same sequence in some common ancestor of X and Y. Due to their common origin, homologous sequences are often similar. However high evolutionary distance may cause the sequences to diverge so much that the sequence similarity is as low as for two random sequences.

As we mentioned in the previous section, an alignment is a data structure that represents sequences and their relation through some evolutionary changes (substitutions, insertions and deletions). It is obtained by adding gap symbols ($-$) into the sequences so that they have the same length and form a 2-dimensional array or a matrix, where each sequence is in its own row. When aligning two homologous sequences, we want to place homologous parts of the sequences into the same columns.

Additionally, we distinguish between two types of homologs: orthologs and paralogs. The orthologs are two different sequences that originated from the common ancestor sequence by a speciation event. The paralogs are two different sequences that evolved by duplication event within the same organism. The distinction is important when aligning repetitive regions; we want to align orthologous copies of the motif, but the paralogous copies are very similar, and it is very hard to distinguish between paralogous and orthologous copies.

1.2 Notation

In this section we summarize notation used in the following chapters.

All sequences, members of sets, vectors, and rows and columns of matrices will be indexed from 0. We will use mostly right-open intervals: $I = [a, b)$ means that $a \in I$, but $b \notin I$.

The element at the k-th position (zero based) of string (or sequence) s will be written as $s[k]$. The substring $s[i]s[i+1]\dots s[j-1]$ is denoted $s[i:j]$. If n is the length of the string s then $s[: i]$ is equivalent to $s[0 : i]$ and $s[i :]$ is equivalent to $s[i : n]$. We will use the terms sequence and string interchangeably.

Let M be a matrix. Then $M[i, j]$ is the element from the i-th row and j-th column of M (indices are zero based). Similarly as for strings, submatrix $M[i : j, k : l]$ is a matrix consisting from the intersection of rows $i, i + 1, \ldots, j - 1$ with columns $k, k + 1, \ldots, l - 1$. If M is of size $n \times m$ then $M[: i, j :]$ is equivalent to $M[0 : i, j : m]$. The term $M[i, :]$ is equivalent to $M[i, 0 : m]$ which is the i-th row of M.

Chapter 2

Hidden Markov Models and Their Decoding

This chapter contains a survey of the relevant literature and overview of methods and models that we will use in this thesis. We describe generative probabilistic models called hidden Markov models (HMMs), their variants, and algorithms that are used with these models. We describe the problem of sequence alignment and several applications of hidden Markov models to sequence alignment and sequence annotation. In this thesis, we study hidden Markov models from two aspects: computational complexity of some HMM problems, and application of HMMs to sequence alignment.

2.1 Hidden Markov Models

Hidden Markov models (HMM) are graphical probabilistic models commonly used in bioinformatics for sequence annotation or sequence alignment. An HMM is a probabilistic finite state machine that in every state emits one symbol. Later we will discuss variants of HMMs that emit more than one symbol, or that emit symbols on multiple tapes. This section describes basic definitions and algorithms that are used with HMMs.

2.1.1 Definitions

HMMs are generative probabilistic models. The generative process of an HMM starts in a random state q sampled according to the *initial distribution* I. When an HMM is in some state q it emits one symbol from alphabet Σ according distribution e_q and moves to another state according to transition distribution a_q. Note that emission and transition distributions can be different for every state. This process produces two sequences: sequence of states $\pi = \pi_0\pi_1\pi_2\dots$ called *state path* and output sequence $X = X_0X_1X_2\dots$ over alphabet Σ. In this work, we will use only discrete versions of HMMs with finite state space and alphabet.

Definition 1. *Any square matrix M of size $n \times n$ is* stochastic *if it satisfies the following properties.*

1. $\forall 0 \leq i < n, 0 \leq j < m, 0 \leq M[i,j] \leq 1$
2. $\forall 0 \leq i < n, \sum_{i=0}^{m-1} M[i,j] = 1$

Note. Stochastic matrix consists from n probability distributions over a set of size n.

Definition 2. *A Hidden Markov Model (HMM) is a tuple $H = (\Sigma, V, I, e, a)$ where $\Sigma = \{\sigma_0, \dots \sigma_{m-1}\}$ is a finite alphabet of size m, $V = \{v_0, \dots, v_{k-1}\}$ is a finite set of state of size k, I is a distribution over V, e is $k \times m$ matrix where each row contains distribution over Σ and a is stochastic matrix of size $m \times m$.*

We will index elements of e and a by subscripts; therefore $e_{u,v}$ is the element in u-th row and v-th column of e.

Example 1. *Consider the HMM $H = (\Sigma, V, I, e, a)$ from Figure 2.1. Alphabet is $\Sigma = \{A, C, G, T\}$ and set of states is $V = \{R, 0, 1\}$. Transition and emission distributions are described in the figure. We can define initial distribution to $I_R = 0.5, I_1 = 0.3$ and $I_0 = 0.2$.*

Definition 3. *Let $H = (\Sigma, V, I, e, a)$ be an HMM. We say that there is* transition *from state u to state v if $a_{u,v} > 0$. We will write transition from u to v as $u \to v$ and $T = \{u \to v \mid a_{u,v} > 0\}$ is the set of all transitions of H.*

State path *$\pi = \pi_0\pi_1\dots\pi_{n-1}$ is a sequence of states. We say that a state path π is* admissible *if $I_{\pi_0} > 0$ and $\pi_{i-1} \to \pi_i \in T$ for all $1 \leq i < n$. Otherwise π is* inadmissible.

Example 2. *Consider the HMM from figure 2.1. Set of transitions is $T = \{R \to R, R \to In, R \to E, In \to In, In \to E, E \to E, E \to In, E \to R\}$. State path $\pi_1 = RRRRInER$ is an admissible state path and $\pi_2 = RRRInREER$ is an inadmissible state path, because it contains a transition with zero probability ($In \to R$).*

Hidden Markov models are generative probabilistic models, meaning that they describe a simple process that can generate a pair of state path and sequence. In following definition we will define probability distribution of pairs (state path, sequence) and probability distribution of sequences generated by an HMM.

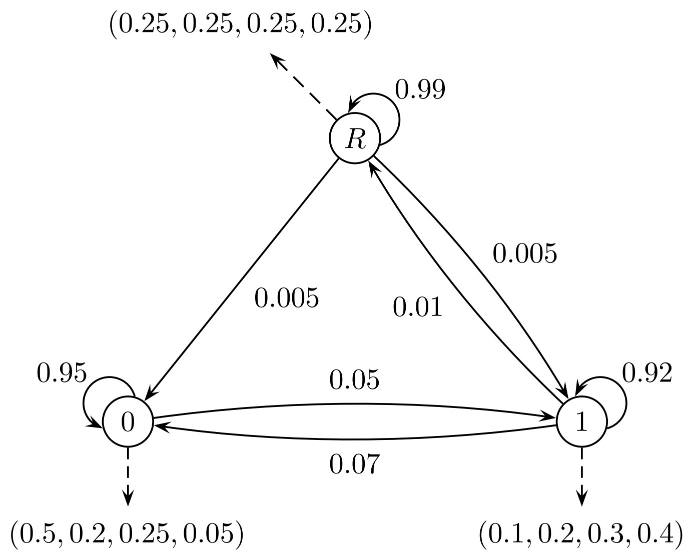


Figure 2.1: An HMM with 3 states that emits symbols from alphabet of size 4. Circles represents states; an arc from state u to v indicates that $a_{u,v} > 0$. The missing arc from state In to R means that $a_{In,R} = 0$. Every four tuple represents the emission distribution of the associated state.

Definition 4. *Let $H = (\Sigma, V, I, e, a)$ be an HMM and $X = X_0X_1 \dots X_{n-1}$ be a sequence over alphabet Σ of length n. Let π be a state path of length n. Then the probability that state path π generated sequence X is*

$$\Pr(X, \pi \mid H) = I_{\pi_0} e_{\pi_0, X_0} \prod_{i=1}^{|X|-1} a_{\pi_{i-1}, \pi_i} e_{\pi_i, X_i}$$

The probability that X was generated by the model H (using any state path) is

$$\Pr(X \mid H) = \sum_{\pi \in V^n} \Pr(X, \pi \mid H)$$

Example 3. *Consider the HMM H from the example 1. Let state path be $\pi = RRInInE$ and generated sequence be $X = ACGTT$. Then the probability that H generates π and X is* $\Pr(X, \pi \mid H) = 0.5 \cdot 0.35 \cdot 0.99 \cdot 0.25 \cdot 0.005 \cdot 0.25 \cdot 0.95 \cdot 0.05 \cdot 0.05 \cdot 0.4 = 0.5143359375 \cdot 10^{-7}$

Note. In our definition of an HMM, the sum of the probabilities of all sequences of length n is 1. Later we will discuss concept of final states. With final states, the sum of the probabilities of all sequences (of all lengths) is one.

We will be interested in the three basic problems in HMMs:

1. Given sequence X and model H. What is the probability that X was generated by model H?
2. Given sequence X, model H and assumption that X was generated by the model H, what is the best explanation of X? By explanation is usually meant state path that generated X. We call the process of computing explanation of sequence X *decoding*.
3. Given training data D (usually sequences with "explanations") and topology of the model (set of states and transitions), what are the best parameters (initial, transition and emission distributions) that explains training data D? This problem is also called *training*

In following sections we will discuss several algorithms for the problems describes above. We will mostly focus on the decoding problem.

2.1.2 The Forward Algorithm

The Forward algorithm computes probability $\Pr(X \mid H)$, probability that a given sequence X of length n was generated by the model (it solves the first problem of HMMs) [22]. The algorithm is based on the dynamic programming. It fills matrix F of size $n \times m$ where m is the number of states of H, $F[i,v]$ is the probability that H generated $X[: i+1]$ with state path that ends in state v. Values $F[i,v]$ are called *forward variables*. $F[i,v]$ can be computed by the following equations (also called the forward equations).

$$F[0,v] = I_v e_{v,X_0}, v \in V \tag{2.1}$$
$$F[i,v] = \sum_{u\in V} F[i-1,u] \cdot a_{u,v} \cdot e_{v,X_i}, v \in V, 0 < i < n \tag{2.2}$$

The probability that H generated X is

$$\Pr(X \mid H) = \sum_{v\in V} F[n-1,v]$$

Using the recurrence equations above, we can compute the probability of X in $O(nm^2)$ time and $O(m)$ memory. If the transition matrix is sparse, then this algorithm can be implemented in $O(n(m+t))$ time where t is the number of transitions. Forward variables are also used in other algorithms.

2.1.3 The Viterbi Algorithm

The Viterbi algorithm is probably the most frequently used decoding algorithm for hidden Markov models [22]. The Viterbi algorithm answers a straightforward question: given the sequence $X = X_0X_1 \ldots X_{n-1}$, what is the most-likely state path π that generates X? Formally, Viterbi algorithm finds a state path maximizing $\Pr(\pi \mid X, H)$. Since

$$\Pr(\pi \mid X, H) = \frac{\Pr(\pi, X, \mid H)}{\Pr(X \mid H)}$$

and quantity $\Pr(X \mid H)$ is fixed, therefore the most probable state path π also maximizes $\Pr(X, \pi \mid H)$.

The Viterbi algorithm is very similar to the Forward algorithm. It starts with computing Viterbi variables $V[i, v]$. Variable $V[i, v]$ stores the probability of the most probable state path that generated $X[: i + 1]$ and ends in state v. The algorithm also computes back-links $B[i, v]$ that contain the previous state in the most probable state path that generated $X[: i + 1]$ and ends in state v. We can compute these values by the following equations (called the Viterbi equations):

$$V[0, v] = I_v e_{v,X_0}, v \in V \tag{2.3}$$

$$V[i, v] = \max_{u \in V} V[i-1, u] a_{u,v} e_{v,X_i}, v \in V, 0 < i < n \tag{2.4}$$

$$B[i, v] = \arg\max_{u \in V} V[i-1, u] a_{u,v} e_{v,X_i}, v \in V, 0 < i < n \tag{2.5}$$

Note. Values $B[0, v], v \in V$ are not needed in the algorithm.

Note the similarity of the Viterbi algorithm and the Forward algorithm. We can obtain the Viterbi equations from the Forward equations by replacing summation with maximization.

Variable $V[n-1, v]$ contains the probability of the most probable state path that generated X and ends in state v. Therefore the state $v_{\max} = \arg\max_{v \in V} V[n-1, v]$ is the last state of the most probable state path. Variable $B[n-1, v_{\max}]$ contains the previous state of the most probable state path. By traversing back through back-links B we can reconstruct the most probable state path in $O(n)$ time.

Time complexity of the Viterbi algorithm is $O(nm^2)$ or $O(n(m+t))$ for sparse transition matrices (m is the number of states and t is the number of transitions).

2.1.4 The Forward-Backward Algorithm and the Posterior Decoding

The *Posterior decoding* is another commonly used decoding method [38, 22]. In contrast to the Viterbi algorithm, the Posterior decoding assigns a label individually to every symbol of an input sequence and does not care about the overall structure of the reconstructed state path.

Given sequence X, posterior decoding finds state path π with the following property:

$$\forall 0 \leq i < n, \pi_i = \arg\max_{v \in V} \Pr\left(\pi_i = v \mid X, H\right)$$

where

$$\Pr\left(\pi_i = v \mid X, H\right) = \sum_{\pi \in V^n, \pi_i = v} \Pr\left(\pi \mid X, H\right)$$

Values $\Pr\left(\pi_i = v \mid X, H\right)$ for every combination of position i and state v can be computed using the Forward-Backward algorithm. In particular,

$$\Pr\left(\pi_i = v, X \mid H\right) = F[i,v]B[i,v] \tag{2.6}$$

In this formula, value $B[i,v]$ is defined as

$$B[i,v] = \sum_{\pi \in V^{n-i}, \pi_0 = v} \prod_{j=1}^{n-i-1} e_{\pi_j, X_{i+j}} a_{\pi_{j-1}, \pi_j} \tag{2.7}$$

Note that the $B[i,v]$ is the backward version of the Forward equations. We can compute these values using the Backward algorithm, which is very similar to the Forward algorithm.

$$B[i,v] = \begin{cases} 1 & \text{if } i = n-1 \\ \sum_{u \in V} e_{\pi_j, X_{i+1}} a_{v,u} B[i+1,u] & \text{otherwise} \end{cases} \tag{2.8}$$

The cell $B[i,v]$ depends on the next positions in sequence X, while $F[i,v]$ depends on the previous positions in sequence X. Another difference is that $B[i,v]$ does include emission of $X[i]$ while $F[i,v]$ does.

We can compute values of $F[i,v]$ and $B[i,v]$ and find the state path in $O(n(m+t))$ time and $O(nm)$ memory. Advantage of the posterior decoding is that it uses information from many slightly suboptimal state paths as opposed to the Viterbi decoding, which uses only one state path. One drawback is that it can reconstruct inadmissible state path. Example is given below.

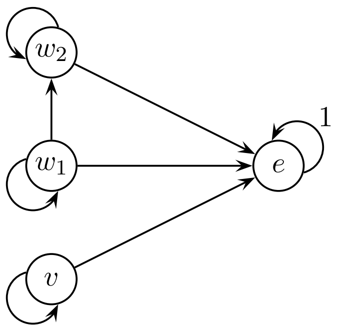


Figure 2.2: Example of an HMM on which the Posterior decoding reconstructs inadmissible state path. All unlabeled transitions are even (0.5). States w_1, w_2, and v emits 0 with probability 1 and state e emits 1 with probability 1. Initial distribution is set to $I_{w_1} = I_{w_2} = 0.3, I_v = 0.4, I_e = 0$.

State/Position	X_0	X_1	X_2
w_1	0.3	0	0
w_2	0.3	0.6	0
v	0.4	0.4	0
e	0	0	1

Table 2.1: Posterior probabilities for the sequence X and the HMM H from the figure 2.2.

Example 4. *Consider an HMM from figure 2.2. Let* $X = 001$*. There are three state paths that have non-zero probability:* $\pi_1 = w_1 w_2 e, \pi_2 = w_2 w_1 e$*, and* $\pi_3 = vve$*. Their probabilities are* $\Pr(\pi_1 \mid X, H) = \Pr(\pi_2 \mid X, H) = 0.3$*, and* $\Pr(\pi_3 \mid X, H) = 0.4$ *respectively. Posterior probabilities are in table 2.1. As we can see, the maximal posterior probability for the first position has state* v*. For the second position it is state* w_2 *and for the third it is state* e*. Therefore PD will reconstruct state path* vw_2e*. However, this state path is inadmissible since* $v \to w_2$ *is not a transition (it has zero probability).*

2.1.5 Sequence Annotation

HMMs can be used for sequence annotation. By sequence annotation we mean assigning "labels" to parts of the input sequence according to their meaning. So far we have described two algorithms that can be used for sequence annotation: the Viterbi algorithm and the Posterior decoding. In both algorithms, the sequence annotation is the output from the algorithm: the decoded state path. Now we give several examples of bioinformatics problems where HMMs were previously used.

Gene finding: Parts of DNA sequences that are in cells translated into proteins are called genes. In gene finding, we want to assign to every symbol of the input sequence a gene label if it is inside a gene or a different label if it is not part of a gene [9, 15, 17, 47]. Additionally, we want to label sequence according to the internal structure of a gene: gene starts with promoter followed by transcription start site and ends with transcription stop site. Between transcription sites are alternating exons and introns [9, 15, 17, 47] (exons encodes proteins, introns are cut out before translation of the exons to protein).

Transmembrane proteins: Some proteins in cells pass through a membrane from one side to the another (usually they pass through the membrane several times). In transmembrane protein prediction, we want to assign labels to the symbols of the input protein sequence to distinguish the parts of the sequence, that are on one side of the membrane from the parts that are on the other side of membrane or inside the membrane [16].

Recombination prediction: Some organisms, for example HIV virus, evolve rapidly and have been classified into several subtypes. Moreover, some viruses are mosaic combination of viruses from different subtypes. For example, beginning and end of the sequence of a virus can be from one subtype and the middle of the sequence is from a different subtype. In recombination prediction, we want to annotate the sequence to distinguish between parts that originate in different subtypes [58, 67].

In general, we have a finite set of labels $C = \{c_0, c_1, \dots, c_{l-1}\}$ and we want to assign one label to every symbol of the input sequence. We do it by assigning one label to every state of an HMM. To predict the annotation of the input sequence X, we can find the most probable state path that could generate X and annotate each symbol of the sequence X according to the label assigned to the state, that generated that symbol. We can formalize it in the following definition.

Definition 5. *Let $H = (\Sigma, V, I, e, a)$ be HMM and $C = \{c_0, c_1, \dots, c_{l-1}\}$ be the finite sets of labels (or colors). Then the* coloring function $\lambda : V^* \to C^*$ *is function that satisfies following properties:*

1. *$\lambda(v) \in C$ for all $v \in V$.*
2. *$\lambda(xy) = \lambda(x)\lambda(y)$ for all $x, y \in V^*$.*

Let X be a sequence generated by state path π. Then annotation Λ of sequence X is $\Lambda = \lambda(\pi)$.

In the sequence annotation problem, we do not know the state path π that generated a given sequence. Our goal is to reconstruct the state path π or at least to give a good approximation of the correct annotation $\Lambda(\pi)$. Note that in general, several state paths can have the same label.

Definition 6. *Let H be an HMM, X be a sequence of length n, Λ be an annotation of sequence X. The probability of annotation Λ given sequence X is*

$$\Pr(\Lambda \mid X, H) = \sum_{\pi \in V^n, \lambda(\pi) = \Lambda} \Pr(\pi \mid X, H) \tag{2.9}$$

Note that $\Pr(\pi \mid X, H) = \frac{\Pr(\pi, X \mid H)}{\Pr(X \mid H)}$.

Example 5. *Let H be the HMM from the example 1. Let $C = \{I, G\}$ and $\lambda(R) = I$ and $\lambda(0) = \lambda(1) = G$. Consider sequence $X = AACT$, which was generated by the state path $\pi = R011$. The correct annotation of X is therefore $\lambda(R011) = IGGG$.*

We can imagine H as very simple (and not very realistic) gene predictor. R represents intergenic regions, state In represents introns and E represents exons. Label I represents intergenic regions and label G represents regions that are genes.

Using this interpretation, the sequence $AACT$ contains gene ACT. There are several state paths consistent with this annotation: if $AACT$ was generated by state path $REEE$ then substring EEE is exon; if it was generated by state path $REIE$ then sequence contain two exons and one intron in the middle. There are 2^3 different state paths π with $\lambda(\pi) = IGGG$. All of those state path support "fact" that $IGGG$ is the correct annotation of X.

Given a sequence X, the natural question is what is the best annotation of X. One measure of quality of an annotation is its probability. The more probable is annotation, the more likely X was generated with some state path with such annotation. We can formulate this in following problem.

Definition 7. *Given HMM H and sequence X, the most probable annotation problem (MPA) is the problem of finding an annotation Λ of X that maximizes the probability*

$$\Pr(\Lambda \mid X, H)$$

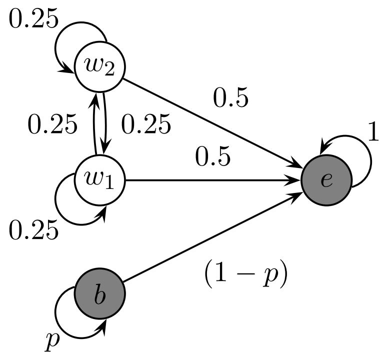


Figure 2.3: Example of an HMM with multiple path problem. States w_1, w_2, b emit symbol 0 with probability 1 and state e emits symbol 1 with probability 1. Initial distribution is set to $I_{w_1} = I_{w_2} = \frac{1}{4}, I_b = 0.5$ and $I_e = 0$. Annotations of the states are represented by the colors of the states ($\lambda(w_1) = \lambda(w_2)$ and $\lambda(b) = \lambda(e)$).

Theorem 1. *Most probable annotation problem is NP-hard.*

This theorem was proved in 2002 by Lyngsø *et al.* and proof can be found in [51]. Their proof was done by a reduction from the maximum clique problem. For input graph with n vertices they construct an HMM H with $O(n^2)$ states and sequence X of the length n. The most probable annotation of X could be converted into maximum clique of input graph. Their result was later strengthened by Brejová *et al.* in a sense that the problem is hard even for some constant-sized HMMs [14].

Theorem 2. *There exists an HMM such that it is NP-hard to find the most probable annotation to a given input sequence X.*

Same paper also contain polynomial Viterbi-like algorithm (Extended Viterbi Algorithm) that finds most probable annotation for special classes of HMMs.

Traditional way to find sequence annotation is to use Viterbi algorithm: given sequence X, we find the most probable state path π, and then compute $\lambda(\pi)$. If the coloring function λ is the identity function, this will find the most probable annotation. In general, the Viterbi algorithm is not even a good approximation of the most probable annotation as shown in the following example.

Example 6. *Consider the HMM from Figure 2.3. Take sequence $X = 0^n1$. Since state e is the only state that can emit 1, every state path with non-zero probability ends in state e. If a state path starts in state b then state path has form b^ne. Annotation of such a* *state path is $\Lambda_1 = \lambda(b)^{n+1}$ and probability of the annotation Λ_1 (and also the state path $b^n e$) is $p^n(1-p)$. If a state path starts in the one of the white states, then a state path has form $w'_0 w'_1 \dots w'_{n-1} e$ where w'_i is either w_1 or w_2. There are 2^n such state paths and each of them has probability $0.5 \cdot 0.25^n$ and annotation $\Lambda_2 = \lambda(w_1)^n \lambda(e)$. Probability of annotation Λ_2 is therefore 0.5^{n+1}. If n is sufficiently high and $\frac{1}{4} < p < \frac{1}{2}$ then the most probable state path is $b^n e$ which corresponds to the annotation Λ_1. However, the most probable annotation is Λ_2 and its probability is exponentially higher then the probability of Λ_1. Therefore the Viterbi algorithm it not even a good approximation of the most probable annotation problem.*

We say that an HMM has the multiple path problem *if it has an annotation that corresponds to more than one state path.*

Note that from the probabilistic nature of the HMMs, the most probable annotation does not have to be the best approximation of the correct annotation. We will discuss alternative decoding criteria in section 2.2.1. .

2.1.6 Training

Training is the process of estimating parameters of the probabilistic models. In this section we briefly describe several approaches for estimating transition and emission distributions of hidden Markov models.

We use the *maximum likelihood approach*. Let H_θ be an HMM where θ contains emission and transition probabilities and let D be training data (the set of pairs (X, π) where X is sequence and π is a state path). We want to find θ that maximizes the likelihood of the data:

$$L(H_\theta \mid D) = \Pr(D \mid H_\theta) = \prod_{(X,\pi)\in D} \Pr(X, \pi \mid H_\theta)$$

Under this scenario, we can use the frequencies of occurred events as the parameters of the model [22]. Let $A_{u,v}$ be the number of transitions from u to v in D. Let $E_{u,x}$ be the number times when state u emitted x in training data D. Then

$$a_{u,v} = \frac{A_{u,v}}{\sum_{w\in V} A_{u,w}} \qquad\qquad e_{u,x} = \frac{E_{u,x}}{\sum_{y\in\Sigma} E_{y,x}}$$

for all $u, v \in V, x \in \Sigma$. Parameters $\theta = (e, a)$ maximize the likelihood [22]. If case of insufficient data, some events that have nonzero probability may not occur in the data and therefore their probability will be estimated to zero. To avoid this behavior, we can use pseudo-counts [22]: we artificially add a constant k_x to the counts of all events x that we expect to have non-zero probability.

Training with Missing Data

In case that parts of the data are missing (for example a part of the state path or part of the sequence) we treat the missing data as a random variables. We can use the following algorithm.

1. Set the initial parameters θ_0. Let $i = 0$
2. Using model H_{θ_i}, compute the expected number of occurrences of all events (the values $A_{u,v}, E_{u,x}, u, v \in V, x \in \Sigma$). Compute the new parameter set θ_{i+1} from the expected counts (using the method described above). Set $i = i + 1$.
3. If stopping criterion was not reached, go to step two. Otherwise set the H_{θ_i} as the final model.

This algorithm is called the Baum-Welch algorithm and it is an instance of the more general Expectation maximization algorithm [22]. Sequence $\{L(H_{\theta_i} \mid D)\}_{i\geq 0}$ is non-decreasing and converges to a local minimum [22]. As a stopping criterion, we can use the number of iterations or the change in the likelihood. The expectations of the number of events can be computed by a variant of the Forward-Backward algorithm.

In step 2 we can replace the Forward-Backward algorithm with the Viterbi algorithm. In such case the Viterbi algorithm computes the most probable values for the missing data and new model is estimated from this data. This approach is called the Viterbi training. While the Viterbi training does not have to converge to local maxima, it is faster in practice [22]. In practice we can use the Viterbi training for the estimation of good starting parameters for the Baum-Welsch training [36].

2.1.7 Variants of Hidden Markov Models

In this section we describe several variants of hidden Markov models. Some of these extensions have more expressive power but mostly they are introduced for simplifying models. We will use all the described variants later in the thesis.

Silent states

One common variant of HMMs are HMMs with silent states. A silent state is a state that does not emit any symbol. In the presence of silent states a state path can be longer than

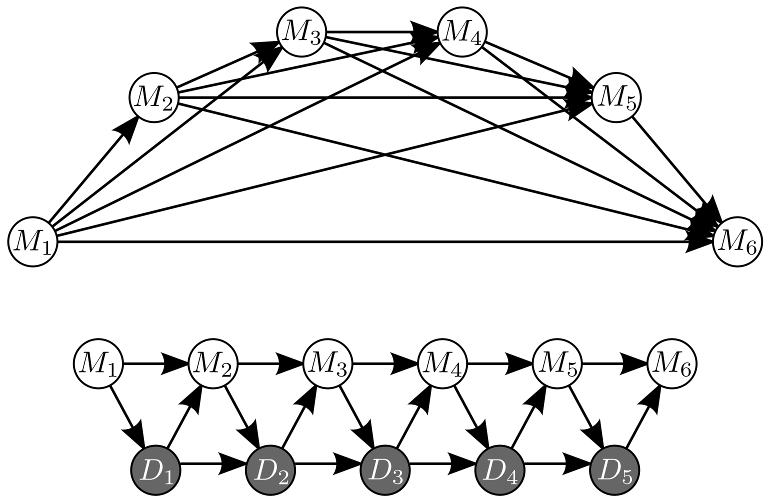


Figure 2.4: Upper model without silent states, lower model has silent states $D_1, \ldots, D_5$ to simplify transitions between states $M_i, 1 \leq i \leq 6$.

the emitted sequence. However, the number of non-silent states in the state path has to be equal to the sequence length. Silent states do not add any expressive power to HMMs, but in some cases they allow to reduce the number of transitions by a factor m. This can be used to decrease the number of parameters and speed up algorithms.

Example 7. *Example of an HMM with silent states that reduces the number of transitions by factor of* $\Theta(m)$.

Consider following HMM $H = (\Sigma, V, I, e, a)$ *with* m *states* $M_1, \ldots, M_m$ *there is transitions from* M_i *to* M_j *if and only if* $i < j$. *Example of such HMM is in Figure 2.4 and it has in general* $\frac{(m-1)(m-2)}{2}$ *transitions.*

To reduce number of transitions, we can create HMM $H' = (\Sigma, V', I', e', a')$ *by removing all transitions and adding chain of delete states* $D_1, \ldots, D_{m-1}$ *with following transitions:* $M_i \to M_{i+1}, M_i \to D_i, D_i] \to M_{i+1}$ *for all* $1 \leq i < m$ *and* $D_i \to D_{i+1}$ *for all* $1 \leq i < m-1$. *Structure of* H' *is in lower part of Figure 2.4. We have changed the number of transitions to* $4m - 5$ *at expense of additional* $m - 1$ *states. In case of sufficiently large* m *(at least* 12*),* H' *is smaller.*

Problem is that for certain transition matrices a, *it is not possible to find transition matrix* a' *such that the distributions of sequences of HMM* H *will be the same as the distribution of sequences of HMM* H'. *However, this type of reduction is used in often in practice, for example in profile HMM [22] which are later explained in section 3.2.2.*

The Viterbi algorithm, the Forward algorithm or Posterior decoding can be implemented *for H' in $O(nm)$ time while these algorithm will have time complexity $O(nm^2)$ for H.*

There should not be the cycle from transitions between silent states, because it causes problems with the order of computations of the Viterbi algorithm, the Forward algorithm and others. For example if we have silent cycle out of states u and v, the recurrency for computing the Forward algorithm contain a cycle: value $F[i,v]$ depends on value $F[i,u]$, and value $F[i,u]$ depends on value $F[i,v]$. Such cycles can be removed from an HMM without affecting of the distribution of the sequence, and for every HMM with silent states there is an HMM without silent states that defines same distribution of sequences [60].

Start and Final state

Sometimes it is useful to have a special start state and a special final state. Start states can be used instead of the initial distribution. State s is a start state, if $I_v = 1$. Conversely, we can model arbitrary initial distribution by a silent start state s with $a_{s,v} = I_v$ for all v.

In contrast, final states affect distribution of the model. An HMM defined in section 2.1.1 defines a distribution over the sequences of the same length. An HMM with final states defines distribution over sequences of all lengths. We denote the set of final states $F \subseteq V$. Transitions from final states are not defined (or set to zero). Emission distribution might be defined (if not, final states are silent). Every state path has to end with a final state, and final state can be only at the end of a state path.

With final states, HMM stops generating a sequence once it reaches some final state. Therefore the sum of the probabilities of all sequences is 1.

Final states slightly affect algorithms. For example in the Forward algorithm we do not have to change recurrences, only the final summation: probability of the sequence is $\sum_{q\in F} F[n-1,q]$. Similarly, in the Viterbi algorithm we have to find $\max_{q\in F} V[n-1,q]$ not $\max_{q\in V} V[n-1,q]$. The Backward algorithm is changed by setting $B[n-1,q]$ to zero for all $q \notin F$.

Advantage of start and final states is, that it is trivial to compose model together. For example to chain two model, we just merge start state of one model with final states of other model. For the purpose of model composition, it is a good practice if all models have single start state and single final state.

High Order HMMs

Sequence X and a state path π generated by an HMM can by considered as sequences of random variables $X_0, X_1, \ldots, X_{n-1}$ and $\pi_0, \pi_1, \ldots, \pi_{n-1}$. Random variables associated with the state path have the Markov property [46], which means that π_i depends only on π_{i-1} or more precisely $\Pr(\pi_i \mid \pi_0, \ldots, \pi_{i-1}) = \Pr(\pi_i \mid \pi_{i-1})$. Similarly, X_i depends only on π_i, that is $\Pr(X_i \mid \pi_0, \ldots, \pi_i, X_0, \ldots, X_{i-1}) = \Pr(X_i \mid \pi_i)$. However, sometimes the ability to look back more than just one state or symbol is useful. We can extend the transition and the emission probabilities to depend on several previous states/emissions.

We will briefly discuss k-th order HMMs where emissions are dependent on previous k emissions. Specifically, X_i depends only on π_i and $X_{i-k}, \ldots, X_{i-1}$. Emission distribution is therefore parametrized by state and previous emissions, which is a string of length at most k (it can be shorter in the beginning of the sequence). $e_{u,x,a}$ is probability that state u emits a under the condition that x is the suffix of the already emitted sequence. Let X be sequence and π be state path. Then definition of probability that X and π were generated by the model changes to

$$\Pr(X, \pi \mid H) = I_{\pi_0} e_{\pi,\varepsilon,X_0} \prod_{i=1}^{n-1} e_{\pi_i, X[i-\min\{k,i\}:i], X_i} a_{\pi_{i-1},\pi_i}$$

where ε is empty string. Other definitions will not change. We can use all algorithms that we have described above, but we have incorporate these new emission distributions.

Generalized HMMs

Consider a state v with a self-transition, i.e. a state for which $e_{v,v} > 0$. The number of steps the model remains in this state is distributed according to the geometric distribution. In particular, the probability that we will leave state v after exactly k steps is $e_{v,v}^{k-1}(1-e_{v,v})$. For some applications this behaviour not appropriate [17, 53].

A generalized hidden Markov model (GHMM) (or hidden semi-Markov model) has with every state v associated a duration distribution d_v. When a GHMM enters state v, it first samples length l according to the distribution d_v. Afterwards it generates string x of length l. To specify the probability $e_{v,x}$, each symbol of generated string x is usually generated independently, which means that $e_{v,x} = \prod_{i=0}^{|x|-1} e_{v,x[i]}$. Output of a GHMM are three sequences: state path $\pi = \pi_0\pi_1 \ldots \pi_{l-1}$, *duration sequence* $D = D_0 D_1 \ldots D_{l-1}$ and sequence $X = X_0 X_1 \ldots X_{n-1}$. The state path and the duration sequence has same length and

$$\sum_{i=0}^{|D|-1} D_i = |X|$$

.

Manipulation with GHMMs is more complicated and technical. Computing this probability of sequence X can be done by a variant of the Forward algorithm. Finding π and D maximizing the probability $\Pr(\pi, D, X \mid H)$ can be found by the variant of the Viterbi algorithm. However, time complexity of those algorithm on GHMM is higher because for computing probability $V[i, v]$ (the probability of the most probable state path ending in state v at position i) we have to consider all possible emission lengths of state v, which is i. The Viterbi algorithm and the Forward algorithm run in $O(n^2m^2)$ time.

2.2 Other Decoding Methods for HMMs

In the previous sections we have described two decoding algorithms: the Viterbi algorithm that finds the most probable state path, and the Posterior decoding that for every symbol of the sequence assigns the state that generated such symbol with maximum probability. We have shown that in some cases these algorithms can recover bad annotation. However, maximizing the most probable annotation is NP-hard and therefore it is not tractable. Additionally, the most probable annotation does not necessary have to be the best approximation of the true annotation.

2.2.1 Highest Expected Gain

In this section we will describe a framework for studying decoding algorithms in a more systematic way. This framework was introduced originally for conditional random fields [31]. To use this framework, we need to define a gain function, which will express similarity (gain) between two annotations or state paths. The higher the gain, the more similar those two annotations are. Gain function is domain specific and can penalize the differences in the domain specific features of the state paths. Gain function is not a similarity in the mathematical sense; it does not even have to be symmetric.

Our goal is to find an annotation that is as similar as possible to the correct annotation. Problem is that we do not know the correct annotation. Our only assumption is that the input sequence came from the model and therefore we will treat the correct annotation as a random variable, with probability distribution defined by the HMM and the observed sequence. We will seek for annotation that maximizes the highest expected gain [58, 60].

Definition 8. *Let H be an HMM and L be the set of all annotations. Any function $f : L \times L \to \mathbb{R}$ is an* annotation gain function*.*

Let Π be the set of all state paths. Any function $f : \Pi \times \Pi \to \mathbb{R}$ is a path gain functions*.*

Note. We will use term gain function instead of annotation/path gain function if it is clear from the context.

Machine learning literature often uses a related term of loss function [44]. Lower loss mean more similar annotations. Therefore instead of maximizing the expected gain we can therefore equivalently minimize the expected loss.

Definition 9. *Let H be an HMM, f be a gain function, X be a sequence generated by H and Λ be an annotation of X. Then the* expected gain *of annotation Λ is*

$$E_{\Lambda_X|X,H}[f(\Lambda_X,\Lambda)] = \sum_{\Lambda_X} f(\Lambda_X,\Lambda)\Pr(\Lambda_X \mid X,H) \tag{2.10}$$

Let π be a state path. Then the expected gain of state path π is

$$E_{\pi_X|X,H}[f(\pi_X,\pi)] = \sum_{\pi_X} f(\pi_X,\pi)\Pr(\pi_X \mid X,H) \tag{2.11}$$

Once we have HMM H, gain function f and the observed sequence X, we try to find the annotation/state path maximizing the expected gain.

$$\Lambda = \arg\max_{\Lambda} E_{\Lambda_X|X,H}\left[f\left(\Lambda_x,\Lambda\right)\right] \tag{2.12}$$

We can express the classical decoding algorithms within this framework to show its universality. We will define two gain functions: f_A which corresponds to the Viterbi algorithm and the most probable annotation problem and f_p which corresponds to the Posterior decoding.

The gain function f_A is simply identity function (definition for state paths is analogous).

$$f_A(\Lambda,\Lambda') = \begin{cases} 1 & \text{if } \Lambda = \Lambda' \\ 0 & \text{if } \Lambda \neq \Lambda' \end{cases} \tag{2.13}$$

For this gain function $E_{\Lambda_X}[f_A(\Lambda_X,\Lambda)] = \Pr(\Lambda \mid X,H)$ and therefore maximizing expected gain is equivalent to the most probable annotation problem. Similarly if we define gain as the identity function over state paths, we obtain the most probable state path problem, which can by solved by the Viterbi algorithm.

Therefore for given sequence finding annotation with highest expected gain is NP-hard if gain function is part of the input. However, for specific gain functions we can maximize expected gain in polynomial time.

The gain function f_P compares two annotations of the same length position by position and assigns score 1 to every position where they are equal.

$$f_P(\Lambda, \Lambda') = \begin{cases} 0 & \text{if } |\Lambda| \neq |\Lambda'| \\ \sum_{i=0}^{|\Lambda|-1} \begin{cases} 1 & \text{if } \Lambda_i = \Lambda'_i \\ 0 & \text{otherwise} \end{cases} \end{cases} \tag{2.14}$$

Similarly, we can define gain function f_P for state paths. In such case maximizing the expected gain is equivalent to the Posterior decoding. Highest expected gain framework give us another interpretation of the scoring function of the Posterior decoding. Let Λ_X have same length as Λ. From linearity of the expectation we have

$$E_{\Lambda_X}[f_P(\Lambda_X, \Lambda)] = \sum_{i=0}^{|\Lambda|-1} E_{\Lambda_X[i]}[f_P(\Lambda_X[i], \Lambda[i])]$$

We say that the i-th label of Λ is correctly predicted if $f_P(\Lambda_X[i], \Lambda[i]) = 1$ (which is true if $\Lambda_X[i] = \Lambda[i]$). Therefore by maximizing f_P we search for an annotation/state path that maximizes the expected number of correctly predicted labels/states.

2.2.2 Maximum Boundary Accuracy Decoding

Maximum boundary accuracy decoding (MBAD) is used in the gene-finder CONTRAST [31]. It was proposed for conditional random fields (CRF), but since CRF are similar to HMM, we define it in the terms of HMMs.

This decoding method maximize the weighted difference between the expected number of true-positive and false-positive coding region boundaries.

Definition 10. *Let $\Lambda = \Lambda_0\Lambda_1 \ldots \Lambda_{n-1}$ be an annotation. A boundary of annotation Λ is every position i where $\Lambda_{i-1} \neq \Lambda_i$.*

Maximum boundary accuracy decoding has one parameter γ. Let $B_{\Lambda'}$ be the set of all boundaries in Λ'. Then MBAD maximizes the following function:

$$f(\Lambda, \Lambda') = \sum_{i \in B_{\Lambda'}} g(\Lambda, \Lambda', i) \tag{2.15}$$

where

$$g(\Lambda, \Lambda', i) = \begin{cases} 1 & \text{if } \Lambda_{i-1} = \Lambda'_{i-1} \text{ and } \Lambda_i = \Lambda'_i \\ -\gamma & \text{otherwise} \end{cases} \tag{2.16}$$

Algorithm for optimizing this gain function is similar to the posterior decoding and details of implementation for HMMs can be found in [60]. The time complexity of the algorithm is $O(nc^2 + nm^2)$ where n is the length of the sequence, m is the number of states and c is the number of labels.

Intuition behind gain function f is following. Like with the Posterior decoding, we want to maximize the number of correctly predicted boundaries (the Posterior decoding maximizes the number of correctly predicted states). The difference is in the γ. If $\gamma = 0$ then almost every possible boundary have positive gain and therefore the reconstructed annotation will contain many false-positive boundaries with very small expected gain. Positive γ cause that boundaries with small posterior probability will have negative expected gain and therefore it is less likely that they appear in the optimal annotation.

2.2.3 Highest Expected Reward Decoding

The Highest Expected Reward Decoding (HERD) is the extension of maximum boundary accuracy decoding. We have developed this decoding for prediction of recombination of HIV virus. Further details can be found in [58, 60] and in Chapter 3, where we extend HERD to reduce the amount of systematic errors.

Genome of some viruses (for example HIV or HCV virus) can be divided into several subtypes. Moreover, it is possible that virus is mosaic recombination of viruses from different subtypes (we call this virus recombinant). In the problem of recombination detection we try to decide if given sequence X is recombinant sequence. If X is recombinant then we want to find original subtypes of every part of a sequence X. Recombinations can be modeled by jumping HMMs [65] which are HMM with topology specific to this domain. In this application is hard to find exact recombination point since annotations with slightly shifted boundaries has similar probabilities. Therefore we have defined the following gain function.

We say that boundary on position i is correctly predicted, if it satisfy following conditions.

1. There is boundary between same labels in the correct annotation on position j and j is within distance W from i ($|i - j| \leq W$).

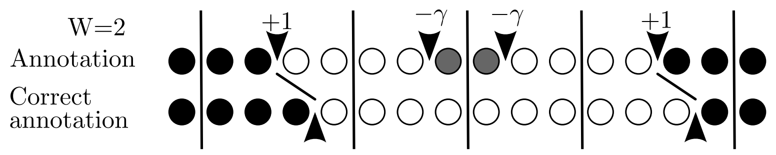


Figure 2.5: Triangles corresponds to the boundaries. Vertical lines corresponds to the windows of size $2W$, or smaller to avoid overlaps. Windows around boundaries represents regions where we search for the corresponding boundaries in the correct annotation. The first and the fourth boundary are correctly predicted because in the correct annotation is boundary between same colors within distance $W = 2$. This picture was taken from [60].

2. There is no other boundary between i and j.

Boundaries are illustrated in figure 2.5.

Let x be the number of correctly predicted boundaries and y be the number of other boundaries in the proposed annotation. Then $f(\Lambda, \Lambda') = x - \gamma y$ where γ is defined constant. If $W = 1$ then this is equivalent to the Maximum Boundary Accuracy Decoding. Intuition behind this gain function is that we want to amplify the expected gain for the boundary if there are many annotations with similar boundary.

Optimizing this criteria can be done in $O(|X|W|C||T| + n|C|^2W^2)$ time and $O(\sqrt{|X|}|C||V| + W|C||V| + n|C|^2)$ memory. Experimental evaluation, optimization algorithm and implementation details can be found in [58, 60].

2.2.4 Distance Measures on Annotations

Another approach to solve similar problem was proposed by *Brown, Truszkowski* in [16]. Originally their implementation was aimed at prediction of boundaries in transmembrane proteins [16], but later they successfully adapted their algorithm to jumping HMMs [67]. Their approach is trying to solve same problem as HERD: the exact boundaries of an annotations are hard to find and grouping similar annotation is useful. At first we give few definitions.

Definition 11. *Let d be any distance measure defined on annotations. Ball of radius r around annotation Λ is*

$$B_d(\Lambda, r) = \{\Lambda' \mid d(\Lambda, \Lambda') \leq r\}$$

Definition 12. *Let $\Lambda = \Lambda_0\Lambda_1 \ldots \Lambda_{n-1}$ be an annotations. Footprint of Λ is maximal subsequence of Λ that does not contain two identical consecutive labels.*

Definition 13. *Let $b_i(\Lambda)$ be i-th boundary of Λ and $b(\Lambda)$ be the number of boundaries in Λ. Border shift distance d_b is*

$$d_b(\Lambda, \Lambda') = \begin{cases} \infty & \textit{if } \Lambda \textit{ and } \Lambda' \textit{ have different footprint} \\ \max_{i=0}^{b(\Lambda)-1} d_i(\Lambda) - d_i(\Lambda') & \textit{otherwise} \end{cases}$$

and border shift sum distance d_s is

$$d_s(\Lambda, \Lambda') = \begin{cases} \infty & \textit{if } \Lambda \textit{ and } \Lambda' \textit{ have different footprint} \\ \sum_{i=0}^{b(\Lambda)-1} d_i(\Lambda) - d_i(\Lambda') & \textit{otherwise} \end{cases}$$

To predict the recombinations of sequences or the annotations of the transmembrane proteins *Brown, Truszkowski* maximize following function

$$f_d(\Lambda, \Lambda') = \begin{cases} 1 & \mathrm{i}f \Lambda \in B_d(\Lambda', r) \\ 0 & \text{otherwise} \end{cases}$$

As distance d, they have considered Hamming distance, Border shift distance and Border shift sum distance [16]. If we set $r = 0$ then f_d is same as f_A and therefore finding annotation that maximize f_d is NP-hard. Additionally, if $r \geq n$ then the problem is equivalent to finding the most probable footprint, and finding the most probable footprint is NP-hard, which we show in the Chapter 3.

Maximizing f_d is NP-hard but finding the annotation with footprint F and maximal expected gain can be done in polynomial time [16]. Therefore *Brown and Truszkowski* used following heuristic algorithm: At first sample the state paths to get the most probable footprint F with high probability, then use the polynomial algorithm for finding the annotation with footprint F and the highest expected gain.

Gain function f_d is similar to f_A: the annotation is considered correct if it is same/very similar to the correct one. MBAD, HERD and the Posterior decoding do not care about overall structure of annotation, they construct annotation from highly probable features. However, decoding function f_d take in to account overall structure of the sequence.

2.3 Sequence Alignments

In this section we introduce the sequence alignment problem, basic algorithms for computing optimal alignments of two sequences.

Parts of two sequences are homologous, if they have evolved from same sequence in their common ancestor. Aim of sequence alignment is to identify homologous sequences.

Sequences can be modified by different evolution events: mutation of a residue into another residue, deletion of a part of the sequence, insertion of residues into the sequence. There are also large-scale rearrangement events like duplications (some substring is duplicated and copied into other part of the sequence), inversions (some substring is reversed) or translocations in which part of the sequence change position. We will ignore them in this thesis, as they cannot be represented by traditional alignments. An alignment is a data structure that represents comparisons of two or more sequences. We obtain an alignment of k sequences by inserting dashes into individual sequences so that they have the same length. We can represent an alignment as a matrix or a table. Each row of the alignment is a sequence with inserted dashes, and each column is a list of residues from all rows at the same position.

An alignment has the following biological meaning: homologous residues (those that have evolved from a common ancestor) are in same column. Dashes represents either the parts of sequence that were deleted during evolution (deletions) or positions where some residues were inserted into some other sequence (insertions). In an alignment it is not possible to distinguish between insertions and deletions – we cannot tell if something was inserted into one sequence or if there was deletion in the other sequences. Therefore we will refer to insertions and deletions as to *indels*.

Example 8. *Consider the following evolutionary history of hypothetical DNA sequences of two living organisms X and Y. There was a ancestral sequence P which evolved into X' and Y' and after that X' evolved into X and Y' evolves into Y. These sequences are shown bellow:*

```
X:      C C     G C G A C C T T G C             A C C A
X':     C C     G           T T G C             A G C A
P:      A C T G G           T C G C T G A G C T A G C A
Y':     T C T G G           C C           G C T A G C A
Y:      T C T A G           C C           G A T A G C A
```

During evolution from P to X', four events occurred: deletion of sequences "TG" and "TGAGCT", and mutations of two bases. During evolution from X' to X, one base was changed, and sequence "CGACC" was inserted. Similarly, during evolution from P to Y', two bases mutated, and two sequences were deleted. During evolution from Y' to Y one base mutated, and sequence "CGACC" was inserted.

From evolutionary history described above, we can create an alignment of sequences X

and Y by removing ancestral sequences, and removing columns that contain only gaps and replacing gaps with dashes (gap symbols).

```
X:     C C - - G C G A C C T T G C - - - A C C A
Y:     T C T A G - - - - - C C - - G A T A G C A
```

In this alignment, symbols that are in the same column are truly homologous (they evolved from the same symbol in P). As you can see, homologous symbols do not have to be equal.

There is only one alignment that reflects the true evolutionary history. Our goal is to find that alignment, or at least alignment, that is as similar as possible. The alignment shown above is a *global alignment* because it is an alignment of whole sequences X and Y. A *local alignment* is an alignment of parts of sequences: a local alignment of sequences X and Y is a global alignment of strings $\bar{X}$ and $\bar{Y}$ where $\bar{X}$ is a substring of X and $\bar{Y}$ is a substring of Y. Since global alignments do not consider rearrangement events[1], local alignments are useful to align sequence parts that did not underwent such events. We will mostly consider global alignments, but most of the methods can be extended also to local alignments.

In this section we will review basic methods for constructing alignments. We will discuss basic scoring schemes and algorithms that find optimal alignment under these schemes.

2.3.1 Scoring Schemes

Since we want to construct alignments that have biological meaning, we have to develop a method for assessing the quality of an alignment. One way of doing so is to define a scoring scheme, which assigns to every alignment a real number (called score). The alignments similar to the true alignment should have higher score than the alignments that differ from the true alignment. Once we have a scoring scheme, we will search for the alignment of the input sequences with the highest score.

Typical scoring schemes used in practice score each column of an alignment without gaps independently. Gaps are scored by a penalty that depends on the length of the gap (the number of consecutive dashes). Score of an entire alignment is the sum of the scores of all ungapped columns plus the sum of the scores of all gaps.

In particular we will assume that all sequences are from a finite alphabet Σ. Specifically, the DNA alphabet contain 4 symbols A, C, G, T, and protein alphabet contains 20 codes

[1]Duplication, reversal or translocation.

of amino acids. We will score a column containing residues a and b by $S[a, b]$ where S is a matrix of size $|\Sigma| \times |\Sigma|$ called *substitution matrix*. A gap of length x has a score $g + dx$, where g is the gap opening penalty and d is the gap extension penalty. Both are usually negative, since we want alignments that contains many columns with same or similar symbols. Positive gap penalty causes tendency towards alignments with many gaps, which reduce the number of aligned residues. We call this gap scoring scheme an affine gap model [22].

Matrices and gap penalties, used as scoring schemes, are usually derived from frequencies of the substitutions and indels in alignments created manually by biologists. Example of such matrices are PAM or BLOSUM matrices[22]. Additionally, scoring matrices can be also derived from theoretical models of evolution, one such example is Jukes-Cantor model[22].

2.3.2 Needleman-Wunsch Algorithm

The alignment with the highest score can be found by the Needleman-Wunsch algorithm [22]. This algorithm uses an arbitrary score table S, affine gap model with gap penalty d and gap opening penalty $g = 0$ (if $g = 0$ then this model is also called linear gap model). To align sequences X and Y of length n and m respectively, we define matrix M of size $n \times m$. Element $M[i, j]$ will be the score of the best alignment of sequences $X[: i]$ and $Y[: j]$. We can compute $M[i, j]$ by the following equations:

$$M[-1, -1] = 0 \tag{2.17}$$

$$M[-1, i] = i \cdot d, 0 < i < m \tag{2.18}$$

$$M[i, -1] = i \cdot d, 0 < i < n \tag{2.19}$$

$$M[i, j] = \max \begin{cases} M[i-1, j-1] + S(X_i, X_j) \\ M[i, j-1] + d \\ M[i-1, j] + d \end{cases} , 0 \leq i < n, 0 \leq j < m \tag{2.20}$$

By computing $M[n-1, m-1]$, we have the score of the optimal (highest-scoring) alignment of X and Y [22].

To cope with gap opening penalty, we have to slightly change the algorithm. We define two other matrices M_X and M_Y of same size as M. Element $M_X[i, j]$ will contain the highest score of an alignment of sequences $X[: i]$ and $Y[: j]$ that ends with a gap in sequence X. M_Y is analogous. Values of these matrices can be computed by the following recurrences.

$$M[-1,-1] = 0 \tag{2.21}$$

$$M[-1,i] = M_X[-1,i] = i \cdot d + g, 0 < i < m \tag{2.22}$$

$$M[i,-1] = M_Y[i,-1] = i \cdot d + g, 0 < i < n \tag{2.23}$$

$$M_X[i,-1] = -\infty, 0 \leq i < n \tag{2.24}$$

$$M_Y[-1,i] = -\infty, 0 \leq i < m \tag{2.25}$$

$$M[i,j] = \max \begin{cases} M[i-1,j-1] + S(X_i, X_j) \\ M_X[i,j] \\ M_Y[i,j] \end{cases}, 0 \leq i < n, 0 \leq j < m \tag{2.26}$$

$$M_X[i,j] = \max \begin{cases} M[i-1,j] + g + d \\ M_X[i-1,j] + d \end{cases}, 0 \leq i < n, 0 \leq j < m \tag{2.27}$$

$$M_Y[i,j] = \max \begin{cases} M[i,j-1] + g + d \\ M_Y[i,j-1] + d \end{cases}, 0 \leq i < n, 0 \leq j < m \tag{2.28}$$

To compute the value of $M[i,j]$ (and optionally M_X and M_Y), these equations use only values from neighbouring cells which have at least one coordinate smaller. Therefore we can order computation of rows of M so that when we compute value $M[i,j]$, the necessary values are already computed. Let F be the function, that takes as input matrix M and coordinates (i,j) and computes $M[i,j]$ according equation 2.20. Let F' be the same function as F, but with max replaced with arg max. Function $F'(M,(i,j))$ will thus return which cell was used in computation of $F(M,(i,j))$. Note that if g is not zero, then F would take as input the matrices M, M_X, and M_Y and compute $M[i,j], M_X[i,j]$, and $M_Y[i,j]$ according equations 2.26-2.28

The Needleman-Wunsch algorithm can be implemented by the following code. For simplicity we show only computation of matrix M.

```
Initialize M[i,-1] and M[-1,i]
for i in 0...n-1
  for j in 0...m-1
    M[i,j] = F(M,(i,j))
(i,j) = (n-1,m-1)
```

```
6  while i > 0 or j > 0
7    (i',j') = F'(M,(i,j))
8    (a,b) = (X[i],Y[j])
9    if i' = i then a = '-'
10   if j' = j then b = '-'
11   print column of alignment (a,b)
12   (i,j) = (i',j')
```

Lines 2-5 fills matrix M and lines 6-12 implement the back-tracing procedure. Time complexity of this algorithm is $O(mn)$ and memory requirements are $O(mn)$ since we keep matrix M in memory. The algorithm for affine gap model with $g \neq 0$ has the same complexity.

Note that for computing i-th row of matrix M we need only values from row i and $i-1$. Therefore if we want to know just the score of the optimal alignment, we can compute it in $O(m+n)$ memory: after computing of row i, we can discard row $i-1$. However if we want find the optimal alignment, we have to keep matrix M in the memory or use one of the techniques that are described in Section 2.5.

2.4 Sequence Alignments with Pair HMM

In this section we describe pair hidden Markov models (pHMM), which are commonly used for studying relationships between different sequences, relation between Needleman-Wunsch and pHMM, three standard decoding methods for decoding pHMMs, and survey of literature for examples of using pHMM for sequence alignments.

2.4.1 Pair Hidden Markov Models

Pair hidden Markov models are HMMs that generate output on two tapes, resulting in two emitted sequences. Every state can in one step generate one symbol in each sequence, or one symbol in one of the sequences or no symbols at all. Formally, every state generates a pair of strings (a, b), where a and b are of length at most one. Moreover, every state generate strings of constant length (for example it always generates one symbol on each tape). Formal definition of pair HMMs is given below. We can use pHMM to define probabilistic scoring schemes for alignments.

In particular, symbols generated by the same state are considered homologous (are in same column of an alignment). Symbols that are generated by a state that generates only in one sequence are aligned to a gap.

Definition 14. *A pair hidden Markov model (pHMM) is a tuple $H = (\Sigma, V, I, d, e, a)$, where Σ is a finite alphabet, V is a finite set of states, I is an initial distribution and a is a transition matrix, all defined as in definition 2. d_v^x and d_v^y are state durations of state v in sequence x and y respectively. For all $v \in V$, $d_v^x \in \{0,1\}$ and $d_v^y \in \{0,1\}$. Emission probability matrix e is a $|V| \times (|\Sigma \cup \{\varepsilon\}|)^2$ matrix with the following properties:*

1. $\forall v \in V, \forall \sigma_1, \sigma_2 \in \Sigma \cup \{\varepsilon\} : 0 \leq e_{v,(\sigma_1,\sigma_2)} \leq 1$
2. $\forall v \in V : \sum_{\sigma_1,\sigma_2 \in \Sigma \cup \{\varepsilon\}} e_{v,(\sigma_1,\sigma_2)} = 1$
3. *For all states v if $e_{(v,\sigma_1,\sigma_2)} > 0$ then $d_v^x = |\sigma_1|$ and $d_v^y = |\sigma_2|$*

Definition of a state path is the same as for HMMs with silent states. Restriction on the state durations (condition 3 in the definition) ensures that given a state path π and emitted sequence X and Y, for every symbol from X and Y we can assign a state from π that generated that symbol. However, given only two sequences X and Y and no state path, it is not possible to determine which symbols of the two sequences were generated together.

Definition 15. *Let $\pi = \pi_0 \dots \pi_l$ be a state path. Then the* cumulative duration times *are $D_i^x(\pi) = \sum_{j=0}^{i} d_{\pi_j}^x$ and $D_i^y(\pi) = \sum_{j=0}^{i} = d_{\pi_j}^y$. Additionally, $D_{-1}^x(\pi) = D_{-1}^y(\pi) = 0$. If it will be clear from the context which state path we are using, we will write D_i^x and D_i^y instead of $D_i^x(\pi)$ and $D_i^y(\pi)$.*

Given sequences X and Y and state path π, we can tell which symbols were generated by which states. Since every state v generates exactly d_v^x and d_v^y symbols in X and Y respectively, state π_i generated pair $(X[D_{i-1}^x : D_i^x], Y[D_{i-1}^y : D_i^y])$ States $\pi_0, \pi_1, \dots \pi_{i-1}$ generated first D_{i-1}^x symbols in X and first D_{i-1}^y symbols in Y.

Example 9. *In the figure 2.6 is pair hidden Markov model modeling sequence alignment with affine gap model. State M is called match state and generates pair of aligned residues and corresponds to matrix M from the sequence alignment algorithm. Insert states I_X and I_Y represents gaps, because they generate residues in only one sequence. Insert states correspond to matrices M_X and M_Y from the sequence alignment algorithm from section 2.3.2.*

Note that a state path uniquely define an annotation (match state corresponds to match columns, insert states represent gap). Finding the most probable state path for this HMM is equivalent to the Needleman-Wunsch algorithm with affine gap model [22].

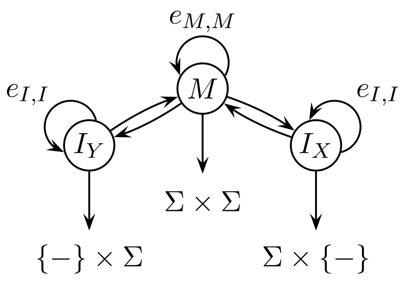


Figure 2.6: Pair hidden Markov model for pairwise alignment. It has two transitions parameters $e_{M,M}$ and $e_{I,I}$, since we set $e_{I,M} = 1 - e_{I,I}$ and $e_{M,I} = \frac{1}{2} - \frac{1}{2}e_{M,M}$. Match state M generates aligned pair of symbols and states I_X and I_Y generates symbols only in X or Y respectively. Initial distribution is uniform.

Definition 16. *Let $H = (\Sigma, V, I, d, e, a)$ be a pHMM, X and Y be arbitrary sequences over Σ and π be a state path. The probability that sequences X and Y were generated by a model H with state path π is*

$$\Pr(X, Y, \pi \mid H) = I_{\pi_0} \left(\prod_{i=1}^{|\pi|} a_{\pi_{i-1},\pi_i} \right) \left(\prod_{i=0}^{|\pi|} e_{\pi_i,(X[D^x_{i-1}:D^x_i],Y[D^y_{i-1}:D^y_i])} \right) \tag{2.29}$$

If $D^x_{|\pi|-1} \neq |X|$ or $D^y_{|\pi|-1} \neq |Y|$ then $\Pr(X, Y, \pi \mid H) = 0$.

Similarly as for HMMs, we can define the probability that sequences X and Y were generated by the model H.

Definition 17. *Let $H = (\Sigma, V, I, e, a)$ be a pHMM and X and Y be arbitrary sequences over Σ. Then probability that sequences X and Y were generated together by model H is*

$$\Pr(X, Y \mid H) = \sum_{\pi} \Pr(X, Y, \pi \mid H) \tag{2.30}$$

2.4.2 Viterbi Algorithm for Pair HMMs

Algorithms operating over pHMMs are similar to those for the regular HMMs, but in general they have higher computational complexity because they combine computation over model states with sequence alignment. In this section, we describe two-dimensional version of the Viterbi algorithm, other algorithms are analogous.

The Viterbi algorithm for HMMs computes variables $V[i, v]$ and $B[i, v]$. Every variable is parametrized by a position in the sequence and a state. For two-dimensional version, we will add position in the second sequence.

Let $V[i, j, v]$ be the probability of the most probable state path that generated $X[: i+1]$ and $Y[: j+1]$ and ended in state v. Clearly, $\max_{v\in V} V[|X|-1, |Y|-1, v]$ is the probability of the most probable state path, that generated X and Y. Let $B[i, j, v]$ be the last but one state of the most probable state path that generated $X[: i + 1]$ and $Y[: j + 1]$ and ended in state v. To make it easier, we expect that all states but one are not silent – they emit symbol in at least one sequence. Let $n = |X|$ and $m = |Y|$.

$$V[-1, -1, v] = I_v, v \in V \tag{2.31}$$

$$V[-2, i, v] = V[j, -2, v] = 0, \forall v \in V, -1 \le i < n, -1 \le j < m \tag{2.32}$$

$$V[i, j, v] = \max_{u\in V} V[i - d_v^x, j - d_v^y, u] a_{u,v} e_{v,(X[i-d_v^x:i],Y[j-d_v^y:j])} \tag{2.33}$$

$$B[i, j, v] = \arg\max_{u\in V} V[i - d_v^x, j - d_v^y, u] a_{u,v} e_{v,(X[i-d_v^x:i],Y[j-d_v^y:j])} \tag{2.34}$$

In equations 2.33 and 2.34 boundaries for i and j are $-1 \le i < n, -1 \le j < m$ and $i > -1$ or $j > -1$.

By finding the last state v of the most probable state path and back-tracing from $B[n - 1, m - 1, v]$, we can reconstruct the most probable state path. Time complexity of this algorithm is $O(nm|V|^2)$ (or $O(nm(|V| + t)$ where t is the number of transitions of H) and memory requirements are $O(nm|V|)$. However, we can use various tricks to decrease memory requirements of such algorithms, as shown in the section 2.5.

The Forward algorithm for GpHMM can be obtained by replacing maximum with addition for computation of $V[i, j, v]$ term. The table B is irrelevant for the Forward algorithm. The Backward algorithm and Forward-Backward algorithm are analogous to the Forward algorithm and their versions for HMM.

2.4.3 Generalized Pair HMMs

A generalized pair HMM (GpHMM) (or pair hidden semi-Markov process) are combination of HMM and GHMM. Every state generates two duration lengths d^x and d^y from some joint distribution associated with the current state $d_v(d^x, d^y)$ and after that it generates two strings x' and y' with lengths d^x and d^y according to the joint distribution $e_{v,(x',y')}$. This probability distribution can by defined for example by pair hidden Markov model. As with GHMMs and unlike pHMMs, the state path is not sufficient to determine which parts of the sequences were generated by which state, we also need two sequences of durations. The Viterbi equations for the GpHMM are following:

$$V[-1,-1,s] = 1 \tag{2.35}$$

$$V[-2,i,v] = V[j,-2,v] = 0, \forall v \in V, -1 \leq i < n, -1 \leq j < m \tag{2.36}$$

$$V[i,j,v] = \max_{u \in V, d^x \leq i, d^y \leq j} V[i-d^x, j-d^y, u] a_{u,v} e_{v,(X[i-d^x:i],Y[j-d^y:j])} d_v(d^x, d^y) \tag{2.37}$$

$$B[i,j,v] = \arg\max_{u \in V, d^x \leq i, d^y \leq j} V[i-d^x, j-d^y, u] a_{u,v} e_{v,(X[i-d^x:i],Y[j-d^y:j])} d_v(d^x, d^y) \tag{2.38}$$

In equations 2.37 and 2.38 boundaries for i and j are $-1 \leq i < n, -1 \leq j < m$ and $i > -1$ or $j > -1$. Additionally, in all equations d_x and d_y can be also bounded by D, the maximum duration length.

Drawback of GpHMM is increased computational complexity. Assume that the emission probability can be computed in $f(n', m')$ time, where n' and m' are the lengths of the emitted sequences. Then the time complexity of the Viterbi algorithm is $O(nm|E|D^2 f(D,D))$ where E is the set of all transitions in a pHMM and D is the maximum duration length [56]. In the case there is not maximum duration length, the time complexity of the Viterbi algorithm is $O(n^2m^2|E|f(n,m))$. For example, if the emission probability is computed by another pHMM with t transitions, then $f(n', m') = O(n'm't)$ and the time complexity of the Viterbi algorithm is $O(n^3m^3|E|t)$ or $O(nm|E|D^4t)$. GpHMMs were successfully used for gene-finding [2, 47, 54, 56].

The Forward algorithm, and the Forward-Backward algorithm are very similar to their versions for pHMM. We will discuss some implementation details for these algorithms in Sections 4.4 and 4.5.

2.4.4 Decoding Methods

In this section we review three decoding methods that were used in literature to reconstruct pairwise alignments: the Viterbi algorithm, the Posterior decoding and the Marginalized posterior decoding.

Let H be an pHMM (or GpHMM) and X and Y be the sequences that we want to align. The probability $\Pr(X, Y \mid H)$ is the probability that X and Y were generated in model H. From state path π in a pHMM we can reconstruct a unique alignment A_π. The model defines the probability that A_π is the true alignment of X and Y under the assumption, that X and Y were generated by model H:

$$\Pr(\pi \mid X, Y, H) = \frac{\Pr(\pi, X, Y \mid H)}{\Pr(X, Y \mid H)}$$

We can use $\Pr(\pi \mid X, Y, H)$ as a score of alignment A_π. Path π with the highest score can be found by a two-dimensional version of the Viterbi algorithm (section 2.4.2). Alignment A_π can be constructed from X, Y and π in a straightforward way: for every match state from π that generated $X[i]$ and $Y[j]$, we add column $(X[i], Y[j])$. For every indel state in π that generates $X[i]$, we add to alignment column $(X[i], '-')$. An indel states for the second sequences are analogous. The two-dimensional Viterbi algorithm is used in most of the software tools we will discuss later.

Alternatively, we can decode pHMMs using a variant of Posterior decoding. Two variants of the Posterior decoding for the pHMMs were described in the literature: the Posterior decoding and the Marginal posterior decoding [50]. Let $\Pr(X[i] \sim Y[j] \mid X, Y, H)$ be the probability that $X[i]$ and $Y[i]$ are aligned: the sum of the probabilities of all alignments that contain column $(X[i], Y[i])$. Let $\Pr(X[i] \sim -_j \mid X, Y, H)$ be the probability that $X[i]$ is aligned to a gap that is in Y between positions j and $j+1$. Similarly let $\Pr(-_i \sim Y[j] \mid X, Y, H)$ be the probability that $Y[j]$ is aligned to a gap in X between positions i and $i+1$. Posterior probabilities defined above can be computed by the two-dimensional version of the Forward-Backward algorithm.

Let alignment A of sequences X and Y have length n and consists of columns $a_0, a_1, \dots,$ a_{n-1}. Each column is a pair $a_i = (x_i, y_i)$ where x_i and y_i are symbols from $\Sigma \cup \{-\}$ [2]. Let $d_A^x(i)$ be the number of non-gap symbols in $x_0, x_1, \dots x_i$, let $d_A^y(i)$ be the number of non-gap symbols in $y_0, y_1, \dots, y_i$ and define $d_A^x(-1) = d_A^y(-1) = 0$. In this notation, $A[0:i]$ is an alignment of $X[: d_A^x(i)]$ and $Y[: d_A^y(i)]$. Then the posterior probability of an alignment column a_i is

$$P(a_i) = \begin{cases} \Pr(x_i \sim y_i \mid X, Y, H) & \text{if } x_i \text{ and } y_i \text{ are not gap symbols} \\ \Pr\left(x_i \sim -_{d_A^y(i)-1} \mid X, Y, H\right) & \text{if } y_i \text{ is gap symbol and } x_i \text{ not} \\ \Pr\left(-_{d_A^x(i)-1} \sim y_i \mid X, Y, H\right) & \text{if } x_i \text{ is gap symbol and } y_i \text{ not} \end{cases}$$

The Posterior decoding (PD) finds the alignment A that maximizes the product of the posterior probabilities of its columns:

$$A = \arg\max_{A' \in Al(X,Y)} \prod_{0 \le i < |A'|} P(a'_i)$$

[2]Note that x and y cannot be both gap symbols.

where $Al(X, Y)$ denote the set of all alignments of sequences X and Y. Similarly we can define Marginalized posterior decoding (MPD): the marginalized posterior probability $P'(a_i)$ is defined as follows.

$$P'(a_i) = \begin{cases} P(a_i) & \text{if } x_i \text{ and } y_i \text{ are not gap symbols} \\ \sum_{0 \le j < |Y|} \Pr\left(x_i \sim -_j \mid X, Y, H\right) & \text{if } y_i \text{ is gap symbol and } x_i \text{ not} \\ \sum_{0 \le j < |X|} \Pr\left(-_j \sim y_i \mid X, Y, H\right) & \text{if } x_i \text{ is gap symbol and } y_i \text{ not} \end{cases}$$

The MPD finds an alignment A that maximizes the product of the marginalized posterior probability:

$$A = \arg \max_{A \in Al(X,Y)} \prod_{0 \le i < |A'|)} P'(a'_i)$$

The Posterior decoding and the Marginalized posterior decoding were used by Lunter *et al.* and both produced better alignments than alignments found by the Viterbi algorithm (more details in section 2.4.8). Once the posterior probabilities of all possible columns of an alignments are computed (in $O(|X||Y|k^2)$ time where k is the number of states of pHMM), we can find the alignment that maximizes the desired function in $O(|X||Y|)$ time by dynamic programming similar to the Needleman-Wunsch algorithm. The difference is that we use the posterior probabilities instead of the substitution scores and gap scores. The recurrence for PD is following:

$$M[-1, i] = 0, 0 \le i < m \tag{2.39}$$

$$M[i, -1] = 0, 0 < i < n \tag{2.40}$$

$$M[i, j] = \max \begin{cases} M[i-1, j-1] + P((X[i], Y[j])) \\ M[i, j-1] + P((-_i, Y[j])) \\ M[i-1, j] + P((X[i], -_j)) \end{cases} , 0 \le i < n, 0 \le j < m \tag{2.41}$$

The recurrence for the MPD is analogous and the rest of the algorithm (finding the alignment) is done exactly as in the Needleman-Wunsch algorithm (see Section 2.3.2). The time complexity of the PD and MPD is $O(|X||Y|k^2)$ [50].

2.4.5 Pair Hidden Markov Models with Gene Structure

In this section we describe several pair hidden Markov models (or generalized pair hidden Markov models) with gene structures incorporated into their topology. These models were used either to align coding DNA or proteins to a genome or to find genes. Every gene consists of two types of sequences: exons (which encode amino-acids), and introns, which are removed before translation. The total length of remaining exons have to be multiple of three; each triplet of nucleotides is called codon and it encodes one of 20 amino-acids. However, the length of individual exons is not necessary the multiple of three. The coding sequence of a gene begins with a start codon (sequence ATG) and the last codon of a gene is called stop codon (sequences TAG, TAA or TGA). An intron begins with 5' splice site (also called donor site), and ends with 3' splice site (or acceptor site). Acceptor and donor sites are sequence motifs of fixed length [49, 71].

We introduce several comparative gene finders. Comparative gene finders use evidence from two organisms to find genes. They use pHMM to simultaneously align and annotate two sequences. The advantage finding genes in two organisms simultaneously is that we can use the evidence from two related organisms to detect genes, that are in both organisms.

Meyer *et al. (2002)* developed comparative gene finder *DoubleScan*. Generally, DoubleScan has 54-state GpHMM with following substructures: substructure that emits exons, substructure that emits introns and substructure that generate intergenic regions. Each substructure has three copies in the model: one for emitting in both sequences, and one for emitting in one sequence.

Decoding method of DoubleScan is the Viterbi algorithm, with stepping stone algorithm (see Section 2.5.1). They use BLASTN [3] for computing initial seed of local alignments. They restrict the Viterbi algorithm to follow the alignments in the subset allowing tolerance 15 bases [56].

SLAM [2] is a comparative gene finder based on a generalized pair hidden Markov model [47] with some states being also high-order states (with dependence on previous emissions). It predicts gene structures for a pair of related eukaryotic organisms. SLAMs decoding method is the Viterbi algorithm. To reduce the running time of the VA, SLAM align sequences within 3 bases around alignment obtained by AVID global alignment tool [11]. Topology of the model can be found in figure 2.7.

TWAIN is comparative gene-finder based on GpHMM with intereresting decoding algorithm [54]: at first it independently annotates input sequences using gene-finder TIGRscan [53], finding signals (splice sites, start/stop codons, ...). Then creates parse graphs for each sequence: signals form vertices, which corresponds to a state of the GpHMM. Two signals are connected with an edge if one can follow other in a gene. The probability

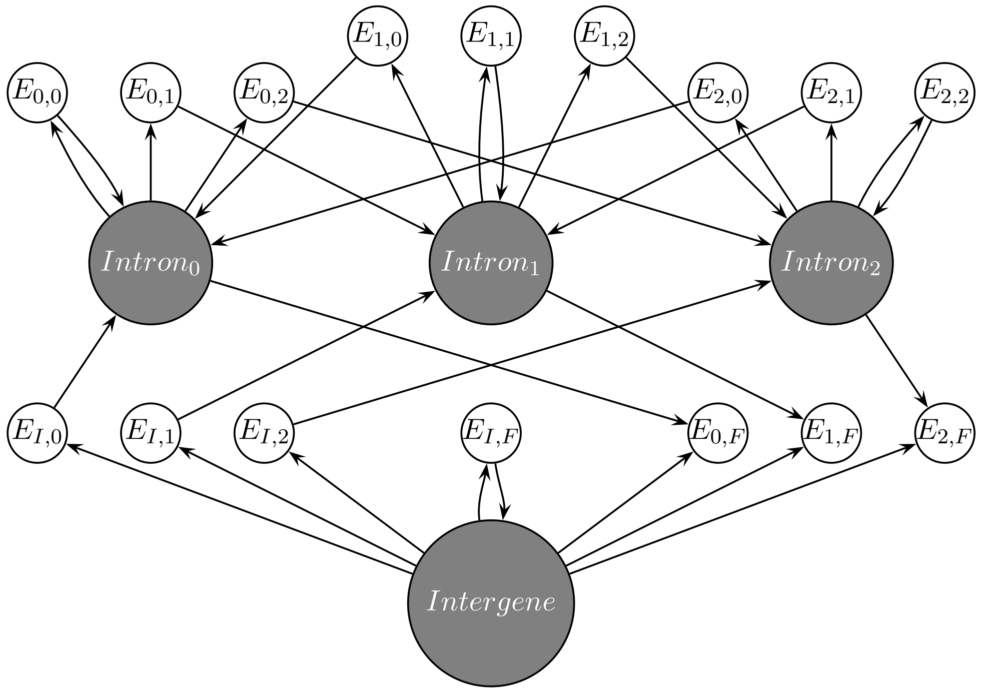


Figure 2.7: Topology of GpHMM used by SLAM (states for genes on reverse strand are omitted). Gray states has self-loops, which are omitted from the Figure. Emissions of shaded states are modeled by the basic three state pHMM. White states represent exons. Each has associated a duration distribution and emissions are modeled by three-state pHMM using 5-th order states that emit whole codons at once. Since introns can be inside a codon, the model contains an exon state for every possible interruption: $E_{i,j}, 0 \leq i, j < 3$ is an exon that begins with end of the interrupted codon of length $((3 - i) \mod 3)$ and ends with the start of a codon of length j. I stands for start codon and F stand for stop codon. $Intron_i$ models intron interrupting codon at the i-th position.

of an edge is determined by the probability of the most probable state path between corresponding two states in the TIGRscans HMM.

TWAIN then creates graph G by cartesian product of two parse graphs, omitting the vertices corresponding to pairs of different signals. Each node in the graph corresponds to a state of the GpHMM and a cell in the dynamic programming matrix of the Viterbi algorithm. Edges between cells correspond to an alignment generated by a single state from GpHMM. The Viterbi algorithm is computed only on cells corresponding to graph G which significantly reduces running time [54].

GeneWise predict genes by aligning a protein to similar gene structures in DNA [9]. It uses probabilistic transducers instead of pHMM. Both transducers and pHMM are probabilistic finite state machines, but probabilistic transducers transform one sequence into the other sequence. It is easy to compose transducers, while maintaining probabilistic interpretation of the resulting model. Composition of two transducers A and B is transducer C that that on the input sequence applies transducer A and then transducer B.

GeneWise model was created by composition of a gene prediction model S which translates genomic sequence to protein sequence and a protein homology model T which maps protein sequence to a homologous protein sequence.

Gene prediction model S consists of a single exon state which translates series of codons into amino-acids, and three submodels for modeling introns. Protein homology model T is a simple pHMM from figure 2.6 defined over protein alphabet [9].

The aim of *Pairagon* is to find local alignments of complementary DNA (cDNA) and genome [49], By aligning experimentally obtained cDNA sequences to the genome we are able to confirm intron and exon structures of genes. Pairagons HMM model consists of a simple pair HMM submodel, which aligns cDNA to DNA and a 5-state submodel for intron structures. The whole topology is in figure 2.8.

Decoding was done by the Viterbi algorithm. Runtime of the algorithm was improved by the stepping-stone algorithm described in Section 2.5.1 and memory requirements were improved using the Treeterbi algorithm [39], which is similar to the On-line Viterbi algorithm [66].

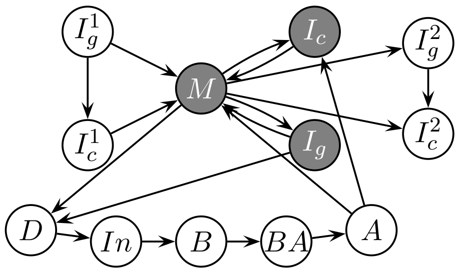


Figure 2.8: Topology of Pairagon GpHMM. All states except states D, B and A have a self-transition. Shaded states corresponds to exons: M emits aligned pairs of symbols, I_c is insertion in the cDNA and I_g is insertion in the genome. States I_c^1, I_g^1, I_c^2 and I_g^2 corresponds to unaligned parts of the DNA and cDNA in the beginning and the end of the sequences. States D, In, B, BA, A correspond to intron structure: Donor, Intron, Branch, Branch Acceptor, and Acceptor.

2.4.6 Non-Geometric Indel Models

In the simple pHMM described in figure 2.6, gap length has geometric distribution: the probability that a gap has length n is $e_{M,I}e_{I,I}^{n-1}(1-e_{I,I})$ (note that the probability that at particular position will be gap with length zero is $1-e_{M,I}$). The Viterbi algorithm is usually computed in log-space: instead of computing product of probabilities of events[3], we compute the sum of logarithms of those probabilities, because computation in log-space is numerically more stable. The Viterbi algorithm for the simple HMM will become the same as the Needleman-Wunsch algorithm. Gap penalty will be $\log(e_{M,I})+\log(1-e_{I,I})+(n-1)\log(e_{I,I})$. By setting $d=\log(e_{I,I})$ and $g=\log(e_{M,I})+\log(1-e_{I,I})-d$ we see that this is exactly affine gap penalty. Therefore we can say that affine gap penalties correspond to geometric distribution of indel lengths.

Using non-affine gap model can improve alignment quality. Problem with geometric distribution (or affine gap penalties) is that they are not realistic [18, 50]. Therefore some other distribution might be more appropriate, for example zeta distribution [18], or combination of several geometric distributions to approximate the distribution of gap length [29, 28].

GpHMM allow us to use arbitrary duration distributions. On the other hand, GpHMM are much slower to decode. One way of incorporating a different gap distribution into pair hidden Markov models without using their generalized version is to use several (for

[3]Event is emission or transition.

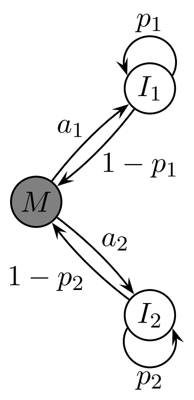


Figure 2.9: Shaded state M represents the match state and states I_1 and I_2 represents indel states in the same sequence. Indel states for the other sequence are omitted.

example two) indel states for every sequence. For example Lunter *et al. (2008)* used two component mixture models: instead of one indel state for every sequence they use two. They report that this improved quality of alignments. Similar approach is used in the multiple sequence aligner FASTA [10].

Modeling non-geometric distributions with several states can be problematic when used with the Viterbi decoding [68]. Set of states with the same emission distribution used for modeling non-geometric distribution is called gadget. We discuss an example of such a problem for the two component mixture model.

Let H be a simple pair hidden Markov model with two pairs of indel states. Let I_1 and I_2 be indel states that generate gaps in the first sequence connected with match state M as in the figure 2.9. Gaps in alignments (in the first sequence) generated by such a model have length distribution $d(n) = (a_1 p_1^{n-1}(1 - p_1) + a_2 p_2^{n-1}(1 - p_2)), n > 0$ and $d(0) = 1 - a_1 - a_2$, where n is the length of the gap, a_1 and a_2 are probabilities of entering state I_1 and I_2 respectively and p_1 and p_2 are probabilities of remaining in state I_1 and I_2 respectively. This is equivalent to the generalized pair hidden Markov model H' with one indel state for every sequence which has $d'(n) = d(n)/(1 - d(0))$ as its duration distribution (in generalized states we want to generate at least one gap. Zero-length gap should be modeled by the probabilities of incoming and outgoing transitions to the generalized state). Both models define the same distribution of alignments and running the Forward-Backward algorithm or the Forward algorithm will give the same results. However, alignments constructed by the Viterbi algorithm can be different for H and H'.

Problem is that in the generalized model the Viterbi algorithm gaps of length n have "score" $d(n)$ but in the non-generalized pair hidden Markov model it will be $m(n) = \max\{a_1 p_1^{n-1}(1-p_1), a_2 p_2^{n-1}(1-p_2)\}, n > 0$ and $m(0) = 1 - a_1 - a_2$. These two scores are different ($d(n)$ is always higher). It is possible that the Viterbi algorithm reconstruct different alignments. Therefore if we are using the Viterbi algorithm, we should either construct a gadget so that $m'(n)$ will be a better approximation of $d(n)$, or use a generalized pair hidden Markov model for the Viterbi algorithm.

2.4.7 Aligning Sequences with Variable Rate of Evolution

The rate of evolution (the expected number of substitution per position in sequence over some period of time) is not constant for the whole genome. It does not have to be constant even within one gene. FEAST is pairwise local alignment tool [36] that takes into account the variable evolution rate. The simple pHMM from figure 2.6 is optimized for one fixed rate of evolution. FEAST contain k such submodels, each trained for a different rate of evolution. Submodels are connected with a single silent state. Since FEAST is a local alignment tool, it also contains one additional submodel for generating an unaligned pair of sequences at both ends of the sequences.

To construct an alignment FEAST uses the Viterbi algorithm. Like many local aligners, FEAST uses a seeding heuristics to reduce computational complexity of finding local alignments. At first it uses six different space seeds to get candidate seed and then extends those seeds using x-drop heuristic [4]. The extension is done by an ungapped version of the Forward algorithm, in contrast with the Viterbi algorithm usually used for this purpose.

Estimation of parameters was done by expectation maximization approach (with Baum-Welsch or Viterbi training). They forced gap parameters to be the same in all submodels.

Different rates of evolution were also used in the whole genome aligner GRAPe [64]. GRAPes HMM consists of two submodels: one with fast evolution rate and two component geometric mixture model for indels and one with the lower evolution rate and geometric distribution of indel lengths. GRAPe uses the Posterior decoding as a decoding method.

2.4.8 Biases In Alignments

Lunter *et al. (2008)* conducted an extensive survey concerning biases in alignment. They considered three types of biases associated with gaps [22]. By *true alignment* we mean an alignment that corresponds to the actual evolution history. Since true alignments are unknown for real data we can simulate evolution on randomly generated sequences, this obtaining a dataset of "true" alignments. Lunter *et al.* studied following biases.

- *Gap wander* means, that a gap is in a different location that in the true alignment. Is is due to random short similarities around the borders of gaps that are indistinguishable from true homologies.
- *Gap attraction* occurs when two gaps are near each other. In such case, merging those gaps and introducing a few mismatches might lead to higher score.
- *Gap annihilation* occurs when there are two gaps of the same length, one in each sequence. Since indels are not so common, removing both gaps while introducing new mismatches might increase the score of an alignment.

Biases are ordered by their frequency from the most occurring to the least occurring [50]. Lunter *et al.* explore there problems with a series of simulations.

They measure the alignment quality by *sensitivity*, which is the ratio of the correctly predicted alignment columns to all homologous columns in the true alignment [50].

In the first experiment, authors use a simple model of evolution obtaining alignments with the expected sequence identity 0.705 with geometric gap model. The sequences were then realigned using the Viterbi algorithm with the same model as was used for simulation. Sensitivity was lowest for the columns near gaps (56%) and the sequence identity for columns near gaps was 85% which does not agree with the expected sequence identity 0.705%. Moving away from gaps the average sequence identity dropped to 0.68%. The increased sequence identity near gaps is due to gap wander bias. The gap attraction effect caused that the number of gaps that are near each other was lower than the expected value obtained from the used gap model.

They also run the Viterbi algorithm parametrized by a range of substitution and indel rates. The highest sensitivity was obtained for the parameters that were identical to the parameters used for simulation. However, even then the sensitivity was only 84% indicating that even if we have the right evolution models, some biases in the alignments are inevitable.

Lunter *et al.* also studied the effect of different decoding methods and different models on the alignment quality. They simulated evolution with parameters that are close to the parameters of human-mouse evolution. They simulated for example the large-scale variation of GC content, GC-content dependent indel and substitution rates and GC-independent local substitution rate variation [50]. They simulated $20,000$ homologous sequence pairs with average length of 700 nucleotides. They add flanking sequences of length 100 nucleotides to the generated sequences to simulate local alignments.

After simulation they realigned homologous sequences using the Viterbi algorithm (VA), the Posterior decoding (PD) and the Marginalized posterior decoding (MPD) with different models: the three state pHMM (H_S); H_S with two indel states for every sequence representing the two component geometric mixture gap model (H_M) and the full model with all parameters that were used for simulation (H_F).

They also introduce two additional measures of the alignment quality. The *false positive fraction (FPF)* is the proportion of the columns that are ungapped in the true alignment but wrongly aligned by an algorithm [50]. The *the nonhomologous fraction (NHF)* is the proportion of columns containing padding sequence among all columns aligned by an algorithm.

The use of the different models has little impact on the sensitivity of the constructed alignments. It is interesting that for the Viterbi algorithm the sensitivity was lower for the full H_3 model than for the simple model H_1. This might be explained by the multiple path problem. With other decoding algorithms the models H_2 and H_3 has slightly higher sensitivity then the H_1 model.

While the use of the "better" model does not significantly improve the quality of alignments, using the Posterior decoding and Marginalized Posterior decoding improved the sensitivity by approximately 2.5% regardless of the model. On the other hand the FPF and the NHF was increased with use of the PD and MPD. The sensitivity of the PD and the MPD were similar but the FPF was lower for the MPD than for PD.

The main outcome of this experiment is that proper decoding method can improve the alignment quality while the use of a simpler model doest not significantly reduce the alignment quality. However, Lunter *et al.* use in their simulations models only relatively simple models of the evolution.

2.5 Algorithmic Improvements

Needleman-Wunsch algorithm and decoding algorithm for HMMs and pHMMs uses dynamic programming. In this section we review several algorithmic improvements to these algorithms. Some of the techniques will not be universally applicable. With these techniques, alignment algorithms could be used with sequences much longer than several thousand bases.

2.5.1 Restricting Search Space

One commonly used technique for speedup (and decreasing memory requirements) of sequence alignment is to restrict the search space of dynamic programming. We compute alignment only in some parts of matrix M and we assume that omitted parts of matrix correspond to alignments with low score. These techniques are also applicable for pHMMs.

If the two sequences are quite similar, the optimal global alignment will not be too far from the main diagonal of matrix M. Therefore it is not necessary to compute parts of matrix M that are too far from the main diagonal [19] (distance from diagonal a is user-defined value or can be computed during alignment [33]). However, this method is not useful for local alignment or global alignment of distant sequences. Now we will discuss some more advanced techniques to restricting search space of dynamic programming.

Seeds

Seeds are a technique frequently used to reduce time complexity of local alignment. They were popularized by BLAST algorithm [3]. A *seed* is a short alignment which is likely to be a part of an alignment with high score. After a seed is found, it is extended with an extension algorithm to a local alignment.

Seeds that cannot be extended to high-scoring alignments are discarded. Such candidate seeds are called false positives. Alignments that were not found by our heuristics are called false negatives. It is important that heuristics used to find seeds has a small number of false negatives and a large number of true positives, otherwise many true high-scoring local alignments will not be found. On the other hand, high number of false positives implies longer running time.

The most traditional approach is to take as seeds all pairs of positions i and j, such that $X[i : i + \tau] = Y[j : j + \tau]$ for some constant τ. This approach is used in BLAST [3]. Various generalization were developed to improve trade off between speed and accuracy, such as seeds with mismatches [40], space seeds [52], vector seeds [13] or daughter seeds [21].

Extension of a seed to a full alignment is done in both directions, usually using equation 2.20 (equation is altered to reverse direction). Extension is stopped, when some criterion is reached. For example, BLAST introduced X-drop heuristics: extension stops if the score of an alignment is lower than the best score that was seen so far minus some user-defined constant [4].

Stepping-Stone Algorithm

Stepping-stone algorithm (SSA) [56, 49] is suitable for global alignment algorithms. The idea is to use good local alignments as anchors. An anchor is similar to a seed: it is a short alignment which we expect to be in the optimal global alignment. However, local alignment tools give local alignments that do not have to be consistent with each other. A set of local alignments is consistent if all local alignments can be together in one global alignment.

SSA chooses a consistent set of local alignments by a simple greedy method, always adding highest-scoring alignment consistent with those selected so far. Selected anchors will be extended to a global alignment. However, since local alignments may contain errors, SSA will relax them. If X_i and Y_j were aligned in some anchor, then X_i can be aligned to positions from $j - \tau$ to $j + \tau$ in the global alignment[4] for some user defined constant τ. Similarly, Y_j can be aligned to positions from $i - \tau$ to $i + \tau$. This technique is also called *banding*, and it is often used. We used this technique in chapter 4.

2.5.2 Reducing Memory Complexity

One general technique to reduce the memory requirements of dynamic programing is *checkpointing* [30].

In order to compute the i-th row of matrix M, we need only row $i - 1$. As mentioned in section 2.3.2, to compute the score of the best alignment we need to store only two consecutive rows. However, if we want to recover the optimal alignment, after we have computed its score, we need all rows of matrix M again, in decreasing order.

Checkpointing solves this problem by storing every k-th row of matrix M, $\lceil n/k \rceil$ rows in total. While back-tracing, we will remember an additional block B of consecutive k rows in interval $[ik, (i+1)k)$. We can compute such a block in $O(kn)$ time using the basic dynamic programming, starting from row number ik which is stored in memory. Overall, we recompute every block at most once, and therefore the time complexity will be $O(mn)$. If we set $k = \sqrt{n}$ then the memory complexity will be $O(m\sqrt{n})$. By using l recursive applications of this technique we can obtain algorithm with memory complexity $O(m\sqrt[l]{n})$.

Checkpointing can be used to decrease the memory complexity of the Viterbi algorithm with back-tracing procedure and the Posterior decoding to $O(\sqrt{n}m)$. Similarly, this technique can be used to reduce memory complexity of the algorithms for pHMMs to

[4]Or aligned to a gap in that region.

$O(nm\sqrt{n})$ where n is the length of the longer sequences and m is the number of states of HMM.

Even more reduction in memory requirements can be obtained using the Hirschberg algorithm [35]. The idea is following: if we want an alignment of sequences X and Y that has bases X_i and Y_j aligned, we have to do dynamic programming only in submatrices $M_1 = M[0 : i, 0 : j]$ and $M_2 = M[i : n-1, j : m-1]$. If $i = \lceil n/2 \rceil$ then the total number of cells in those matrices is roughly half of the number of cells in M. The Hirschberg algorithm incorporates a procedure to compute j such that $X_{\lceil n/2 \rceil}$ is aligned to Y_j in the optimal alignment. This procedure takes $O(nm)$ time and $O(n+m)$ memory (alignment algorithm that keeps only one row of M in memory). If X_i is aligned to a gap, it finds j such that X_i is aligned to gap that comes right after X_j. The algorithm then uses j to determine submatrices M_1, M_2 and finds alignments in submatrices recursively. The optimal alignment is a concatenation of optimal alignments in matrices M_1 and M_2.

Since total size of the two subproblems is always at most half of the size of the original problem, the running time of the algorithm will be roughly double of the standard dynamic programming. The Hirschberg algorithm keeps in memory only a constant number of rows of M, and therefore the memory requirements are $O(n+m)$. The Hirschberg algorithm reduces memory more than checkpointing, but checkpointing can be applied to a wider class of algorithms.

2.5.3 Exploiting Sequence Repetition

There are two similar techniques to accelerate sequence alignment. Both divides the dynamic programming matrix into blocks (square of rectangle submatrices). Each block has input and output cells. Input cells are in the left and bottom border of the submatrix and output cells are in the top and right border of the submatrix (if the computation is performed in top-right order). Each submatrix transforms the input cells into output cells (the computation of dynamic programming). It is possible to use either compression or pre-computation to compute alignment in time proportional to the total length of the borders of the blocks instead of the total size of the block (block of size $t \times u$ has border length $O(t+u)$ and size $O(tu)$).

The first such technique is called Four-Russian technique [5, 33] and can be used for computing edit distance or sequence alignment with some restriction on the scoring matrix. It divides the dynamic programming matrix into equally sized blocks and pre-computes the output cells for all possible input cells and all possible blocks (the differences between values of the input cells are small, so all possible inputs can be enumerated). By setting the block size to the $O(\log(n))$, the total running time of the algorithm is $O(n^2/log(n)^2)$ in the unit-cost model [33].

Instead of dividing matrix into equally sized submatrices, we can run LZ78 factorization [45] on both sequences. LZ78 factorization is compression technique which divide sequence into sequence of $O(n/log(n))$ segments, where each segment s has predecessor segment s_p, where $s = s_p c$, c is a single character. We can use these segments to construct blocks. Each of such blocks has three predecessors blocks that are obtained by using predecessor segments [20]. Crochemore *et al.* developed technique to compute the values of the output cells from the predecessor blocks and input cells in $O(b)$ time where b is the total border length of the block. It utilizes the $O(A + B)$ algorithm for computing row minima and maxima in totally a monotone matrix of size $A \times B$ [1]. The total time complexity of this algorithm is $O(n^2/\log n)$ time [20].

Similar idea was also used for accelerating dynamic programming for decoding and training of HMMs [69]. The computation of the Forward algorithm can be decomposed as an series of matrix multiplication. Using LZ78 factorization, we can divide this into multiplication of $n/\log(n)$, where multiplication of matrices in each block is done in $O(m^3)$ where m is the number of states of HMM. Using this, we can compute the Forward algorithm in $O(nm^3/log(n))$ time which lead to $O(\log(n)/m)$ speed-up [69]. Similarly, it is possible to use this technique to the Viterbi algorithm. In this case we replace summation with maximum in the matrix multiplication operation [69].

Chapter 3

Two Stage Decoding of HMMs

In this chapter we will study several decoding techniques, which are aimed to reduce systematic errors in sequence annotation. We will illustrate our approach using the HIV recombination prediction domain, discussed in Section 2.2.3. Here, the goal is to determine which parts of a recombinant virus sequence originate in which subtype. For this task, we have previously developed the HERD decoding method [58, 60] (Section 2.2.3) which performed better than the Viterbi algorithm. However, in some cases the resulting annotation contains some systematic errors, for example frequent switching between different subtypes. One such example can be found in Figure 3.1. To mitigate such effects, one can use a two-stage decoding strategy. First we search for the most probable footprint F. The footprint is what is left from labeling after keeping only one label from each group of consecutive identical labels. Formally it was defined in section 2.2.4 by definition 12. In the second stage we find the annotation with footprint F that maximizes the expected reward (or some other criterion). The second stage is forced to keep the structure of footprint, and therefore rapid changing of types as in Figure 3.1 is not possible, unless the footprint contains a large number of recombinations.

More generally, in the first stage we search for the best guide, which is a general constraint on the allowed annotation, for example in the form of footprint or a set of labels. In the second stage, we find the optimal annotation constrained by the selected guide. Example of such algorithm is the most probable ball problem [16, 67] discussed in Section 2.2.4. In this this algorithm, the two-stage approach was motivated by computational complexity, since optimizing the border shift distance is NP-hard [16]. Our motivation to use the two-stage algorithms is to improve their accuracy and decrease the amount of systematic errors.

In this chapter, we will focus mainly on the computational complexity of the first

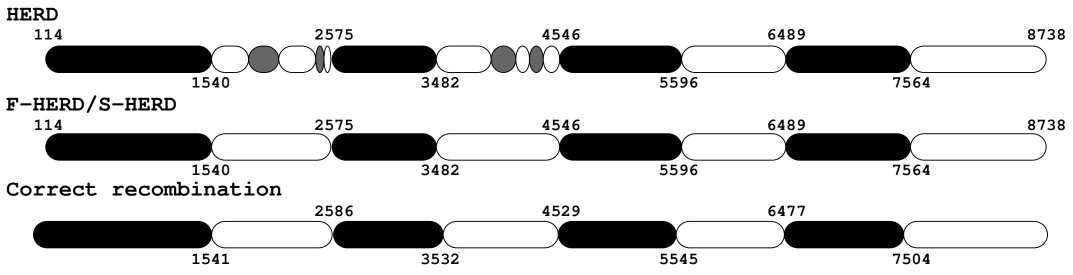


Figure 3.1: Each row corresponds to one annotation of the same HIV recombinant, colors represent virus subtypes. Unaltered HERD algorithm predicts a wrong subtype and also contains too many segments. But restricting to the specified footprint (F-HERD) or to the specified set (S-HERD) provides an almost correct annotation without these errors.

step: finding the most probable guide. Complexity of the second step, namely finding the annotation with highest expected gain without guides we studied in [58, 60]. Footprint and restriction guides can be easily incorporated to traditional decoding methods, but set guide will likely to lead to NP-hard problems. At first we give a formal definition of the two-stage algorithm. Then we show an experiment illustrating effects of the two-stage decoding on the HIV recombination detection problem. Further in this chapter we prove that finding the most probable guide is for certain guide functions NP-hard.

3.1 Formal Definition

In this section we will define two-stage decoding algorithms, and the generalization of the footprint function from the first stage, the guide function. Additionally to the footprint, we define a different guide function called annotation set. A footprint was already defined in definition 12, and we denote the footprint of annotation Λ as $F(\Lambda)$.

Note. The following definitions could also use state paths instead of annotations. However, annotations are more general, because by setting λ to the identity function, all annotations become state paths.

Definition 18 (Annotation set). *Let $\pi = \pi_1\pi_2\dots\pi_n$ be a state path and λ be coloring function as in definition 5. The* Annotation set *of labeling Λ is the set of labels visited on the state path:*

$$S(\pi) = \{\lambda(\pi_1), \lambda(\pi_2) \dots \lambda(\pi_n)\}$$

Annotation set of an annotation $\Lambda = \Lambda_1\Lambda_2 \dots \Lambda_n$ is the set

$$S(\Lambda) = \{\Lambda_1, \Lambda_2, \dots \Lambda_n\}$$

A state set *is an annotation set when the labeling function is the identity function.*

Two-stage decoding system can be described in Highest Expected Gain framework introduced in section 2.2.1. At first, we define guide and guide function.

Definition 19 (Guide function, guide relation, guide)**.** *Let H be an HMM, and f be an annotation gain function. Let L be the set of all annotations, W be an arbitrary set. We call any function $R : L \to W$ a* guide function *and $R(\Lambda)$ is the* guide *of annotation Λ.*

A guide relation *is a relation $\hat{R} \subseteq L \times W$ with the property that for all $\Lambda \in L$, $(\Lambda, R(\Lambda)) \in \hat{R}$.*

We will define the probability of guide r given sequence X:

$$\Pr(r \mid X, H) = \sum_{\Lambda \in L, R(\Lambda) = r} \Pr(\Lambda \mid X, H) \tag{3.1}$$

A natural way of defining guide relation is the following: $(\Lambda, r) \in \hat{R}$ if and only if $R(\Lambda) = R$. However, in some cases it might be useful to relax the restriction provided by the guide.

Note that we can see a guide as a generalization of an annotation. However, our use of guides is quite different from our use of annotations. While annotations are the final product, purpose of guides is to capture some important properties of the input, that can be subsequently used to find the correct annotation. Finally, we can formally define the two-stage decoding:

Definition 20 (Two-stage decoding)**.** *Let H be an HMM, f be an annotation gain function, R be a guide function and W be a set of all possible guides. Let X be a sequence of interest and L be the set of all annotations of X. In the* two-stage decoding *we search for annotation Λ, defined as:*

$$\Lambda = \arg \max_{\Lambda' \in L, (\Lambda', r) \in \hat{R}} E_{\Lambda_X \mid X, H}[f(\Lambda_X, \Lambda')] \tag{3.2}$$

where guide r is defined as

$$r = \arg\max_{r' \in W} \Pr(r' \mid H, X) \tag{3.3}$$

Finding the most probable guide can be decoupled from finding the annotation with highest expected reward. Implementing search with guides is straightforward, and we will discuss it later.

Note that we could use a different method to select guide r. While it is possible to use gain functions for defining optimization criteria to select guide (and altering the equation 3.3), we did not study such variations.

3.2 Experiments

In this section we apply the two-stage decoding methods on the HIV recombination detection problem. Our goal is to illustrate the effect and usefulness of the two-stage decoding algorithms, not to compare it with the other methods for recombination detection. We will describe the HIV recombination detection domain, two two-stage decoding algorithm and show experiment on the artificial dataset of recombinant HIV sequences.

In section 2.2.3 we described HERD decoding. It was used with jumping HMMs [65] for detecting recombinations in HIV virus. However, using this criteria lead to two artefacts:

- Producing annotations with many short recombinations.
- Producing recombinations of wrong virus subtype.

Example of both type of errors are illustrated in Figure 3.1.

3.2.1 Application Domain

We use the problem of recombination detection in HIV virus for the demonstration of usefulness of two-stage decoding. This section contains background information about data used in the experiment.

The genome of the HIV virus is an RNA sequence roughly 9000 bases long. Mutation rate of the HIV virus is high [65], and additionally HIV virus is categorized into several *subtypes* [62] based on sequence similarity. Phylogenetic tree of subtypes is show in figure 3.2. Subtypes A and F are further divided into *sub-subtypes* A1, A2 and F1, F2. Due to high sequence similarity, subtypes B and D are sometimes recognized as sub-subtypes of subtype BD [62].

Additionally to high mutation rate, *recombination* occurs in evolution of HIV virus [62]. Recombination of two source viruses v_1 and v_2 is a mosaic virus which contains parts from both v_1 and v_2. The relative position of individual parts is retained, so each region of

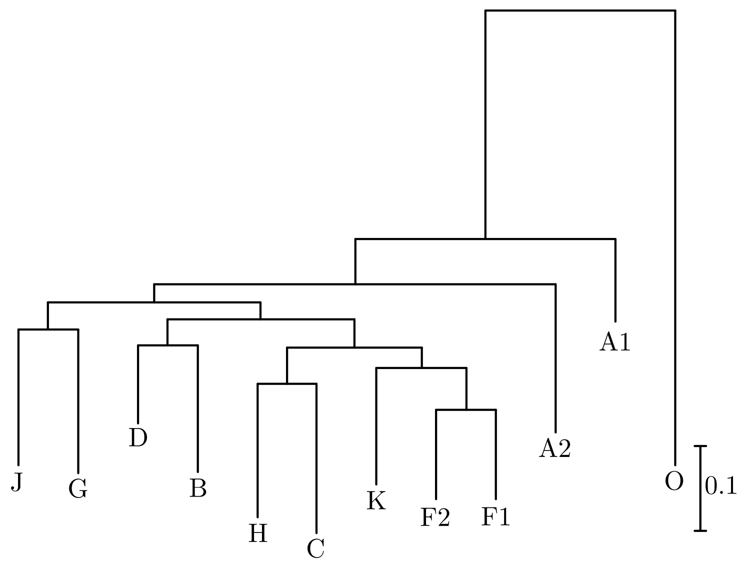


Figure 3.2: Phylogenetic tree of the HIV-M1 group of genomes with outgroup sequences from the of HIV-O group generated by phyml [32] from the database distributed with jpHMM [65]. Figure was originally used in [60]. The lengths of the branches correspond to evolution distances.

the recombinant virus originates from the corresponding region in the source virus. There can be multiple recombination points within the same virus, and since recombination can occur between two recombinants, there can be multiple source subtypes or sub-subtypes in one recombinant.

Given a recombinant sequence X, the goal of recombination detection is to label each base x of sequence X with the subtype from which x originates. The input sequence does not have to contain recombination. In such case the annotation of the sequence contain only single subtype. The set of annotation symbols therefore contains all subtypes/sub-subtypes of the HIV virus.

3.2.2 Model

In our experiments, we use jumping HMM (jpHMM) developed for recombination detection by *Schultz et al. (2006).* The main building block of the jpHMM are profile Hidden Markov Models (profile HMMs).

Profile HMMs are HMMs with a special topology used for matching certain motifs [22]. Profile HMM of length n consists of match states $M_1, M_2, \ldots, M_n$, insert states $I_0, I_1, \ldots, I_n$, silent delete states $D_1, D_2, \ldots, D_n$ and silent init state and silent final state.

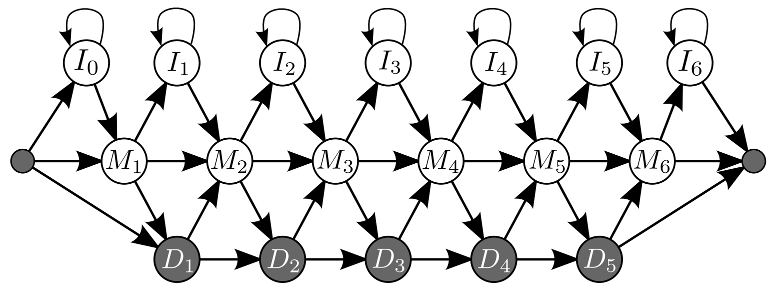


Figure 3.3: Example of a profile HMM of length 6. Shaded states are silent. The leftmost shaded state is the init state, the rightmost shaded state is the final state. Figure is from [60].

All states are connected by transitions as in the figure 3.3. A motif is represented by the chain of match states, each with its own emission distribution. Insert states model (short) insertions between positions of the motif. Delete states allow to skip some parts of the motif. A profile HMM can be obtained from an alignment of similar sequences; match states then represent columns of the alignment. Not all of the columns correspond to match states; columns that contains gaps above some threshold are usually omitted [22].

Jumping HMMs are built upon profile HMMs [65]. We start with an alignment of all HIV sequences and construct a profile HMM for each subtype or sub-subtype (it depends on whether we want to distinguish between sub-subtypes). We then add global init state and global final state to connect profiles in parallel. Column numbers of each match, insert, and delete states are preserved, so we know the corresponding column for all submodels. Transitions between different profiles are added to model recombinations: These "jump" transitions always end in the state that corresponds to the nearest column of the original alignment following their source column. Jump from column c_1 in subtype A to column $c_2 > c_1$ in subtype B is allowed if and only if there are no states in the profiles of A or B corresponding to columns between c_1 and c_2. A simplified topology of a jpHMM is in figure 3.4. More details can be found in [65].

3.2.3 Algorithms

In this section we describe the HERD algorithm and how to modify the Viterbi algorithm and the HERD to use footprint and the annotation set as the guides. We denote HERDs gain function, described in section 2.2.3, as f_{HERD} (note that the gain function of the Viterbi algorithm is f_V). Details of the algorithm that optimizes this gain function can

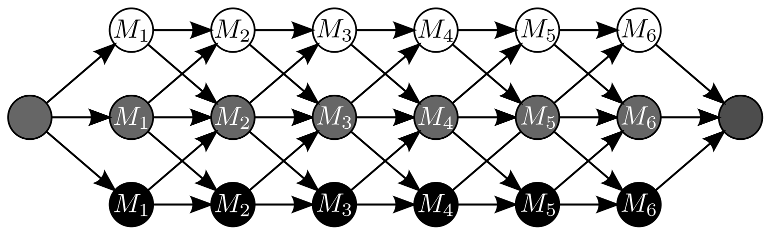


Figure 3.4: The simplified structure of a jpHMM. Insert and delete states are omitted from the figure for readability. Picture is taken from our previous work [60]

be found in [58, 60]. We use the footprint and the annotation set as a guide functions. In case of footprint, we use the natural way to define guide relation, however in case of the annotation set we use guide relation $\hat{R}_S$ defined as follows:

$$(\Lambda, r) \in \hat{R}_S \Leftrightarrow R(\Lambda) \subseteq r$$

The reason for using this guide relation is that it allows us to use simpler and faster optimization algorithm. This is not a problem because HERD have problem with detecting additional wrong subtypes, not missing the correct one. In our tests it never occurs that the resulting annotation had different set than the most probable set from the first stage of the algorithm. In the following text, we will use the term *segment*, which is the maximal single-colored consecutive subsequence of annotation.

The original HERD algorithm can be decomposed into two phases. In the first phase, it constructs an *annotation graph*. An annotation graph is a graph in which each path from the start vertex to the end vertex represents an annotation. The length of such a path equals to the expected gain of the corresponding annotation. The longest path in the annotation graph therefore corresponds to the annotation with the highest expected gain. Annotation graph is directed and acyclic and its structure is described in Figure 3.5. In the second phase we find the longest path in this graph and convert it to the annotation. Since the annotation graph is directed and acyclic, it is possible to find such a path in polynomial time [60].

To add guides, we need to change the second phase of the algorithm. We will be searching for an annotation restricted to the guide with the highest expected gain. If the guide function is an annotation set, we simply remove vertices with colors that are not in annotation set from the annotation graph and use the original algorithm described in [60]. Since trimming the annotation graph can be done trivially in linear time, this does not change the computational complexity of the HERD algorithm.

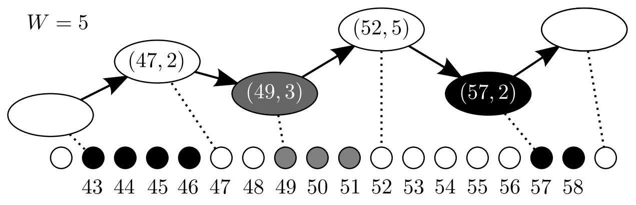


Figure 3.5: Part of a path in an annotation graph with the corresponding annotation. A node with color c and caption (b, w) represents the vertex (b, c, w) of the graph. The dashed lines connect each vertex to its corresponding segment. (b, c, w) corresponds to the beginning of c-colored annotation segment starting at position b. End of the segment is determined by the following vertex in the path; therefore each edge in the graph corresponds to a possible annotation segment. The length of an edge is the expected gain of the corresponding annotation segment. If w is less than maximal window width W, which is a parameter of the HERD algorithm, the edges from such vertex are allowed only to vertices of the form $(b + w, *, *)$ where $*$ can be any valid value. If $w = W$, then outgoing edges are of the form $(b + W + k, *, *)$ where $k \le 0$. Figure is from our previous work [60]

Using the footprint as a guide requires more changes to the algorithm then just removing vertices with colors that are not in the footprint; we also alter the dynamic programming formula for computing the longest path. We order vertices of the annotation graph topologically from start to end vertex into sequence $v_0, v_2, \ldots v_{n-1}$ where n is the number of vertices in the annotation graph, v_0 is the init vertex and v_{n-1} is the end vertex. This will be the order in which the dynamic programming equations will be computed. Let $f = f_0 f_2 \ldots f_{k-1}$ be the footprint we use as a guide and $V[i, j]$ be the length of the longest path ending in v_i that has footprint $f[: j]$. Therefore the length of the longest path with footprint f is $V[n-1, k]$. If E is the set of all edges and $C(v_i, v_j)$ is the weight of an edge (v_i, v_j), then $V[i, j]$ can be computed by following equations:

$$V[0, 0] = 0 \tag{3.4}$$

$$V[0, j] = -\infty, 0 < j \le k \tag{3.5}$$

$$V[i, j] = \max_{i', (v_{i'}, v_i) \in E} \begin{cases} C(v_i', v_i) + V[i', j] & \text{if } \lambda(v_i) = \lambda(v_{i'}) \\ C(v_i', v_i) + V[i', j-1] & \text{if } \lambda(v_i) \neq \lambda(v_{i'}) \text{ and } \lambda(v_{i'}) = f_{j-1} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Note that since we have ordered vertices topologically, to compute $V[i, j]$ we use only values of $V[i', j']$ where $i' < i, j' \le j$. Therefore we can compute V in increasing order of

i. The longest path can be computed using back-tracing as in section 2.3.2. The time and memory complexity of this part of the algorithm increased by the factor of k. Therefore the time complexity increases to $O(mWC|E|+mkC^2W^2)$ where C is the number of colors (labels), W is the window size, m is the length of the sequence and $|E|$ is the number of transitions in the HMM. Space requirements increased to $O(\sqrt{m}Cn + WCnk + mkC^2)$. However, in practice guide contains only a small number of colors and as a result, graph is much smaller then in the unguided version. Implementation details of other parts of an algorithm can be found in [60].

As we will show in this chapter, finding the most probable set and the most probable footprint are NP-hard and NP-complete problems. Therefore we used heuristic algorithm implemented in the from program balls [16], which is also described in section 2.2.4. Note that the balls algorithm estimates the footprint by sampling state paths and selecting the most frequent footprint of sampled paths. For using the set as a guide, we have chosen the set of labels used in the footprint guide, since these sets were almost always correct.

Altering the Viterbi algorithm to use set as an restriction is as simple as for the HERD; we only need to to ignore states that are not in the guide set. We can do it by removing state that are not in the guide set from the HMM without normalizing the transition probabilities.

Optimizing the Viterbi algorithm with footprint f as an guide can be done by computing following equations:

$$V[0,v,0] = I_v e_{v,X_0}, \lambda(v) = f_0 \tag{3.7}$$

$$V[0,v,0] = 0, \lambda(v) \neq f_0 \tag{3.8}$$

$$V[0,v,l] = 0, l > 0 \tag{3.9}$$

$$V[i,v,l] = \max_{u\in V} \begin{cases} V[i-1,u,l]a_{u,v}e_{v,X_i} & \text{if } \lambda(v) = \lambda(u) \\ V[i-1,u,l-1]a_{u,v}e_{v,X_i} & \text{if } \lambda(u) = f_{l-1} \text{ and } \lambda(v) = f_l \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

(3.11)

where $v \in V, 0 < i < n$. Computation of back-links $B[i,v,l]$ are analogous to the computation of $V[i,v,l]$ as in the Viterbi algorithm. Note that these equations are the original Viterbi equations from Section 2.1.3 with additional dimension: the position l in the footprint. The time complexity of this algorithm is $O(nm^2k)$, where k is the length of the footprint. This algorithm is k times slower than the Viterbi algorithm.

3.2.4 Data and Results

We created an artificial dataset of recombinant HIV sequences and run HERD and the Viterbi algorithm (VA) with two guide functions: annotation set and footprint. HERD was run with default parameters, $\gamma = 0.2, W = 10$. The model was trained using jpHMM program on the alignment of HIV sequences distributed with jpHMM [65]. We removed 10% of sequences from this alignment before training data and used them for evaluation. We refer to the alignment distributed with jpHMM as the *source alignment*.

Artificial testing sequences were generated by alternating segments of two real sequences from different subtypes. These alternating segments were selected from an alignment of original sequences. We did experiments with two types of segment lengths: either $200 - 300$ (short) or $950 - 1050$ (long) with additional $0 - 750$ and $0 - 500$ bases for the first and for the last segment respectively. The Length of each segment was drawn from the uniform distribution.

Apart from two different recombination length, we also evaluated algorithm of recombinants of sequences from same subtypes but different sub-subtypes, which gave us 4 different data sets. For all datasets we generated 150 recombinants of two sequences. For subtypes we use subtypes A, BD, C, F, G. For sub-subtypes we have used sub-subtype pairs A1-A2, B-D and F1-F2.

To measure accuracy of the algorithms and amount of systematic errors, we use the following metrics.

1. **%id**: Percentage of the bases with correctly predicted label.
2. **Segment specificity**: Percentage of the number of correctly predicted segments out of the number of all predicted segments. Segment is correctly predicted if its boundaries are within 10 bases of the correct boundaries.
3. **Segment sensitivity**: Percentage of the number of correctly predicted segments out of the number of segments in the correct annotation.
4. **%correct footprints**: Percentage of correctly predicted footprints.
5. **Footprint distance**: Edit distance of predicted footprint and the correct footprint normalized by the length of the correct footprint (averaged over all samples).
6. **Set specificity**: Percentage of the number of correctly predicted label types out of the total size of predicted label set.
7. **Set sensitivity**: Percentage of the number of correctly predicted label types out of the total size of correct label set.

Aim of measures $1-3$ is to quantify the performance of the algorithm in terms of ability to correctly predict the correct annotation. Measures $4-7$ quantify the amount of previously mentioned systematic errors in the predicted annotation.

We use three versions of HERD and VA in our experiments: original algorithm, version with annotation set as a guide denoted by S-HERD (S-Viterbi) and version using footprint as a guide denoted by F-HERD (F-Viterbi). The results of experiments are in the table 3.1.

In general, we expect short recombination length to be harder than long recombination length and distinguishing between sub-subtypes to be harder than distinguishing between subtypes. Both of these expectations were observed in the results. In all experiments on recombinants with long recombination length, footprint algorithms outperformed their non-guided versions in all metrics, especially with the metrics $4-7$: for HERD, the fraction of correct footprint was increased from 76.67% to 100% in case of subtypes and from 21.33% to 98% in case of sub-subtypes (similarly for VA). However on recombinants with short recombination length, used footprints were wrong most of the time and in the sub-subtypes dataset, all of the algorithms predict zero correct footprints. In fact, accuracy of F-HERD and F-Viterbi decreased in all metrics except for annotation set specificity and sensitivity. This and increase in footprint distance suggest the problem might be cause by heuristic methods for finding the best footprint and that improvements in these methods are needed; to check this, we computed the probability of predicted and correct footprints. For short recombination length and sub-subtypes, 88.67% of predicted footprints have lower probability than the correct footprint, 2.67% have the same probability and 8.67% have higher probability than the correct footprint. For the same recombination length and subtypes, 60.67% of the predicted footprints have lower probability and 39.33% have higher probability than the correct footprint. For the long recombination lengths, footprints which were not correctly predicted have higher probability than the correct footprint.

Methods with set as an guide performs slightly worse than footprint based methods on the data sets with long recombination lengths, but the metrics were still significantly better than unguided HERD algorithm. Additionally, using set as an guide for short recombination length was not performing worse than original algorithm; accuracy was similar.

	%id	**segment sp.**	**segment sn.**	**%correct footprints**	**footprint distance**	**set sp.**	**set sn.**
subtypes, short recombination length							
HERD	88.09	65.59	**67.71**	30.67	0.0728	56.67	**100.0**
S-HERD	**88.55**	**68.17**	**67.71**	**37.33**	**0.0699**	**99.33**	**100.0**
F-HERD	84.71	65.02	58.19	2.67	0.1653	**99.33**	**100.0**
Viterbi	86.33	62.73	61.31	19.33	0.1097	68.67	**100.0**
S-Viterbi	**87.01**	**64.33**	**62.12**	**26.0**	**0.100.08**	**99.33**	**100.0**
F-Viterbi	84.75	61.99	55.57	2.67	0.1653	**99.33**	**100.0**
sub-subtypes, short recombination length							
HERD	**72.72**	32.79	**27.56**	**0.0**	**0.3416**	19.33	**100.0**
S-HERD	72.33	**35.77**	26.59	**0.0**	0.3937	**100.0**	**100.0**
F-HERD	68.05	30.65	16.97	**0.0**	0.4777	**100.0**	**100.0**
Viterbi	69.34	30.54	21.04	**0.0**	0.4729	32.0	99.33
S-Viterbi	**70.21**	**32.28**	**21.37**	**0.0**	**0.4715**	**100.0**	99.33
F-Viterbi	67.91	26.18	14.67	**0.0**	0.4777	**100.0**	**100.0**
subtypes, long recombination length							
HERD	93.86	50.20	61.82	76.67	0.0579	77.33	**100.0**
S-HERD	94.12	53.46	63.68	98.67	0.0030	**100.0**	**100.0**
F-HERD	**94.19**	**53.57**	**63.83**	**100.0**	**0.0**	**100.0**	**100.0**
Viterbi	93.82	49.09	59.40	86.67	0.0230	86.67	**100.0**
S-Viterbi	93.96	50.24	60.10	98.0	0.0046	**100.0.0**	**100.0**
F-Viterbi	**93.99**	**50.61**	**60.33**	**100.0**	**0.0**	**100.0.0**	**100.0**
sub-subtypes, long recombination length							
HERD	84.10	19.87	25.63	21.33	0.4089	29.33	**100.0**
S-HERD	83.90	24.87	28.55	47.97	0.1026	98.65	**100.0**
F-HERD	**85.03**	**26.65**	**30.24**	**98.0**	**0.0038**	**98.67**	**100.0**
Viterbi	84.75	22.46	26.18	40.0	0.1520	56.67	**100.0**
S-Viterbi	85.23	**24.61**	**27.70**	64.0	0.0654	**98.67**	**100.0**
F-Viterbi	**85.39**	24.57	**27.70**	**98.0**	**0.0038**	**98.67**	**100.0**

Table 3.1: Results of experiments on four data sets. Note that larger values are better for all columns except *footprint distance* column where a lower value is better. The best value out of each column is shown in bold.

3.3 Computational Complexity Problems

In the previous experiments, we demonstrated that the two-stage algorithm can be a useful technique to improve HMM decoding algorithm. In this and following sections we will discuss theoretical aspects of defining guides and finding the most probable guides. In particular, we show that the following problems are NP-hard: *the most probable set*, *the most probable footprint*, and *the most probable restriction*. The last problem is variant of the most probable set. In this section, we define these problems and state the main results. The following sections then discuss individual problems and prove the results.

Definition 21 (The most probable set problem)**.** *Given an HMM H, sequence X of length n and a number $p \in [0,1]$, decide if there exists a set of states S such that $\Pr(S(\pi) = S, X \mid H) \geq p$.*

Theorem 3. *The most probable set problem is NP-hard.*

In the most probable set problem, we consider only paths that use all of the states from the annotation set. As in the experiments, it is more natural to include also paths that use only a subsets of annotation set. Therefore we might be interested in maximizing

$$\Pr(S(\pi) \subseteq S, X \mid H) = \sum_{\pi' \subseteq \pi} \Pr(S(\pi') = S, X \mid H)$$

However, this probability is trivially maximized by the set of all labels. To get a meaningful problem definition, we can restrict the size of the annotation set to be l:

Definition 22 (The most probable restriction problem)**.** *Given an HMM H, sequence X, integer l and number $p \in [0,1]$, decide if there exists a subset of states S of size l, such that $\Pr(S(\pi) \subseteq S, X \mid H) \geq p$.*

Theorem 4. *The most probable restriction problem is NP-complete.*

Finally, we also show that the most probable footprint problem is NP-complete.

Definition 23 (The most probable footprint problem)**.** *Given an HMM H, sequence X of length n and a number $p \in [0,1]$, decide if there exists a footprint F such that $\Pr(f(\pi) = F, X \mid H) \geq p$.*

Theorem 5. *The most probable footprint problem is NP-complete even if a labeling function is the identity function.*

Note that if we allow labeling function to be arbitrary, we can reduce the most probable annotation problem to the most probable footprint problem, we only need to make sure that it is not possible to have two consecutive states with same label in possible state path. It can be done by adding states with new label in the middle of the transitions, that generates new symbol and adding such symbols between symbols of the input sequence. In such case, the most probable annotation and the most probable footprint will be identical.

First we show combinatorial proof of theorem 3. Then we show proof of theorem 4. Finally we show proof of theorem 5 by showing 8-state HMM for which is NP hard to find the most probable footprint, then we extend this proof to theorem prove theorems 3 and 4.

3.4 The Most Probable Set Problem

In this section, we prove NP-hardness of the most probable set problem, theorem 3.

We will use a reduction from the maximum clique problem [26]. Given a graph $G = (V, E)$ and a clique size k, we first choose a suitable threshold $k' \geq k$, as detailed below, and construct a graph $G' = (V', E')$ such that G' has a clique of size k' if and only if G has a clique of size k. This is achieved simply by adding $k' - k$ new vertices and connecting each of the new vertices to all other vertices in V'. As long as $k' - k$ is not too large, this transformation can be done in polynomial time.

In the next step, we use G' and k' to construct an HMM $H_{G'}$, an input sequence and a probability threshold. Every state set in $H_{G'}$ with probability above threshold for the given sequence corresponds to a clique of size k' in graph G'. We will use the following straightforward way of converting a graph to an HMM.

Definition 24. *Let $G = (V, E)$ be an undirected graph (without self-loops). Then the* graph HMM *H_G is defined as follows:*

- *Its set of states is $V \cup \{\psi\}$, where $\psi \notin V$ is a new state called the error state.*
- *Its emission alphabet is $\{0, 1\}$.*
- *Each state $v \in V$ has initial probability $I_v = 1/|V|$, the error state has initial probability $I_\psi = 0$.*
- *Each state $v \in V$ emits 0 with probability 1, the error state emits 1 with probability 1.*

- *Transitions with non-zero probability between states $u, v \in V$ correspond to edges in E:*

$$a_{u,v} = \begin{cases} \frac{1}{|V|} & \{u,v\} \in E \\ 0 & \textit{otherwise} \end{cases}$$

- $a_{u,\psi} = 1 - \sum_{v \in V} a_{u,v}$ *and* $a_{\psi,u} = 0$ *for all* $u \in V$*. The error state has a self-transition with probability 1:* $a_{\psi,\psi} = 1$*.*

The purpose of error state ψ is to level the probabilities of admissible state paths without ψ to the same probability. Since ψ is the only state that generates 1, only sequences of form $X = 0^n$ can be generated by admissible state paths without the error state. Emission probabilities on such strings are 1, initial probability and all transitions are equal to $|V|^{-1}$, and therefore $\Pr(\pi, X = 0^n \mid H_G, n) = |V|^{-n}$. Each such path corresponds to walk in graph G. Each set of states in H_G naturally corresponds to a subset of vertices. The probability of the set is proportional to the number of corresponding walks in the graph G. Therefore we will count the number of special walks in induced subgraphs of G.

Definition 25. *Let $G = (V, E)$ be a graph and $S \subseteq V$ be an arbitrary set of its vertices. Then walk $w = v_1 v_2 \dots v_n, (v_i \in V)$ of length $n - 1$* covers *set S, if and only if $\{v_1, v_2, \dots, v_n \mid 1 \leq i \leq n\} = S$. We say that walk w covers graph G if w covers the whole set of vertices V.*

In other words, a walk covers set S if it uses only vertices from S and each vertex from S is used at least once.

Let $Y(n, G)$ be the number of graph covering walks of length $n - 1$. Since a walk of length $n - 1$ contains n vertices, $Y(n, G) = 0$ if $n < |V|$. We consider a special case of this formula for complete graphs: let K_k be complete graph with k vertices, then $D(n, k) = Y(n, K_k)$. The following claim clearly holds:

Lemma 1. *If G is a graph with k vertices and $n \geq k$, then $Y(n, G) \leq D(n, k)$ with equality only for $G = K_k$.*

In our reduction, we use HMM $H = H_{G'}$ and $X = 0^n$ for a suitable choice of n and k' discussed below. We will set threshold p to $D(n, k')/|V|^n$. Clearly, if the input graph G has a clique S of size k, graph G' has a clique S' of size k' and there are $D(n, k')$ walks of length $n - 1$ that cover S'. Each such walk corresponds to one state path, and therefore the probability of the set of states S' is exactly p.

n/k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	$\mathbf{M_n}$
0	**1**									0
1		**1**								1
2			**2**							2
3			2	**6**						3
4			2	18	**24**					4
5			2	42	**144**	120				4
6			2	90	600	**1200**	720			5
7			2	186	2160	7800	**10800**	5040		6
8			2	378	7224	42000	100800	**105840**	40320	7
9			2	762	23184	204120	756000	**1340640**	1128960	7
10			2	1530	72600	932400	5004720	13335840	**18627840**	8
$\mathbf{n_k}$	0	1	2	3	4	6	7	8	10	
$\mathbf{M_{n_k}}$	0	1	2	3	4	5	6	7	8	

Table 3.2: Values of $D(n,k)$, n_k, M_n, and M_{n_k} for small values of n and k. Empty cells contain zeros.

In order to prove the opposite implication "if there exists a set of states with probability p then there is clique of size k in G", we need suitable choices of n and k'. Table 3.2 shows values of $D(n,k)$ for small values of n and k. For a fixed length of walk n, the number of walks in K_k initially grows with increasing k, as we have more choices which vertex to use next. However, when k approaches n, $D(n,k)$ starts to decrease, because the walks are more constrained by the requirement to cover every vertex. We are particularly interested in the value of k where $D(n,k)$ achieves the maximum value for a fixed n. In particular, we use the following notation:

$$M_n = \min\left\{k; D(n,k) = \max_{0\le k'\le n} D(n,k')\right\}$$

Note that if there are multiple values of k achieving maximum, we take the smallest one as M_n. In our reduction, we would like to set n to be the smallest value such that $M_n = k$, but we were not able to prove that such n exists for each k. Therefore we choose as n the smallest value such that $M_n \ge k$, and we denote this value n_k. As k' we then use M_{n_k}. The following lemma states important properties of n_k and M_{n_k}.

Lemma 2. *The value of n_k is at most $\lceil k \ln k \rceil$, and n_k and M_{n_k} can be computed in polynomial time.*

Before proving this lemma, we finish the proof of the reduction. Let us assume that there is a set of states S such that $\Pr(s(\pi) = S, X \mid H, n) \geq p$. This means that if we consider walks in the subgraph $G'(S)$ induced by the set S, we get $Y(n, G'(S)) \geq D(n, k')$. We will consider three cases:

- If S is a clique and $|S| \geq k'$, we have the desired clique in graph G', and therefore there is also a clique of size k in graph G.
- If S is a clique and $|S| < k'$, then by definition of M_n we have $Y(n, G'(S)) = D(n, |S|) < D(n, M_n) = D(n, k')$. This is a contradiction with our assumption.
- If S is not a clique, then by Lemma 1 and definition of M_n we have $Y(n, G'(S)) < D(n, K_{|S|}) \leq D(n, M_n) = D(n, k')$. Again we get a contradiction with the inequality $Y(n, G'(S)) \geq D(n, k')$.

Therefore, we have proved that G contains a clique of size k if and only if the most probable set of states in $H_{G'}$ that can generate X has probability at least p. Moreover, we can construct n_k, M_{n_k}, $H_{G'}$, X, and p in polynomial time. To complete this proof we need to prove Lemma 2. We start by proving another useful lemma.

Lemma 3. *The following recurrence holds for $2 \leq k \leq n$:*

$$D(n, k) = (k-1)D(n-1, k) + kD(n-1, k-1).$$

In addition, $D(n, n) = n!$, $D(n, 1) = 0$ for $n > 1$, and $D(n, k) = 0$ for $k > n$.

Proof. Clearly, $D(n, n) = n!$ since walks of length $n-1$ correspond to permutations of vertices. If $n > 1$ then $D(n, 1) = 0$, since K_1 does not contain any edges. If $k > n$, $D(n, k) = 0$ since a walk of length $n-1$ can pass through at most n vertices.

Now let $2 \leq k \leq n$. Let $v(w)$ be the number of different vertices covered by walk w. Let w be a walk of length $n-1$ with $v(w) = k$, and let w' be a walk obtained by taking the first $n-1$ vertices of walk w. Then $v(w')$ is either k or $k-1$.

Every walk w' of length $n-2$ with $v(w') = k$ can be extended to a walk w of length $n-1$ in K_k in $k-1$ ways, because as the last vertex of w we can use any vertex except the last vertex of w'. Therefore there are $(k-1)D(n-1, k)$ different walks w in K_k with property $v(w') = k$.

On the other hand, if $v(w') = k-1$, we can create a walk w'' in K_{k-1} by renumbering the vertices in w' so that only numbers $\{1, \dots, k-1\}$ are used (if the vertex missing in w' is i, we replace j by $j-1$ for every vertex $j > i$). The same representative w'' is shared by k different walks w, because to create w from w'', we need to choose the missing vertex i from all k possibilities, renumber vertices to get w' and then to add the missing vertex i at the end of the walk. Therefore there are $kD(n-1, k-1)$ walks with the property $v(w') = k-1$. Combining the two cases we get the desired recurrence. □

Proof. [Proof of Lemma 2] Assume that $k \geq 3$. Clearly, $D(n,k) \leq k(k-1)^{n-1}$, since $k(k-1)^{n-1}$ is the number of all walks of length $n-1$ in K_k. However, this number includes also walks that avoid some vertices. The number of such walks can be bounded from above by $k(k-1)(k-2)^{n-1}$ where we choose one of the k vertices to avoid and then consider all possible walks on the remaining $k-1$ vertices. In this way we count some walks multiple times; nonetheless by Bonferroni inequality we obtain bound

$$D(n,k) \geq k(k-1)^{n-1} - k(k-1)(k-2)^{n-1}$$

For $k \geq 4$ we therefore have that if

$$(k-1)(k-2)^{n-1} < k(k-1)^{n-1} - k(k-1)(k-2)^{n-1}$$

then $D(n, k-1) < D(n,k)$. By taking logarithm of both sides of the inequality, we obtain $n > f(k)$ where

$$f(k) = 1 + \frac{\ln(k^2-1) - \ln k}{\ln(k-1) - \ln(k-2)}$$

Let $n = \lceil f(k) \rceil$ for some $k \geq 4$ and consider row n in Table 3.2. We have that $D(n, k-1) < D(n,k)$ and since function f is increasing, we also we have that $D(n, k'-1) < D(n, k')$ for all $k' \leq k$ (we have proved it only for $k' \geq 4$, but it is easy to see that it is also true for $2 \leq k' \leq 3$). The maximum in row n is therefore achieved at some position $M_n \geq k$. Recall, that n_k is the smallest n such that $M_n \geq k$. Therefore $n_k \leq \lceil f(k) \rceil$. The function $k \ln k / f(k)$ is decreasing and its limit is 1 as k approaches ∞. Therefore $\lceil f(k) \rceil \leq \lceil k \ln k \rceil$, which gives us the inequality $n_k \leq \lceil k \ln k \rceil$. This inequality can also be easily verified for $k < 4$. Since $M_n \leq n$, we also have $M_{n_k} \leq \lceil k \ln k \rceil$.

We can compute n_k and M_{n_k} by filling in table $D(m,j)$ for all values of m and j up to $\lceil k \ln k \rceil$ using the recurrence from Lemma 3. Since $D(n,k) \leq k^n \leq n^n$, we can store $D(m,j)$ in $O(k \cdot \text{polylog}(k))$ bits. Therefore computing the desired values n_k and M_{n_k} can be done in polynomial time. □

Note that it is not clear if the most probable set of states problem is in NP. In particular, given a set of states S, it is NP-hard to find out if its probability is greater than some threshold p, even if this threshold is 0, as we show next.

Theorem 6. *Given HMM H, sequence X of length n and a subset of state space S, the problem of deciding if $\Pr(s(\pi) = S, X \mid H, n)$ is non-zero is NP-complete.*

Proof. The proof is a reduction from the problem of finding Hamiltonian path.

Let $G = (V, E)$ be a graph and H_G be the corresponding graph HMM as in Definition 24. Let $X = 0^{|V|}$. It is easy to see that $\Pr(s(\pi) = V, X \mid H_G, |X|) > 0$ if and only if G contains a Hamiltonian path. □

Theorem 7. *The most probable set problem is fixed-parameter tractable. There is an algorithm with time complexity $O(2^m m^2 n)$ where m is the number of states of the HMM and n is the length of the input sequence.*

Proof. Let V be the set of states of the HMM, and X the input sequence of length n. We can find the most probable set of states in time $O(2^m m^2 n)$ by a dynamic programming algorithm similar to the Forward algorithm. We define $F[i, S, v]$ to be the sum of probabilities of all states paths π of length i such that $s(\pi) = S$, π ends in state v and generates the first i characters of sequence X. The probability of a set S is the of all state paths that generated X with state set S:

$$\Pr(S \mid X) = \sum_{v \in V} F[n, S, v] \tag{3.12}$$

We can find the most probable state set by enumerating all sets and choosing the one with the highest probability. To compute $F[n, S, v]$, we use the following equation:

$$F[i,S,v] = \begin{cases} I_v e(v, X[1]) & i = 1, S = \{v\} \\ \sum_{u \to v} a_{u,v} e(v, X[i]) \left(F[i-1, S \backslash \{v\}, u] + F[i-1, S, u]\right) & i > 1, v \in S \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

Computing F can be done in $O(2^m m^2 n)$ time assuming that manipulation with the numeric representation of probabilities is in constant time. Assuming that the parameters of HMM are rational numbers, we can represent all quantities with the polynomial number of bits and the resulting time complexity will be higher by a polynomial factor. □

3.5 The Most Probable State Restriction

The most probable restriction problem is following: given HMM, decide if there exists such subset of the set of states, that have size l and probability at least p. We prove that this problem is NP-complete.

Proof. (Theorem 4) We will prove NP-hardness by a reduction from 3-SAT. Consider an instance of 3-SAT with the set of variables $U = \{u_1, u_2, \dots, u_n\}$ and the set of clauses $C = \{c_1, c_2, \dots, c_m\}$. Based on sets U and C, we construct an HMM H as follows. The set of states V will contain all positive and negative literals. The emission alphabet Σ contains all clauses, all variables and a special error symbol ψ. The initial probability of each state is $1/(2n)$, and the transition probability between any two states is also $1/(2n)$. State for a literal u emits with probability $1/|\Sigma|$ every clause that contains u. State for literal u also generates the positive form of the literal with probability $1/|\Sigma|$. For proper normalization, it also generates the error symbol with probability $1 - \sum_{x \in C \cup U} e(v, x)$.

We also create string $X = u_1 u_2 \dots u_n c_1 c_2 \dots c_m$ and set the size of the restriction l to equal the number of variables n. Every state path π that can generate X has probability $(2n|\Sigma|)^{-|X|}$; we set threshold p to this value. Each variable u_i in the first part of X can be generated only by states u_i and $\bar{u}_i$; therefore at least one of these states needs to be in the path. The first portion of the path thus traverses l (equals to n) different states; only these states can be used to emit the second part of the sequence. Since each of u_i $(1 \leq i \leq n)$ has to be generated, exactly one of the states u_i, $\bar{u}_i$ will be in the state path. Therefore the set of states used by a state path corresponds to assignment of variables. If this assignment is satisfying, it is possible to generate the rest of the sequence: $c_1 c_2 \dots c_m$, and state path is admissible. If the assignment is not satisfying, there exists clause c_j that cannot be generated states were used in the first part of the sequence (and it is not possible to add additional state into the state set) and therefore any state path using this set of states will be inadmissible. Therefore if there is admissible state path that generates X, then there is an satisfying assignment for the instance of the SAT problem.

Note that given a restriction S, we can easily verify if its probability is at least p by a variant of the Forward algorithm in which we allow only states in S. Therefore the problem is in NP. □

Theorem 8. *The most probable restriction is fixed parameter tractable.*

Proof. We can use the same algorithm as in the proof of the theorem 7, with the difference that we compute the probability of only for sets of size at most l. During the computation

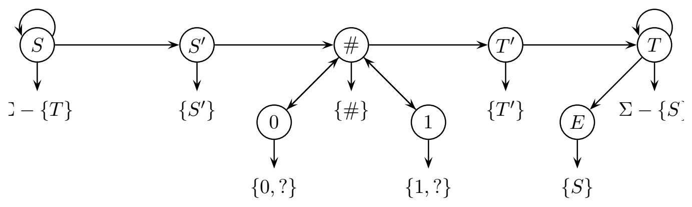


Figure 3.6: The HMM from the proof of Theorem 5. States are shown as circles; under each we list the set of symbols that the state emits with non-zero probability. Each of these symbols is emitted with probability $1/k$, where k is the size of the set. The HMM always starts in state S. All outgoing transitions from a particular state have the same probability.

we use only subsets of states with size at most l and therefore the time complexity is

$$O\left(\sum_{l=0}^{l} \binom{m}{k} m^2 n\right) \tag{3.14}$$

where m is the number of states, n is the length of the sequence and l is the size of the restriction.

3.6 The Most Probable Footprint

In this section we show proof that the most probable footprint problem is NP-hard. Given HMM H, sequence X and the probability p, goal is to decide if there exists a footprint F with the probability at least p. We show result for constant-sized HMM and the identity function as an labeling function (proof for arbitrary labeling function is trivial).

Proof. (Theorem 5) We will prove NP-hardness by a reduction from the maximum clique problem using the HMM in Figure 3.6 with eight states and alphabet $\Sigma = \{S, S', T, T', \#, 0, 1, ?\}$.

Let $G = (V, E)$ be an undirected graph with n vertices $V = \{1, 2, \ldots, n\}$. We will encode it in a sequence X over alphabet Σ as follows. For every vertex $v \in V$, we create a block X_v with $2n + 3$ symbols: $X_v = S'\#b_{v,1}\#b_{v,2}\# \ldots \#b_{v,n}\#T'$, where $b_{i,j} = 1$ if $i = j$, $b_{i,j} =?$ if $(i, j) \in E$ and $b_{i,j} = 0$ otherwise. Sequence X is a concatenation of blocks for all vertices with additional first and last symbols: $X = SX_1X_2 \ldots X_nT$.

All state paths that can generate X have a similar structure. The first symbol S and several initial blocks are generated by state S, one block, say X_i, is generated by states S', $\#$, 0, 1, and T' and the rest of the sequence, including the final symbol T, is generated by state T. We will say that a state path with this structure *covers* the block X_i. Note that state E is never used in generating X, its role is to ensure that the probability of self-transition is the same in states S and T. All state paths that can generate X have the same probability $q = \Pr(\pi, X \mid H) = 2^{-2n^2-2n}3^{-n-1}7^{-2n^2-n+1}$.

We say that a state path π is a *run* of footprint F, if π can generate X, and $f(\pi) = F$. Every footprint that can generate X has the following structure: $F = SS'\#c_1\#c_2\# \dots \# c_n\#T'T$, where $c_i \in \{0,1\}$. The probability of footprint F is qk, where k is the number of its runs. Also note that every run of F covers a different X_i, because once X_i and F are fixed, the whole path is uniquely determined.

We will now prove that the graph G has a clique of size at least k if and only if there is a footprint for sequence X with probability at least qk. First, let R be a clique in G of size at least $k > 0$. Consider the footprint $F = SS'\#c_1\#c_2\# \dots \#c_n\#T'T$ where $c_i = 1$ if $i \in R$ and $c_i = 0$ otherwise. For any $i \in R$, there is a run π_i of F that covers X_i. This run will use state 1 for generating each $b_{i,j}$ such that $j \in R$ and thus both $b_{i,j} \in \{?, 1\}$ and $c_j = 1$. For $j \notin R$ we have $b_{i,j} = 0$ and $c_j = 0$, thus they will use state 0 in π. Since there is a different run for every $i \in R$, footprint F has at least k runs.

Conversely, let F be a footprint with probability at least $qk > 0$ and thus with at least k runs. We will construct a clique of size at least k as follows. Let R be the set of all vertices i such that F has a run that covers X_i. Clearly the size of R is at least k. Since F has non-zero probability, it has the form $SS'\#c_1\#c_2 \dots \#c_n\#T'T$ for $c_i \in \{0,1\}$. For all $i \in R$, $c_i = 1$ because the i-th block has $b_{i,i} = 1$. Therefore for all $i, j \in R$, we have $b_{i,j} \in \{1, ?\}$, which means that $(i,j) \in E$ or $i = j$. This implies that R is indeed a clique.

To summarize, given graph G and threshold k, we can compute in polynomial time sequence X and threshold qk such that G has a clique of size at least k if and only if sequence X has a footprint with probability at least qk. This completes our reduction.

The problem is in NP (even if HMM is not fixed, but given on input), because given an HMM H, sequence X and a footprint F, we can compute the probability $\Pr(f(\pi) = F, X \mid H, |X|)$ in polynomial time by a dynamic programming algorithm which considers all prefixes of X and all prefixes of F. If probability p and parameters of HMMs are given as rational numbers, we can compute all quantities without rounding in polynomial number of bits. $\square$

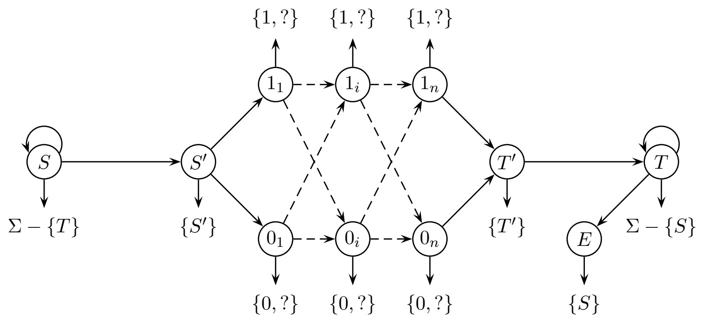


Figure 3.7: The HMM from the proof of Theorem 3 for $n = 3$. States are shown as circles; next to each we list the set of symbols that the state emits with non-zero probability. Each of these symbols is emitted with probability $1/k$, where k is the size of the set. The HMM always starts in state S. All outgoing transitions from a particular state have the same probability.

3.6.1 The Most Probable Set Problem

In this section, we show an alternative proof of NP-hardness of the most probable set problem. We show how to modify the proof of theorem 5 to obtain a proof of theorem 3.

Proof. (Theorem 3) Let $G = (V, E)$ be an undirected graph with n vertices as in the proof of theorem 5. Note that $V = \{1, 2, \ldots, n\}$. Graph G will be encoded in sequence X as in the proof of theorem 5, but without # symbols. Let X_i be the block that represents vertex i. Then $X = SX_1 \ldots X_nT$.

Instead of using one fixed HMM, we expand the middle part of the model so that it generates blocks of length $n + 2$ (n vertices and S', T' states). We remove state # and expand states 0 and 1 into n copies denoted 0_v and 1_v for $v \in V$. Transitions are from S' to 0_1 and 1_1, from 0_n and 1_n to T' and for all $i < n$ from c_i to d_{i+1} for $c, d \in \{0, 1\}$ as in figure 3.7.

State paths that generate X have a similar structure as in the proof of theorem 5. State S generate the first symbol S, and some initial blocks. One block X_i is generated by some of the states $S', T', 0_v, 1_v, v \in V$. Rest of the sequence is generated by the terminal state T. Construction of the HMM and sequence X guarantees that for all $v \in V$ exactly one of $0_v, 1_v$ is in the state path. We say that a state path with this structure *covers* block X_i. All state paths that can generate X have the same probability $q = \Pr(\pi, X \mid H) = 2^{-n^2-3n+1}6^{-n^2-n}$.

We say that a state path π is a *run* of set D, if π can generate X and $s(\pi) = D$. The probability of set D is qk, where k is the number of runs of D. Every run of D covers a different block, since once the block is known, the path is uniquely determined.

We will now prove that the graph G has a clique of size at least k if and only if there is a set of states for sequence X with probability at least qk. First, let R be a clique in G of size at least $k > 0$. Consider the set $D = \{S, S', c_1, c_2, \ldots, c_n, T', T\}$ where $c_i = 1_i$ if $i \in R$ and $c_i = 0_i$ otherwise. For any $i \in R$, there is a run of D that covers X_i. This run will use state 1_j for generating each $b_{i,j}$ such that $j \in R$ and thus both $b_{i,j} \in \{?, 1\}$ and $c_j = 1_j$. For $j \notin R$ we have $b_{i,j} = 0$ and $c_j = 0_j$, thus the run will use state 0_j. Since there is a different run for every $i \in R$, set D has at least k runs.

Conversely, let D be a set of states with probability at least $qk > 0$ and thus with at least k runs. We will construct a clique of size at least k as follows. Let R be the set of all vertices i such that D has a run that covers X_i. Clearly the size of R is at least k. Since D has non-zero probability, it has the form $\{S, S', c_1, c_2, \ldots, c_n, T', T\}$ for $c_i \in \{0_i, 1_i\}$. For all $i \in R$, $c_i = 1_i$ because the i-th block has $b_{i,i} = 1$. Therefore for all $i, j \in R$, we have $b_{i,j} \in \{1, ?\}$, which means that $(i, j) \in E$ or $i = j$. This implies that R is indeed a clique.

Given graph G and threshold k, we can compute in polynomial time sequence X and threshold qk such that G has a clique of size at least k if and only if sequence X has a set of states with probability at least qk. This completes our reduction. $\square$

3.6.2 The Most Probable Restriction

Finally, we show that the proof from the previous section is also proving NP-hardness of the most probable restriction problem (Theorem 4).

Proof. (Theorem 4) Consider the proof of Theorem 3 from Section 3.6.1. Any set for for sequence X that have non-zero probability on sequences that encodes graph with n vertices have size $n + 4$. Therefore if we set $l = n + 4$ (l is the restriction parameter), the proof of theorem 3 is also a proof of this theorem. $\square$

Chapter 4

Alignment with Tandem Repeats

A *tandem repeat* is a region in a genomic sequence that contains two or more consecutive copies of some *motif*, which is a short genomic sequence. The instances of the motif in a tandem repeat are called *repetitions*. Tandem repeats, like other sequences, undergo evolution, and events like substitutions, insertions, and deletions occurs in the individual repetitions. Therefore individual repetitions are only approximate copies of a motif. More than 2% of the human genome is covered by short tandem repeats, and they occur in many genes and regulatory regions [27]. Additionally, recent short insertions in the human genome are mostly caused by tandem duplication [55]. Most of the tandem repeats have evolved by tandem segmental duplications, which means that a tandem repeat was created by several successive duplication events. The consequence is that the homologous tandem repeats in two related sequences may contain a different number of copies of the original motif. Note that it is possible that none of the current repetition is the exact copy of the original motif.

Example 10. *Below is an example of a sequence with tandem repeat with motif AT and* 12 *repetitions. One repetition contains insertion (I in the annotation) and another repetition contains substitution (M in the annotation). The repetitions are annotated by alternating . and _ characters.*

```
Sequence:   ACGTCGATGCATATATATATATATAATATCTATATGAGCTGATGCTAGCTAC
Annotation:           __..__..__..__I..__M.__..
```

Note that while it is not clear to which repetition the insertion belongs, we do not find it important.

Aligning homologous tandem repeats is hard, because it is not clear which repetitions are orthologous (which originate in the same ancestral repetition). Tandem repeats do not only affect quality of alignment within them, but the error can spread into adjacent columns of an alignment, as we show in Section 4.6.3, Figure 4.5. These inaccuracies can spread further and cause artifacts in the results of comparative genomic methods that use sequence alignments.

In this chapter, we introduce a tractable model that explicitly accounts for tandem repeats. We use the maximum expected gain framework to explore several decoding criteria for our model. We evaluate our model with different decoding criteria using simulated data.

4.1 Related Work

Alignment methods that take into consideration tandem duplications were first studied by Benson [6], who proposed an extension of the Needleman-Wunsch algorithm. They extend the scoring scheme by adding a duplication event, which creates a tandem repeat. A duplication consists of copying some substring U in one input sequence into multiple tandem copies of U in the second sequence. The score of the duplication event is the sum of the duplication initiation cost, duplication extension cost (multiplied by the number of repetitions), and the maximum alignment score for aligning the sequence U to its repetitions in the second sequence. Time complexity of this algorithm is $O(n^4)$, and Benson also proposed a heuristic algorithm to compute the alignment in a reasonable time. Several variants of this problem were also studied [63, 8]. Another approach to align sequences with tandem repeats is to use a lossy compression scheme to collapse tandem repeats, and then align compressed sequences [24].

A traditional approach to deal with tandem repeats is to mask them in both sequences and then align masked sequences by some alignment algorithm. Masking means replacing tandem repeats and other low complexity regions either with lower-case letters (soft-masking) or with N symbols (hard-masking, N represents any base). Masking is done by some method for finding tandem repeats, such as Tandem Repeat Finder (TRF) [7], TANTAN [25], mreps [41], or ATRhunter [70].

Methods for aligning sequences with tandem repeats mentioned above were not based on probabilistic models. The first probabilistic method specifically targeting tandem repeats was introduced by Hickey and Blanchette [34]. They developed a context-sensitive model based on pair Tree-Adjoining grammars. Their model does not explicitly model an arbitrary number of repeats; it focuses on short context-sensitive indels caused by tandem duplications. Each such context sensitive indel was modeled by assuming that the indel sequence and previous/following aligned pair of sequences evolved from single common ancestor. The time complexity of their decoding algorithm is $O(n^2L^2)$, where n is the length of the sequences and L is the maximal length of context sensitive indels.

Another, partially probabilistic method was developed by Kováč *et al. (2012).* The aim of the method was to align repetitive motifs occurring in some protein families (for example zinc finger proteins), which are similar to tandem repeats in DNA. Their method focused on correctly aligning individual occurrences of the motifs. They combined a profile HMM of the motif and a pair HMM, and developed a new decoding algorithm similar to the Viterbi algorithm. Despite the use of probabilistic models, their method was not a probabilistic model, because scores from different models were combined in an ad-hoc manner.

Hudek [37] developed an algorithm which combines the Viterbi and the Posterior decoding. The goal of the algorithm was to reduce misalignments due to short tandem repeats with the motif length 1, for example AA in the first sequence and $AAAAAAAAA$ in the second sequence. The algorithm does not search for alignment, but for the most probable segmentation; alignment is divided into segments, such that each block contain gaps in only one sequence. Within one block, algorithm considers all possible evolutionary histories of such block; the probability of the block is the sum of the probabilities of all alignments of sequences within block. The probability of the segmentation is the product of probabilities of all blocks along with transition probabilities between blocks. This was implemented using pHMM with two colors (each color for one segment) and the algorithm effectively search for the most probable annotation (segmentation). Authors give the algorithm for searching for the most probable segmentation in $O(n^4)$ time, where n is the length of the sequences. Additionally, authors provide heuristic with time complexity $O(n^2k)$, where k is the limit for the maximal block length.

4.2 Finding Tandem Repeats

Although our main goal is to align sequences with tandem repeats, our model will feature submodels that can be used to find the repeats in a single sequence. In this section, we describe our repeat submodel and repeat model from the TANTAN repeat finder [25]. We start by describing non-probabilistic approach to finding repeats, which we also use in the preprocessing phase of our method.

4.2.1 Tandem Repeat Finder

Probably the most popular method for finding tandem repeats is the Tandem Repeat Finder (TRF) [7]. Tandem repeat finder finds the position of a tandem repeat (referred as an *interval*), consensus sequence (an approximation of the original motif), alignment of the consensus sequence and the input sequence, and various other information about each repeat.

The method consists of two components: detection and analysis. The detection component tries to find a set of candidate tandem repeats by analyzing the distances between occurrences of the same k-tuple (substring of length k). It uses several statistical criteria to detect repeats and distinguish between tandem repeats and non-tandem repeats [7].

The analysis component aligns a candidate consensus using wraparound dynamic programming [57] with the surrounding sequence. If the alignment is not successful (candidate consensus has to be aligned at least twice to be successful), candidate repeat is discarded. Otherwise, the final consensus sequence is computed from the alignment along with other statistics about tandem repeat.

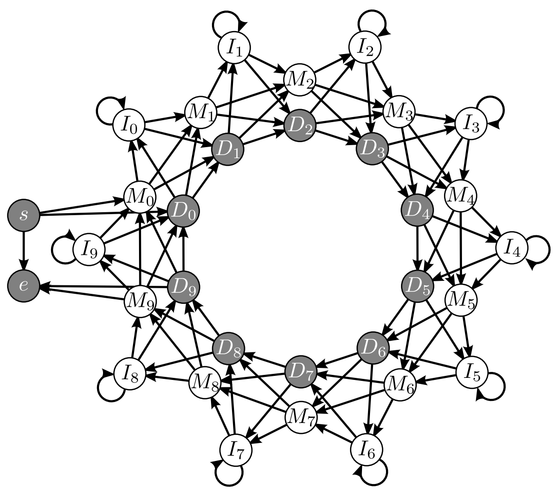
Tandem repeats found by the TRF are often redundant. They can overlap, have slightly different consensus motifs, the motifs can be shifted cyclically, and can have different lengths, as in the following example:

```
Sequence  X:        CACCGCCACCACCGTAG
Consensus ACCACC:    ACCACCACCACC      2 repetitions
Consensus ACC:       ACCACCACCACC      4 repetitions
consensus CAC:      CACCACCACCACC      4.3 repetitions
```

The sequence X contains a tandem repeat and there are three consensus sequences with different number of repetitions. Note that the repetition of the consensus can be incomplete; as with the CAC consensus where we have 4.3 repetitions (0.3 stands for the last C in the repetitive sequence). Note that the purpose of this example to illustrate possible redundancies in the TRF output. If we actually run TRF on sequence X, the output would be only the repeat with consensus CAC.

4.2.2 Sunflower Model

The *sunflower model* is an HMM that represents tandem repetious of one previously specified motif $C = c_0 \dots c_{k-1}$. We have developed this model as a part of our model for aligning sequences with tandem repeats, which we describe in section 4.3, but it can be


(a) Cyclic profile model

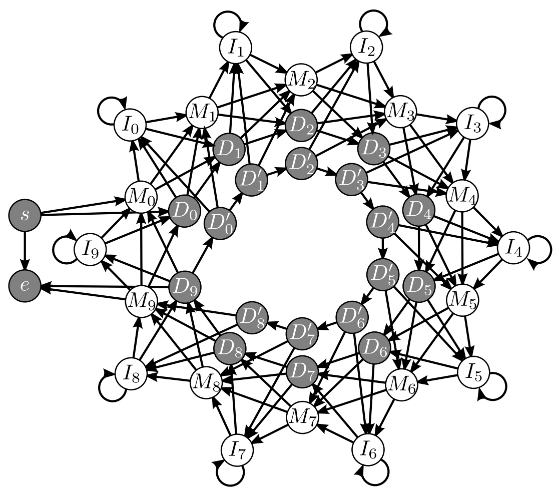

(b) Sunflower model

Figure 4.1: Example of the Sunflower model with the motif of length 10. White states emit one character and gray states are silent states. States s and e are initial and final silent states. Sunflower model has additional delete states $D'_0, \dots D'_8$ to remove silent cycle from the model.

also used as a simple tool for finding tandem repeats and we use it to improve a candidate set of tandem repeats initially constructed by the TRF program (see Section 4.5).

The sunflower model is an extension of the profile HMM described in Section 3.2.2 (see Figure 3.3 on page 55). In particular, the sunflower model is a circular version of the profile HMM with match states representing motif C. This model implicitly assumes that repetitions evolved from motif C independently of each other, as of at single point of time, motif C was copied several times forming tandem repeat, which then underwent several simple evolution events (substitutions, deletions and insertions). This does not accurately reflect a typical evolution history of a tandem repeat, where individual repetitions arise over time from existing copies which could differ from the original consensus C. However, trying to model such evolution would lead to a much more complicated model.

Our model contains typical profile HMM states $M_0, \dots, M_{k-1}, I_0, \dots, I_{k-1}$ and $D_0, \dots, D_{k-1}$. The transitions between the states are similar to profile HMM: $M_i \to M_{i\oplus1}, M_i \to I_i, M_i \to D_{i\oplus1}, I_i \to I_i, I_i \to M_{i\oplus1}, I_i \to D_{i\oplus1}, D_i \to D_{i\oplus1}, D_i \to M_{i\oplus1}, D_i \to I_i$ for all $0 \leq i < k$, where $\oplus$ is $+$ modulo k. As in the profile HMM, states D_i are silent. We add silent start state s and silent final state e with transitions $s \to M_0, s \to D_0, D_{k-1} \to e, M_{k-1} \to e$ and transition $s \to e$ to model empty tandem repeats. The whole model topology is shown in Figure 4.1a.

The problem is that the cyclic profile contains a cycle of silent states, which causes problems with training and decoding algorithm (see Section 2.1.7). Removing these states would lead to additional $\theta(k^2)$ edges. Therefore, we have decided to remove the transition between delete states D_{k-1} and D_0. To compensate for the lost possibility of deleting an arbitrary part of the tandem repeat, we add an additional chain of delete states $D'_0, \dots, D'_{k-2}$, which are accessible only from state D_{k-1} by transition $D_{k-1} \to D'_0$, and their outgoing transitions are similar to delete state transitions: $D'_i \to M_{i+1}, D'_i \to I_i$ for $0 \le i \le k-2$ and $D'_i \to D'_{i+1}$ for $0 \le i < k-2$. Note that D'_{k-2} is connected only to M_{k-1} to avoid passing a full circle by delete states. The full model is in Figure 4.1b. We call this model the *Sunflower* model.

This model has $4k+1$ states and $12k+1$ transitions, out of which $2k+1$ states are silent. There are $14k+2$ parameters to train for a motif C (including emissions of insert and match states over the alphabet if size 4). Models with such a large number of parameters are hard to train, so we reduced the number of parameters as follows. First, we tied similar transitions, so that they have the same probability. We used the set of parameters p_{ab} where $a, b \in \{m, i, d, \cdot\}$, where m stands for any match state, i stands for any insert state, d states for any delete state (from both chains), and $\cdot$ is either start or final state. Therefore the probability of transition from match state to delete state is p_{md} and probability of transition from insert state to final state is $p_{i\cdot}$. Probabilities were set so that $\sum_{b\in\{m,i,d\}p_{ab}} = 1$ for all possible a. Therefore, transitions from states M_{k-1}, D_{k-1} and D'_{k-1} do not sum to 1, because they are either missing one transition or having one additional transition. For those states, the probability of transitions were normalized in order to form a probability distribution.

To reduce the number of emission parameters, all insert states share the same emission distribution. For the emission distribution of match state M_i, we assumed that the consensus base c_i from the motif evolved over evolutionary time t according to Jukes-Cantor model[1]. Jukes-Cantor model is a theoretical model of evolution that assumes constant rate of evolution. Under this model base B_1 evolved over time t to different base B_2 with probability $1/4(1-\exp(-4t/3))$. The probability that B_1 after time t will be still (or again) B_1 is $1/4(1+3\exp(-4t/3))$ [22]. Time t was the same for all match states. Therefore emissions of all states are specified using only 4 parameters (1 for match states

[1] The parameter t of Jukes-Cantor model is not time as measured by years, but rather a branch length in an evolutionary tree obtained by multiplying of substitution rate and time. More details can be found in [22].

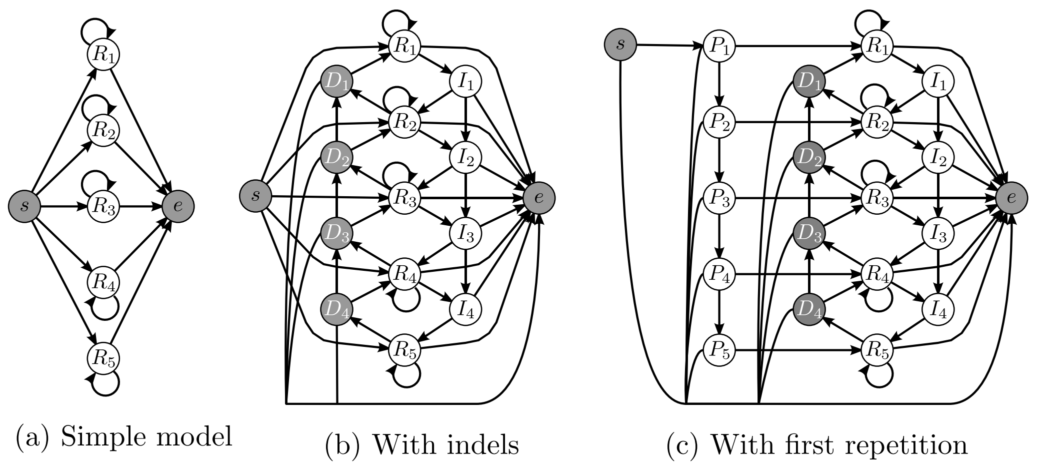


Figure 4.2: Three variants of the core of the TANTAN model. Gray states are silent. The first one is a simplified model, only containing repeat states. The second was used by Frith [25], allowing insertions and deletions. The last one is our extension with additional prefix states $P_1, \dots P_K$ which model the first repetition.

and 3 for insert states).

To use the sunflower model models for finding tandem repeats, we have to add state B that models non-repetitive part of the sequence. Let S_C be the sunflower model for motif C. We add the transitions from B to the start state of S_C and from the final state of S_C to the state B. The probability of transition from the state B to the submodel S_C is p_r, the probability of repeat starting at particular position in the sequence. Probability of transition from final state of S_C to B is 1. There is also transition $B \to B$ with probability $1 - p_r$. We can use the Viterbi algorithm with this model to find all occurrences of tandem a repeat with motif C. However, with this model we can search only for a specific motif. We call this method of finding tandem repeats the Sunflower repeat finder (SRF).

4.2.3 TANTAN

TANTAN is a high-order HMM aimed at finding tandem repeat developed by Frith [25]. Unlike the Sunflower model, TANTAN models tandem repeats with an arbitrary motif. Its only restriction is the maximal length of the motif K. In this section, we describe the core of the model; the part that models tandem repeats. This core can be transformed to the model usable for search by adding a background state B as we described for the Sunflower model.

The principal idea is to use the state of order l to model a repeat with motif length

l. TANTANs high order states uses less information than standard high order states. Emission of symbol $X[i]$ in a standard state of order l depends on subsequence $X[i-l:i]$. Emission in TANTANs states of order l depends only on the symbol $X[i-l]$. The model consists of K high-order states R_l, $1 \leq l \leq K$ called repeat states, where state R_l is of order l. Emission of state R_l is set so that the state emits the same symbol as $X[i-l]$ with a high probability. By adding transition $R_l \to R_l$ we obtain an HMM modeling tandem repeats without indels with a motif of length l. By connecting states $R_1, \ldots, R_K$ to a single start and final state as shown in Figure 4.2a, we obtain an HMM that models repeats with the motif lengths 1 through K.

This simplified model does not allow insertions and deletions in repetitions. Indels are handled similarly to a profile HMMs, by adding insert states $I_1, \ldots, I_{K-1}$ and silent delete states $D_1, \ldots, D_{K-1}$ connected as in the Figure 4.2b. There are however significant differences from profile HMMs. In a profile HMM, delete states are used to skip at least one match state, while in TANTAN a delete state is used move to a state with a lower order. Conversely, insert states allows us to move in the opposite direction; using insert state causes an increase of the order of the repeat state. It is also possible to move from the insert state I_j to the insert state I_{j+1}, which is not possible in a profile HMM.

One disadvantage of the TANTAN model is that it does not model the first occurrence of the motif sequence, since repeat states model only the subsequent repetitions of the sequence. This caused problems when we wanted to incorporate TANTAN as a submodel for aligning sequences. Therefore we have added an additional chain of prefix states $P_1, \ldots, P_K$ modeling the first repetition. Transitions from the start state are now only to the final state (modeling an empty sequence) and the state P_1. State $P_l, 1 \leq l \leq K$ has transitions to the final state, repeat state R_l and the state P_{l+1}, if such a state exists.

We set emission distribution of the insert and prefix states to the background probability: the distribution of the bases in DNA. Emission state of the state R_l was derived using Jukes-Cantor model with parameter t, where the emission $X[i]$ evolved from $X[i-l]$ over time t.

The TANTAN model evolution of repetitions differently than the Sunflower model. While the Sunflower model assumes that the repetitions evolved independently from the single common ancestral motif, TANTAN assumes that each repetition evolves from the previous repetition. Therefore the substitutions, insertions, and deletions carry over to the following repetition.

4.3 Models for Aligning with Tandem Repeats

In this section we describe models that we have designed and used in our methods. We describe the model in an dual way; as a generalized pHMM and as an equivalent non-generalized pHMM (emissions length is at most 1 in both sequences). The distribution of generated alignments and annotations (with certain annotation functions) are same in both models, but clearly the distribution of state paths will be different due to different sets of states.

The generalized model is obtained by taking the simple 3-state pHMM model from section 2.4 and adding a single generalized pair state R, called the *repeat state*, which in a single step generates tandem repeats in both sequences. The overall topology of GpHMM is illustrated in the Figure 4.3.

To made this model more flexible, the emission distribution of state R is defined by an additional pHMM. Since the repetitions in the tandem repeat are very similar to each other, we did not try to model the evolution of repetitive parts of the sequences. We assumed that repeat evolved from the original motif by one event, and then repetitions evolved independently. The goal of state R is to model such evolution in single generalized emission. Model was constructed from the Sunflower models. Let C be the set of all motifs that can occur in the alignment. For each motif $c \in C$ we created sunflowers S_c^X and S_c^Y. Sunflower S_c^X is the sunflower model with motif c which generates symbols only in sequence X; S_c^Y generates symbols in Y. We connect S_c^Y and S_c^X by a transition from the final state of S_c^X to the start state of S_c^Y with probability 1 and thus getting model H_c that generates repeats in both sequences. We connected models H_c for all $c \in C$ in parallel by a single start state and a single final state as in Figure 4.3. The probability of the transition from the start state of R to the start state of model H_c was proportional to the distribution of motifs $\Pr(c)$. Note that the size of this model is determined mostly by the total length of all motifs, and therefore this model can be very large, even infinite if we consider all possible sequences as possible motifs for tandem repeats. To keep the model size reasonable, we computed a set of candidate motifs C and use it for construction of the model. More details are in section 4.5. We call this generalized model the sunflower field (SFF) model.

Note that the state R defined as above can generate arbitrary long sequences. Additionally, the Sunflowers are combined in a way, which does not produce an alignment of tandem repeats. Its task is to filter tandem repeats out of alignment, so that they do not cause biases in the parts of an alignment emitted by other states (match and indel

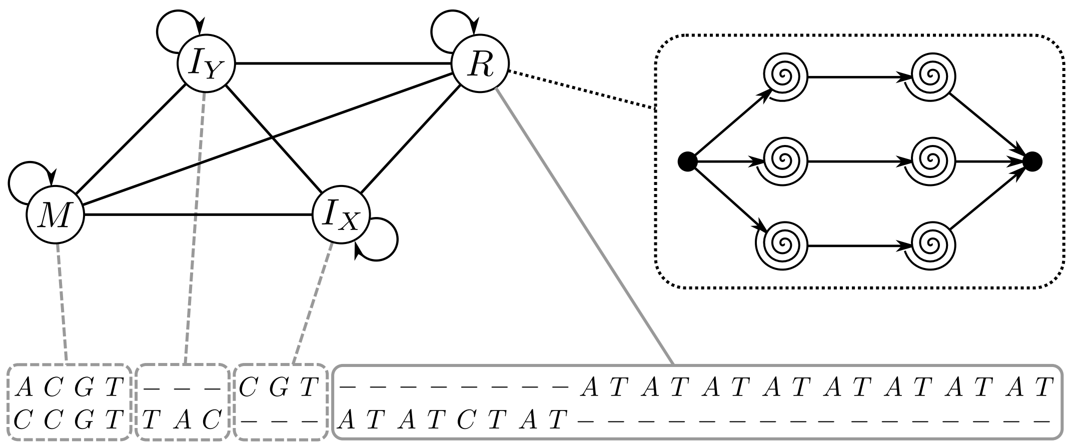

I_Y
R
M
I_X
A C G T
C C G T
– – –
T A C
C G T
– – –
– – – – – – – – – A T A T A T A T A T A T A T A T
A T A T C T A T – – – – – – – – – – – – – – – – –

Figure 4.3: Topology of the Repeat pHMM. We extended the 3-state pHMM with one generalized state R, which in one emission generates tandem repeats in both sequences. The emission distribution of state R is defined by another pHMM shown in a dotted box (*). Bellow the model we show an example of emitted alignments. Dashed gray lines corresponds to multiple emissions from the same states, while the full line represents one emission from a generalized state.
(*) Black states in this submodel are silent start state and final state. Spirals represents sunflower submodels. They are in pairs of two identical models S_c^X and S_c^Y, one for generating tandem repeat in one sequence, other for generating tandem repeat in the other sequence. There are multiple pairs of submodels, each for modeling different motif c.

states). Alignment of homologous tandem repeats is done later in a post-processing step (see Section 2.4.4).

We also experimented with using a TANTAN-like model for defining emissions distribution of state R. We used the TANTAN model with the prefix states from Section 4.2.3. Similarly as with the sunflower model, we created two copies of TANTAN, each generating repeats in only one sequence and connect them together exactly as we would connect the sunflower models. Since TANTAN model does not rely on a motif, it is not necessary to create more copies of the TANTAN HMM and its general topology looks like SFFs with only one motif. Since TANTAN is a high order HMM, the resulting model is a high order and generalized pair HMM. We refer to this model as TANTAN pHMM (TTP). The advantage of this model over SFF is in it's size, since TTP will have in practice fewer states than SFF; the size of SFF model depends on the total length of all motif strings while the size of TTP model depends on the maximum motif length (we set this limit to

50 bases). Therefore the SFF model can be exponentially larger than the TTP.

Another difference between the SFF and the TTP is in the assumed evolution of orthologous tandem repeats. The SFF model assumes that repetitions accumulated substitutions independently, while the TTP model assumes that each repetition evolve from the previous occurrence of the repetition. In the SFF model, we require that the tandem repeats in both sequences share motif, but there is no dependence between the two sequences in the TTP model; there is no penalty for them having different motifs.

In reality, tandem repeats at orthologous positions in two species share common ancestor and therefore share part of their evolutionary history. However, it is possible that they were consequently modified by additional evolution events after speciation (including more tandem duplications). In our model, we ignore such complex evolution of repetitions, because it would lead to very complex model and increase the difficulty in decoding and training. Kováč *et al.* developed a method with limited dependence by adding repeat submodels emitting copies in the two sequences at same time [42].

Both SFF and TTP are defined as 4 state high order GpHMMs. In general, using generalized models increases the time complexity of decoding algorithms quadratically. Therefore we also used expanded versions of the models: we replaced the generalized state R with the pHMM submodel defining emissions of R. All transitions entering into R were replaced by transitions to the start state of the submodel, and all outgoing transitions from R were replaced by transitions starting in the final state of the submodel. This transformation does not change the distribution of alignments generated by the model.

4.3.1 Parameter Estimation

We used the alignment of the human chromosome 15 and its orthologous sequences in the dog genome as the *source alignment* for the estimation of the parameters of the used models. In our experiments, we have used $310,091$ consensus sequences found by the TRF program on the source alignment as motifs for building SFF. The probability of choosing a particular motif is the observed frequency of the motif in the TRF output. Since resulting model is large, we limit Sunflower submodels (see Section 4.5.

We set the parameters of the Sunflower submodel manually: the insert and delete rates were set to 0.005, the match states allows substitutions in motif sequence according to the Jukes-Cantor model with parameter $t = 0.05$. The emission of the insert states were set according to the frequency of the bases in the input human-dog alignment. Parameters of the TANTAN submodel were estimated by the Baum-Welch algorithm [22] on 500 repeats sampled from the SFF model.

We have annotated the source alignment using TRF program; each alignment column was annotated as a repeat if any of its bases were annotated as repeat. We assumed that all columns annotated as repeats were generated by state R and all other columns were generated by respective state: M, I_X, or I_Y. Note that is is clear which non-repeat column was generated by which state. Therefore could estimate the emission distributions and transition distributions from the frequencies of occurred transitions in transitions in the source alignment (see Section 2.1.6).

4.4 Decoding Methods

The right selection of the decoding method can have significant effect on the quality[2] of predictions of a method. In sections 2.1.5, 2.2 and Chapter 3, we described several decoding algorithms whose aim was to increase the accuracy of the sequence annotation. Similarly, Lunter *et al. (2008)* studied the effect of different decoding algorithms on the quality of the constructed alignments (see Section 2.4.8) and showed that the Viterbi algorithm was not the best method for constructing sequence alignments. Therefore it is worth to consider domain-specific decoding methods for constructing alignments.

In this section we describe several optimization criteria that we used with the 3-state model, the SFF model and the TTP model: the Viterbi algorithm (VA), the posterior decoding (PD), the marginalized posterior decoding (MPD), the block Viterbi algorithm (BVA) and the block posterior decoding (BPD). The first three methods are used with the non-generalized versions of the models, the latter two methods (block methods) are used with GpHMMs. We have already described the first three algorithms in Section 2.4.4. While the first three methods are general purpose decoding methods of pair HMMs, we introduce BPD and BVA as an domain-specific decoding methods. Their goal is to remove tandem repeats out scope of the three state part of the SFF model.

Before continuing, we make additional argument for the use of the domain-specific decoding methods instead of finding the most probable state path, the most probable annotation, or in case of the pHMM, the most probable alignment. When constructing the model, we do many simplifications of the biological processes. This is because because without these simplifications, we would end up with very large and impractical models. We can use the domain-specific decoding method as an compensation for such simplifications;

[2]The term quality is intentionally loose.

we can practically move the complexity from the model into the decoding method.

In Section 2.2.1 we defined a gain function as a functions of two annotations. For pair HMMs we define a gain function as a function of two annotated alignments consisting of alignment columns along with annotation symbols. This is necessary, because we use generalized models, and therefore annotation symbols do not uniquely determine the alignment (there can be multiple alignments with the same annotation).

We represent columns of an annotated alignment using indices to sequences and annotation symbols. In the case of generalized models, which can generate more than one column in one step, only the first column will contain annotation symbol, other columns will contain symbol $\varnothing$. Formally, an *annotated alignment* of length t of sequences $X = x_0x_1 \dots x_{n-1}$ and $Y = y_0y_2 \dots y_{m-1}$ is represented as a sequence of tuples $(u_0, a_0, b_0), \dots, (u_{t-1}, a_{t-1}, b_{t-1})$ where $a_i \in \{0, \dots, n-1\} \cup \{-_0, \dots, -_n\}$ and $b_i \in \{0, \dots, m-1\} \cup \{-_0, \dots, -_m\}$, and u_i is either an annotation symbol or symbol $\varnothing$. We will refer to triple (u_i, a_i, v_i) as an *annotated alignment column*. Number i represents i-th symbol in the corresponding input sequence and $-_i$ represents a gap in the sequence before position i (if i is the length of the sequence, it represents the gap at the end of the sequence). For example $(I_X, 47, -_{42})$ means that x_{47} is aligned to a gap that is between y_{41} and y_{42} and that this column has annotation I_X. Naturally, indices in a_i and b_i have to be in non-decreasing order, and each non-dashed symbol has to be in the corresponding sequence in the alignment exactly once. Additionally, a_0 has to be 0 or $-_0$, a_t is n or $-_n$. The constrains for b are analogous. The first annotation symbol $u_0 \neq \varnothing$ and if some u_i is equal to $\varnothing$ then such a column was emitted by the same emission as the previous column. If we use a non-generalized model, the annotated alignment cannot contain symbol $\varnothing$. The reason for using annotation symbol $\varnothing$ is that we need to distinguish between different generalized emissions. We can define the probability of an annotated alignment Λ as the sum of the probabilities of the state paths that generate Λ. We denote this probability as $\Pr(\Lambda \mid X, Y)$.

We are interested in the *repeat annotation* with annotation function λ_R. For generalized SFF and TTP (and 3-state pHMM), λ_R is the identify function. For the non-generalized versions, $\lambda_R(u) = R$ for all states u from the submodels modeling tandem repeats.

Similarly as for the regular HMMs, we can define a gain function $G(\Lambda, \Lambda')$ that corresponds to "similarity" of two annotated alignments Λ and Λ'. Note that for HMMs, symbol Λ represents annotation; with pHMMs, we use this symbol for an annotated alignment. Since a pHMM defines the probability distribution Λ_T of the correct annotated alignments, we can define the expected gain of an annotated alignment given sequence X and Y and gain function G:

$$E_{\Lambda_T|X,Y}\left[G(\Lambda_T,\Lambda)\right]=\sum_{\Lambda_T}G(\Lambda_T,\Lambda)\Pr\left(\Lambda_T\mid X,Y\right) \tag{4.1}$$

Since we do not know the correct annotated alignment, we search for the annotated alignment Λ^* with the highest expected gain:

$$\Lambda^*=\arg\max_{\Lambda}E_{\Lambda_T|X,Y}\left[G(\Lambda_T,\Lambda)\right] \tag{4.2}$$

Finally, we express the optimization criteria of various decoding methods using the highest expected gain framework.

The Viterbi algorithm and block Viterbi algorithm Gain function for the Viterbi algorithm assigns +1 if the predicted annotated alignment is identical to the true annotated alignment. In both cases, the labeling function is the identity function. The optimization algorithms were described in sections 2.4.2 and 2.4.3. The time complexity of VA is $O(nmE)$ where E is the number of non-zero transitions in the model and n and m are lengths of the two sequences. We will discuss the time complexity of BVA at the end of this section.

We make a distinction between the VA and BVA when using the SFF or the TTP models. Using the VA on the expanded model is referred to as the VA. Using the VA on the generalized version of the model will be referred to as the BVA. The difference between these two versions is that in the VA, we account for only one state path through the repeat submodel. The BVA however sums all possible paths through the repeat submodel, and therefore abstracts from the exact realization of the alignment of tandem repeats to their consensus.

Posterior decoding The Viterbi algorithm awards non-zero gain only if an annotated alignment is entirely correct. The gain function of the posterior decoding is more granular. In the traditional version (discussed in Section 2.4.4) it assigns +1 for every correctly predicted alignment column ignoring column labels. An annotated alignment column (u_i,a_i,b_i) is correctly predicted if there is an alignment column (v,a_i,b_i) for arbitrary v in the true annotated alignment.

We will consider a stricter version of the PD in which the corresponding annotated alignment column in the correct annotated alignment has to be identical (annotations have to be same as well). This stricter condition is aimed for the expanded versions of SFF and TTP, since without this condition the PD would treat repeats as gaps despite having orthologs in the other sequence (our models treat such repeats independently and models them formally as gaps). For the 3-state HMM, this stricter condition is equivalent to the original version because each alignment column determine the used state. The running time of the PD is again $O(nmE)$.

The difference from the original PD is in the used recurrence. In stricter version we also keep the used label c in the recurrence M. The interpretation of $M[i,j,a]$ if following. The score of the best annotated alignment that aligning sequences $X[:i]$ and $Y[:j]$, where the last annotation symbol is a. The recurrence is following:

$$M[-1,i,a] = 0, 0 \leq i < m \tag{4.3}$$

$$M[i,-1,a] = 0, 0 < i < n \tag{4.4}$$

$$M[i,j,a] = \max_{b \in C} \begin{cases} M[i-1,j-1,b] + \Pr\left((X[i],a,Y[j])\right) \\ M[i,j-1,b] + \Pr\left((-_i,a,Y[j])\right) \\ M[i-1,j,b] + \Pr\left((X[i],a,-_j)\right) \end{cases}, 0 \leq i < n, 0 \leq j < m \tag{4.5}$$

C is the set of all annotation symbol and $\Pr(A)$ is the posterior probability of the annotation alignment column A. The posterior probability of an annotated alignment column A is the sum of the probabilities of all annotated alignments that contain A. It can be computed by the Forward-Backward algorithm.

Marginalized posterior decoding Marginalized posterior decoding is similar to the posterior decoding. The only difference is that gaps that differ only in their position in the sequence are considered identical. Therefore we treat column $(u,i,-_j)$ to be same as $(u,i,-_k)$; gaps in the second sequence are treated symmetrically. The optimization of this gain function is almost identical to the optimization of posterior decoding. The only difference is that after computation of the posterior probabilities for all annotated columns, we replace probability $(u,i,-_j)$ with the sum of $(u,i,-_l)$ for all l. The algorithm has also time complexity $O(nmE)$. As with PD, this decoding method is not used on the generalized models.

Block posterior decoding Block posterior decoding is based on the posterior decoding and is aimed at generalized models. BPD scores whole emissions instead of individual

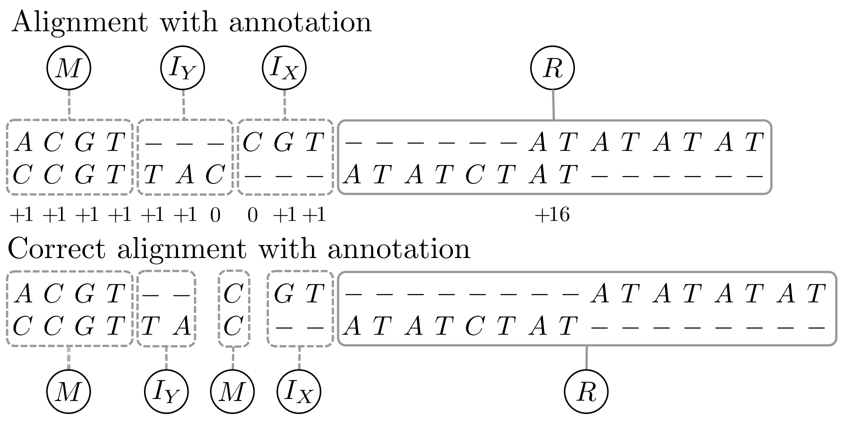

Alignment with annotation
M
I_Y
I_X
R
A C G T – – – C G T – – – – – – A T A T A T A T
C C G T T A C – – – A T A T C T A T – – – – – –
+1 +1 +1 +1 +1 +1 0 0 +1 +1 +16
Correct alignment with annotation
A C G T – – C G T – – – – – – – – A T A T A T A T
C C G T T A C – – A T A T C T A T – – – – – – – –
M
I_Y
M
I_X
R

Figure 4.4: An example of the BPD gain function. Gray solid boxes enclose single blocks, dashed boxes combine multiple block (each alignment column in such a box is one block). Block from repeat state have gain +16 because it emits 16 sequences in exactly same regions of the input sequences. Other alignment columns get +1 only if there is same alignment column in the correct alignment that was generated by same state.

columns. The segment of an annotated alignment that was emitted by one emission is called a *block*. In particular, the states M, I_X and I_Y emit one-column blocks, and state R emits multicolumn blocks. An annotated alignment can be divided into blocks using $\varnothing$ symbol, which mark continuation of a block. The gain function scores each block individually. A one-column block $(u_i, a_i, b_i), u_i \in \{M, I_X, I_Y\}$ gets score +1 if there is an identical column in the true annotated alignment; otherwise gain for such a block is 0. A block of form $\Lambda_E = (u_i, a_i, b_i)(\varnothing, a_{i+1}, b_{i+1}) \dots (\varnothing, a_j, b_j)$ where $(j+1)$-th column does not contain $\varnothing$ (or there is no $(j+1)$-th column) gets score $+l$ if the block is correct. The gain l is the number of emitted non-gap indices in the block. For example block $(R, 4, -_8)(\varnothing, 5, -_8)(\varnothing, 6, 8)$ contains 4 non-gap indices: $4, 5, 6$ in the first sequence and 8 in the second sequence. The block is considered correct if exactly the same regions in X and Y form a block with the same annotation in the true annotated alignment.

The reason for using score $+l$ instead of +1 is that otherwise the decoding method would be biased towards short blocks. Instead of l, the number of emitted symbols, we could use the number of emitted columns. However we would then discriminate the submodels that would tried to align repeats (although we did not used such models). An example of this gain function in Figure 4.4.

To optimize this gain function, we have to compute posterior probabilities for all potential blocks. This can be done by the Forward-backward algorithm. In a GpHMM, each block is given by a state and two intervals; one in sequence X and one in sequence Y. In our models, only state R is generalized and the overall number of states is constant. The number of possible blocks is therefore $O(n^2m^2)$. The expected gain of the block is its posterior probability multiplied by the number of emitted symbols. After computing the expected gains of individual blocks, we can compute the highest scoring annotated alignment in $O(n^2m^2)$ time by the dynamic programming will be described in the end of this section.

If we use the Forward algorithm for pHMM for computation of the emissions of state R, then time complexity of the Forward-Backward algorithm (computation of the posterior probabilities of all blocks) would be in $O(n^3m^3E)$ time, where E is the number of transitions in the repeat submodel; $O(nmE)$ term is computing the emission probability. With some pre-processing, we can lower the $O(nmE)$ term to $O(k)$ where k in the number of sunflower pairs in the repeat submodel (in case of TTP, $k = 1$).

Let us consider the repeat submodel H with k sunflower pairs $S_1^X, S_1^Y, \ldots, S_k^X, S_k^Y$. Let p_i be the probability of entering sunflower S_i^X. The probability of transition from S_i^X to S_i^Y is 1. We can write the probability of generating an arbitrary pair of sequences X' and Y' by H as

$$\Pr\left(X', Y' \mid H\right) = \sum_{i=1}^{k} p_i \Pr\left(X' \mid S_i^X\right) \Pr\left(Y' \mid S_i^Y\right) \tag{4.6}$$

By pre-computing all of the emission terms $\Pr\left(X' \mid S_i^X\right)$ and $\Pr\left(Y' \mid S_i^Y\right)$, we can compute the posterior probability of the block in $O(k)$ time. Naive computation of probability $\Pr\left(X' \mid S_i^X\right)$ for all intervals X' would take $O(n^3E)$ time, but this can be further improved. When computing $\Pr\left(X' \mid S_i^X\right)$ using the forward algorithm, we can use the values from the forward table to compute the probability of emitting any prefix of X' by looking at the forward probability of final state for the corresponding column. We thus need to run the forward algorithm only once for every suffix of X. The optimization for the other sequence is analogous. Using these techniques, we can pre-compute the emission terms in $O((n^2 + m^2)E)$ time, and the overall time complexity of computing the posterior probabilities of blocks is $O(kn^2m^2 + (n^2 + m^2)E)$.

Once the posterior probabilities are computed, we find the best-scoring alignment using similar dynamic programming as for the PD. The difference is that blocks now are longer than one column and therefore the recurrence uses

$$M[-1,i,a] = 0, 0 \leq i < m \tag{4.7}$$

$$M[i,-1,a] = 0, 0 < i < n \tag{4.8}$$

$$M[i,j,a] = \max_{b \in C} M[i-i', j-j', b] + \Pr\left((X[i':i], a, Y[j':j])\right) \tag{4.9}$$

where $0 \leq i' \leq i < n, 0 \leq j' \leq j$ and $i - i' > 0$ or $j - j' > 0$ (at least one base was emitted). The $\Pr\left((X[i':i], a, Y[j':j])\right)$ is the posterior probability of a block with label a containing sequences $X[i':i]$ and $Y[j':j]$. The time complexity of this step is $O(n^2m^2)$ and therefore the overall time complexity of BPD is $O(kn^2m^2 + (n^2+m^2)E)$.

We can use the same techniques to optimize the BVA; the most expensive step in the BVA is the computation of the block probabilities for state R, namely $e_{v,(X',Y')}$. We can use same trick to pre-compute all emission terms in $O((n^2+m^2))$ time, afterwards we can compute block probability in $O(k)$. The complexity of the Viterbi algorithm (see Section 2.4.3) will be $O(kn^2m^2)$ and the overall time complexity of BVA is $O(kn^2m^2+(n^2+m^2)E)$.

Post-processing The problem with the SFF and the TTP models is that they model repeat sequences in the sequences X and Y independently and do not try to align tandem repeats at orthologous locations. Therefore we post-process the alignment using the 3-state pHMM by realigning segments of alignments that are annotated as repeats. We also include adjacent gaps into this post-process step.

4.5 Optimizations

In this section, we summarize additional techniques we have used to decrease the running time of the decoding algorithms. The fastest decoding algorithms described above run in $O(nmE)$ time, which is still prohibitive for longer sequences. Additionally, block-based methods are even slower. We used the following optimizations:

We implemented a standard technique of banding (Section 2.5.1). We first align input sequences using Muscle [23] with default parameters. The final alignment methods were then restricted to be within 30 bases from this guide alignment. This technique reduces the $O(nm)$ factor from the time complexity to $O((n+m)d)$ where d is the maximum distance from the guide alignment.

The size of the SFF model is enormous. It is not practical to use a model with $310,091$ sunflower pairs. Therefore we used the TRF program [7] to find consensus motifs in the input alignment and used only those to build SFF model. Note that the transition probabilities to individual sunflower pairs were kept the same as in the original model. In case that TRF found a consensus that was not in the original set of consensus sequences used to build the SFF model, we assign it the probability of the least frequent consensus from the original set. This technique is not applicable to TTP model.

To further reduce the running time of the block-based algorithms (BVA and BPD), we restrict the emissions of the state R to specific intervals in both input sequences. First, we find the candidate set of tandem repeat intervals for each input sequence independently. Let T_X and T_Y are the sets of candidate intervals (not necessarily disjoint) for sequences X and Y respectively. We restrict emission of state R only to the pairs of intervals i_X, i_Y where i_X and i_Y have beginning and end within 10 bases from some interval from T_X and T_Y respectively, or if one of i_X or i_Y is an empty interval (there can be a tandem repeat in only one sequence). By this heuristic, state R can have emissions in at most $(400|T_X| + n)(400|T_Y| + m)$ positions. We applied this method to both SFF and TPP models.

We used the following algorithm to select candidate intervals T_X and T_Y:

1. Run the TRF program on the input sequences X and Y. Obtain sets of intervals T_X and T_Y and lists of consensus sequences C_X and C_Y.
2. For every consensus $c \in C_X \cup C_Y$, build the SRF model S_c (Sunflower repeat finding model from Section 4.2.2).
3. Each model S_c is run on the sequences X and Y using the Viterbi algorithm. All found repeat intervals are added into corresponding sets T_X and T_Y.

The reason for using this method instead of using only intervals from TRF is that running TRF on the input sequences independently can cause several problems. For example, if sequence X contains three repetitions of motif m and sequence Y contains only one, then the TRF would return interval only for sequence X. Additionally, consensus can be found in both sequences but can be rotated, and therefore intervals in both sequences would be shifted.

4.6 Experiments

In sections 4.3 and 4.4 we proposed new methods for aligning sequences that contain tandem repeats; SFF and TTP model. We also described five decoding methods, out of which two are new (BVA and BPD). To show the usefulness of the proposed methods, we investigate the effects of using these methods instead of standard methods for aligning sequences, which were described in Section 4.1. We evaluated these methods on the simulated data generated from a model trained on the human-dog alignment. Evaluating model on real data is hard, because we do not know the correct alignment of the real DNA sequences, and thus we have to use indirect methods for evaluation. We do not present such a comparison in this thesis, but in [59] we compare several methods on real data counting the number of genes that map through each alignment to the other genome to a plausible gene structure.

4.6.1 Simulated Data

We trained the SFF model, the TTP model, and the 3-state HMM on the human-dog alignment as described in section 4.3.1. Then we sampled 200 test alignments each of length at least 200 bases from the SFF submodel with the condition, that the number of repetitions in the tandem repeat has to be at least three in both sequences; we were sampling emission from state R until the condition was met. The reason for this is that otherwise we would have many segments of alignments that are annotated as repeats, but in fact contain only one copy of the motif. We refer to these alignments (along with the sampled repeat annotation) as the correct alignments and correct annotation (repeat intervals and consensuses). The program Context [34] was trained of 200 separate alignments sampled from the same SFF model.

4.6.2 Accuracy Measures

We use several measures to compare the correct alignment and predicted alignments. It is important to use multiple measures for evaluating predicted alignments, because the measure determine the optimal gain function; when comparing using measure $\beta(\Lambda, \Lambda')$, we can alway search for an alignment that optimizes the expected value of $\beta(\Lambda, \Lambda')$. However in practice we are looking for methods that give good results under multiple measures.

The first measure we used was the error rate; the fraction of incorrectly predicted columns of an alignment. It was measured only on the alignment columns generated from non-repeat states during sampling, because SFF and TTP do not model alignment of repetitive regions.

Additionally, we investigate the accuracy of predicting exact repeat block boundaries in alignments by measuring the number of correctly predicted repeat blocks; a block is correctly predicted if it contains identical parts of the sequences as the corresponding block in the correct alignment. We report the block sensitivity, which is the number of correctly predicted blocks divided by the number of all repeat blocks in the correct alignment, and block specificity, which is the number of correctly predicted blocks divided by the number of all predicted repeat blocks. We also investigate the accuracy of prediction of repeat annotation for individual bases; the repeat sensitivity and specificity.

4.6.3 Results

We have conducted several experiments to compare the accuracy of the predicted alignment, and measure the effects of using different models and algorithms. First we compare the accuracy of our SFF-based methods with standard alignment methods. The results are in the Table 4.1. We choose the Viterbi algorithm with the 3-state model as a baseline method. From standard methods, we also run the Muscle software [23], the Context [34], and the posterior decoding on the 3-state HMM with and without hard-masking of tandem repeats. The hard-masking was done using the TRF program. None of the standard methods provides repeat annotation, and therefore we report only the alignment error. The only exception is the PD with masking, where we annotated predicted columns as repeats if at least one base in the column was masked.

We run all decoding algorithms described in section 2.4.4 with SFF model and compared it with standard alignment methods. In general, using the SFF model decreased the alignment error rate by $15-30\%$ compared to the baseline method (see table 4.1). Other standard methods also had in general higher error rate than the SFF-based methods. The high error rate of the Context program could be caused by insufficient training data or some software issues. We run muscle with default parameters, and therefore its scoring scheme was not tailored to our data. Higher error rate of PD with hard-masking might be caused by the insufficient accuracy of the tandem repeat annotation.

The 15% decrease of the error rate was obtained by replacing the 3-state model with the Sunflower Field model, additional improvements was due to use of the different decoding algorithms, with the marginalized posterior decoding having the lowest error rate. The Viterbi based methods had consistently higher error rate than posterior based methods

	Alignment	Repeat		Block	
Algorithm	error	sn.	sp.	sn.	sp.
3-state VA (baseline)	4.78%				
Context	5.98%				
Muscle	5.62%				
3-state PD	**4.41%**				
3-state masked PD[†]	5.03%	99.23%	74.16%	7.66%	7.24%
SFF MPD	**3.37%**	**95.97%**	97.78%	**43.07%**	44.87%
SFF PD	3.53%	95.86%	97.87%	42.70%	47.37%
SFF BPD	3.51%	93.09%	**98.07%**	36.50%	41.67%
SFF BVA	3.91%	93.26%	97.96%	35.77%	40.66%
SFF VA	4.04%	95.29%	97.85%	42.70%	**48.95%**

Table 4.1: Accuracy of decoding methods on simulated data. [†]Columns with at least one masked character are considered as repeats.

and the block-based versions of the algorithms decreased the error rate by at most 3.3% compared to the non-block based methods. Surprisingly, block based methods had poorest performance in the block sensitivity and specificity and repeat sensitivity. These measures are closer to what block methods optimize, so we expected the opposite. Reason for this is perhaps due to the use of approximate intervals obtained by TRF and SRF. Note that the block-based methods use both predicted intervals and motifs to restrict the search space due to their high running time. In contrast, the remaining methods use only motifs (see section 4.5).

Indeed when we replace predicted motifs and intervals with the correct ones, accuracy of the block-based methods improved significantly (see Table 4.2). Results in Table 4.2 help us to quantify the effect of not using correct repeat intervals and thus can be seen as an upper bounds for the performance of our models achievable by improving repeat annotation. There is clearly room for improvements of repeat intervals; we could perhaps try to use other programs that detect tandem repeats, like ATRhunter [70] or mreps [41].

The change from predicted to real intervals had even more pronounced effect on the TTP model (see Table 4.3); using the correct intervals, the TTP model has only slightly worse error rate than the SFF, which is expectable, since the data were generated from the SFF model. However, using predicted intervals, TTP has a significantly higher error rate,

	Alignment	Repeat		Block	
Algorithm	error	sn.	sp.	sn.	sp.
SFF MPD	**3.37%**	**95.97%**	97.78%	**43.07%**	44.87%
SFF PD	3.53%	95.86%	97.87%	42.70%	47.37%
SFF BPD	3.51%	93.09%	**98.07**%	36.50%	41.67%
SFF BVA	3.91%	93.26%	97.96%	35.77%	40.66%
SFF VA	4.04%	95.29%	97.85%	42.70%	**48.95%**
SFF MPD$^\circ$	**3.02**%	**98.93**%	99.64%	77.01%	76.17%
SFF PD$^\circ$	3.42%	98.84%	99.51%	75.91%	80.93%
SFF BPD$^{\circ\circ\circ}$	3.21%	97.70%	**99.87%**	80.66%	**94.44**%
SFF BVA$^{\circ\circ\circ}$	3.71%	98.12%	99.85%	**81.75**%	92.18%
SFF VA$^\circ$	3.94%	98.54%	99.45%	75.55%	83.47%

Table 4.2: Accuracy of decoding methods using real motif and/or real intervals on simulated data. $^\circ$method uses motifs from the correct repeat blocks. $^{\circ\circ\circ}$method uses the correct motifs and intervals from the correct repeat blocks.

	Alignment	Repeat		Block	
Algorithm	error	sn.	sp.	sn.	sp.
3-state VA (baseline)	4.78%				
SFF BPD	**3.51%**	93.09%	**98.07%**	**36.50%**	**41.67%**
SFF BVA	3.91%	**93.26%**	97.96%	35.77%	40.66%
TTP BPD	5.05%	61.38%	97.48%	0.00%	0.00%
TTP BVA	6.17%	67.86%	96.51%	0.00%	0.00%
SFF BPD$^{\circ\circ\circ}$	**3.21%**	97.70%	99.87%	80.66%	**94.44**%
SFF BVA$^{\circ\circ\circ}$	3.71%	**98.12%**	99.85%	**81.75**%	92.18%
TTP BPD$^{\circ\circ}$	3.42%	60.45%	**99.90**%	0.36%	0.46%
TTP BVA$^{\circ\circ}$	3.83%	61.74%	99.88%	0.00%	0.00%

Table 4.3: Accuracy of decoding methods using real motif and/or real intervals on simulated data. $^{\circ\circ}$method uses intervals from the correct repeat blocks. $^{\circ\circ\circ}$method uses the correct motifs and intervals from the correct repeat blocks.

	Alignment	Repeat		Block	
Algorithm	error	sn.	sp.	sn.	sp.
3-state VA (baseline)	4.78%				
SFF MPD	3.37%	95.97%	97.78%	**43.07%**	**44.87**%
SFF MPD¥	3.63%	**96.03%**	97.74%	42.70%	43.33%
SFF MPD¥¥	**3.36**%	95.99%	**97.81**%	40.88%	43.08%

Table 4.4: Accuracy for marginalized posterior decoding with different models or real motifs. ¥Parameters of the three-state submodel were estimated from human-chicken alignment. ¥¥Parameters of SFF submodel were perturbed randomly.

which means that TTP is even more sensitive to the intervals that are used. The high error rate and low repeat and block prediction accuracy could be caused by improperly modeling the first repetition, because annotations of the TTP methods almost always skip the first copy of the repeat motif. Additionally, in TPP the tandem repeat motifs in sequences X and Y are not dependent between sequences, which can also increase error rate. This could be improved by merging the separate chain of prefix states $P_1, \ldots, P_K$ for sequences X and Y into a chain of match states aligning the first repetition.

Since we use the same model for generating data and testing, we tried to quantify the effect of misspecification of the model parameters. The SFF model has two principal types of parameters; parameters of the 3-state HMM and parameters of the Sunflower model. We investigate the effect of misspecification for both types independently. For the 3-state model, we have estimated the parameters from human-chicken alignments (chromosome 20 in human genome) instead of human-dog alignments. For the second type, we have perturbed the each parameter of the Sunflower model randomly by an additive term from 0.02 to 0.05. On these models, we run marginalized posterior decoding. Table 4.4 shows that our method is quite robust. Our method is more sensitive to change in the training alignments for the 3-state submodel, since the error rate for MPD increased by 8%, but it is still significantly better than the baseline method. Perturbing parameters for Sunflower submodel did not have significant effect on error rate (increase by 0.5%).

We were also investigating the use of the different different repeat finding methods for masking (see Table 4.5). We use the Sunflower model (modified for finding tandem repeats as described in section 4.2.2), TRF, and TRF with non-overlapping repeats (we select maximal scoring non-overlapping set of tandem repeats out of TRF output). We

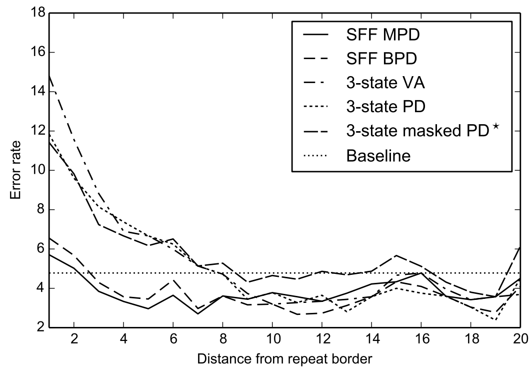
	Alignment	Repeat		Block	
Algorithm	error	sn.	sp.	sn.	sp.
3-state Viterbi (baseline)	4.78%				
3 state masked PD, TRF†	5.03	**99.23**	74.16	7.66	7.24
3 state masked PD, Sunflower†	5.21	98.30	68.65	**9.12**	7.53
3 state masked PD, TRF*†	**4.77**	88.17	**79.63**	7.66	**9.13**
3 state masked PD, $^{\circ\circ\circ\dagger}$	**4.50**	**100.00**	**97.23**	**45.99**	**61.13**
3 state masked VA, TRF†	5.89	**99.25**	73.29	5.47	5.14
3 state masked VA, Sunflower†	5.82	98.32	67.90	**5.84**	4.89
3 state masked VA, TRF*†	**5.10**	88.33	**78.68**	5.47	**6.47**
3 state masked VA, $^{\circ\circ\circ\dagger}$	**5.02**	**100.00**	**95.41**	**45.99**	**44.21**

Table 4.5: Accuracy of decoding method using the simple 3-state HMM and hard-masking. The best and the second best are bold. †Columns with at least one masked character are considered as repeats. $^{\circ\circ\circ}$method uses the correct motifs and intervals from the correct repeat blocks. *Non-overlapping set of repeats from TRF output with maximal total score is used.

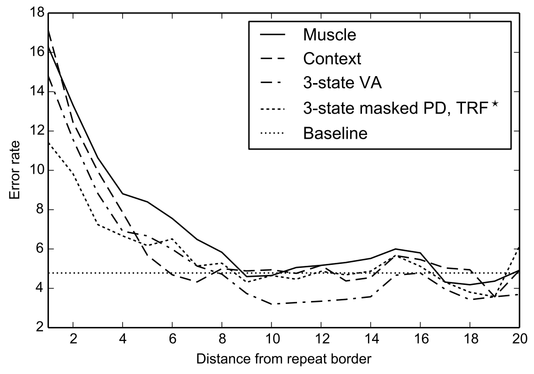
used hard-masking as described in section 4.1. For decoding we used both the VA and the PD with the 3-state model. Apart from masking the correct intervals, the best performing masking was one where we masked non-overlapping repeats from TRF.

In the last experiment, we investigate the relation between the error rate and the distance from the nearest repeat. An alignment column was considered as a repeat if in the correct alignment at least one base of the column was annotated as a repeat. The distance from the repeat is measured by the number of columns. When the distance from the repeat is high, our model is almost identical to a 3-state pHMM, and therefore the performance of both models should be similar in such regions. However, we do expect models incorporating repeats to be more precise in the regions near tandem repeats. Figure 4.5 contains plots for this metric for various combinations of parameters. In general, all algorithms have the highest error rate near boundaries of the repeat, with the error rate lowering with the distance from the repeat. Most methods reach the baseline error rate within distance 10 bases of the repeat border.

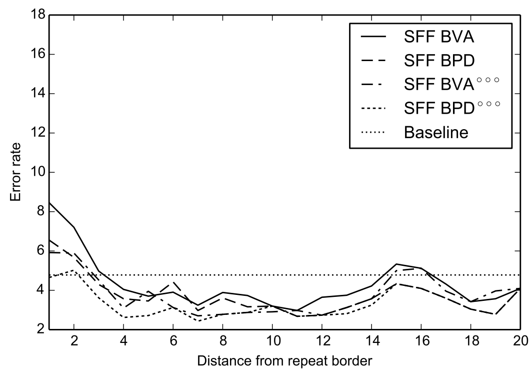
Figures 4.5a and 4.5b suggests that the right selection of decoding method can improve error rate near repeats, but most of the drop in the error rate was caused by using the



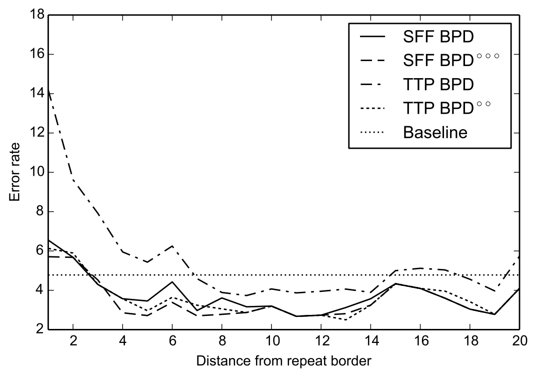
(a) Our methods and traditional methods



(b) Other methods



(c) Block decodings



(d) SFF and TTP

Figure 4.5: The relation between the error rate and the distance from the nearest repeat. On the Y axis is error rate of the algorithm; the number of incorrectly aligned columns. On the X axis is the distance from the nearest repeat. Baseline is the overall error rate, with no relation to the distance from the repeat border, for 3-state model with the VA. $^{\circ}$method uses motifs from the correct repeat blocks. $^{\circ\circ}$method uses intervals from the correct repeat blocks. $^{\circ\circ\circ}$method uses the correct motifs and intervals from the correct repeat blocks. *Non-overlapping set of repeats from TRF output with maximal total score is used.

SFF model. As with other metrics, TTP with correct intervals performs similarly to SFF model, but TTPs error rate near repeats raises significantly where intervals were obtained from the TRF (see Figure 4.5d). Figure 4.5c demonstrates that for block methods, using BVA instead of BPD almost double the error rate near repeat. Overall these plots confirm our expectation that the improvement of the SFF model is concentrated near tandem repeat borders.

Chapter 5

Conclusion

In this thesis we studied the use of hidden Markov models for sequence annotation and sequence alignment and provided both theoretical and practical contributions.

Chapter 3 considers two-stage decodings for sequence annotation. In a two stage decoding, we at first compute the guide and then find an annotation optimizing a chosen gain function using the guide as a restriction. Guides can be used to decrease running time, but we have shown experimentally that guides can also improve the accuracy. Then we focus on the computational problems of computing guides. We gave the combinatorial proof that the most probable set problem is NP-hard. We also show that this problem is fixed-parameter tractable when the fixed parameter is the number of states of the HMM. It is not clear, if the most probable set problem is in NP. Then we show a reduction from 3-SAT to the most probable restriction problem proving that this problem is NP-complete. Similarly to the most probable set problem, this problem is also fixed-parameter tractable. Finally, we focus on the most probable footprint problem. We showed a constant-sized HMM for which finding the most probable footprint is NP-hard even if we use the identity function as a labeling function. We also show how to alter the proof of the most probable footprint to the proofs of the most probable set and the most probable restriction.

There are still some open problems left. The most probable footprint is a special case of the most probable ball problem (with the border shift sum distance, where r is the radius) which was studied by Brown and Truszkowski [16]; we described it in Section 2.2.4. This problem is NP-hard even for $r = 0$ [16], if multiple states can have same label. Additionally, if $r \geq n$, where n is the length of the input sequence, the most probable ball problem is equivalent to the most probable footprint problem. Our results thus imply that the most probable ball problem is NP-hard even if all states have unique labels. It is still open problem, if the most probable ball problem is NP-hard for $r < n$ and HMMs with the identity function as an annotation function.

From the practical point of view, two-stage algorithms could be used for pair hidden Markov models, or in different application domains; for example the gene finding.

In Chapter 4 we study the sequence alignment problem using pair HMMs. We propose the new SFF model that incorporates tandem repeats, which are prevalent in the genomic sequences. In addition to the new model, we explored several decoding criteria including, the block Viterbi algorithm and the block posterior decoding, which treat tandem repeats as blocks. On the simulated data, the SFF model alone decreased the error rate by at least 15% compared to the standard three-state model with the Viterbi algorithm. The decrease in the error rate was concentrated near repetitive intervals.

There are several possible directions for further research. For example our model does not take into account dependencies between repetitions, and therefore tandem repeats in different sequences are independent (apart from using the same motif). We could model the evolution of repeats more realistically either in the SFF model, or we could incorporate it into the decoding algorithm. Similar improvement can be done into the TTP model, which does not model the first repetition well and allows the use of the different motifs in homologous repeats. This problem could be solved by modeling the first repetition as a pair profile HMM and other repetitions would be modeled by the TANTAN model. Other improvement can be incorporating additional submodels modeling other genomic features, for example gene structures. It might be interesting to study the ways of incorporating different, but overlapping features into the same model (for example the tandem repeats inside of gene structures). Finally, we can extend this method to align multiple sequences. From the practical point of view, our software needs to be optimized, so it can be used on genomic sequences of arbitrary length.

We studied the decoding algorithms in all chapters of this thesis. The selection of a proper decoding method is neglected problem, and many applications simply choose the Viterbi algorithm. While it is true that for many applications the Viterbi algorithm is optimal, our experience shows that choosing the proper decoding method can significantly improve the accuracy of predicted annotations or alignments. We believe that the selection of the decoding function should be an important part of designing a method that uses a probabilistic model, because domain-specific decoding algorithms can decrease some statistical biases and can be used as a compensation for simplifications done during designing of the HMM. Finally, there are many interesting computational problems that arise in the area of decoding algorithms, from the most probable annotation problem to the optimization of custom gain functions.

Bibliography

[1] Alok Aggarwal, Maria Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric Applications of a Matrix-Searching Algorithm. *Algorithmica*, 2:195–208, 1987.

[2] M Alexandersson, S Cawley, and L Pachter. SLAM: Cross-Species Gene Finding and Alignment with a Generalized Pair Hidden Markov Model. *Genome Res.*, 13:496–502, Mar 2003.

[3] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman. Basic Local Alignment Search Tool. *J Mol Biol*, 215(3):403–10, Oct 1990.

[4] S F Altschul, T L Madden, A A Schaffer, J Zhang, Z Zhang, W Miller, and D J Lipman. Gapped BLAST and PSI-BLAST: a New Generation of Protein Database Search Programs. *Nucleic Acids Res*, 25(17):3389–402, Sep 1997.

[5] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics—Doklady*, 11(5):1209–1210, 1970.

[6] G. Benson. Sequence alignment with tandem duplication. *Journal of Computational Biology*, 4(3):351–357, 1997.

[7] G. Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2):573–580, Jan 1999.

[8] Sèverine Bérard, François Nicolas, Jérôme Buard, Olivier Gascuel, and Eric Rivals. A fast and specific alignment method for minisatellite maps. *Evolutionary Bioinformatics Online*, 2:303, 2006.

[9] E Birney, M Clamp, and R Durbin. GeneWise and Genomewise. *Genome Res.*, 14:988–995, May 2004.

[10] Robert K Bradley, Adam Roberts, Michael Smoot, Sudeep Juvekar, Jaeyoung Do, Colin Dewey, Ian Holmes, and Lior Pachter. Fast Statistical Alignment. *PLoS Comput Biol*, 5(5):e1000392, 05 2009.

[11] Nick Bray, Inna Dubchak, and Lior Pachter. AVID: A Global Alignment Program. *Genome Res*, 13(1):97–102, Jan 2003.

[12] Alvis Brazma, Helen Parkinson, Thomas Schlitt, and Mohammadreza Shojatalab. A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays. http://www.ebi.ac.uk/microarray/biology_intro.html, 2001.

[13] Broňa Brejová, Daniel G Brown, and Tomáš Vinař. Vector seeds: An extension to spaced seeds. *Journal of Computer and System Sciences*, 70(3):364 – 380, 2005.

[14] Broňa Brejová, Daniel G Brown, and Tomáš Vinař. The most probable annotation problem in HMMs and its application to bioinformatics. *J. Comput. Syst. Sci.*, 73:1060–1077, November 2007.

[15] Broňa Brejová, Daniel G Brown, Ming Li, and Tomáš Vinař. ExonHunter: a comprehensive approach to gene finding. *Bioinformatics*, 21 Suppl 1:i57–65, Jun 2005.

[16] Daniel G Brown and Jakub Truszkowski. New decoding algorithms for hidden Markov models using distance measures on labellings. In *Asia Pacific Bioinformatics Conference (APBC)*, 2010.

[17] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78 – 94, 1997.

[18] Reed A Cartwright. Problems and solutions for estimating indel rates and length distributions. *Mol Biol Evol*, 26(2):473–80, Feb 2009.

[19] K M Chao, W R Pearson, and W Miller. Aligning two sequences within a specified diagonal band. *Comput Appl Biosci*, 8(5):481–7, Oct 1992.

[20] Maxime Crochemore, Gad M Landau, and Michal Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 679–688, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[21] Miklós Csűrös and Bin Ma. Rapid Homology Search with Two-Stage Extension and Daughter Seeds. In Lusheng Wang, editor, *Computing and Combinatorics*, volume 3595 of *Lecture Notes in Computer Science*, pages 104–114. Springer Berlin / Heidelberg, 2005.

[22] R Durbin, S Eddy, A Krogh, and G Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[23] Robert C Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[24] Valerio Freschi and Alessandro Bogliolo. A lossy compression technique enabling duplication-aware sequence alignment. *Evolutionary Bioinformatics Online*, 8:171, 2012.

[25] M. C. Frith. A new repeat-masking method enables specific detection of homologous sequences. *Nucleic Acids Res*, 39(4):e23, 2011.

[26] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[27] Rita Gemayel, Marcelo D Vinces, Matthieu Legendre, and Kevin J Verstrepen. Variable tandem repeats accelerate evolution of coding and regulatory sequences. *Annual Review of Genetics*, 44:445–477, 2010.

[28] O Gill and Courant Inst Nyu. PLANAR: RNA Sequence Alignment using Non-Affine Gap Penalty and Secondary Structure, 2006.

[29] Ofer Gill, Yi Zhou, and Bud Mishra. Aligning Sequences with Non-Affine Gap Penalty: PLAINS Algorithm, a Practical Implementation, and its Biological Applications in Comparative Genomics. In *Advances In Bioinformatics And Its Applications*, 2004.

[30] J A Grice, R Hughey, and D Speck. Reduced space sequence alignment. *Comput Appl Biosci*, 13(1):45–53, Feb 1997.

[31] Samuel S Gross, Chuong B Do, Marina Sirota, and Serafim Batzoglou. CONTRAST: a discriminative, phylogeny-free approach to multiple informant de novo gene prediction. *Genome Biology*, 8(12):R269, 2007.

[32] Stéphane Guindon and Olivier Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696 – 704, 2003.

[33] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology.* Cambridge Univ. Press, 2007.

[34] Glenn Hickey and Mathieu Blanchette. A probabilistic model for sequence alignment with context-sensitive indels. *Journal of Computational Biology*, 18(11):1449–1464, 2011.

[35] DS Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18:341–343, June 1975.

[36] AK Hudek and DG Brown. FEAST: Sensitive Local Alignment with Multiple Rates of Evolution. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 8(3):698 –709, may-june 2011.

[37] Alexander Karl Hudek. *Improvements in the Accuracy of Pairwise Genomic Alignment.* PhD thesis, University of Waterloo, 2010.

[38] Lukas Kall, Anders Krogh, and Erik L L Sonnhammer. An HMM posterior decoder for sequence feature prediction that includes homology information. *Bioinformatics*, 21 Suppl 1:i251–257, 2005.

[39] Evan Keibler, Manimozhiyan Arumugam, and Michael R Brent. The Treeterbi and Parallel Treeterbi algorithms: efficient, optimal decoding for ordinary, generalized and pair HMMs. *Bioinformatics*, 23(5):545–54, Mar 2007.

[40] W James Kent. BLAT–the BLAST-like alignment tool. *Genome Res*, 12(4):656–64, Apr 2002.

[41] Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Research*, 31(13):3672–3678, 2003.

[42] Peter Kováč, Broňa Brejová, and Tomáš Vinař. Aligning sequences with repetitive motifs. In *Information Technologies - Applications and Theory (ITAT)*, pages 41–48, 2012.

[43] Anders Krogh, Björn Larsson, Gunnar von Heijne, and Erik LL Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *Journal of Molecular Biology*, 305(3):567 – 580, 2001.

[44] J Lember and AA Koloydenko. A constructive proof of the existence of viterbi processes. *Information Theory, IEEE Transactions on*, 56(4):2017 –2033, april 2010.

[45] A Lempel and J Ziv. On the Complexity of Finite Sequences. *Information Theory, IEEE Transactions on*, 22(1):75 – 81, jan 1976.

[46] David A Levin, Yuval Peres, and Elizabeth L Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[47] Pachter Lior, Alexandersson Marina, and Simon Cawley. Applications of Generalized Pair Hidden Markov Models to Alignment and Gene Finding Problems. *Journal of Computational Biology*, 9, July 2004.

[48] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities. *Bioinformatics*, 26(16):1958–64, Aug 2010.

[49] DV Lu, RH Brown, M Arumugam, and MR Brent. Pairagon: a highly accurate, HMM-based cDNA-to-genome aligner. *Bioinformatics*, 25:1587–1593, Jul 2009.

[50] Gerton Lunter, Andrea Rocco, Naila Mimouni, Andreas Heger, Alexandre Caldeira, and Jotun Hein. Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *Genome Res*, 18(2):298–309, Feb 2008.

[51] Rune B Lyngsø and Christian N S Pedersen. The consensus string problem and the complexity of comparing hidden Markov models. *Journal of Computer and System Sciences*, 65(3):545 – 569, 2002.

[52] Bin Ma, John Tromp, and Ming Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–5, Mar 2002.

[53] W H Majoros, M Pertea, and S L Salzberg. TigrScan and GlimmerHMM: two open source ab initio eukaryotic gene-finders. *Bioinformatics*, 20(16):2878–9, Nov 2004.

[54] WH Majoros, M Pertea, and SL Salzberg. Efficient implementation of a generalized pair hidden Markov model for comparative gene finding. *Bioinformatics*, 21:1782–1788, May 2005.

[55] P. W. Messer and P. F. Arndt. The majority of recent short DNA insertions in the human genome are tandem duplications. *Mol Biol Evol*, 24(5):1190–1197, 2007.

[56] IM Meyer and R Durbin. Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, 18:1309–1318, Oct 2002.

[57] EW Myers and W Miller. Approximate Matching of Regular Expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[58] Michal Nánási, Tomáš Vinař, and Broňa Brejová. The Highest Expected Reward Decoding for HMMs with Application to Recombination Detection. In A. Amir and L. Parida, editors, *Combinatorial Pattern Matching, 21th Annual Symposium (CPM 2010)*, volume 6129 of *Lecture Notes in Computer Science*, pages 164–176, Brooklyn, New York, USA, 2010. Springer.

[59] Michal Nánási, Tomáš Vinař, and Broňa Brejová. Probabilistic approaches to alignment with tandem repeats. *Algorithms for Molecular Biology*, 9(1):3, 2014.

[60] Michal Nánási. Biological sequence annotation with hidden Markov models. Master's thesis, Comenius University, 2010.

[61] Lior Pachter, Marina Alexandersson, and Simon Cawley. Applications of generalized pair hidden Markov models to alignment and gene finding problems. *Journal of Computational Biology*, 9(2):389–399, 2002.

[62] D. L. Robertson, J. P. Anderson, J. A. Bradac, J. K. Carr, B. Foley, R. K. Funkhouser, F. Gao, B. H. Hahn, M. L. Kalish, C. Kuiken, G. H. Learn, T. Leitner, F. McCutchan, S. Osmanov, M. Peeters, D. Pieniazek, M. Salminen, P. M. Sharp, S. Wolinsky, and B. Korber. HIV-1 nomenclature proposal. *Science*, 288(5463):55–6, 2000.

[63] Michael Sammeth and Jens Stoye. Comparing tandem repeats with duplications and excisions of variable degree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):395–407, 2006.

[64] R Satija, J Hein, and G A Lunter. Genome-wide functional element detection using pairwise statistical alignment outperforms multiple genome footprinting techniques. *Bioinformatics*, 26(17):2116–20, Sep 2010.

[65] Anne-Kathrin Schultz, Ming Zhang, Thomas Leitner, Carla Kuiken, Bette Korber, Burkhard Morgenstern, and Mario Stanke. A jumping profile Hidden Markov Model

and applications to recombination sites in HIV and HCV genomes. *BMC Bioinformatics*, 7:265, 2006.

[66] Rastislav Šrámek, Broňa Brejová, and Tomáš Vinař. On-Line Viterbi Algorithm for Analysis of Long Biological Sequences. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *Algorithms in Bioinformatics*, volume 4645 of *Lecture Notes in Computer Science*, pages 240–251. Springer Berlin / Heidelberg, 2007.

[67] Jakub Truszkowski and Daniel Gregory Brown. More accurate recombination prediction in HIV-1 using a robust decoding algorithm for HMMs. *BMC Bioinformatics*, 12:168, 2011.

[68] Tomáš Vinař. *Enhancements to Hidden Markov Models for Gene Finding and Other Biological Applications*. PhD thesis, University of Waterloo, October 2005.

[69] Oren Weimann. *Accelerating dynamic programming*. PhD thesis, Massachusetts Institute Of Technology, Cambridge, MA, USA, 2009.

[70] Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding approximate tandem repeats in genomic sequences. *Journal of Computational Biology*, 12(7):928–942, 2005.

[71] Marketa Zvelebil and Jeremy Baum. *Understanding Bioinformatics*. Garland Science, 1 edition, 2007.