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

Biological sequence annotation with hidden Markov models

(Master's thesis)

Michal Nánási

Thesis advisor: Mgr. Bronislava Brejová, PhD.

Bratislava, 2010

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Biological sequence annotation with hidden Markov models

Master's Thesis

Study Program: Informatics
Branch of Study: 9.2.1 Informatics
Supervisor: Mgr. Broňa Brejová PhD.
Bratislava, 2010

Bc. Michal Nánási

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.

Contents

List of Tables

List of Figures

Abstract

Author:	Michal Nánási
Title:	Biological sequence annotation with hidden Markov models
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Departement of Computer Science
Supervisor:	Mgr. Broňa Brejová PhD.
Number of Pages:	54
Year:	2010

Hidden Markov models (HMMs) are an important tool for modeling biological sequences and their annotations. By sequence annotation we mean an assignment of labels to each symbol according to its function. For example, in gene finding we want to distinguish the regions of DNA that encode proteins from surrounding non-coding sequence. A hidden Markov model defines a probability distribution over all annotations of a given sequence X.

HMMs are traditionally decoded by the Viterbi algorithm. Viterbi algorithm finds the most probable annotation for a subset of HMMs. In general, the sequence annotation is NP-hard and the Viterbi algorithm is used as a heuristic algorithm.

Recently it has been shown that other decoding methods give better results than Viterbi in specific applications. We propose a new decoding method that allows uncertainty in region boundaries by considering all annotations with slightly shifted boundaries as the same. Our method (the highest expected reward decoding – HERD) is an extension of the framework of maximum expected boundary accuracy decoding introduced by Gross *et al.*.

We evaluate HERD on the problem of detecting viral recombination in the HIV genome and compare it with an existing tool called jumping HMM which uses the Viterbi algorithm. HERD has better prediction accuracy in terms of the feature sensitivity and specificity, where feature is a single-colored block in an annotation.

Keywords: hidden Markov models, sequence annotation, viral recombination

Chapter 1

Introduction

In this thesis we propose a new algorithm for decoding hidden Markov models (HMMs) and apply it to the problem of recombination detection in HIV genomic sequences. Modern sequencing technologies produce large amounts of data and therefore we need efficient algorithms for analyzing such sequences. The HMMs have numerous applications in analysis of biological sequences, for example in gene finding [BK97, BBLV05], prediction of the structure of the transmembrane proteins [KLvHS01] or the detection of protein family membership [Edd98]. We are particularly interested in the use of HMMs for annotations of biological sequences. By annotating sequences we mean assignment of labels to each symbol according to its function. For example, in gene finding we want to distinguish between regions of DNA that encode proteins from non-coding sequences. For our needs, a DNA molecule can be represented as a finite sequence of four nucleotides – adenine, cytosine, guanine and thymine or `A,C,G,T` in short.

HMMs are probabilistic generative models. They are similar to probabilistic finite state machines. Every HMM is characterized by a finite set of states, emission distribution of each state and the transition probabilities between states. In each step an HMM emits a symbol according to its current state, then it changes the state according to its transition distribution. Every HMM defines a probability distribution over all state paths and sequences. Every state has assigned a meaning or function, so we can deduce the function of particular parts of the sequence from a state path. The problem is that in almost all applications we observe the sequence, but the state path remains hidden. If we fix the observed sequence, the HMM defines a probability distribution of state paths. The most probable state path can be found by the classical Viterbi algorithm in polynomial time [For73]. Formal definition of the HMM and description of several known algorithm can be found in Chapter 2.

Recently it has been shown that other methods give better result than the Viterbi algorithm in specific applications [KKS05, GDSB07, BT10]. One reason is that in various HMMs several states may have same meaning. For example, most HMMs used for gene

finding have many states that model coding parts of the DNA. We can label states with color so that the states generating sequences with same function will have the same color. Each state path can be then transformed into an annotation which is a sequence of colors. Annotations are different from state paths, because multiple state paths can share the same annotation. Therefore the probability of an annotation is the sum of probabilities of all state paths with that annotation. In some applications, the most probable annotation can be more accurate than the most probable state path. While finding the most probable state path can be done in polynomial time, finding the most probable annotation is NP-hard [LP02, BBV07].

To overcome this problem, we seek objectives that can be computed effectively and have better results than the Viterbi algorithm for a particular task. As our application domain we choose the viral recombination detection problem. Genomes of some viruses, such as human immunodeficiency virus (HIV), can form of a mosaic recombination of several types of the same virus [RAB+00]. Our goal is to detect if a sequence of the HIV genome belongs to a known family or if it is a recombination of viruses from different families. Current tools for such application use the Viterbi algorithm [SZL+06] (see details in Chapter 3). This application is important for monitoring the HIV epidemics.

In Chapter 4 we define a new objective function appropriate to our application domain and present a polynomial-time algorithm for finding an annotation optimising our objective. To bypass NP-hardness of finding the most probable annotation, our objective function assigns score locally for each recombination point. Finding the exact recombination point is difficult, because there the annotations with slightly shifted recombination points have similar probabilities. Our gain function overcomes this problem by considering all nearby recombination points as equivalent. We expressed our objective in terms of gain functions [HKS+09]. We will also consider several variants of our objective function to improve the performance of our algorithm.

Finally, in Chapter 5 we show the results of several experiments on artificial and real data and compare our algorithm with other decoding methods. We show, that our decoding method have higher predicting accuracy than the existing tool that uses the Viterbi algorithm [SZL+06] and that our decoding method is effective for recombination detection in HIV genome.

Chapter 2

Hidden Markov Models

A *hidden Markov model (HMM)* [DEKM98] is a frequently used generative probabilistic model, that generates finite sequences over some alphabet. HMMs have numerous applications in computational biology, but also in speech recognition, image processing and other areas. In this chapter we define these model and related problems. We show some existing algorithms and their limitations.

2.1 Definitions

An HMM can be seen as probabilistic finite state machine [HU79] that generates finite sequences over some alphabet Σ. Each state has its own emission distribution and transition distribution. Every second the machine will every second toss a coin and generate a symbol from the emission distribution of the current state. After that the machine will toss a coin again and move to another state. Once a machine reaches a final state, it halts. We will define an extended variant of HMMs, allowing silent states.

Definition 1 (Hidden Markov Model with silent states (HMM)) *Let* $H = (\Sigma, V, Q, e, a)$*, where* Σ *is a finite* alphabet*,* $V = \{v_0, v_1, \ldots, v_{M-1}\}$ *is the set of* states *where* $M = |V|$*,* $Q \subseteq V$ *is the set of* silent states*,* $e : \Sigma \times V \rightarrow < 0, 1 >$ *are* emission probabilities *and* $a : V \times V \rightarrow < 0, 1 >$ *are* transition probabilities*.*

H is hidden Markov model with silent states, if it satisfies following conditions:

1. *For every* $u \in V \backslash Q$, $\sum_{a \in \Sigma} e_{a,u} = 1$
2. *For every* $u \in Q, a \in \Sigma$, $e_{a,u} = 0$
3. *For every* $u \in V \backslash \{v_{M-1}\}$, $\sum_{v \in V} a_{u,v} = 1$
4. *For every* $u \in V$, $a_{v_{M-1},u} = 0$

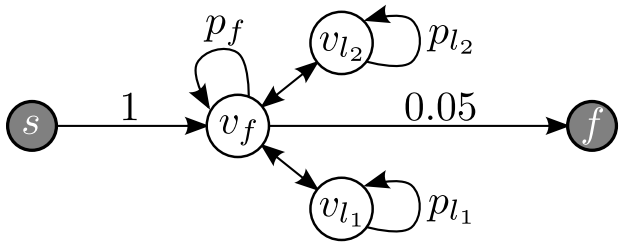

Figure 2.1: The HMM that models the process from example 1. State v_f represents the fair die, state v_{l_1} represents the first loaded die, and state v_{l_2} represents the second loaded die.

5. $v_0, v_{M-1} \in Q$

We will denote the state v_0 as the start state *and the state v_{M-1} as the* final state.

The conditions 1, 2 and 3 ensure that every state has a proper emission and transition distribution. As we can see, the silent states does not emit any symbol. The fourth condition states that once the process goes into final state it ends. The last condition is only technical; we use it to simplify equations.

Notation 1 *We say that there is a* transition *from state $u \in V$ to state $v \in V$, if and only if $a_{u,v} > 0$. We write that transition as $u \to v$. Let $T = \{u \to v\}$ be the set of all transitions. Finally, a* state path *is a finite sequence over V which start with start state and end with final state.*

Example 1 *Inspired by [DEKM98]. Consider the following process. We play dice in a casino. There are three dice. One fair die and two loaded. In the casino they usually use the fair die, but sometimes they secretly change it for the loaded one. One of the loaded dice has probability* 0.5 *of one and probability* 0.1 *of the numbers from two to six. The second loaded die has probability* 0.5 *of six and probability 0.1 of the numbers from one to five. The casino changes the die with certain probability after each roll. After a roll with the fair die, the casino will keep the fair die with probability p_f, with probability $0.95 - p_f$ will change the die and with probability* 0.05 *casino will end the game. The casino chooses one of the loaded dices with equal probability. After each roll with the first loaded die casino will keep that die with probability p_{l_1} and witch probability $1 - p_{l_1}$ will change that die to the fair one. The same applies tor the second loaded dice, but it has the keeping probability p_{l_2}.*

This process can be modeled by HMM in Figure 2.1. Start state s and final state f are the only silent states. Emission distribution of the state v_f is $e_{f_v,x} = 1/6, x \in \{1, 2, \ldots, 6\}$.

State v_{l_1} emits the symbol 1 with probability 1/2 and other symbols with probability 1/10. Therefore $e_{f_{l_1},1} = 1/2$ and $e_{f_{l_2},x} = 1/10, x \in \{2,3,\ldots,6\}$. The emission probabilities of state v_{l_2} are $e_{f_{l_2},x} = 1/10, x \in \{1,2,\ldots,5\}$ and $e_{f_{l_2},6} = 1/2$.

This HMM have 9 transitions $T = \{s \to v_f, v_f \to v_f, v_f \to f, v_f \to v_{l_1}, v_f \to v_{l_2}, v_{l_1} \to v_{l_1}, v_{l_1} \to v_f, v_{l_2} \to v_{l_2}, v_{l_2} \to v_f\}$. Transition probabilities are $a_{s,v_f} = 1$, $a_{v_f,v_f} = p_f$, $a_{v_f,f} = 0.05$, $a_{v_f,v_{l_i}} = (0.95 - p_f)/2$, $a_{v_{l_i},v_{l_i}} = p_{l_i}$ and $a_{v_{l_i},v_f} = 1 - p_{l_i}, i \in \{1,2\}$.

Sequences $(s, v_f, v_f, v_f, v_{l_1}, v_f, f)$ or $(s, v_f, v_f, v_f, v_{l_2}, f)$ are state paths although the second sequence contains nonexistent transition $v_{l_2}f$. Sequence (v_f, v_f, v_f, f) is not state path, because it does not start in state s.

A hidden Markov model is a machine that generates pairs (π, X), where π is a state path and X is a sequence. Usually we observe only the sequence and the state path remains hidden (this is the reason, why we called them *hidden* Markov models). Now we will define the probability distribution over all pairs of (π, X), but first we need several technical definitions.

Definition 2 *Let $\pi = \pi_1\pi_2\ldots\pi_l, \pi_i \in V$ be a state path, and $0 < i_1 < i_2 < \cdots < i_k \leq l$ be the indexes of all non-silent states from π. Then $\pi^Q = \pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}$ is the* non-silent state path *from path π.*

Example 2 *Consider the HMM from the example 1. Let π the state path $\pi = (s, v_f, v_f, v_f, v_{l_1}, v_f, v_{l_2}, v_f, f)$ be an state path. Then corresponding non-silent state path is $\pi^Q = (v_f, v_f, v_f, v_{l_1}, v_f, v_{l_2}, v_f)$.*

Definition 3 *Let $H = (\Sigma, V, Q, e, a)$ be an HMM and $X = x_1x_2\ldots x_n$ be a finite sequence over Σ. Let $\pi = \pi_1\pi_2\ldots\pi_l$ be a state path and $\pi_1^Q\pi_2^Q\ldots\pi_N^Q$ be a non-silent state path from π. Path π is* consistent *with X if and only if $N = n$.*

In other words, the state path π is consistent with the sequence X if the sequence X could be generated by π, at least in terms of the number of its non-silent states.

Definition 4 *Let H be an HMM and X be a sequence of length n. Let π be the state path and π^Q the corresponding non-silent state path. If π is not consistent with X, then probability that H will generate X and π is 0. Otherwise*

$$\Pr(X, \pi) = \left(\prod_{i=1}^{l-1} a_{\pi_i,\pi_{i+1}}\right) \cdot \left(\prod_{i=1}^{n} e_{\pi_i^Q,x_i}\right) \tag{2.1}$$

Example 3 *Let H be the HMM from example 1. Let $X = 1263526546$ be the sequence generated by H with state path $\pi = (s, v_f, v_f, v_f, v_f, v_{l_1}, v_{l_1}, v_f, v_{l_2}, v_{l_2}, v_f, f)$. Then the probability of the sequence X generated by state path π is*

$$\Pr(X, \pi) = 1 \cdot p_f^4 \cdot \frac{0.95 - p_f}{2} \cdot (1 - p_{l_1}) \cdot \frac{0.95 - p_v}{2} \cdot p_{l_2} \cdot (1 - p_{l_2}) \cdot 0.05 \cdot \left(\frac{1}{6}\right)^4 \cdot \left(\frac{1}{10}\right)^2 \cdot \frac{1}{6} \cdot \left(\frac{1}{10}\right)^2 \cdot \frac{1}{6}$$

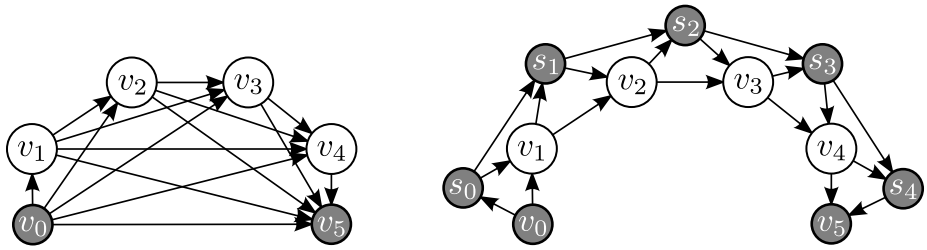

Figure 2.2: Demonstration of reducing the number of transitions. The original HMM is on the left and the resulting HMM is on the right.

Note that if an HMM has a state v_i from which the final state is not accessible (and v_i is accessible from the start state), then the sum of the probabilities of all possible state paths and sequences will be less than one. This is the case because all state paths that goes through v_i will never reach final state and will not be counted. If the HMM contains such states, there is a nonzero probability that it will generate infinite sequence. On the other hand, states that are not accessible from the start state can be removed from an HMM without affecting its probability distribution.

HMMs can be defined in many other ways. For example instead of the start state we could use a special initial distribution over all states, which represents the probability of starting in a particular state. We choose this particular definition, because we need silent states in our application and the silent start state is equivalent to an explicit initial distribution. We can use silent states to reduce the number of transitions. For example consider an HMM consisting of n states $v_0, v_1, \ldots, v_{n-1}$ where state v_i is connected with all states $v_j, j > i$. This HMM has $\Theta(n^2)$ transitions. We can replace these transitions by $n-1$ silent states $s_0, s_2, \ldots, s_{n-2}$ with transitions $s_i \to s_{i+1}, i < n-2$, $v_i \to s_i, i < n-1$ and $s_i \to v_{i+1}, i < n-1$ (in Figure 2.2. This new HMM has an equivalent structure in a sense that it has the same set of non-silent paths with non-zero probability. However the new HMM has fewer parameters and it cannot represent all distributions possible in the original HMM. Note that the reduction to $\Theta(n)$ transitions may decrease running time considerably.

Although silent states are sometimes useful for reducing the overall size of a model, they do not increase the expressive power of HMMs as we show in the next theorem.

Theorem 1 *For every HMM H there exists HMM H' that does not have silent states except start and final state and the distributions of the sequences and non-silent state paths of both HMMs are the same.*

Proof. We show how to remove a single silent state. We can repeat this process until there will be no silent states (except the start and final state).

Let $v_i \in Q$ be the silent state we want to remove. If $a_{v_i,v_i} > 0$ then we have to remove that transition. This transition corresponds to paths v_i^k. Sum of the probabilities of those state paths is $\frac{a_{v_i,v_i}}{1-a_{v_i,v_i}}$. Therefore if v_i has more then one transition, we will remove transition $v_i \to v_i$ from the model and multiply transitions $a_{v_i,v_j} > 0$ by $\frac{1}{1-a_{v_i,v_i}}$. If v_i has only one transition $v_i \to v_i$, we can not simply remove it. Therefore we change the silent state v_i into a non-silent state. We can do this because there is no state path that goes through v_i (every state path must end in the final state).

Now the silent state v_i does not have self-loop and we can remove it. Let $v_k \to v_i$ and $v_i \to v_j$ be transitions in H. We will remove those transitions from the model and add $a_{v_k,v_i} \cdot a_{v_i,v_j}$ to the probability of transition $v_k \to v_j$. We have an equivalent HMM H' without state v_i.

Although our definition of an HMM allows silent states we will enforce some restriction on their usage in order to simplify algorithms and proofs. We disallow silent states with self-loops as well as longer cycles consisting solely from silent states.

Definition 5 *The HMM H is* well defined*, if and only if it does not contain a cycle consisting solely of silent states.*

Note that self-loop is also a cycle.

2.2 Forward Algorithm

Given a well-defined HMM H and sequence X of length n, we may want to compute the probability that sequence X was generated by H. As the state path is hidden, we have to sum over all possible state paths and therefore

$$\Pr(X) = \sum_{\pi} \Pr(X, \pi) \tag{2.2}$$

Computing this quantity in straightforward way will give us an exponential-time algorithm. Therefore we use the forward algorithm [DEKM98]. The forward algorithm is a straightforward use of the dynamic programming technique. Let $F[i,u]$ be the sum of the probabilities of all state paths that start in v_0, generate $X[1:i]$ and end in state u. Then $\Pr(X) = F[n, v_{|V|-1}]$. F can be computed by the following equations.

$$F[0, v_0] = 1 \tag{2.3}$$

$$F[i, v_j] = \sum_{v_k \to v_j} F[i, v_k] a_{v_k,v_j}, v_j \in Q, i \neq 0 \vee v_j \neq v_0 \tag{2.4}$$

$$F[i, v_j] = \sum_{v_k \to v_j} F[i-1, v_k] a_{v_k,v_j} e_{v_j,x}, i > 0, v_j \in V \backslash Q \tag{2.5}$$

We can compute F in order of increasing increasing i and in appropriate order of states v_i so the all terms on the right-hand side of 2.4 are evaluated before $F[i, v_j]$. In HMMs without silent states the order may be arbitrary; 2.4 is never used. Silent states need to be ordered topologically so that for every two silent states v_i, v_j, if $a_{v_i,v_j} > 0$ then $i < j$. Now we can compute $F[i, v_j]$ in the order of increasing j.

Theorem 2 *Let $F[i, v_j]$ be defined by equations 2.3,2.4 and 2.5. Then $F[i, v_j]$ is sum of the probabilities of all state paths that start in v_0, generate $X[1 : i]$ and end in state u.*

Proof. Induction on K, where $K = i \cdot |V| + j$.

1. When $K = 0$, there is nothing to prove, equation 2.3 holds.

2. Let induction hypothesis hold for all $0 \leq i' \cdot |V| + j' \leq K$. Let $i \cdot |V| + j = K + 1$.

   Let $C = \{v_k | a_{v_k,v_j} > 0\}$ be the set of states from which is v_j accessible. Therefore all state paths that generate $X[1 : i]$ and ends in v_j have last but one state from C. We can split those paths into $|C|$ disjoint sets $G_{v_k}, v_k \in C$. We have to consider two options.

   (a) State v_j is silent. This means that X_i was not generated by v_j and therefore we know from induction hypothesis and definition 2.1 that $\Pr(G_{v_k}, X[1 : i]) = F[i, v_k] \cdot a_{v_k,v_j}$. Because these sets are disjoint, we obtain

   $$F[i, v_j] = \sum_{v_k \in C} \Pr(G_{v_k}, X[1 : i]) = \sum_{v_k \in C} F[i, v_k] \cdot a_{v_k,v_j}$$

   (b) State v_j is not silent. Then X_i was generated by v_j, so we know from induction hypothesis and definition 2.1 that $\Pr(G_{v_k}, X[1 : i]) = F[i, v_k] \cdot a_{v_k,v_j} \cdot e_{v_j,X_i}$. Because sets G_{v_k} are disjoint, the probability

   $$F[i, v_j] = \sum_{v_k \in C} \Pr(G_{v_k}, X[1 : i]) = \sum_{v_k \in C} F[i - 1, v_k] \cdot a_{v_k,v_j} \cdot e_{v_k,X_i}$$

Probability $\Pr(X)$ can be also computed by the backward algorithm [DEKM98]. It is based on same principle as the forward algorithm, but it proceeds backwards. It computes the probability that a state path starts in particular state and generates the suffix of the sequence. Formally, $B[i, v_j]$ is the probability, that H generates the $X[i : n]$ and starts in v_j (and v_j does not emit any symbol, even if it is not a silent state). Then $\Pr(X) = B[i, v_0]$. Values $B[i, v_j]$ can be computed by the following equations:

$$B[n, v_{|V|-1}] = 1 \tag{2.6}$$

$$B[i, v_j] = \sum_{v_j \to v_k, v_k \in Q} B[i, v_k] \cdot a_{v_j,v_k} \tag{2.7}$$

$$+ \sum_{v_j \to v_k, v_k \notin Q} B[i + 1, v_k] \cdot a_{v_j,v_k} \cdot e_{v_k,X_i} \tag{2.8}$$

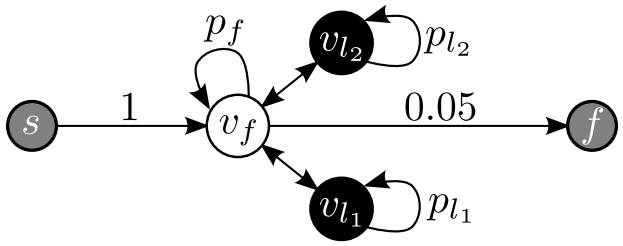

Figure 2.3: The example of ocasionally dishonest casino. The state corresponding to fair dice have white color and the states corresponding to the loaded dice are black.

The proof of correctness is analogous to the proof of theorem 2 and we omit it. Despite the fact, that the backward algorithm is almost the same as forward, we included it because we will need it later in Section 2.7.

2.3 Sequence Annotation

In the previous section we have introduced algorithms for computing the probability of sequence X. Another important task is to find a state path that was generated with X. However in general there may be several state paths that could have been generated with X, albeit with different probabilities. In that case we might be interested in the most probable one. But finding the state path is not always what we want. Sometimes the HMM is too complex, and many states have the same meaning. This is formalized in following definition that assigns colors to classes of states with different meaning and extends the definition to state paths.

Definition 6 (Annotation) *Let $C = \{c_1, c_2, \ldots, c_M\}$ be the finite set of colors and V, Q be the set of states and silent states of the model. Then $\lambda : V^* \to C \cup \{\varepsilon\}$ is* coloring function*, if it satisfies:*

1. $\lambda(v) = \varepsilon$ *if* $v \in Q$
2. $\lambda(v) \in C$ *if* $v \in V \backslash Q$.
3. $\lambda(uv) = \lambda(u)\lambda(v)$ *if* $u, v \in V^+$.

Let $X \in \Sigma^*$ be a finite sequence and π be the corresponding state path that generates X. Then annotation *of X is $\Lambda = \lambda(\pi)$. Usually, we will denote characters of Λ as $\Lambda = \Lambda_1\Lambda_2 \ldots \Lambda_n$.*

Now we can define the probability of an annotation as the sum of the probabilities of all state path with that annotation.

Definition 7 *Let H be an HMM, X be the sequence and Λ be a annotation of the sequence X. Then probability that H generates X with annotation Λ is*

$$\Pr(X, \Lambda) = \sum_{\pi, \lambda(\pi)=\Lambda} \Pr(X, \pi) \tag{2.9}$$

Note that we can compute the conditional probability of Λ for given X simply as $\Pr(\Lambda|X) = \Pr(X, \Lambda)/\Pr(X)$.

Example 4 *Intuitively, we can see the annotations as the meaning of the sequence. Consider the HMM from Figure 2.3 that represents the dishonest casino from example 1 with coloring λ that assigns white color to state v_f and black color to states v_{l_1} and v_{l_2}. In this case the meaning is the honesty of the casino. A part of the sequence is honest if it was generated by the fair die. When we have a sequence of rolls, we want to mark some parts as a honest and the rest as not honest. If we find the most probable annotation, the black part of sequences can be marked as not honest and the white parts can be marked as honest.*

2.4 Viterbi Algorithm

The most commonly used algorithm for decoding HMMs is Viterbi algorithm [For73]. By decoding we mean assigning a state path or annotation to an input sequence X. The Viterbi algorithm computes the most probable state path π that generates a given sequence X. We can use annotation $\lambda(\pi)$ as the predicted annotation of X.

Let $V(i, v_j)$ be the probability of the most probable state path generating $X[1:i]$ and ending in state v_j. It can be computed by the following formula.

$$\begin{aligned} V(0, v_0) &= 1 \\ V(i, v) &= \max_{u \to v} \left(V(i, u) \cdot a_{u,v} \right), v \in Q, i \neq 0 \vee v \neq v_0 \\ V(i, v) &= e_{v,x_i} \cdot \max_{u \to v} \left(V(i-1, u) \cdot a_{u,v} \right), v \in S \backslash Q, i > 0 \end{aligned}$$

As we can see, this part of the Viterbi algorithm is almost the same as the forward algorithm, with only one difference: the summation is replaced by maximum. The proof of correctness is also similar to the proof of theorem 2. The probability of the most probable state path will be stored in $V(n, V_M)$, we should reconstruct it. While computing the table V, we also fill in the table V' of size $n \times M$. All $V'(i, v_j)$ will store a pointer to the previous state v_k on the path to v_j in computing $V(i, v_j)$. Formally,

$$V'(i, v_j) = \arg\max_{v_k} \left(V(i-1, v_k) \cdot a_{v_k, v_j} \right)$$

if $i > 0$ and v_j is not a silent state, or

$$V'(i, v_j) = \arg\max_{v_k} \left(V(i, u) \cdot a_{v_k, v_j} \right)$$

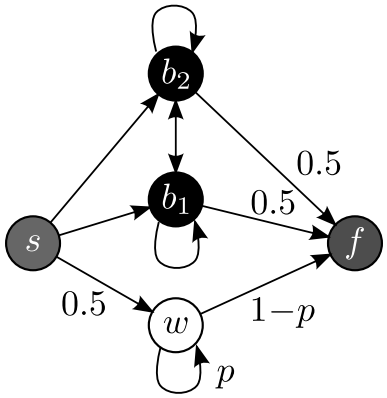

Figure 2.4: The simple HMM with the multiple path problem. The transitions without probabilities shown in the figure have each probability 0.25. This HMM emits symbol 0 with probability 1 in each state (except for first and final state).

if v_j is a silent state. By following the pointers in this table from $V'(n, v_M)$ to $V'(0, v_0)$, we can easily reconstruct the best state path.

The time complexity of the Viterbi algorithm is $O(n \cdot M \cdot D)$, where n is the length of the sequence and M is the number of states and D is the average degree of states in the model. The memory requirements are $O(n \cdot M \cdot D)$, and as we show later in Section 4.4.1, it can be lowered to $O(\sqrt{n} \cdot M \cdot D)$. One advantage of the Viterbi algorithm is that it can be easily implemented with some heuristics optimizations, such as beam search[Low76], that reduces the running time and memory consumption.

Idea behind this optimization is that we did not compute Viterbi values $V(i, v)$ for all state paths, but only for small subset of the states. Let v^* be the state, that maximizes the value $V(i-1, v^*)$ and β be the parameter if the beam search called beam-width. We compute $V(i, v)$ only for those states, which have the higher Viterbi value $V(i-1, v)$ than $\beta V(i-1, v^*)$.

Applying function λ on the most probable state path we get the desired annotation. But this is not always the annotation with the highest overall probability. Consider the HMM from Figure 2.4 and sequence $X = 0^n$. There are two possible annotations. One is entirely black and the second one is white. There is only one path that goes through state w and its probability is $0.5 \cdot (1-p) \cdot p^{n-1}$. Any state path that goes through black states have probability $0.5 \cdot 0.25^n$ and there are 2^n such paths. If $p > 0.25$, the most probable path is the white one. The probability of the white annotation is same, and the probability of the black annotation is $2^n \cdot 0.5 \cdot 0.25^n = 0.5^{n+1}$. If n is large enough and $0.25 < p < 0.5$, the most probable annotation will be the black one. It is also clear that the probability of the most probable state path is exponentially smaller than the probability of the most probable annotation.

The problem is, that the black annotation corresponds to many path of low probability and therefore the Viterbi algorithm will ignore it. We say, that this HMM has a *multiple path problem*.

Definition 8 *[BBV07] Let H be an HMM. We say that H have a* multiple path problem *it there exists a sequence X and two different state paths π_1, π_2 with the same annotation $\lambda(\pi_1) = \lambda(\pi_2)$ and non-zero probability of generating sequence X, $\Pr(X, \pi_i) > 0$.*

Note that also the HMM from Figure 2.3 has multiple path problem.

2.5 The Most Probable Annotation Problem

As we saw in the previous sections, the most probable state path does not always lead to the best annotation. Therefore we need another decoding method. We could try to find the algorithm that will find the most probable annotation. At first, we will define our problem.

Definition 9 (The most probable annotation problem) *[BBV07] Given an HMM H and a sequence X, find the annotation Λ that maximizes $\Pr(\Lambda|X)$.*

This problem is NP-hard. This was prooved by a reduction from the maximum clique problem [LP02]. The second, stronger result is that this problem is NP-hard even for a small fixed HMM [BBV07]. We show the first proof of NP-hardness.

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

Proof. [LP02]. The original proof was constructed for the consensus string problem and the most probable annotation was then reduced to this problem. We show a direct reduction from the maximum clique problem to the most probable annotation problem.

Let $G = (V, E)$ be a graph and $v_1, v_2, \dots v_n$ be the ordering of its vertices. We create an HMM $H_G = (V_G, Q_G, e, a)$ with a separate layer for every vertex v_i. We also define a coloring function λ, for which we will be computing the most probable annotation. The set of the color of the coloring function λ is $C = V \cup \{L\}$.

Layer for vertex v_i will have $n + 1$ silent states $s_{i,j}, 0 \leq j \leq n$. Two silent state $s_{i,j}$ and $s_{i,j+1}$ will be separated by either one non-silent state or a pair of two non-silent states. There are three rules for constructing HMM H_G:

1. There will be a non-silent state $q_{i,i}$ with color $\lambda(q_{i,i}) = v_i$ and transitions $s_{i,i-1} \to q_{i,i}$ and $q_{i,i} \to s_{i,i}$ with probability one.

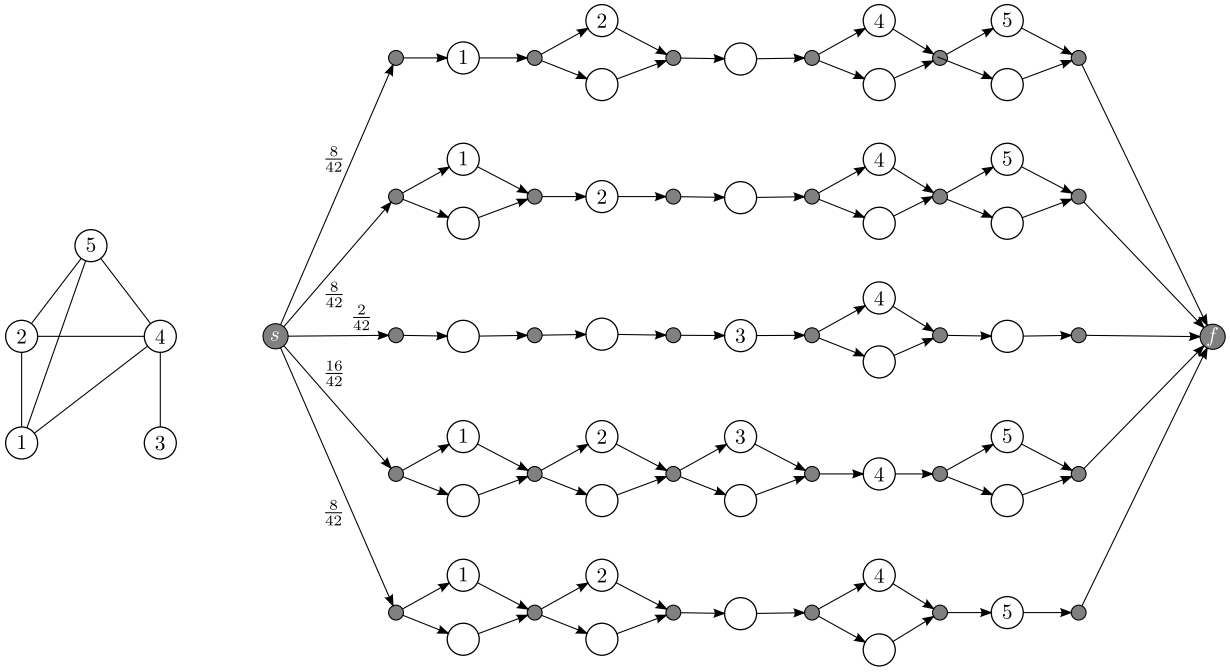

Figure 2.5: On the left side is the graph G and on the right side of the figure is the HMM H_G. The probability of any sequence generated by this HMM is $1/\sum_{v\in V} 2^{deg(v)}$. The numbers in the states in HMM represents the colors. Silent states are gray. Any unlabeled transitions have probability 1 or 1/2. Any non-silent state generates symbol 1 with probability 1. White states without label have color L.

2. For every vertex v_j that is not connected with vertex v_j in graph G there will be state $q_{i,j}$ with color $\lambda(q_{i,j}) = L$ and transitions $s_{i,j-1} \to q_{i,j}$ and $q_{i,j} \to s_{i,j}$ with probability one.

3. For every vertex v_j that is connected with with vertex v_j in graph G there will be two states $q^1_{i,j}, q^2_{i,j}$ with color $\lambda(q^1_{i,j}) = v_j, \lambda(q^2_{i,j}) = L$ and transitions $s_{i,j-1} \to q^k_{i,j}, k \in \{1, 2\}$ with probability $1/2$ and transitions $q^k_{i,j} \to s_{i,j}$ with probability one.

To finish the construction we add start and final state. Silent states $s_{i,n}, 1 \leq i \leq n$ are connected with final state with transition with probability one. From start state there are transitions to all states $s_{i,0}, 1 \leq i \leq n$. Transition from start state to state $s_{i,0}$ have probability

$$\frac{2^{\deg(v_i}}{\sum_{v \in V} 2^{\deg(v_j)}}$$

The example of this construction is in the Figure 2.5.

The probability of any state path from the H_G is $1/\sum_{v \in V} 2^{\deg(v_i)}$ or zero and for every annotation Λ there is at most one state path $p, \lambda(p) = \Lambda$ for every layer. Also every annotation with non-zero probability can have color v_i only on i-th position. Finally if a path goes through through layer v_i and has color v_j on position $j, i \neq j$ then in graph G is the edge (v_i, v_j).

Let $\gamma = \sum_{v \in V} 2^{\deg(v)}$. The probability of any annotation is $k/\gamma, k \in \mathbb{N}$, k is the number of layers which have a path with annotation Λ and non-zero probability. Let $v_{I_1}, v_{I_2}, \ldots, v_{I_k}$ be those layers. Then for every $1 \leq i \leq k$ the I_i-th color of Λ is v_{I_i}. Since every vertex v_{I_i} is connected with vertices $v_{I_j}, j \neq i$ in graph G by an edge, the vertices $v_{I_i}, 1 \leq i \leq k$ induce a clique in graph G. Therefore if H_G has an annotation with probability k/γ then in graph is clique of size k.

Also for every clique in graph G there is an annotation with probability k/γ. Let I be the set of indexes of vertices in that clique. Then the annotation Λ will have on positions I_i the color v_{I_i} and other positions color L. This annotation has probability k/γ. Therefore finding the most probable annotation in H_G is equivalent to finding the maximum clique in graph G. Since finding the maximum clique is NP-hard, the problem of finding the most probable annotation is NP-hard.

2.6 Extended Viterbi Algorithm

Despite the fact that finding the most probable annotation is NP-hard (even for some small HMM), there are effective algorithms, that compute the most probable annotation for some classes of HMMs. For example for HMMs without the multiple path problem the Viterbi algorithm finds the most probable annotation.

However, there is an efficient algorithm that will find the most probable annotation for a larger class of HMMs than HMMs without the multiple path problem.

Definition 10 (Extended annotation) *[BBV07] A* critical edge *is a transition between two states of different color. The extended annotation of a state path $\pi_1\pi_2\ldots\pi_n$ is the pair (Λ, C), where $\Lambda = \lambda(\pi)$ is the sequence of colors, and $C = c_1c_2\ldots c_k$ is the sequence of critical edges in that path.*

The Extended Viterbi algorithm [BBV07] computes the most probable extended annotation in $O(n^2M^3)$ time, where n is the length of the sequence and M the number of states of the HMM.

Definition 11 *[BBV07] An HMM satisfies the* critical edge condition *for an input sequence if any two paths for with the same annotation have the same sequence of critical edges. An HMM satisfies the critical edge condition in general if for all input sequences, the critical edge condition is satisfied.*

In all HMMs that satisfies the critical edge condition can the Extended Viterbi algorithm finds the most probable annotation [BBV07]. There is also generalized version of this algorithm that allows some uncertainty in the sequence of critical edges [BBV07].

2.7 Posterior Decoding

Another way of decoding HMM is the posterior decoding [KKS05, DEKM98]. In this algorithm we assigns the most probable state to each position. Formally, we assign to position i state

$$\arg\max_{v\in V\setminus Q}\{\Pr(\pi_i^Q = v|X)\} \tag{2.10}$$

As we can see, we obtain a non-silent state path. To compute such a state path we need co compute the posterior probabilities of all states and positions by the following formula.

$$\Pr(\pi_i^Q|X) = \frac{F[i,v_j]\cdot B[i,v_j]}{\Pr(X)} \tag{2.11}$$

Note that the $F[i,v]$ and $B[i,v]$ are computed by forward and backward algorithm.

The running time and memory requirements are same as in the Viterbi algorithm, $O(n\cdot M\cdot D)$, where n is the length of the sequence, M is the number of states and D is the average degree of states in the model. In contrast to the Viterbi algorithm, it is harder to use optimization techniques such as beam-search. If we try to do so, the resulting posterior probabilities will be wrong because we will throw away many state paths that have individually low probability, but in sum they can have significant impact.

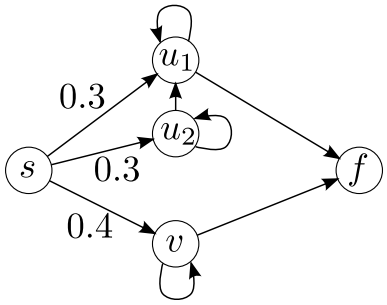

Figure 2.6: The simple HMM, on which posterior decoding generates unprobable state paths. This HMM generates sequences over alphabet $\Sigma = \{0\}$. The start state s and the final state f are silent. Unlabeled transitions have probability 0.5.

The problem with the posterior decoding is that it can produce an annotation with zero probability. Consider the HMM from figure 2.6 and sequence $X = 00$. There are three possible state paths: $svvf, su_1u_1f$ and su_2u_1f, with probabilities $\Pr(svvf, 00) = 0.1, \Pr(su_1u_2f, 00) = 0.075$ and $\Pr(su_2u_1f, 00) = 0.75$. When we compute the posterior probabilities of states, $\Pr(\pi_1^Q = v|00) = 0.4, \Pr(\pi_1^Q = u_i|00) = 0.3, \Pr(\pi_2^Q = v|00) = 0.4), \Pr(\pi_2^Q = u_1|00) = 0.6$ and $\Pr(\pi_2^Q = u_2|00) = 0$. Therefore the posterior decoding of the sequence 00 will be svu_1f, but this state path has zero probability.

Inspite of this drawbacks, the posterior decoding optimises an intuitive objective. It maximizes the expected number of correctly assigned states:

$$\arg\max_{\pi^Q} = \sum_{\pi'^Q} \left(\Pr(\pi'^Q|X) \sum_{i=1}^{|\pi^Q|} (\pi_i^Q = \pi_i'^Q) \right) \tag{2.12}$$

where term $(\pi_i^Q = \pi_i'^Q)$ is one if π^Q and π'^Q are same and zero otherwise.

We can also extend the forward-backward algorithm to optimize the most probable color for each particular position. We just need to sum the posterior probability over all states with the same color. This variant can also lead to annotation with zero probability.

2.8 Gain Functions

In this section we present a more general decoding framework based on gain function [HKS+09]. The previous methods we have searched for an object y with maximum conditional likelihood given the sequence X:

$$y = \arg\max_{y'} \Pr(y' \mid X)$$

where y is a state path or an annotation of sequence X. In this framework we will define a gain function which characterizes the similarity between proposed annotation and the

correct annotation. We will also say that the gain function assigns a reward to the annotation given the correct annotation. The correct annotation of sequence X is annotation Λ' which was generated together with sequence X. Since the correct annotation is hidden, we can not search for the annotation with the highest reward (the most similar to the correct one). But given X we know the distribution of all possible annotations $\Pr(\Lambda|X)$ and therefore we can search for the annotation with the highest expected reward.

Now we formally define gain functions. We will define the gain function for state paths, not for annotations, so that we can express the previous decoding methods in terms of gain functions.

Definition 12 (Gain function) *Let H be an HMM and L be the set of all state paths. Then any function $f : L \times L \to \mathbb{R}$ is a* gain function.

Note. The gain function f does not have to be symmetric (there may be $a, c \in L$, such that $f(a, b) \neq f(b, a)$), but we will consider mostly symmetric functions.

The correct state path of the sequence X is defined analogously as the correct annotation. It is a state path from which was generated the sequence X. Let X the sequence, π_X it's correct state path and π another state path of X. Then *reward* of state path π is $f(\pi_X, \pi)$.

Definition 13 (Expected reward) *Let H be an HMM, f be a gain function, X be a sequence and π be a state path of X. Then the* expected reward *is defined as*

$$E_{\pi_X|X}[f(\pi_X, \pi)] = \sum_{\pi_X} f(\pi_X, \pi) \cdot \Pr(\pi_X \mid X) \tag{2.13}$$

Our goal will be to search for state paths with the highest expected reward. As we will show, classical decoding approaches from previous sections can be expressed by an appropriate gain function.

Theorem 4 *Let H be an HMM and X be a sequence. Let f_V be the gain function defined in the following way.*

$$f_V(\pi_1, \pi_2) = \begin{cases} 1 & \text{if } \pi_1 = \pi_2 \\ 0 & \text{if } \pi_1 \neq \pi_2 \end{cases}$$

Then the state path $\bar{\pi}$ that has the highest expected reward can be found by the Viterbi algorithm. Moreover, the expected reward $E_{\pi_X|X}[f_v(\pi_X, \bar{\pi})] = \Pr(\bar{\pi}, X)$.

Proof. Expected reward of a state path π' is the same as its probability:

$$E_{\pi_X|X}[f_V(\pi_X, \pi')] = \sum_{\pi_X} f(\pi_X, \pi') \cdot \Pr(\pi_X \mid X) = \Pr(\pi' \mid X)$$

Therefore the state path with the highest expected reward is the one with the highest probability. QED

Using the same argument we can define the gain function for the most probable annotation problem.

$$f_A(\pi_1, \pi_2) = \begin{cases} 1 & \text{if } \lambda(\pi_1) = \lambda(\pi_2) \\ 0 & \text{if } \lambda(\pi_1) \neq \lambda(\pi_2) \end{cases}$$

Note that finding the most probable annotation is NP-hard and therefore finding the annotation with the highest expected reward with respect to gain function f_A is also NP-hard.

It is also possible to define a gain function which is optimized by the same state path as the posteriori decoding. Recall that if π is a state path, π^Q is the same state path without silent states.

$$f_P(\pi, \pi') = \sum_{i=1}^{|\pi^Q|} \begin{cases} 1 & \text{if } \pi_i^Q = \pi_i'^Q \\ 0 & \text{if } \pi_i^Q \neq \pi_i'^Q \end{cases}$$

Computing the expected reward of this function is in fact the equation 2.12.

As we saw, the gain functions are universal framework in which we can express many decoding methods. This general framework is relatively new and an apparent for the stochastic context free grammars [HKS+09]. Before it has been used with particular gain functions designed for the particular application. In the next section we show such decoding method that was proposed for the de novo gene prediction [GDSB07].

2.9 Maximum Expected Boundary Accuracy Decoding

The maximum expected boundary accuracy decoding maximizes a weighted difference between the expected number of true-positive and false-positive coding region boundary predictions [GDSB07]. Boundary is a change of the color in the annotation and we denote them as the position in the annotation. For example the boundary i in annotation Λ means that $\Lambda_i \neq \Lambda_{i-1}$. This decoding was proposed for gene prediction where it is important to find the exact positions of boundaries, since small error in position may significantly change the predicted protein.

This decoding method was originally proposed for the conditional random fields. Since conditional random fields are similar to hidden Markov models, we demonstrate this method on decoding HMMs.

Definition 14 (Maximum boundary accuracy decoding) *[GDSB07] Let $\Lambda = \Lambda_1\Lambda_2 \dots \Lambda_l$ be an annotation. Let B be the set of all boundaries, $B_\Lambda = \{i|\Lambda_i \neq \Lambda_{i-1}\}$. Let gain function f be defined in following way:*

$$f_M(\pi, \pi') = \sum_{i \in B_\Lambda} \begin{cases} \kappa & \text{if } \lambda(\pi_i^Q) = \lambda(\pi_i'^Q) \text{ and } \lambda(\pi_{i-1}^Q) = \lambda(\pi_{i-1}'^Q) \\ -1 & \text{otherwise} \end{cases} \tag{2.14}$$

Then in the maximum boundary accuracy decoding *we search for the state path with the highest expected reward.*

This objective can be computed efficiently in $O(n|T|+n|C|)$ time, where n is the length of the sequence, T is the set of all transitions and C is the set of colors. We do not present the algorithm for finding the annotation with highest expected reward here, because it is a special case of our algorithm which can be found in Chapter 4.

2.10 Distance Measures on Annotations

This decoding method is taylored on problems, where there are many annotations with similar probability and slightly shifted boundaries between colors. For such applications the decoding method should take into account these similar annotations [BT10].

We express this method in terms of gain functions. Let d be some distance measure defined on annotations and $\bar{B}_d(\Lambda, r) = \{\Lambda' : d(\Lambda, \Lambda') \leq r\}$ be the ball of radius r with centre Λ. We define a gain function in the following way:

$$f_{\bar{B}}(\pi, \pi') = \begin{cases} 1 & \text{if } \lambda(\pi') \in \bar{B}_d(\lambda(\pi), r) \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

The expected reward of state path π with respect to function $f_{\bar{B}}$ is the sum of the probabilities of all state paths π' for which $\lambda(\pi') \in \bar{B}_d(\lambda(\pi), r)$ holds.

Brown *et al. (2010)* were using several distance functions for finding the annotation accumulating annotations with slightly shifted boundaries. Now we define those distance functions.

Definition 15 (Hamming distance) *[BKR02] Let a, b are two words of the same length over alphabet Σ. Then Hamming distance of a and b is*

$$d_H(a, b) = \sum_{i=1}^{|a|} (a_i \neq b_i)$$

where $(a_i \neq b_i)$ is one if $a_i \neq b_i$ and zero otherwise.

Note that the Hamming distance is in fact the number of incorrectly assigned colors.

The let $\Lambda = c_1^{k_1} c_2^{k_1} \dots c_m^{k_m}$ be an annotation where $c_i \neq c_{i+1}$. Then the footprint of the annotation Λ is the sequence $c_1 c_2 \dots c_m$.

Definition 16 (Border shift distance) *[BT10] Let Λ_1, Λ_2 be two annotations of same sequence X. Let $B_{\Lambda_1} = \{(b_{1,1}), \dots (b_{1,k})\}$ and $B_{\Lambda_2} = \{b_{2,1}), \dots (b_{2,k})\}$ be ordered sets of the boundaries of the annotations Λ_1, Λ_2. Then* border shift distace *is:*

$$d_b(\Lambda_1, \Lambda_2) \begin{cases} \infty & \textit{if } \Lambda_1 \textit{ and } \Lambda_2 \textit{ have different footprints} \\ \max_{i=1..k} |b_{1,i} - b_{2,i}| & \textit{otherwise} \end{cases}$$

Note that if Λ_1 *and* Λ_2 *have same footprint then* $k_1 = k_2$.

Definition 17 (Border shift sum distance) *[BT10] Let* $\Lambda_1, \Lambda_2, B_{\Lambda_1}$ *and* B_{Λ_2} *be define exactly as in the definition 16. Then the* border shift sum distace *is:*

$$d_b(\Lambda_1, \Lambda_2) \begin{cases} \infty & \text{if } \Lambda_1 \text{ and } \Lambda_2 \text{ have different footprints} \\ \sum_{i=1}^{k} |b_{1,i} - b_{2,i}| & \text{otherwise} \end{cases}$$

Since for any distance function $d(\Lambda_1, \Lambda_2) = 0$ if and only if $\Lambda_1 = \Lambda_2$ finding annotation with highest expected reward is NP-hard even for $r = 0$. Therefore Brown *et al.* were using heuristic algorithms for finding the annotations in the reasonable time. They have considered several distance measures, such as Hamming distance, border shift distance and border shift sum distance refined as follows.

In Chapter 4 we will consider a similar objective to the border shift distance. We will bypass the NP-hardness by scoring the boundaries independently instead of scoring the annotations as a whole.

Chapter 3

Applications of HMM to Viral Recombination

HMMs have numerous application. The goal of this chapter is to explain in more details their use for detecting HIV viral recombinations. We will test our new decoding method on this particular application domain. We also describe sequence alignment and the profile HMMs, because they form the basis of jumping HMMs.

3.1 Sequence Alignment

Sequence alignment is one of the basic task in analyzing biological sequences. To create alignment we have to add *gap symbols* - to several input sequences, so that they have equal length.

As we saw in Figure 3.1, there are many possible alignments for any set of sequences. For example, we can create an alignment by adding $|Y|$ gap symbols before beginning of the X and $|X|$ gap symbols after the end of the $|Y|$. Such an alignment is literally meaningless. To distinguish between good and bad alignments of given sequences we will

1. `AAATGTAGAGATAAG---AAGTTCAATGGAACAGGCCCAT`
   `AAGTGT---GATGATAAAAGGTTCAATGGGACGGGGCCAT`

2. `AAATGTAGA-GATAAGAAGTTCAATGGAACAGGCCCAT`
   `AAGTGT-GATGATAAAAGGTTCAATGGGACGGGGCCAT`

Figure 3.1: The first alignment is a part of the multiple alignment of many different HIV virus genomes The second alignment is an alignment of the same two sequences and was created manually.

use a scoring scheme. Our goal is to produce an alignment that reflects the evolutionary history of the sequences.

In evolution theory a sequences car evolve one from another one by a series of mutation events. An event may be an insertion or deletion of a residue from the sequence or a substitution of a residue in the sequence. Insertions and deletions are called *indels*, because it is hard to distinguish them. For example in the first alignment in Figure 3.1, the sequence `AGA` could be deleted from the second sequence or be inserted into the first sequence. We say that two sequences are homologous if they evolved from one sequence called a common ancestor. We want to have a scoring system such that homologous sequences will have an alignment with high score, whereas alignments of non-homologous sequences will have low score.

At first we will ignore insertions and deletions and consider only the substitutions. We assume that all evolution events are independent. Let R be the probabilistic model, in which symbol q has frequency p_q, and two sequences X and Y are independent (they are not homologous). Then the joint probability of both sequences in this model is

$$\Pr(X, Y \mid R) = \prod_i p_{x_i} \cdot \prod_i p_{y_i} \tag{3.1}$$

Now consider another model M, in which symbols x_i and y_i are derived from some symbol c_i through substitutions with probability p_{x_i,y_i}. Then the probability of both sequences is

$$\Pr(x, y \mid M) = \prod_i p_{x_i,y_i} \tag{3.2}$$

The ratio of two probabilities, can be seen as a measure of X and Y being homologous.

$$\frac{\Pr(x, y \mid M)}{\Pr(x, y \mid R)} = \prod_i \frac{p_{x_i,y_i}}{p_{x_i} p_{y_i}} \tag{3.3}$$

If the result is greater than one, the sequences are more likely to be homologous than to be from a random model and vice versa. Since the computation of the equation 3.3 is not numerically stable, we use logarithms.

$$\log\left(\frac{\Pr(x, y \mid M)}{\Pr(x, y \mid R)}\right) = \sum_i \log\left(\frac{p_{x_i,y_i}}{p_{x_i} p_{y_i}}\right) \tag{3.4}$$

Then a positive score value means homologous sequences and negative non-homologous.

The score of a pair of symbols x_i and y_i from the same column of an alignment is $\log\left(\frac{p_{x_i,y_i}}{p_{x_i} p_{y_i}}\right)$. Values $\log\left(\frac{p_{x_i,y_i}}{p_{x_i} p_{y_i}}\right)$ we will be arranged to a score matrix. One question remains. What are the probabilities p_q and p_{x_i,y_i}? The frequencies p_q can be observed in sequences, but in sequences we do not observe p_{x_i,y_i}. There are commonly used matrices,

```
CCAGTAGATCCT---AACCTAGAACCCTGGAACCATCCAGGAAGTCAG
CCAGTAGATCCT---AACCTAGAGCCCTGGAATCATCCAG---GTCAG
CCAGTAGATCCT---AGGCTAGAGCCCTGGAACCATCCAGGAAGTCAG
CCA---GACCCT---AACCTAGAGCCCTGGAAGCATACAGGAAGTCGG
CCAGTAGATCCT---AGCCTAGAGCCCTGGAACCATCCAGGAAGTCAG
CCAGAAGATCCTCCCAGCTTGAGC---TGGAACCATCCAGGAAGGCAG
CCAGTACATCCT---AGCCTACAGCCCTGGAACCATCCAGGAAGTCAG
```

Figure 3.2: Example of a multiple alignment.

PAM, BLOSUM [DEKM98] used as score matrices. The BLOSUM matrix is created from manually created and verified alignments and PAM matrix is derived from a sequence alignments and a theoretical model of evolution.

Now we have to cope with gaps. Let g be the length of the gap. We assume, that probability of a gap of length g at position h is

$$\Pr(g, h \mid M) = f(g) \prod_{h \leq i < h+g} p_{x_i} \tag{3.5}$$

To get the measure of sequence X being homologous to an empty sequence Y we take the ratio of two probabilities.

$$\frac{\Pr(g, h \mid M)}{\Pr(g, h \mid R)} = \frac{f(g) \prod_{h \leq i < h+g} p_{x_i}}{\prod_{h \leq i < h+g} p_{x_i}} = f(g) \tag{3.6}$$

Now we see, that the score of the gap depends only in the gap length and it is $\gamma(g) = \log(f(g))$. Usually the scoring function for the gaps is linear. There are two variants of scoring function $\gamma(g)$. The first is the $\gamma(g) = -gd$ and the $\gamma(g) = -e - (g-1)d$ where d is gap penalty and the e is the opening gap penalty. The opening gap penalty is commonly greater than the gap penalty, because we usually prefer fewer but longer gaps than many short gaps.

To find an alignment the highest score we can use the Needleman-Wunsch algorithm that have time complexity $O(|X||Y|)$. Since for long sequences this algorithm is unpractical, we usually have to use heuristic algorithms, for example BLAST [DEKM98].

Multiple Alignment

The multiple alignment is the alignment of two or more sequences. The meaning of multiple alignment is similar to the meaning of the alignment of two sequences. In a multiple sequence alignment the homologous residues are aligned in the same columns. The residues are homologous if they evolved from common ancestral residue. Homologous residues may

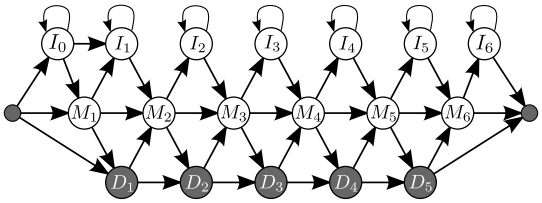

Figure 3.3: Example of profile HMM with length 6

have same function in all sequences of the alignment or be in same position in the structures that sequences creates. In Figure 3.2 we can see a proportion the multiple alignment of several HIV sequences.

There are many scoring methods for multiple alignment. Most of them assigns score individually to each column of the alignment as described above. The simplest scoring method uses a sum of pairwise scores between all pairs of sequences in the alignment. If the score of the alignment is the sum of the scores of all columns in the alignment, we can find the multiple alignment with the highest score by dynamic programming in $O(2^k n^k)$ where k is the number of sequences and n is the length of the sequences [DEKM98]. If the number of sequences is fixed, the time complexity is polynomial. But in case of the variable number of each sequence, finding the multiple alignment with the highest score is for the most score systems NP-hard [WJ94]. In practice exists various heuristic algorithm that can align large number of long sequences in the reasonable time [DEKM98].

3.2 Profile HMMs

A sequence family is a set of sequences that are in some way similar, for example in their sequence or function. A profile HMM can effectively represent a family of similar sequences. It can be used for example to search for new family members in a sequence database.

A profile HMM [DEKM98] of length k consists of $3 \cdot (k+1)$ states: the start state B, the end state E, the *match states* $M_1, M_2, \ldots, M_k$, *insert states* $I_0, I_1, \ldots I_k$ and *delete states* $D_1, D_2, \ldots D_{k-1}$. These states are connected by transitions as follows (see also Figure 3.3.

1. There are transitions from state B to states M_1, I_0 and D_1
2. There are transitions from states D_{k-1}, I_k, M_k to state E.
3. For every i, there are transitions from M_i to states M_{i+1}, I_i and D_{i+1} if such state exists.
4. For every i, there are transitions from I_i to I_i, I_{i+1}, D_{i+1} and M_{i+1} if such state exists.

The match states represent the main structure of the family members. In ideal case the i-th match state represents the i-th symbol of the sequence. Since the insertion or deletion may have happened, we need insert and delete states. Other way how to represent a family of the sequences is the multiple alignment. But if we have a large number of sequences such a representation is too big. We can see the profile HMM as the "compression" of a multiple alignment.

Each match state of the profile corresponds to some column in the alignment. Not all columns in the alignment have necessarily a corresponding match state. For example we do not want to have match state for columns that consist almost entirely of gaps. To create a profile HMM we choose the columns from which we create the match states. There are various heuristics to do this, for example Schultz *et al. (2006)* used only columns from multiple alignment that did not have any gaps. The emission probabilities are usually derived from frequencies of symbols in the column. The transition probabilities between match, delete and insert states can also be derived from observed frequencies [DEKM98].

We have used the profile HMM as a part of a larger model, but usually they are used to detect membership in a sequence family. To provide score for a particular membership of a sequence to the family, we can use the probability of the most probable state path found by Viterbi algorithm or the probability of the sequence computed by forward algorithm [DEKM98]. Both of these probabilities are typically compared with a simple background random model of sequences in a log-odd score, similar to pairwise alignment [DEKM98].

3.3 Jumping HMM

Human immunodeficiency virus (HIV) is a major pathogen causing the pandemic of the Acquired Immune Deficiency Syndrome (AIDS). The HIV genome mutates rapidly and currently is divided into three main phylogenetic groups M, N and O. Those groups were created by three different transmissions of SIV virus from chimpanzee to humans [SZL$^+$06]. Most HIV infections are caused by HIV-1 M group viruses. This group is divided into 10 subtypes and some subtypes are divided into sub-subtypes, as show in Figure 3.4. To classify a newly sequenced virus into known subtypes, we can use profile HMMs. But some HIV genomes are a mosaic recombination of viruses from two or more different subtypes [RAB$^+$00]. These viruses are called circulating recombinant forms (CRFs). Our goal is do decide, whether the input sequence belongs to certain virus family or if it is a CRF. If it is a CRF, we want to reconstruct recombination points and to find the source family of each homogeneous block. The HCV virus have similar structure, and therefore the same methods can be used to detect recombination in the genome of HCV.

Now we describe in brief the process of the recombination of the HIV genome. Unlike other viruses, the HIV is diploid. It contains two full-length RNA strands, each able to

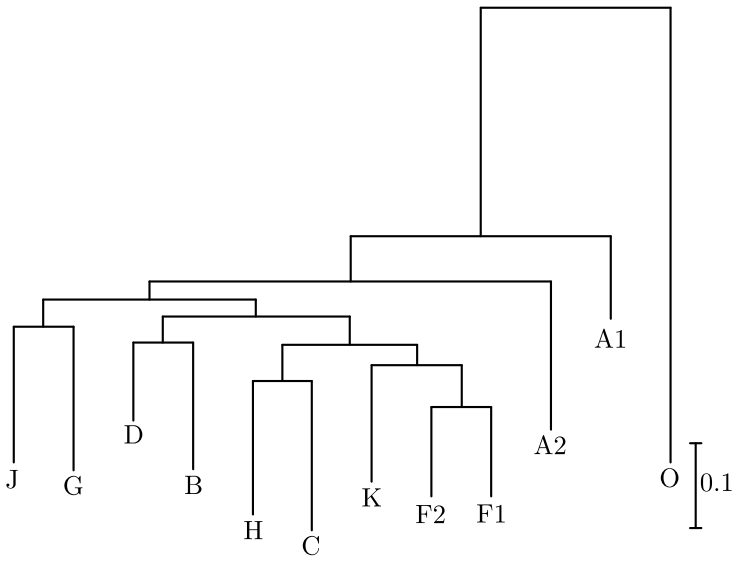

Figure 3.4: Phylogenetic tree of the HIV-M1 group of genomes with branch of HIV-O group generated by phyml [GG03]. As the source alignment we have used the database distributed with jpHMM [SZL+06]. The lengths of the branches corresponds to the evolution distances. The subtypes A and F are divided into sub-subtypes A1,A2 and F1,F2.

replicate in a cell. Once the viron infects the cell, it injects into the cell both strains of RNA and various proteins including reverse transcriptase. The RNA of the virus is transcribed into double stranded DNA by reverse transcriptase. After the reverse transcription, the resulting DNA is incorporated into the cell's nucleus where it may be replicated. Once the RNA of the virus leaves the nucleus, it is translated into proteins that are essential to the survival of the virus and assembled viron leaves the cell [Bur97].

If two virons infects the same cell, the pairs of RNA strains can be mixed into new "heterozygotous" viron during the viron assembly. This new viron have one strain from each of the two parent virons. Once this viron infects an other cell, the two strains can be recombined into new strain during the reverse transcription [Bur97].

Currently the most commonly used recombination detection tool is Jumping profile HMM (jpHMM) [SZL+06]. Now we describe the model behind this tool.

Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be the set of all sequences in the database which is divided into k subtypes $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$. Let A be the multiple alignment of all sequences in $\mathcal{S}$. In the jpHMM, every subtype $\mathcal{S}_i$ is represented by profile P_i. The profile P_i was constructed from subalignment of A composed of sequences in $\mathcal{S}_i$.

To model recombination, we have to add transitions between states of the different profiles. To keep the model simple and of reasonable size, we have limit their number.

Definition 18 *[SZL+06] Let T be a state from profile P_i and R be a state from profile P_j,*

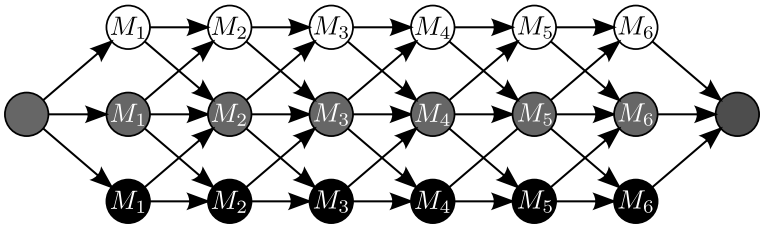

Figure 3.5: The simplified structure of jpHMM. Insert and delete state are mitted from the figure for readibility.

$i \neq j$. We say that state T is to the right of state R, if and only if the column r_t, from which was the state T created is strictly to the right of a column r_R, from which was the state R created ($r_t > r_j$).

The relation to be strictly to the right *implies a partial ordering over the set of states on the jpHMM.*

The jumps (transitions) between the profiles are constructed by the following rules [SZL+06].

1. For a two profiles P_i and P_j, the algorithm can jump from a match state of S_i only to a match state or a delete state of S_j, and from an insert state or delete state of S_i a jump is possible only to a match state of S_j.

2. It is possible to jump from state T in profile P_i to the leftmost state R in P_j, that is strictly to the right of T.

3. It is not possible to jump from state $M_{i,k}, D_{i,k}$ or $I_{i,k}$ to state R in P_j if R is strictly to the right from the state $M_{i,k+1}$.

The first condition reduces the number of jump transitions in the model. The last two conditions ensure that all insertions and deletions are caused by insert and delete states and not by jumps. The simplified structure of jpHMM we can see in the Figure 3.5.

To complete the construction of the model, we describe the beginning and the end of the jpHMM. The starting state of the model is B and end state E. Instead of one insert state before each profile, there is one global insert state for the beginning of the model. This state is reachable directly from B and it has transitions to the first match state of each profile. There is also a similar insert state at the end of the model. There are also two special delete states, one reachable from state B and one with transition to E. Both are connected with every match state, so that only partial sequences in the family can be generated. Finally, there are transitions from state B to the first match state in each profile and transitions from the last state of each profile to the state E.

Schultz *et al.* use for finding annotation the Viterbi algorithm optimized with beam-search. This algorithm is described in Chapter 2. In jumping HMM there is difference between the annotation and state path. At first, the jumping HMM have multiple path problem. In this application we are not interested in the state path, which correspond to the alignment of the sequence to the source database. We are interested in the annotations, which correspond to the division of the sequence into blocks from different subtypes. Also the annotations with slightly shifted boundaries between different colors have similar probability and therefore we want to consider all those annotations while searching for the recombination points. In the next chapter we will introduce the new gain function that is designed for this application.

Chapter 4

HERD: The Highest Expected Reward Decoding

In this chapter we define a new gain function, that allows some uncertainty in boundaries in annotation and algorithm that finds the annotation with the highest expected reward with respect to this function. We show the algorithm that finds the annotation with the highest expected reward with respect to our gain function. Our algorithm is non-trivial extension of the maximum expected boundary accuracy decoding [GDSB07].

4.1 The Highest Expected Reward Decoding

We seek for a gain function that is appropriate for our application domain. This gain function should consider the annotation with slightly shifted boundaries as the same. To bypass NP-hardness, we will have to define this gain function such that it will assign score individually to each boundary. But first we need some technical definitions.

Definition 19 (Boundary) *Let $\Lambda = \Lambda_1\Lambda_2 \dots \Lambda_n$ be an annotation. Let Λ_0 and Λ_{n+1} be the new unique colors. We say that there is* boundary *b in Λ if $\Lambda_b \neq \Lambda_{b-1}, 1 \leq b \leq n+1$.*

Note. We added the two special colors Λ_0 and Λ_{n+1} in order to achieve that the start and end of the annotation are boundaries.

Definition 20 (Feature) Feature *in an annotation is a maximal contiguous region of labels with the same color.*

We will define a buddies of boundaries. The pair of the boundaries in two annotations are buddies if they divide two similar features in both sequences.

Definition 21 (Buddy) *Let Λ and Λ' be two annotations of the same sequence and $W \in \mathbb{N}$. Boundaries i in Λ and j in Λ' are called* buddies *if*

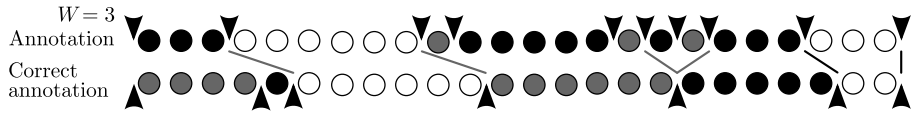

Figure 4.1: The arrows represent boundaries. Black lines connect boundaries that are buddies, the gray lines connect boundaries that would be buddies if the condition iii was dropped from the definition 21

(i) $\Lambda_{i-1} = \Lambda_{j-1}$ *and* $\Lambda_i = \Lambda_j$

(ii) $|i - j| < W$

(iii) There is no other boundary at positions $\min\{i, j\}, \ldots, \max\{i, j\}$ *in either* Λ *or* Λ'.

We denote the parameter W *as the* window size.

The intuition behind the condition iii is that we want buddies to be boundaries of practically identical features in the two annotations that differs only slightly in the exact boundary position. Consider the example in Figure 4.1 and the two boundaries connected by the leftmost gray line. Without condition iii they would be buddies, but the black features in their left are are completely disjoint. A similar situation occurs at the second gray line. Another useful property is that each boundary can have at most one buddy (the last two gray lines). The last two gray lines in Figure 4.1 illustrate that this would not be true without condition iii.

Definition 22 (The highest expected reward decoding problem) *Let gain function* $G(\Lambda, \Lambda')$ *assigns reward* $+1$ *for every boundary in* Λ *that has a buddy in* Λ' *and reward* $-\gamma$ *to all other boundaries in* Λ. *The* highest expected reward decoding problem (HERD) *is finding the annotation* Λ *with the highest expected gain*

$$E_{\Lambda'|X}[G(\Lambda, \Lambda')] = \sum_{\Lambda'} G(\Lambda, \Lambda') \Pr(\Lambda'|X) \tag{4.1}$$

Note. The highest expected reward decoding is for $W = 1$ equivalent to maximum accuracy boundary decoding.

Example of a reward $G(\Lambda, \Lambda')$ is in Figure 4.2. The expected reward of annotation Λ can be decomposed as a sum of partial expected rewards for individual boundaries in Λ. In particular, let B be the set of all boundaries in Λ and $p_{\Lambda,b}$ be the probability that boundary b in Λ has buddy. Formally,

$$p_{\Lambda,b} = \sum_{\Lambda', b \text{ has a buddy in } \Lambda'} \Pr(\Lambda'|X)$$

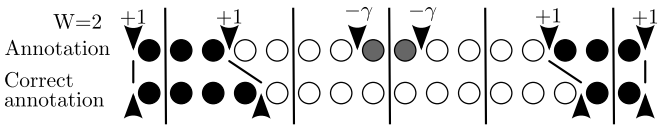

Figure 4.2: Example of an annotation with reward $4 - 2\gamma$.

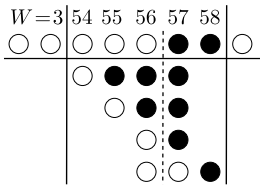

Figure 4.3: Decomposition of $p(i, c_1, c_2, w_l, w_r)$ for $W = 3$. The condition *iii* from 21 restricts annotation to the ones, that don't change color between the buddies.

Let $R_\gamma(p) = p - \gamma \cdot (1 - p)$. Then from linearity of expectation we can do following decomposition:

$$E_{\Lambda'|X}[G(\Lambda, \Lambda')] = \sum_{b \in B} R_\gamma(p_{\Lambda,b}) \tag{4.2}$$

The quantity $R_\gamma(p_{\Lambda,b})$ is the expected reward of boundary b in the annotation Λ.

We can divide the HERD algorithm into two phases. In the first phase, we compute posterior probabilities $p_{\Lambda,b}$ for all possible boundaries. In the second phase we find the annotation Λ with the highest expected reward.

4.1.1 Computing Posterior Probabilities

In this section we show how to compute posterior probabilities $p_{\Lambda,b}$. Let b be the boundary in Λ. The term $p_{\Lambda,b}$ can be expressed as $\sum_{b'} p_{\Lambda,b,b'}$ where

$$p_{\Lambda,b,b'} = \sum_{\Lambda', b \text{ has a buddy } b' \text{ in } \Lambda'} \Pr(\Lambda'|X)$$

From condition *ii* from definition 21 we know that the buddy b' can be only at positions $b - W, \ldots, b + W$. The condition *iii* restrict the possible positions of buddy b' even more. Let $b - w_L$ be the rightmost boundary in Λ that is to the left of b and $b + w_R$ be the leftmost boundary in Λ that is to the right of b. The possible positions for b' are $b - \min\{W, w_L\} + 1, \ldots, b + \min\{W, w_R\} - 1$. Let $w_l = \min\{W, w_L\}$ and $w_r = \min\{W, w_R\}$. The only information about annotation Λ, that we need to compute $p_{\Lambda,b}$ are the values of $b, \Lambda_{b-1}, \Lambda_b, w_l, w_r$ and therefore instead of $p_{\Lambda,b}$ we will write the $p(b, \Lambda_{b-1}, \Lambda_b, w_l, w_r)$. We

divide the possible buddies of b into two sets. The first set B_1 consists of potential buddies $b-w_l, \ldots, b$ and the second set B_2 consists of buddies $b+1, \ldots, b+w_r$. Figure 4.3 illustrates these two sets. The annotations with buddies b' from B_1 have one white label at position $b'-1$ and labels from b' to b are black (both inclusive), all other labels can be arbitrary. The annotations for boundaries b' from B_2 have a symmetric structure: the labels $b, \ldots, b'-1$ are white and the label b' is black. Therefore we can write $p_{\Lambda,b,b'} = \Pr(\Lambda'_{b'\ldots b} = \Lambda_{b-1}\Lambda_b^{b-b'+1}|X)$ if $b' \in B_1$ and $p_{\Lambda,b,b'} = \Pr(\Lambda'_{b-1\ldots b'-1} = \Lambda_{b-1}^{b'-b+1}\Lambda_b|X)$ otherwise. Remind that the $\Lambda_{i\ldots j}$ is the sequence $\Lambda_i\Lambda_{i+1}\ldots\Lambda_j$ and Λ_i^n is the sequence $\Lambda_i\Lambda_i\ldots\Lambda_i$ of the length n. Therefore The $\Pr(\Lambda'_{b'\ldots b} = \Lambda_{b-1}\Lambda_b^{b-b'+1}|X)$ is the conditional probability, that annotation Λ' of the sequence X has on position b' color Λ_{b-1} and on positions $b'+1\ldots b$ color Λ_b. Now we can express $p_{\Lambda,b}$ more precisely.

$$p(b, c_1, c_2, w_l, w_r) \sum_{w=1}^{w_l} \Pr\left(\Lambda'_{b-w\ldots b} = c_1 c_2^w | X\right) + \sum_{w=2}^{w_r} \Pr\left(\Lambda'_{b-1\ldots b+w-1} = c_1^w c_2 | X\right) \tag{4.3}$$

The terms $\Pr\left(\Lambda'_{b-w\ldots b} = c_1 c_2^w|X\right)$ and $\Pr\left(\Lambda'_{b-1\ldots b+w-1} = c_1^w c_2|X\right)$ are similar, both can be computed by an extension of the forward-backward algorithm. For simplicity we now show how to find these quantities for HMMs without silent states and defer the discussion of general algorithm for well-defined HMMs to Section 4.2.

We will compute $\Pr(\Lambda_{i\ldots i+w} = c_1 c_2^w \mid W)$ by combining all state paths that end at position i with color c_1 and those starting in position $i+1$ with color c_2 and continuing with the same color for the next w states.

$$\Pr(\Lambda_{i\ldots i+w} = c_1 c_2^w \mid X) = \sum_{v \to v', \lambda(v)=c_1, \lambda(v')=c_2} F[i, v] \cdot a_{v,v'} \cdot B[i+1, v', w] \tag{4.4}$$

Term $F[i, v]$ from the equation can be computed by the forward algorithm and the term $B[i, v, w]$ by simple extension of the backward algorithm as follows. Value $B[i, v, w]$ is the sum of the probabilities of all state paths, that start at position i in state w, generate tail of the sequence and the colors of the first w non-silent states are $\lambda(v)$. Value $B[i, v, 0]$ is computed by backward algorithm. The rest of B can be expressed by following equations.

$$B[i, v, w] = \sum_{v \to v', \lambda(v)=\lambda(v')} B[i+1, v, w-1] \cdot e_{v,X_i} \cdot a_{v,v'} \tag{4.5}$$

Unlike the backward algorithm defined in Section 2.2 we slightly changed the interpretation of the quantity $B[i, v, 0]$. In this case the state v emits the symbol X_i if v is non-silent state.

The computation of the term $\Pr\left(\Lambda'_{b-1\ldots b+w-1} = c_1^w c_2|X\right)$ is symmetric to the computation of the term $\Pr\left(\Lambda'_{b-w\ldots b} = c_1 c_2^w|X\right)$ and therefore we omitted it.

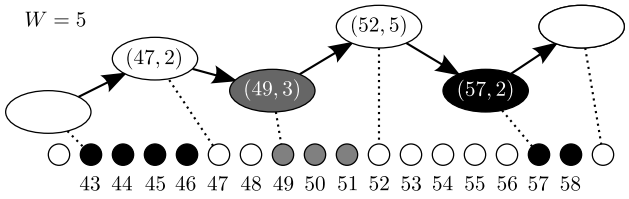

Figure 4.4: Part of a path in an annotation graph that represents one annotation. A node with color c and caption (b, w) represents the vertex (b, c, w) of the graph. The dashed lines connect each vertex to its corresponding feature.

4.1.2 Finding the Annotation

Once we have computed the posterior probabilities, we can find the annotation with the highest expected reward. We will do it by constructing a directed acyclic graph, in which every path from the starting vertex to the end vertex corresponds to an annotation of sequence X and the weight of that path will be reward of that annotation. Conversely, for every annotation of X there will be a path with these properties. We will refer to this graph as to the *annotation graph*.

The vertices of the annotation graph graph will be triples $(b, c, w), 1 \leq b \leq n, c \in C, w \leq W$ and a special start vertex $(0, \Lambda_0, 1)$ and end vertex $(n + 1, \Lambda_{n+1}, 1)$. All rules apply to the regular vertices as well as the start and end vertex. A vertex represents the feature of color c with left boundary at b and length exactly w if $w < W$, or at least W otherwise. There will be an edge between vertices (b_1, c_1, w_1) and (b_2, c_2, w_2) if and only if $b_2 = b_1 + w_1$ or $b_2 \geq b_1 + W, w + 1 = W$ and $c_1 \neq c_2$ with weight $R_\gamma(p(i_2, c_1, c_2, w_1, w_2))$. We call edges for $w_1 = W$ long-distance edges. Figure 4.4 shows an example of annotation graph.

Each edge corresponds to the feature of color c_1 at positions $b_1, \ldots, b_2 - 1$. The weight of the edge corresponds to the expected reward of boundary b_2. Since $b_2 > b_1$, the annotation graph is acyclic and therefore we can find the longest path from the start vertex to the end vertex in linear time [CSRL01]. This path will correspond to the state path with the highest expected reward as we show int he next theorem. But first we need one technical definition.

Definition 23 *The path $w = w_1 w_2 \ldots w_k$ correspond to annotation Λ, if for all i the edge (w_i, w_{i+1}) correspond to i-th feature of Λ.*

Theorem 5 *The algorithm described above find the annotation of sequence X with the highest expected reward.*

Proof. First we show that there is one to one correspondence between the paths in graph

and annotations. In the second part of the proof we show that the weight of the path is equal to the expected reward of corresponding annotation.

Let $w = (0, \Lambda_0, 1)(b_1, c_1, w_1)(b_2, c_2, w_2) \dots (b_k, c_k, w_k)(n+1, \Lambda_{n+1}, 1)$ be a path in the annotation graph. Then the only one corresponding annotation to this path is $\Lambda = c_1^{b_2-b_1} c_2^{b_3-b_2} \dots c_{k-1}^{b_k-b_{k-1}}$. We can also reverse this assignment and get the path from annotation. It easy to check, that all these assignments are injections and therefore there is one to one correspondence between the paths and annotations.

Now we prof the rest. Let $w = (b_1, c_1, w_1)(b_2, c_2, w_2) \dots (b_k, c_k, w_k), b_1 = 1, b_k = n+1$ be the path and $\Lambda = c_1^{b_2-b_1} c_2^{b_3-b_2} \dots c_{k-1}^{b_k-b_{k-1}}$. The weight of the path is

$$\sum_{i=1}^{k-1} R_\gamma(p(b_{i+1}, c_i, c_{i+1}, w_i, w_{i+1}))$$

which is exactly the expected reward. QED

The construction described above has one drawback. The number of long-distance edges is quadratic in the length of the sequence. We can lower this number by adding a special collector vertices to our construction. The collector vertices have similar function as the silent states in the HMM. In particular, we add to construction vertices (b, c), $1 \le b \le n, c \in C$ and replace long distance edges with these edges for every b_1, w, c_1, c_2:

- Edge from (b_1, c_1) to $(b_1 + 1, c_1)$ with zero weight.
- Edge from (b_1, c_1, W) to (b_1, c_1) with zero weight.
- Edge from (b_1, c_1) to $(b_1 + W, c_2, w), c_1 \neq c_2$ with weight $R_\gamma(p(b_1 + W, c_1, c_2, W, w))$.

This construction is correct because we replace every long-distance edge with exactly one chain of collector vertices with same total weight and we did not connect any vertices with collector vertices that were not connected by long-distance edge.

With collector edges, the resulting graph has only $O(nW^2C^2)$ edges and therefore we can find the longest path in $O(nW^2C^2)$ time. The overall running time of our algorithm is $O(nW|E| + nW^2C^2)$. Note that the time complexity is linear in the length of the sequence for a constant-sized HMM. The HERD algorithm is slower than the Viterbi algorithm by a factor of W.

4.2 Extension to HMMs With Silent States

The algorithm for computation of $\Pr(a_{i \dots i+w} = c_1 {c_2}^w \mid X)$ does not work on HMMs with silent states. We can replace silent states with additional edges, but in the worst case

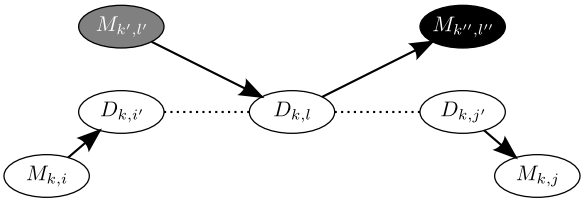

Figure 4.5: The submodel of jpHMM that breaks simplicity.

we add $O(|V|^2)$ edges to the model, increasing the running time. A good example is the jpHMM from Chapter 3. jpHMM contains a chain of delete states for every profile. Removing silent states in the jpHMM will add $O(kn^2)$ edges, where n is the length of the profile and k is the number of profiles. Thus if the sequence has also length n, the running time of the HERD would be $O(kn^3W + nW^2k^2)$. To improve running time we show two extensions of our algorithm. One works on a large class of HMMs with silent states and the other works on any well-defined HMM.

4.2.1 Simple HMMs

In general, we do not assign colors to silent states ($\lambda(u) = \varepsilon$ for a silent state). We define an alternative coloring function c' that assigns colors to silent states.

Definition 24 *Let H be an HMM and let $c : V \to C \cup \{\varepsilon\}$ be its coloring function. Then an* alternative coloring function *is function $\lambda' : V \to C$ such that for every non-silent state $v, \lambda'(v) = \lambda(v)$.*

Definition 25 *State path $u_1u_2 \dots u_k$ is* **simple** *if and only if the number of consecutive state pairs (u_i, u_{i+1}) with $\lambda'(u_i) \neq \lambda'(u_{i+1})$ is at most 1.*

Definition 26 *Let H be an HMM, $Q \subseteq V$ be the set of silent states. HMM H is* simple*, if and only if there exists alternative coloring function λ' such that every state path $uu_1u_2 \dots u_kv, u, v \in V \backslash Q, u_i \in Q$ is simple.*

Intuitively we require that any path from say a white non-silent state to a black non-silent state through several silent states contains first several (possibly zero) white states followed by several black states and thus changes the color exactly once.

Unfortunately, jpHMM from Chapter 3 is not simple (if it contains more than 2 profiles). From match state $M_{k,i}$ it is possible to get to the match state $M_{k,j}, j > i$ through delete states. Since both match states have the same color k, all delete states between them need to have color k in alternative coloring function. Let $D_{k,l}$ be a delete state on such a path between match states $M_{k,i}$ and M_{k_j} (see Figure 4.5). There exists a match state $M_{k',l'}$

from a different profile from which a jumping edge leads to $D_{k,l}$. There also exists an edge from $D_{k,l}$ to a match state $M_{k'',l''}, k \neq k''$. Since $D_{k,l}$ has color k, the path $M_{k',l'}D_{k,l}M_{k'',l''}$ is not simple.

We show how to modify jpHMM so that it is simple. As we can see in Figure 4.5 we have to remove either jumps from delete states to match states of different profiles or from match states to delete states of different profiles. Now we show, that if we remove all jumps from match states to delete states from another profile, the resulting HMM will be simple.

Let λ be the alternative coloring function defined in following way. Let D_{ij} be the delete state from jpHMM, that is in j-th profile with color c_j. Then $\lambda'(D_{ij}) = c_j$. To other silent states in the model we will assign new unique color c_u. Since transitions to delete states from another profiles are not allowed, therefore all states paths $uu_1 \dots u_k v, u, v \in V \backslash Q, u_i \in Q$ is simple, because all u_i has same color as u. The other silent states in jpHMM are at beginning or the end of the sequence. The first ones are not accessible from any non-silent states and there is no path from ending silent states to non-silent state. Therefore this changed jpHMM is simple.

Since we have removed from the model delete to match transitions, we doubled the jump transitions from the match states to the delete states to preserve the probability of jump from one profile to another profile.

Now we will use the simplicity of HMM to reduce the time complexity of our algorithm. Let H be a simple well defined HMM and λ' be the corresponding alternative coloring function. We show how to compute $\Pr(a_{i\dots i+w} = c_1 {c_2}^w \mid X)$. Let $F[i, v, w], w \in \{0, 1\}$ be the sum of probabilities of all state paths ending in state v and generating the first i symbols of X such that the last w symbols have annotation $\lambda'(v)$. Let $B[i, v, w], 0 \leq w \leq W$ be the sum of probabilities of all state paths starting in state v and generating symbols $x_i, \dots, x_n$ such that the first symbols $x_i, \dots, x_{i+w-1}$ have annotation $\lambda'(v)$. Then

$$\Pr(\Lambda_{i\dots i+w} = c_1 {c_2}^w \mid X) = \frac{\sum_{v \to v', \lambda(v)=c_1, \lambda(v')=c_2} F[i, v, 1] \cdot a_{v,v'} \cdot B[i+1, v', w]}{\Pr(X)} \tag{4.6}$$

To prove this equality, consider a state path s that satisfies $\Lambda_{i\dots i+w} = c_1 {c_2}^w$. Let u be the i-th non-silent state in s and let v be the $i+1$-th non-silent state in s. Clearly, state u is responsible for emission of color Λ_i and v emitted color Λ_{i+1}. States u and v may be in s connected by several silent states, and the path $u = s_j, s_{j+1}, \dots, s_k = v$ is simple. Because $\lambda(u) = c_1$ and $\lambda(v) = c_2$, there exists a unique index l where this path changes color from c_1 to c_2. State path s is counted in $F[i, s_l, 1] \cdot a_{s_l, s_{l+1}} \cdot B[i+1, s_{l+1}, w]$.

Finally, we show how to compute F and B. Recall that v_0 is the start state of H and v_{M-1} is the final state. Values $F[i, v, 0]$ can be computed by the standard forward algorithm and $B[i, v, 0]$ by the backward algorithm from Section 2.2.

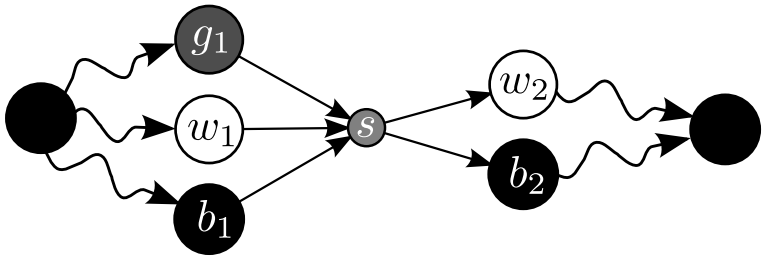

Figure 4.6: Example of an HMM with one silent state s on which the simple extension of the forward algorithm does not work.

$$F[i,v,1] = \sum_{u \to v, v \notin Q} F[i-1,u,0] \cdot e_{v,x_i} a_{u,v} + \sum_{u \to v, v \in Q, \lambda'(u)=\lambda'(v)} F[i,u,1] \cdot a_{u,v}$$

$$B[i,u,1] = \sum_{u \to v, u \notin Q} B[i+1,v,0] \cdot e_{u,x_i} \cdot a_{u,v} + \sum_{u \to v, u \in Q, \lambda'(u)=\lambda'(v)} B[i,v,1] \cdot a_{u,v}$$

$$B[i,u,w>1] = \sum_{u \to v, u \notin Q, \lambda'(u)=\lambda'(v)} B[i+1,v,w-1] \cdot e_{u,x_i} \cdot a_{u,v} + \sum_{u \to v, u \in Q, \lambda'(u)=\lambda'(v)} B[i,v,w] \cdot a_{u,v}$$

The states of HMM have to be topologically sorted as in forward algorithm in Section 2.2.

This part of the algorithm works in $O(nW|E|)$ time as in the case of HMMs without silent states. Jumping HMMs after our modification are simple HMMs and therefore algorithm described above can be used on them. Space complexity is $O(nW|V|)$ and as we will discuss in Section 4.4 it can be improved to $O(\sqrt{n}W|V|)$.

4.2.2 Well Defined HMM

The algorithm from the previous section does no work correctly on general well-defined HMMs. Consider the HMM in Figure 4.6. Relevant part of the HMM has one silent state s, one gray state g_1, two white-colored states w_1, w_2, and two black-colored non-silent states b_1, b_2. Without loss of generality, let color of state s be black. Then any state path that goes through $g_1 s w_2$ will not be counted but those state paths generates valid annotation. Therefore we have to had additional information about a color of the last emitted symbol.

We will redefine F and B by adding parameter c, color of the last non-silent state. Let $F[i,v,w,c]$ be the sum of probabilities of all state paths ending in state v and generating the first i symbols of X so that the symbols at position $i-w+1,\ldots i$ have color c. Let $B[i,v,w,c]$ for $w \geq 1$ be the sum of probabilities of all state paths starting in state v

and generating symbols $x_i \dots x_n$ such that symbols $x_i, \dots x_{i+w-1}$ have color c. Note that the values $F[i, v, 0]$ and $B[i, v, 0]$ are defined as in the previous section. Obviously, if v is not silent state and $c \neq \lambda(v)$, then $F[i, v, w, c] = V[i, v, w, c] = 0$. Note that now we use coloring function, not the alternative coloring function, and therefore silent states do not have assigned any color.

With these changes we will compute quantity $\Pr(\Lambda_{i\dots i+w} = c_1 {c_2}^w \mid X)$ using the following formula.

$$\Pr(\Lambda_{i\dots i+w} = c_1 {c_2}^w \mid X) = \frac{\sum_{v \to v', \lambda(v')=c_2} F[i, v, 1, c_1] \cdot a_{v,v'} \cdot B[i+1, v', w, c_2]}{\Pr(X)} \tag{4.7}$$

Since v' has a color, it is a non-silent state. Therefore we sum the probabilities of all state paths, that have previous color c_1 ($F[i, v, 1, c_1]$) and next w symbols have color c_2 ($B[i+1, v', w, c_2]$).

The forward and backward quantities can be computed by the following equations. Note that sates of HMM have to be topologically sorted as in forward algorithm. Values $F[i, v, 0]$ and $B[i, v, 0]$ are computed by the forward and backward algorithm in Section 2.2.

$$\begin{aligned}
F[i, v, 1, c] &= \sum_{u \to v, v \notin Q, \lambda(v)=c} F[i-1, u, 0] \cdot e_{v,x_i} \cdot a_{u,v} + \sum_{u \to v, v \in Q} F[i, u, 1, c] \cdot a_{u,v} \\
B[i, v, 1, c] &= \sum_{u \to v, v \notin Q, \lambda(u)=c} B[i+1, u, 0] \cdot e_{v,x_i} \cdot a_{u,v} + \sum_{u \to v, v \in Q} B[i, u, 1, c] \cdot a_{u,v} \\
B[i, v, w>1, c] &= \sum_{u \to v, v \notin Q, \lambda(u)=c} B[i+1, u, w-1, c] \cdot e_{u,x_i} \cdot a_{u,v} + \sum_{u \to v, v \in Q} B[i, u, w, c] \cdot a_{u,v}
\end{aligned}$$

Time and space complexity of the algorithm increases by a factor of C. Time complexity $O(nWC|E| + nC^2W^2)$ and space complexity is $O(nC|V| + WC|V| + nC^2)$. As in the previous section, we can improve space complexity to $O(\sqrt{n}C|V| + WC|V| + nC^2)$.

4.3 Different Gain Functions

In Section 2.8 we have defined the gain function $G(\Lambda, \Lambda')$. In this section we show how to change the algorithm to cope with various generalizations of this gain functions. Our framework is flexible and therefore we we need to change only small parts of the algorithm.

Our gain function does not take into account the distance of boundaries that are buddies, as long as they are within W symbols of each other. We may want to penalise distant buddies in order to predict breakpoints more precisely. For example, we may assign score β^d to buddies in distance d, where $0 < \beta \leq 1$. We go further and allow almost any function to be used as the score.

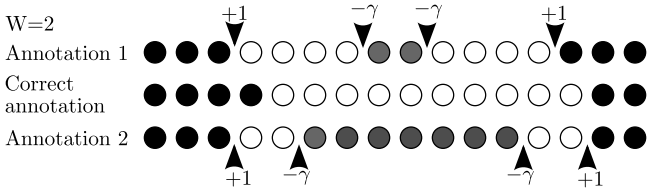

Figure 4.7: Annotation 1 and Annotation 2 have the same score $(2 + -2\gamma)$ in the gain function $G(\Lambda, \Lambda')$, but the first one is better.

Definition 27 *Let $f : \mathbb{N} \rightarrow < 0, 1 >$ be a monotonically non-increasing function. Then $G_f(\Lambda, \Lambda')$ is a gain function that assigns score $f(|i - j|)$ to each boundary i in Λ that has a buddy j in Λ' and score $-\gamma$ to boundaries in Λ without a buddy.*

Note that condition $|i - j| < W$ still holds and therefore $f(n)$ can be described by $W + 1$ values.

Given a sequence X, in the highest expected reward decoding we want find the annotation maximizing $E_{\Lambda'|X}[G_f(\Lambda, \Lambda')]$. Let $B(\Lambda)$ be set of all boundaries in Λ. Using linearity of expectation, $E_{\Lambda'|X}[G_f(\Lambda, \Lambda')] = \sum_{b \in B(\Lambda)} R_{f,\gamma}(b \mid \Lambda, X)$, where $R_{f,\gamma}(b \mid \Lambda, X)$ is the expected gain of boundary b in annotation Λ given sequence X. Previously, considering only constant f we had $R_{f,\gamma} = p_{\Lambda,b} - \gamma(1 - p_{\Lambda,b})$, where $p_{\Lambda,b}$ is posterior probability that boundary b in Λ has a buddy. The formula is somewhat mode complicated for general f.

$$R_{f,\gamma}(b \mid \Lambda, X) = \sum_{b' \in B'(b,\Lambda)} \Pr(b, b'|\Lambda, X) \cdot f(|b - b'|) - \gamma(1 - p_{\Lambda,b}) \tag{4.8}$$

where $B'(b, \Lambda)$ is the set of all possible buddies of b in any annotation given Λ and $\Pr(b, b' \mid \Lambda, X)$ is the posterior probability that b and b' are buddies, which can be expressed as in Section 4.1.

$$\Pr(b, b' \mid \Lambda, X) = \Pr(\Lambda'_{\min\{b,b'\} \ldots \max\{b,b'\}} = \Lambda_b^{|b-b'|-1} \Lambda_{b+1} | X)$$

We will consider another generalisation of the gain function motivated by the example from Figure 4.7. The two different labeling with same score. One have Both of these annotations shown in the figure have same score, but the second one has eight incorrect labels, while the first one has only four. In order to avoid such cases, we will define a gain function that also receives reward β for each correctly predicted label.

Definition 28 *Let $f : \mathbb{N} \rightarrow < 0, 1 >$ be a monotonically non-increasing function. Then*

$$G'_f(\Lambda, \Lambda') = G_f(\Lambda, \Lambda') + (|\Lambda| - d_H(\Lambda, \Lambda')) \cdot \beta$$

Recall, that d_H is a Hamming distance defined in Section 2.10.

With this gain function, annotation 1 in figure has a higher score than annotation 2. Now we show, how to find a labeling with highest expected gain.

As before, we are searching for the annotation, that maximizes

$$\begin{aligned} E_{\Lambda'|X}[G'_f(\Lambda,\Lambda')] &= E_{\Lambda'|X}[G_f(\Lambda,\Lambda')] + E_{\Lambda'|X}[(|\Lambda| - d_H(\Lambda,\Lambda'))\cdot\beta] \\ &= E_{\Lambda'|X}[G_f(\Lambda,\Lambda')] + \beta\cdot E_{\Lambda'|X}[(\sum_{i=1}^{|\Lambda|}(\Lambda_i=\Lambda'_i))\cdot\beta] \\ &= E_{\Lambda'|X}[G_f(\Lambda,\Lambda')] + \beta\cdot\sum_{i=1}^{|\Lambda|} E_{\Lambda'|X}[(\Lambda_i=\Lambda'_i)\cdot\beta] \\ &= \sum_{b\in B(\Lambda)} R_{f,\gamma}(b\mid\Lambda,X) + \beta\cdot\sum_{i=1}^{|\Lambda|}\Pr(\Lambda_i=\Lambda'_i)|X)\cdot\beta \end{aligned}$$

The posterior probability $\Pr(\Lambda_i = \Lambda'_i \mid X)$ can be computed by formula

$$\Pr(\Lambda_i=\Lambda'_i\mid X) = \sum_{v\notin Q, c(v)=\Lambda_i} \frac{F[i,v,0]\cdot B[i,v,0]}{e_{v,\Lambda_i}\cdot\Pr(X)} \tag{4.9}$$

To compute an annotation with the highest expected reward using G' as gain function, we have to change the algorithm more than with the previous gain function. We have to alter the weight of each edge in the annotation graph from Section 4.1 increasing it by the sum of the posterior probabilities of the colors on corresponding feature. For each edge from vertex (i,w,c) to (j,w',c') with weight d we assign new weight $d+\beta\sum_{k=i}^{j-1}\Pr(\Lambda_k = c\mid X)$.

We can also adapt the construction with collector vertices by altering the weight of the edges between collector vertices. In particular, the edge from (b_1,c_1) to (b_1+1,c_1) will have weight $\beta\Pr(\Lambda_{b_1}=c_1\mid X)$ and the edge from (b_1,c_1) to $(b_1+W,c_2,w), c_1\neq c_2$ will have weight $\beta\Pr(\Lambda_{b_1\ldots b_1+W-1}=c_1^W\mid X)$. All other edges will be altered as in previous case. This alteration does not affect time and space complexity of the HERD algorithm if we precompute partial sums of the posterior probabilities $\Pr(\Lambda_i = c\mid C)$ in $O(|C|n|T|)$ time. Then we will be able to alter every edge in the $O(1)$ time.

4.4 Implementation Details

We have implemented several variants of the HERD algorithm in C++. Our implementation works only on simple HMMs, in order to achieve a reasonable space complexity. We implemented all variants of the gain functions but we restricted our implementation only on the simple HMMs.

4.4.1 Memory Management

Memory requirements of our algorithm depend on the length of the sequence and the size of the HMM. Recall that asymptotic memory complexity is $O(n|V|W + n|C|^2W)$, where n is the length of the sequence. For example, a typical HIV genome has length almost $10,000$ and the jumping HMM we use has $365,596$ states. If we use double precision floating point numbers, then we need $108GB$ of RAM to store the tables even for $W = 1$, which does not allow to run our algorithm on today's desktop computers. In this section we discuss techniques for improving the memory complexity of our algorithm. The unoptimized algorithm can be expressed by the following pseudocode:

```
1 F = ComputeF()
2 B = ComputeB()
3 for i in 1...length(sequence):
4     Pr[i] = ComputePr (F[i],B[i+1])
```

Here `ComputePr(...)` is the summation 4.6, $F[i]$ are values $F[i, c, w]$ for all $c \in C, 0 \leq w \leq W$ and $B[i]$ is similar. We will call $F[i]$ and $B[i]$ rows, despite the fact that they have more dimensions. In order to compute $F[i]$ (or $B[i]$), we need only $F[i-1]$ (or $B[i+1]$) respectively. We can compute $\Pr(\Lambda_{i...i+w} = c_1{c_2}^w \mid X)$ in the order of increasing (or decreasing) i, and therefore we have to keep in memory only two rows of $F[i]$ (or $B[i]$) respectively. We implemented the algorithm with decreasing i, because rows of B are larger than rows of F. The pseudocode of the optimized algorithm:

```
1 F = Compute](
2 B = ComputeInitialRowB(i)
3 for i in length(sequece)-1...1:
4   Pr[i] = ComputePr (F[i],B)
5   B = ComputeBFromPreviousRow(i-1,B)
```

By remembering only two rows of B, we save some space, but it is not enough. We cannot do the same for B, because we would have to recompute $F[i]$ each time from $F[0]$ and thus the time complexity will raise by factor of n.

To compute F efficiently, we have used the classical checkpointing [TH98]. We divided F into $\lceil n/k \rceil$ buffers, each of k rows (except the last one, which may be smaller). For each buffer we keep in memory the first row. We also keep one whole buffer. If we need a row from a buffer, that is not in memory, we recompute that buffer starting at its first row and return the corresponding row. We need rows in decreasing order, and therefore we recompute each buffer at most twice. Therefore this trick does not affect the asymptotic time complexity of our algorithm, but may double the actual running time.

Overall we keep $r(k) = n/k+k+O(1)$ rows in memory. Function $r(k)$ has the minimum at $k = \sqrt{n}$, which is the buffer size we use.

Memory requirements are $O(\sqrt{n}|V| + W|V| + nC^2)$ for simple HMMs and $O(\sqrt{n}|V|C + W|V| + nC^2)$ for well-defined HMM. In our implementation checkpointing can be omitted for small input.

Chapter 5

Experiments

We have conducted several experiments to measure the accuracy of HERD algorithm. In each experiment we measure the following attributes:

1. Base accuracy – the fraction of the correctly predicted colors.
2. Reward – the reward of the predicted annotation with respect to correct annotation.
3. Feature specificity and sensitivity – feature is correctly predicted if it's boundaries are misplaced by at most W bases. Sensitivity is the ratio of the number of correctly predicted features and the total number of features in the correct annotation. Specificity is the ratio of the number of correctly predicted features and the total number of predicted features.
4. Average error in placement of correctly predicted boundaries – correctly predicted boundary is boundary of correctly predicted feature and we measure its average distance to the correct position.

We use synthetic data generated from a small toy HMM as well as real HIV sequences and simulated HIV sequences generated from the model. We compared the HERD algorithm with Viterbi algorithm and in case of the toy HMM also with the extended Viterbi algorithm. For all experiments on HIV data, we use jpHMM as the implementation of the Viterbi algorithm. In some experiments we compare HERD also with posterior decoding.

Note that average error in placement of boundaries is not very accurate measure, because for every run this quantity is mean over different set of boundaries.

5.1 Toy HMM

The first experiments were done on a small toy HMM [BBV07] shown in Figure 5.1. This HMM has the multiple path problem, and therefore finding the best annotation should be

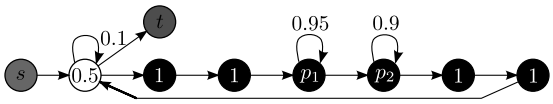

Figure 5.1: A toy HMM. Toy HMM emmits symbols over the binary alphabet. The numbers inside states represent the emission probability of symbol 1. The starting state s and the final state t are silent. This HMM was insprired by models of CT-rich intron tails in gene finding [BBV07]. The probabilities p_1 and p_2 are parameters and during testing were set to different values.

Table 5.1: The prediction accuracy on randomly generated data generated from the HMM in Figure 5.1. (i) The fraction of correctly predicted bases with respect to correct annotation. (ii) Reward $G(\Lambda, \Lambda')$ of the predicted annotation and the correct annotation. For evaluation the gain function parameters were set to $W = 5$ and $\gamma = 1$. (iii) The feature specificity (sp.) and sensitivity (sn.), (iv) average error in boundary placement

Algorithm	% bases correct$^{(i)}$	Gain $^{(ii)}$	Feature sp.$^{(iii)}$	Feature sn.$^{(iii)}$	Avg. err. $^{(iv)}$
HMM parameters $p_1 = 0.9$**,** $p_2 = 0.9$					
HERD $W = 5, \gamma = 1$	88.7%	**12.7**	**75.9%**	**66.9%**	1.8
HERD $W = 1, \gamma = 1$	47.5%	3.0	55.1%	17.8%	**0.0**
HERD $W = 1, \gamma = 0.1$	90.4%	2.4	51.8%	66.0%	0.9
Viterbi	89.4%	8.9	66.3%	47.3%	0.7
Extended Viterbi	**91.2%**	10.3	69.9%	56.2%	0.8
HMM parameters $p_1 = 0.7$**,** $p_2 = 0.8$					
HERD $W = 5, \gamma = 1$	77.6%	**5.9**	**54.8%**	39.3%	1.37
HERD $W = 1, \gamma = 1$	47.5%	3.0	55.0%	17.7%	**0.0**
HERD $W = 1, \gamma = 0.1$	79.6%	-2.7	38.2%	**43.9%**	0.9
Viterbi	75.0%	3.6	51.2%	25.7%	0.4
Extended Viterbi	**79.7%**	4.1	49.0%	31.3%	0.6

more accurate than annotation found by the Viterbi algorithm. We tested the toy HMM with various combinations of p_1 and p_2. From each HMM we have randomly generated 5000 sequences of mean length about 500 bases. We report the results for two instances of toy HMM, specifically $p_1 = p_2 = 0.9$ and $p_1 = 0.7, p_2 = 0.8$. The results are summarised in Table 5.1. Since the extended Viterbi algorithm (EVA) finds the most probable annotation in this HMM, it outperforms the Viterbi algorithm. The HERD with parameters $W = 5$ and $\gamma = 1$ has higher reward than other decoding methods. This is not surprising, because the data were generated from the same model and the HERD optimizes the expected reward. The HERD is worse in terms of the fraction of correctly predicted labels, because it ignores small differences in the boundary position. Finally, the HERD have also the best results in terms of feature specificity and sensitivity.

The results also show that the parameter setting is crucial for our method. HERD with parameters $W = 1, \gamma = 1$ have poor results in almost all measures. The performance was improved when we set the parameter γ to 0.1. The reason is that for $W = 1$, we sum fewer state paths to the posterior probability of the boundary and therefore most boundaries usually have negative expected reward. The HERD with parameters $W = 1$ is equivalent to maximum expected boundary accuracy decoding [GDSB07].

5.2 Detection of HIV Recombination

Since running the HERD algorithm on the whole HIV genome and whole jpHMM is computationally intensive, we evaluated our algorithm on shorter sequences. The tests were done on 1696 columns of the whole genomic alignment of the HIV virus, starting at position 6925. In this way we can run test with a higher number of sequences than Schultz *et al. (2006)* in a reasonable time. First we have sampled data from the model to estimate optimal parameters for our algorithm. Then we have tested the algorithm on artificial recombination data. In the end we have done test on several real full-length recombined sequences.

5.2.1 Sampled data

We have sampled 800 sequences together with their annotations from the model with jumping probability $P_j = 10^{-5}$. Recall that the P_j is the probability of a jump between two profiles in jpHMM. Unlike Schultz *et al.* we increase the jumping probability to have recombined sequences in the sampled data. The lower P_j correspond to the sequences without recombination and higher P_j correspond to the recombinant sequences. We run the HERD algorithm for various parameter values to find optimal parameter settings, for our algorithm. The Figure 5.2 demonstrates the impact of the penalty γ on the prediction accuracy. Based on the results, we set the value of γ to 2.2. With larger values of γ

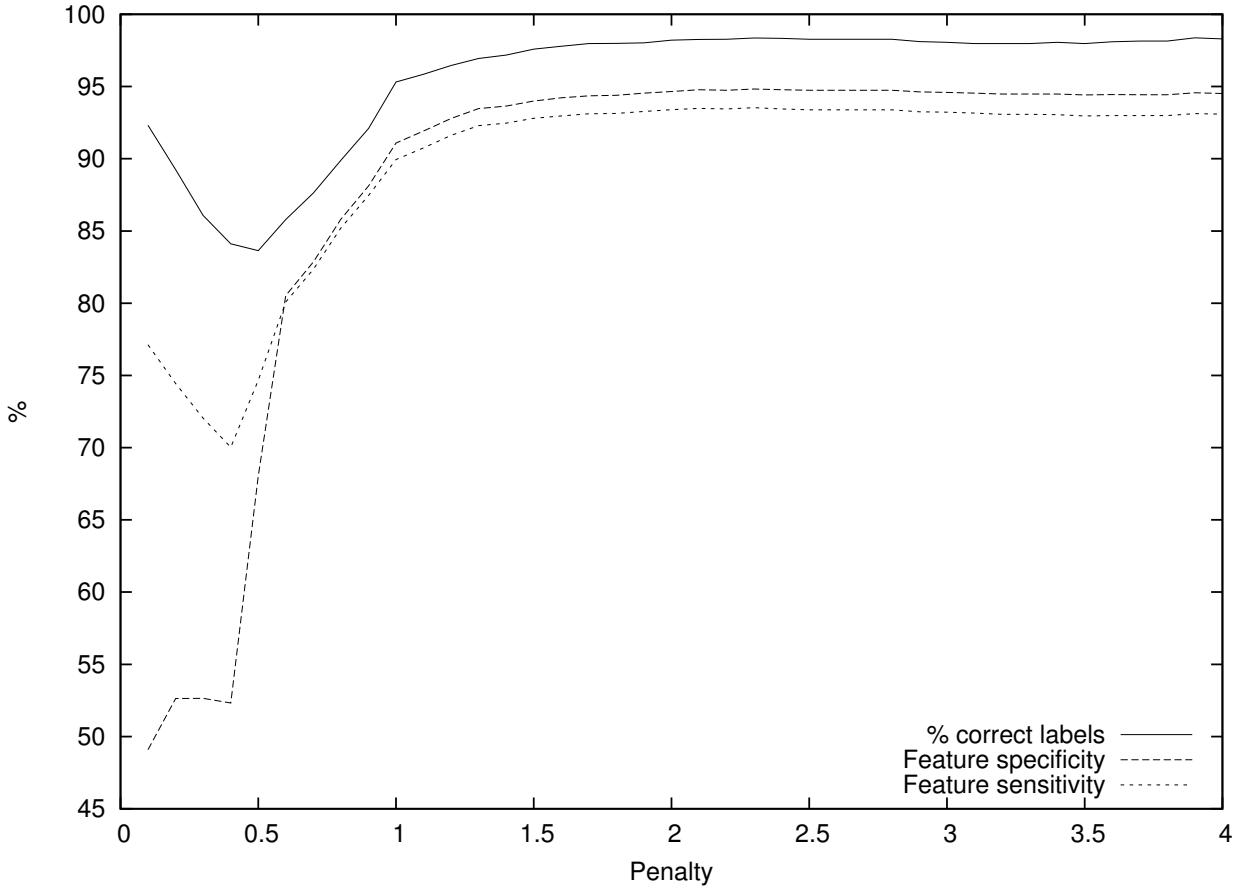

Figure 5.2: The prediction accuracy of the HERD algorithm for fixed window size $W = 10$ and variable value of penalty γ. The jump probability was set to $P_j = 10^{-5}$. The hithest highest feature specificity was with $\gamma = 2.3$ and the highest feature sensitivity was with $\gamma = 2.1$.

Table 5.2: The prediction accuracy on data generated from the jpHMM. The table columns are same as in Table 5.1 with one expection: the evaluation parameters for the gain functions are $W = 10$ and $\gamma = 2.2$.

Algorithm	% bases correct	Gain	Feature sp.	Feature sn.	Avg. err.
HERD, $W = 10, \gamma = 2.2, P_j = 10^{-5}$	98.27%	**3.96**	94.74%	93.45%	0.97
HERD, $W = 1, \gamma = 2.2, P_j = 10^{-5}$	81.33%	2.99	75.49%	71.25%	**0.39**
HERD, $W = 1, \gamma = 0.1, P_j = 10^{-5}$	99.31%	2.98	87.03%	91.68%	0.56
Viterbi, $P_j = 10^{-5}$	99.49%	3.79	92.4%	91.06%	0.55
Viterbi, $P_j = 10^{-9}$	99.18%	3.68	90.59%	88.23%	0.51
Posterior, $P_j = 10^{-5}$	**99.59%**	2.94	**94.96%**	**94.31%**	0.55

the feature sensitivity and specificity have slightly decreased, because with larger γ our algorithm did not have enough confidence to use the jump between colors. Figure 5.3 shows that increasing window size improves the performance of our algorithm. We set the window size to 10. The larger values would probably improve prediction performance of our algorithm even more but it would also increase running time. The value $W = 10$ is a good compromise between the accuracy and the running time.

The comparison with other decoding methods is in Table 5.2. We compared our algorithm with the Viterbi algorithm, the posterior decoding and the maximum accuracy boundary decoding (equivalent to HERD with $W = 1$). We run Viterbi algorithm with its default value of the jumping probability ($P_j = 10^{-9}$) and also with the jump probability we used for generating test data ($P_j = 10^{-5}$). HERD algorithm again achieved the highest reward. As we have expected, the posterior decoding has the highest base accuracy which is equal to its gain function. Perhaps unexpectedly, the posterior decoding has also the highest feature specificity and sensitivity. In general, the posterior decoding can in uncertain regions change the color frequently, but in the sampled data this event apparently did not occur. The Viterbi algorithm outperforms the HERD in terms of the base accuracy for both jump probabilities, but HERD have significantly better performance in with respect to the feature specificity and sensitivity. The maximum accuracy boundary decoding have comparable performance to the Viterbi algorithm (with exception to feature specificity where was Viterbi algorithm better). One surprising result was the high feature sensitivity and specificity of the posterior decoding.

5.2.2 Artificial Recombinations

To test if our algorithm correctly detects recombination in the HIV genome we have artificially created recombinant sequences from the real ones. We removed 62 sequences from

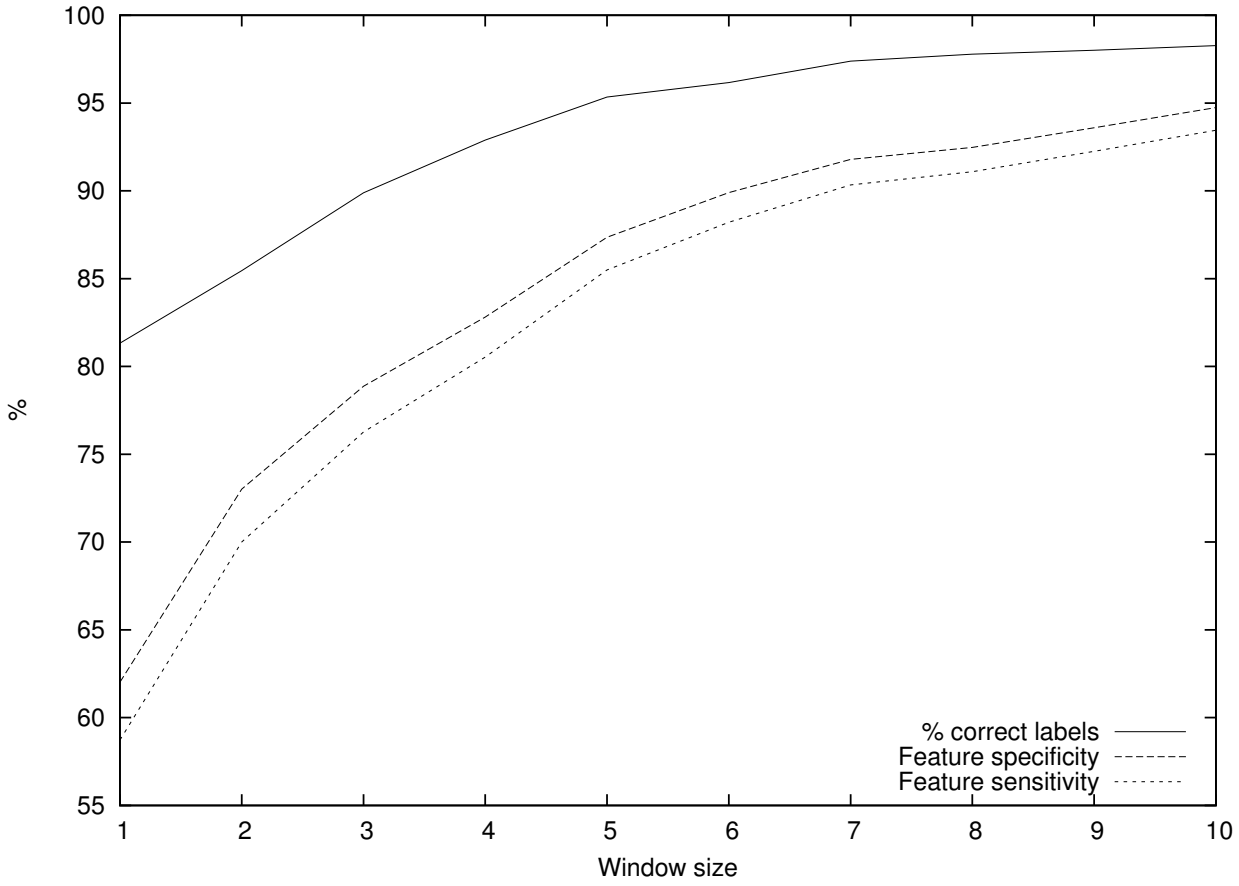

Figure 5.3: The prediction accuracy of the HERD algorithm for the fixed penalty $\gamma = 2.2$ and variable window size W. The jump probability was set to $P_j = 10^{-5}$. The accuracy raises with growing window size.

the source alignment. Specifically, from each of the subtypes $A1, B, C, D, F1$ we took 10 sequences, from subtypes $G, A2$ and $F2$ we took $5, 3$ and 4 sequences respectively. From these sequences we created artificial recombinants with recombination every 300 bases. The remaining sequences we used to estimate the paramteters of the jpHMM. Again, these tests were done on 1696 columns of the whole genomic alignment of the HIV virus, starting at position 6925.

With artificial recombinants we do two tests. At first we took subtypes A, B, C, G and from each pair of subtypes we have created 50 recombinant sequences. Each recombinant is a mosaic of two real sequences, one from each subtype. From three pairs of sub-subtypes $A1$ and $A2$, $F1$ and $F2$, B and D of the same type we created together 170 artificial recombinant sequences. Note, that subtypes B and D have very small phylogenetic distance, and therefore they should be considered as sub-subtypes [RAB$^+$00].

We test the HERD algorithm with the parameters γ and W derived from sampled data, but our algorithm did not have the expected prediction accuracy. Therefore we have run the HERD for various values of parameter γ. The results are in the Figure 5.4. Unlike on the sampled data, the prediction accuracy of our algorithm has significantly decreased for the larger values of γ. The reason for this is that in this test the recombined data does fit well our model and therefore the posterior probabilities of the boundaries are significantly smaller than on the sampled data. To overcome this problem we have decreased the penalty γ to 1.

We compared our algorithm with the Viterbi algorithm, the posterior decoding and the maximum accuracy boundary decoding. The results of these experiments are in Table 5.3. In the recombination between different subtypes the Viterbi algorithm have the best average base accuracy. The HERD with $\gamma = 1$ have significantly higher feature specificity and sensitivity than the Viterbi algorithm and even outperforms the posterior decoding in some measures. Unlike on the sampled data, the maximum accuracy boundary decoding has poor performance.

The results on recombinations within the same subtype are similar to the results on the recombinations between different subtypes, but all methods performs worse then in the previous case. This is expected, because the sequences from different sub-subtypes are more similar one to another and therefore it is harder to distinguish between them.

We have also tested our algorithm on the non-recombined sequences that we removed from an alignment. The Viterbi for any tested jumping probability and the HERD for jumping probability $P_j = 10^{-9}$ detected all sequences to be free of recombinations and classified them to correct subtype. HERD with jumping probability $P_j = 10^{-5}$ correctly predicts the non-recombined type in 83.9% sequences and in the rest of the sequences HERD detected a false recombination.

We also conduct experiments with the different gain functions from Section 4.3. We demonstrate them on the recombination detection of the recombinants within the same

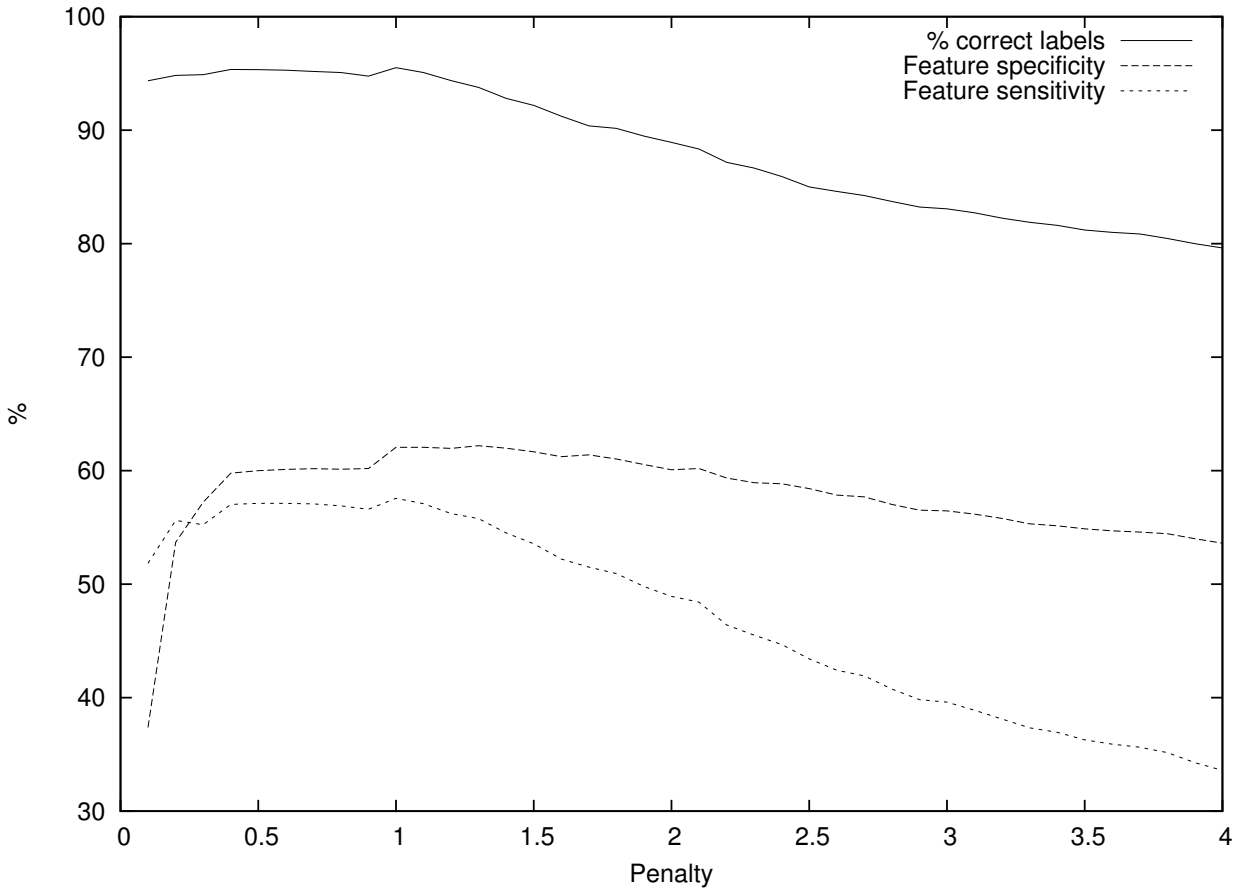

Figure 5.4: The prediction accuracy of the HERD algorithm for the fixed window size $W = 10$ on the artificial recombinants between different subtypes.

subtype. At first, we set weight β of the posterior probability to values $1/n, 10/n, 100/n$ and $1000/n$ where n is the length of the sequence. As we expected, adding the parameter β increased the number of correctly predicted labels and also the larger values of β increased the feature specificity and sensitivity. We have also tried to penalise the more distant boundaries to decrease the error of the prediction. The best results were achieved for the function $f(n) = 0.9^n + 0.4$, but this extension decreased the feature sensitivity and specificity of the HERD algorithm. The results are in Figure 5.4.

These tests shows that parameter estimation is for HERD important and that there is one principal problem. On the recombinant sequences HERD gave good results with $P_j = 10^{-5}$, but on sequences without recombination this parameter leads to undesired recombinations. This is not surprising, because the recombinations are infrequent which correspond to small value of P_j, but if the recombination occurs then in sequence is more recombinations and this corresponds to the larger value of P_j. In practice, we could perform a likelihood ratio test with nested models [Fel04], where we optimize the P_j for the input sequence, and the null model will be jpHMM with $P_j = 0$. If the test predicts that we have recombined sequence, we will use the higher P_j for recombination detection.

5.2.3 Circular Recombinant Forms

We have also run our algorithm on 12 naturally occurring recombinants (CRF02-CRF08, CRF10-CRF14). We used only 12 sequences, because the annotations of others CRFs were created with help of the jumping HMM, and therefore the comparison would not be very informative. We set the parameters of our algorithm to $P_j = 10^{-9}, W = 10, \gamma = 1.5, \beta = 10$. We run the Viterbi algorithm with the original parameter $P_j = 10^{-9}$. Since the reference annotations contain unannotated regions of uncertain origin, we have removed such regions from our statistics. Our algorithm performs slightly better than the Viterbi algorithm, but the performance of both algorithm are poor. The results are in Table 5.5. Note that we do not know the methods by which was annotated the sequences we used for evaluation and therefore we are not sure if we had correct annotations.

Table 5.3: The prediction accuracy on the HIV recombination data. The table columns are the same as in Table 5.1 with one expection: the evaluation parameters for the gain function are $W = 10$ and $\gamma = 1$.

Algorithm	% bases correct[(i)]	Gain [(ii)]	Feature sp.[(iii)]	Feature sn.[(iii)]	Avg. err.
Sequences with artificial inter-subtype recombination					
HERD $W = 10, \gamma = 2.2, P_j = 10^{-5}$	87.17%	2.44	59.34%	46.41%	1.97
HERD $W = 10, \gamma = 1, P_j = 10^{-5}$	95.5%	**2.49**	**62.05%**	57.56%	2.37
HERD $W = 1, \gamma = 0.1, P_j = 10^{-5}$	81.28%	1.06	37.19%	29.33%	**1.36**
Viterbi $P_j = 10^{-9}$	95.06%	2.01	53.08%	47.13%	1.82
Viterbi $P_j = 10^{-5}$	**96.27%**	1.88	51.83%	48.21%	1.86
Posterior decoding $P_j = 10^{-5}$	95.77%	2.26	60.35%	**57.72%**	2.08
Sequences with artificial intra-subtype recombination					
HERD $W = 10, \gamma = 1, P_j = 10^{-5}$	91.5%	**1.75**	**46.63%**	41.88%	2.65
HERD $W = 1, \gamma = 0.1, P_j = 10^{-5}$	74.34%	1.35	28.7%	18.5%	2.8
Viterbi $P_j = 10^{-5}$	93.17%	1.21	37.56%	35.17%	2.9
Viterbi $P_j = 10^{-9}$	88.01%	1.32	32.84%	26.08%	2.71
Posterior decoding $P_j = 10^{-5}$	**93.97%**	1.3	43.1%	**42.61%**	**2.61**
Sequences without recombination					
HERD, $W = 10, \gamma = 1, P_j = 10^{-9}$	**100.0%**	**2.0**	**100.0%**	**100.0%**	**0.0**
HERD, $W = 10, \gamma = 1, P_j = 10^{-5}$	93.7%	1.5	83.9%	83.9%	**0.0**
Viterbi $P_j = 10^{-9}$	**100.0%**	2.0	**100.0%**	**100.0%**	**0.0**
Viterbi $P_j = 10^{-5}$	**100.0%**	2.0	**100.0%**	**100.0%**	**0.0**

Table 5.4: The prediction accuracy on the HIV recombination data – the sequences with artificial intra-subtype recombination. The table columns are the same as in Table 5.1 with one expection: the evaluation parameters for the gain function are $W = 10$ and $\gamma = 1$.

Algorithm	% bases correct$^{(i)}$	Gain $^{(ii)}$	Feature sp.$^{(iii)}$	Feature sn.$^{(iii)}$	Avg. err.
The gain function with posterior probabilities					
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = 0$	91.5%	**1.75**	46.63%	41.88%	2.65
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = \frac{1}{n}$	93.73%	1.58	46.19%	43.53%	2.67
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = \frac{10}{n}$	94.02%	1.49	45.46%	43.16%	2.58
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = \frac{100}{n}$	94.04%	1.61	47.35%	47.18%	**2.6**
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = \frac{1000}{n}$	**94.11%**	1.5	**47.66%**	**47.67%**	2.78
Weighted gain function					
HERD $W = 10, \gamma = 1, P_j = 10^{-5}$	91.5%	**1.55**	44%	40.97%	**2.37**
HERD $W = 10, \gamma = 1, P_j = 10^{-5}, \beta = \frac{10}{n}$	**94.09%**	1.49	**44.73%**	**43.0%**	**2.37**

Table 5.5: The prediction accuracy on 12 naturally occurring recombinants. The table columns are same as in Table 5.1 with one exception: The evaluation paramtered for gain function are $W = 10, \gamma = 1.5$ and $\beta = 10$

Algorithm	% bases correct$^{(i)}$	Gain $^{(ii)}$	Feature sp.$^{(iii)}$	Feature sn.$^{(iii)}$	Avg. err.
HERD, $\gamma = 1.5, P_j = 10^{-9}, \beta = \frac{10}{n}$	**65.22%**	-10.5	**7.23%**	**10.02%**	2.71
Viterbi, $P_j = 10^{-9}$	64.51%	**-10.29**	6.4%	9.26%	**1.58**

Chapter 6

Conclusion

In this thesis we have studied hidden Markov models (HMMs) and their application to the viral recombination detection problem. In Chapter 2 we have defined hidden Markov models and summarized the well known decoding algorithms as well as recent decoding algorithms that give more accurate results in particular applications than the traditional methods. We express all those algorithm in terms of gain functions [HKS+09].

The third chapter discusses the application of the HMMs to the viral recombination problem in the HIV genome. In the Chapter 4 we have introduced a new gain function specifically designed for this application. We have developed an efficient algorithm that optimizes this function. We also suggest several generalizations of our gain function to improve accuracy of our decoding method. We call our method the highest expected reward decoding (HERD). HERD is designed for HMMs where the detection of exact annotation boundaries is difficult or when we are not interested in the exact boundaries.

We have implemented the algorithm and compared it with other decoding algorithms. The results in Chaper 5 showed that HERD has higher prediction accuracy than the Viterbi algorithm, which is currently used on the viral recombination problem. In particular, HERD has higher feature specificity and sensitivity than the Viterbi algorithm and similar sensitivity but higher specificity than the posterior decoding.

The experiments show that our method is very sensitive to correct parameter settings. The open question remains how to estimate the parameters in principal way. Optimizing the parameters to sequences and annotations sampled from the model did not work well on sequences that were not directly generated from the model. Other question is how the HERD would perform on other application domain, such as transmembrane proteins where similar methods were successful [BT10].

The web page of HERD algorithm is `http://compbio.fmph.uniba.sk/herd`.

Bibliography

[BBLV05] Brona Brejova, Daniel G. Brown, Ming Li, and Tomas Vinar. ExonHunter: a comprehensive approach to gene finding. *Bioinformatics*, 21(S1):i57–i65, 2005. Intelligent Systems for Molecular Biology (ISMB 2005).

[BBV07] Brona Brejova, Daniel G. Brown, and Tomas Vinar. The most probable annotation problem in HMMs and its application to bioinformatics. *Journal of Computer and System Sciences*, 73(7):1060–1077, 2007.

[BK97] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78 – 94, 1997.

[BKR02] Abraham Bookstein, Vladimir A. Kulyukin, and Timo Raita. Generalized hamming distance. *Inf. Retr.*, 5(4):353–375, 2002.

[BPSS01] Alvis Brazma, Helen Parkinson, Thomas Schlitt, and Mohammadreza Shojatalab. A quick introduction to elements of biology - cells, molecules, genes, funcional genomics, microarrays. *EMBL-EBI, http://www.ebi.ac.uk/microarray/biology_intro.html*, 2001.

[BT10] 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.

[Bur97] D S Burke. Recombination in HIV: an important viral evolutionary strategy. *Emerging Infectious Diseases*, 3(3):253 – 259, 1997.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[Edd98] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–63, 1998.

[Fel04] Joseph Felsenstein. Inferring phylogenies. *Sinauer Associates*, 2004.

[For73] G. David Forney Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[GDSB07] 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.

[GG03] 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.

[HKS+09] Michiaki Hamada, Hisanori Kiryu, Kengo Sato, Toutai Mituyama, and Kiyoshi Asai. Prediction of RNA secondary structure using generalized centroid estimators. *Bioinformatics*, 25(4):465–473, 2009.

[HU79] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[KKS05] 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.

[KLvHS01] Anders Krogh, Björn Larsson, Gunnar von Heijne, and Erik L.L Sonnhammer. Predicting transmembrane protein topology with a hidden markov model: application to complete genomes. *Journal of Molecular Biology*, 305(3):567 – 580, 2001.

[KPD+04] Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[Low76] B. T. Lowerre. *The Harpy speech recognition system*. PhD thesis, Carnegie-Mellon Univ., Pittsburgh, PA., 1976.

[LP02] 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.

[RAB+00] 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.

[SZB+09] Anne-Kathrin Schultz, Ming Zhang, Ingo Bulla, Thomas Leitner, Bette Korber, Burkhard Morgenstern, and Mario Stanke. jpHMM: improving the reliability of recombination prediction in HIV-1. *Nucleic Acids Research*, 37(Web Server issue):W647–651, 2009.

[SZL+06] 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.

[TH98] C. Tarnas and R. Hughey. Reduced space hidden Markov model training. *Bioinformatics*, 14(5):401–6, 1998.

[WJ94] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal Computational Biology*, 1(4):337–48, 1994.

Abstrakt

Autor:	Michal Nánási
Názov práce:	Biological sequence annotation with hidden Markov models
Univerzita:	Univerzita komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci diplomovej práce:	Mgr. Broňa Brejová PhD.
Rozsah práce:	54
Rok:	2010

Skryté Markovove modely (HMM) sú dôležitým nástrojom na modelovanie biologických sekvencii a ich anotácii. Anotovaním sekvencie myslíme priradenie popisov jednotlivým symbolom sekvencie podľa ich významu. Napríklad ak hľadáme gény, tak sa snažíme rozdeliť DNA sekvenciu na časti, ktoré kódujú proteíny (gény) a na tie, ktoré génmi nie sú. Skryté Markovove modely definujú pravdepodobnostnú distribúciu sekvencii a ich anotácii.

Dekódovanie zo skrytých Markovovych modelov je obyčajne realizované pomocou Viterbiho algoritmu. Viterbiho algoritmus nájde najpravdepodobnejšiu anotáciu len pre podtriedu všetkých skrytých Markových modelov. Vo všeobecnosti je anotácia sekvencii NP-ťažká a preto Viterbiho algoritmus môžeme použiť len ako heuristickú metódu.

V posledných rokoch sa ukázalo, že v určitých aplikáciach iné dekódovacie metódy majú lepšie výsledky ako Viterbiho algoritmus. V tejto práci predkladáme novú dekódovaciu metódu, ktorá berie do úvahy neurčitosť v presnej polohe hraníc jednotlivých regiónov. Naša metóda považuje anotácie, ktoré sa len trochu líšia v hraniciach regiónov za rovnaké. Nazvali sme ju dekódovanie s najvyššou predpokladanou odmenou (the highest expected reward decoding – HERD) a je založená na dekódovaní pomocou maximalizácie strednej hodnoty presnosti hraníc (maximum expected boundary accuracy decoding) [GDSB07].

Náš algoritmus sme testovali na probléme detekcie rekombinácií v genóme vírusu HIV a porovnali sme ho s existujúcim nástrojom, ktorý sa volá preskakujúce HMM (jumping HMM). HERD má lepšiu presnosť predikcie v rámci špecifickosti a senzitívnosti správne predikovaných jednofarebných regiónov.

Kľúčové slová: Skryté Markovove modely, anotácia sekvencií, rekombinácie vírusov