Enhancements to Hidden Markov Models for Gene Finding and Other Biological Applications

by

Tomáš Vinař

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2005

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, we present enhancements of hidden Markov models for the problem of finding genes in DNA sequences. Genes are the parts of DNA that serve as a template for synthesis of proteins. Thus, gene finding is a crucial step in the analysis of DNA sequencing data.

Hidden Markov models are a key tool used in gene finding. Yhis thesis presents three methods for extending the capabilities of hidden Markov models to better capture the statistical properties of DNA sequences. In all three, we encounter limiting factors that lead to trade-offs between the model accuracy and those limiting factors.

First, we build better models for recognizing biological signals in DNA sequences. Our new models capture non-adjacent dependencies within these signals. In this case, the main limiting factor is the amount of training data: more training data allows more complex models. Second, we design methods for better representation of length distributions in hidden Markov models, where we balance the accuracy of the representation against the running time needed to find genes in novel sequences. Finally, we show that creating hidden Markov models with complex topologies may be detrimental to the prediction accuracy, unless we use more complex prediction algorithms. However, such algorithms require longer running time, and in many cases the prediction problem is NP-hard. For gene finding this means that incorporating some of the prior biological knowledge into the model would require impractical running times. However, we also demonstrate that our methods can be used for solving other biological problems, where input sequences are short.

As a model example to evaluate our methods, we built a gene finder ExonHunter that outperforms programs commonly used in genome projects.

Acknowledgements

I would like to thank all the people, who contributed to this thesis. Thanks to both of my supervisors Ming Li and Dan Brown. During my years of PhD studies, they provided me with tremendous amount of support and extraordinary freedom to pursue my own curiosity, yet they were always eager to work on the problems with me and give me a guidance; to Broňa Brejová, my wife, my best friend, and also my closest research collaborator.

I would also like to thank members of my committee Therese Biedl, Ian Munro, Burkhard Morgenstern, and Romy Shioda for their guidance and insightful comments.

Special thanks to people who helped me in the beginnings of my research career by many hours spent in helpful discussions: Jonathan Badger, Haoyong Zhang, John Tsang, and Michael Hu. Thanks to Martin Demaine and Therese Biedl, who always encouraged me to start new things, and who helped me to set up bioinformatics problem sessions. Special thanks to Therese, under whose guidance we wrote our first research paper.

Thanks to all the other people with whom I had a pleasure to co-author research papers and reports: Jonathan Buss, Erik Demaine, Chrysanne DiMarco, Mohammadtaghi Hajiaghayi, Angele Hamel, Masud Hasan, Ian Harrower, Sandra Romero Hidalgo, Gina Holguin, Joe D. Horton, Alejandro Lopez-Ortiz, and Cheryl Patten.

Last, but definitely not least, I would like to thank my parents, for supporting me and encouraging me in all my endeavors.

To my brother,
who left us early.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hidden Markov models are a prime tool in the analysis of biological sequences. A wide variety of important problems in computational biology, such as gene finding (Burge, 1997), predicting the topology of transmembrane proteins (Krogh et al., 2001), predicting protein secondary structure (Bystroff et al., 2000), identifying protein families (Eddy, 1998), or searching for homologies (Brejová et al., 2004b), benefit from methods and algorithms based on hidden Markov models. As of July 2005, these methods support the basic research of almost 1500 genome projects in progress all over the world (Bernal et al., 2001).[1]

In this thesis, we explore several possible extensions to hidden Markov models for biological sequences. By careful study of the properties of biological sequences, we can design new algorithms that extend the modeling capabilities of hidden Markov models to achieve improved accuracy. However, such extensions are not free. In real applications, we need to balance such model extensions with a variety of limiting factors. We show trade-offs that exist between model faithfulness and the amount of data available for training (Chapter 2), the feasibility of running time of required algorithms (Chapter 3), and sometimes even with the inherent intractability of underlying problems (Chapter 4).

In hidden Markov models, many components work together to achieve high prediction accuracy. In such a system, a seemingly modest improvement of one component may lead surprisingly to improved performance of other components of the system as well. We observe such behaviour in hidden Markov models in our experiments in Chapters 2 and 3.

While our results can be generalized to many scenarios, in this thesis we focus on the *annotation problem*. Here, we first train the hidden Markov model to generate pairs of sequences and their annotations, *maximizing the likelihood* of a training set of known sequences, and afterwards, we *decode* new sequences by finding their most probable annotation in the trained model. We demonstrate our annotation methods on the problem of *gene finding*. In the rest of this chapter, we introduce and review previous work on hidden Markov models, on the annotation problem in general, and on gene finding specifically.

[1] The Genomes Online Database created by Bernal et al. (2001) is continuously updated with references and progress of the current genome projects worldwide.

1.1 Sequence Annotation and Hidden Markov Models

In this section we present an overview of hidden Markov models in context of the problem of *sequence annotation*, which is the central problem of this thesis. In this problem, we are given a sequence of observations. Our goal is to find particular features of this sequence (called *labels*). Our task is to assign a label to each observation in the sequence.

For example, we may have the sequence of the colors of roulette spins for an evening: red for odd numbers, black for even numbers, or green for zeroes. Even though there are numbers on the roulette wheel, we will be interested only in the colors of these numbers, so the sequence of observations will be a sequence of colors from the set $\{\text{red}, \text{black}, \text{green}\}$. We may suspect that the house occasionally changes the balance of the wheel so that the black numbers come more often to gain more profit from unsuspecting gamblers. We want to detect this trend in the data, and identify the spins from fair and unfair wheels. Thus, we are looking to label each color in the sequence of observations by one of the labels from the set $\{\text{fair}, \text{loaded}\}$.

Or, we may have a sequence of observations of smoke levels measured by a fire monitoring device. In this scenario, we want to recognize parts of the sequence that may correspond to a fire and raise an alarm. On the other hand, we do not want to raise an alarm when somebody is cooking dinner.

The sequence does not need to correspond to observations of some physical phenomenon. Tagging the words of a sentence with their parts of speech is a common example of the annotation problem in the field of natural language processing. In areas such as bioinformatics, we often annotate DNA or protein sequences. In these problems, we are usually interested in functional or structural features of the sequence, and we want to mark every symbol in the sequence with its particular feature. For example, in gene finding, a DNA sequence can be functionally split into *introns*, *exons*, and *intergenic regions*.

The problems above have some features in common. For all, it is very hard to define straightforward rules resulting in good annotation. Given this complexity, if we attempt to formulate these problems as combinatorial or optimization problems based on common sense, we may end up with a variety of different formulations, each yielding different results. Thus, developing algorithms to solve these problems, or sometimes even attempting to provide clear formulations of these problems, is an exceptionally hard task.

The sequences also may not contain enough information to resolve the questions we want to ask. In case of the dishonest casino, if re-balancing the wheel is done very often, we may never be able to detect which parts of the sequence correspond to a loaded wheel. In fact, we only stand a chance if the casino is forced to keep the same wheel balance for long stretches of spins.

Probabilistic modeling helps us to attack these problems. To solve a particular problem, we require a *training set* T containing sequences properly annotated with their labels. Based on this training set, we design and train the parameters of a *generative probabilistic model*. We can imagine a generative model as a black box that upon request generates (based on some stochastic process) a sequence and its corresponding annotation. Thus the generative model defines a probability distribution over the space of all pairs of sequences of a given

length and their possible annotations.

For example, in case of the dishonest casino, we can use the following model. If the wheel is fair at the moment, colors are generated from the fair distribution with the label "fair". If the wheel is loaded, colors are generated from a specific biased distribution and the label will be "loaded". In the training process, we can learn from the training set not only the bias of the loaded wheel, but also how often the casino manager changes the wheel from fair to loaded and vice versa.

When we are presented with a new unlabeled sequence, we need to *decode the sequence*. In the process of decoding, we will choose labels for all observations in the sequence, based on the probability distribution defined by the generative model. There are many definitions of decoding, the advantages and disadvantages of which we will discuss in some length below, but the annotation that maximizes the joint probability of the sequence and its corresponding annotation, or the *most probable annotation*, is often chosen.

For example, in the dishonest casino example, we would get a sequence of spin colors for the evening, and we would assign either the label "fair" or "loaded" to each spin to maximize the probability of generating both the sequence and the annotation in the trained probabilistic model.

This method yields the best results if the generative model is a true and exact representation of the process underlying the observation. However, even in such a perfect case the results may not be perfect. Typically, though, the generative probabilistic model is merely a good approximation of the true process, developed for the purposes of a particular decoding algorithm. A generally accepted principle is that more *faithful* generative models give better performance in decoding. That is, the more the pairs of sequences and their corresponding annotations generated by the generative model resemble the real sequences, the better the model's performance will be in annotating new sequences. Even though this principle sounds reasonable, and we generally concentrate on methods of increasing the faithfulness of the generative models we use, we also demonstrate in Sections 3.4 and 4.1 that this principle is not necessarily applicable in all cases.

In the rest of this section, we introduce hidden Markov models (HMMs), a family of generative probabilistic models that are popular for annotation problems in areas such as natural language processing, speech recognition, and bioinformatics

1.1.1 Hidden Markov Models

A *hidden Markov model* (HMM) is a generative probabilistic model, consisting of *states* and *transitions*. A particular HMM defines a random process for generating strings of a prespecified length n, labeled with the sequence of states by which the symbols of the string were generated.

The process starts in a start state σ. In each step, a symbol is first randomly generated according to the *emission probabilities* of the current state, which we define shortly, and then a random transition is followed to another state according to the transition probabilities of the current state. The model finishes after n such steps. We call one instance of such generative process a *run*.

Each state u is a Markov chain of order o_u. Let $e_u(x_1, \ldots, x_{o_u}, x)$ be the emission probability of generating character x in state u, provided that the previous o_u symbols are $x_1, \ldots, x_{o_u}$.

In what follows, we will often use $e_u(s_1 \ldots s_i)$ or simply $e_u(s_i)$ instead of $e_u(s_{i-o_u}, \ldots, s_{i-1}, s_i)$ if we consider the emission probability in the context of a longer sequence $s = s_1 s_2 \ldots s_n$. Also, in special cases, such as the beginning of the sequence where it is impossible to use Markov chains of order o_u, we will automatically consider the Markov chains of appropriate lower orders, without explicitly mentioning it.

We now define an HMM and the associated generative process formally.

Definition 1. *A* hidden Markov model (HMM) *is a six-tuple $(V, \Sigma, o, e, a, \sigma)$, where V is a set of states, Σ is an alphabet, o is a function, where o_v specifies the order of each state $v \in V$, the function $e : V \to (\Sigma^* \times \Sigma \to [0, 1])$ defines emission probabilities, the function $a : V \times V \to [0, 1]$ defines transition probabilities, and $\sigma \in V$ is the start state.*

For all $u \in V$, $\sum_{v \in V} a_{u,v} = 1$, and for all $s \in \Sigma^*$ and $u \in V$, $\sum_{x \in \Sigma} e_u(s, x) = 1$. For all states u, the values of the emission function $e_u(s, x)$ must depend only on the last o_u characters of the string s, so that every state is an o_u-th order Markov chain.

Definition 2. *For a given HMM $(V, \Sigma, o, e, a, \sigma)$, a state path $\pi = \pi_1 \ldots \pi_n$ consisting of n states from V, and a string $s = s_1 \ldots s_n$ of symbols from Σ, we define the probability of generating the sequence s by the state path π as follows:*

$$\Pr(\pi, s) = e_{\pi_1}(s_1) \cdot \prod_{i=2}^{n} a_{\pi_{i-1}, \pi_i} \cdot e_{\pi_i}(s_i), \tag{1.1}$$

if $\pi_1 = \sigma$. Otherwise, $\Pr(\pi, s) = 0$.

An HMM can be depicted as a graph, where states constitute vertices and positive probability transitions constitute edges. An example of such a representation of an HMM is shown in Figure 1.1. The HMM generates sequences over the alphabet $\{a, c, g, t\}$. It has two states, A and B, state B being the start state. The parts of the sequence generated in state B are longer (expected length 100) and have composition bias towards the symbols a and t. The parts of the sequence generated in state A are shorter (expected length 10) with a bias towards g, c. The sequence composition biases are represented by the emission probabilities, while the likelihood of changing between the two states is represented by the transition probabilities.

1.1.2 Algorithms for Decoding Hidden Markov Models

To decode a given sequence s, we want to find the "best" annotation, or the sequence of the states, corresponding to s. This decoding problem can be formulated in a variety of ways, depending on which annotation is considered the "best".

The most commonly used decoding in HMMs is *finding the most probable state path.*

Definition 3 (Most probable state path). *The HMM defines for every state path π the probability that the sequence s was generated by this state path* $\Pr(\pi|s)$. The most probable

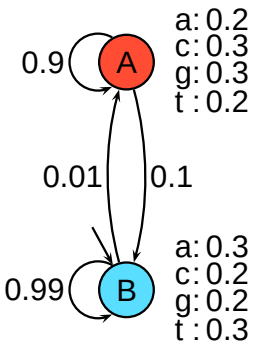

Figure 1.1: **Example of a hidden Markov model.** The HMM in the figure has two states, A and B, with different sets of emission probabilities. The state B is marked as a start state by an arrow.

state path *for sequence s is the state path π that maximizes* $\Pr(\pi|s)$*. Only state paths starting with the start state σ are considered.*

This thesis focuses mostly on decoding using the most probable state path. However, in this section we also mention other methods for decoding HMMs. Each of these methods have some desirable and some undesirable properties. Moreover, some of the algorithms, originally introduced in the context of decoding HMMs, will be useful later for computing the probabilities necessary for training and decoding of HMMs that are extended in various ways.

1.1.2.1 Computing the Most Probable State Path

The following algorithm computes the most probable state path π for a given sequence s in a given HMM.

Theorem 4 (Viterbi algorithm, Viterbi (1967); Forney (1973)). *For a given sequence s of length n and a hidden Markov model with m states, the most probable state path can be computed in $O(nm^2)$ time.*

Proof. First observe that for any path π, $\Pr(\pi|s) = \Pr(\pi, s)/\Pr(s)$. Therefore we can concentrate on computing the path π which maximizes $\Pr(\pi, s)$, since the denominator remains constant. For a given prefix of the sequence s and the state path π, we can compute[2]:

$$\Pr(\pi_{1\ldots i}, s_{1\ldots i}) = \Pr(\pi_{1\ldots i-1}, s_{1\ldots i-1}) \cdot a(\pi_{i-1}, \pi_i) \cdot e_{\pi_i}(s_i) \tag{1.2}$$

Let $P(i, v)$ be the probability of the most probable path through the model generating the string $s_{1\ldots i}$ and finishing in the state v. Based on equation 1.2, we have the following

[2]Note that here we do not consider the transition that logically belongs to the i-th step of the generative process

recurrence:

$$P(i, v) = e_v(s_i) \cdot \max_{u \in V} P(i-1, u) \cdot a(u, v) \tag{1.3}$$

We set the base cases $P(1, \sigma) = e_\sigma(s_1)$ for the start state σ, and $P(1, v) = 0$ for all the states that are not the start state. Or, if the start state is not specified uniquely, but instead a prior distribution on initial state is given, we can set the probabilities of $P(1, *)$ accordingly.

Note that the probability of the most probable path corresponds to $\max_{v \in V} \Pr(n, v)$. The values of $P(i, v)$ can be computed in order of increasing values of i by dynamic programming. Computing each value of $P(i, v)$ takes $O(m)$ time, where m is the number of states of the HMM. In addition to the values of $P(i, v)$, we will also remember which state u maximized the recurrence 1.3 when we computed $P(i, v)$. In this way, we can reconstruct the most probable state path by tracing back through the dynamic programming table.

Therefore, the most probable path can be computed in time $O(nm^2)$, where n is the length of the sequence s, and m is the number of states of the HMM. □

1.1.2.2 Posterior Decoding

While the definition of decoding using the most probable state path is very intuitive, it sometimes does not reflect our needs. *Posterior decoding*, instead of computing the globally optimal state path, concentrates on a single position in the sequence. For this position, we try to answer a different question: what is the most likely state that could have generated the symbol at this position?

This question cannot be simply answered by finding the most probable path using the Viterbi algorithm. Many different state paths in the HMM can generate the same sequence s, and for a particular position i many of them will agree on the same state. To compute the probability of state v at a given position, we need to add the probabilities of all paths that pass through state v at position i.

Definition 5 (Posterior state probability). *The* posterior probability $\Pr(\pi_i = v \mid s)$ *of state v at position i of sequence s is the sum of the probabilities of all paths passing through state v at position i:*

$$\Pr(\pi_i = v \mid s) = \sum_{\pi : \pi_i = v} \Pr(\pi \mid s) \tag{1.4}$$

Note that there exists a similarity to the case of finding the probability of the most probable path. Specifically, $\Pr(\pi_i = v \mid s)$ is proportional to $\Pr(\pi_i = v, s)$, and therefore when computing the highest posterior probability state, we can compute the joint probability instead of the conditional probability.

Definition 6 (Posterior decoding). *In* posterior decoding*, we compute the highest posterior probability state at each position i of the sequence s.*

Sometimes, it is beneficial to use posterior decoding instead of the most probable path decoding. Consider an example of three paths π, π', and π'', where π is the most probable

path, and π' and π'' are suboptimal paths with probability close to the probability of the most probable path. Suppose that at a given position i, $\pi_i = u$, while $\pi'_i = \pi''_i = v$. Then the best posterior annotation may differ from the annotation obtained using the most probable path at position i: the posterior annotation gives state v, while the most probable state path gives state u. State v in this case may actually be the better answer: if the parameters of the model were to change only a little, one of the paths π' and π'' might easily become the highest probability path.

On the other hand, if we look at the posterior annotation as a sequence of states, it is often a composition of unrelated high probability annotations. There is nothing in the definition to enforce dependencies between consecutive positions. If the posterior annotation has state u at position i and state v at position $i+1$, it is possible that the transition (u, v) has very low probability $t_{u,v}$ or even does not exist at all in the HMM. Thus, the posterior annotation π may have very small (or even zero) probability of having generated s. The posterior annotation is a good decoding method if we are interested in local properties of the annotation, but fails if we are interested in globally optimal solutions (such as annotations of structural elements of DNA or protein sequences).

To compute the posterior probability, $\Pr(\pi_i = v \mid s)$, of state v at position i of the sequence s, we observe that the probability can be decomposed as follows:

$$\Pr(\pi_i = v \mid s) = \sum_{w \in V} \frac{F_i(v, s) \cdot t_{v,w} \cdot B_{i+1}(w, s)}{\Pr(s)}, \tag{1.5}$$

where $F_i(v, s) = \Pr(\pi_i = v, s_1 \ldots s_i)$, the probability of generating first i symbols and ending in the state v, is called the *forward probability* of state v at position i, and $B_{i+1}(w, s) = \Pr(\pi_{i+1} = w, s_{i+1} \ldots s_n)$, the probability of starting in state w and generating sequence $s_{i+1} \ldots s_n$, is called the *backward probability* of state w at position $i+1$.

Theorem 7 (Forward algorithm, Baum and Eagon (1967)). *All forward probabilities for a given sequence s and a given hidden Markov model can be computed in $O(nm^2)$ time.*

Proof. Using equation 1.2, we can derive the following recurrence relation between values of $F_i(*, s)$ and $F_{i-1}(*, s)$.

$$F_i(v, s) = \sum_{w \in V} F_{i-1}(w, s) \cdot t_{w,v} \cdot e_v(s_i), \tag{1.6}$$

where we define $F_1(\sigma, s) = e_v(s_1)$ for the start state σ, and $F_1(v, s) = 0$ for all other states. Therefore the values of F can be computed by dynamic programming, progressing in order of increasing values of i, in $O(nm^2)$ time. □

The similar *backward algorithm*, progressing right-to-left instead of left-to-right computes all backward probability values in time $O(nm^2)$. Using Formula 1.5 and the results of the forward and backward algorithms, we can compute the posterior probabilities of all states at all positions of the sequence s in $O(nm^2)$ time.[3]

[3]Technically, the forward-backward algorithm computes only the joint probability $\Pr(\pi_i = v, s)$. However,

1.1.2.3 Combining Viterbi and Posterior Decoding

We can overcome some of the disadvantages of posterior decoding by adding a post-processing step. We first compute all posterior state probabilities, using the forward-backward algorithm. In the post-processing step, we are restricted to the paths that are *consistent with the HMM topology* (thus using only transitions present in the HMM topology are used), and we then find the path that maximizes either the sum (Kall et al., 2005) or the product (Fariselli et al., 2005) of the the posterior state probabilities. This is done by a dynamic programming algorithm similar to the Viterbi algorithm in time $O(nm^2)$.

These objective functions are based on interpreting the posterior probabilities as a positionally independent error measure. In the case of sums, we are trying to minimize the number of incorrectly predicted states, while in the case of products, we are trying to maximize the probability of a completely correct prediction.

1.1.3 Training Hidden Markov Models

The only step remaining in our scheme for designing an HMM for the annotation problem is estimating the parameters to create as faithful a model as possible. The most common approach to the problem can be defined formally as follows. We are given a training set T, and we want to find an HMM that maximizes the likelihood (probability) of generating the training set, subject to constraints on model topology. This process is usually split into two steps.

First, we choose the *model topology*: the directed graph consisting of the states of the HMM and the transitions between the states that have non-zero probability. In this step, we can apply in a very intuitive way any prior knowledge we may have about the particular problem. In fact, the most successful HMMs typically have a manually created topology.

Second, given the model topology, we estimate the emission and transition probabilities of the model to maximize the probability of generating the training set T.

These steps can be combined together if we choose a complete graph as the model topology. However, such a model often contains too many parameters to train accurately. For this reason, choosing the right topology is a very important step in the design of successful HMMs.

There are two potential scenarios under which we may need to train the HMM, depending on the composition of the training set T. If the training set T contains sequences together with their annotations (where the structure of the HMM allows an easy mapping of each sequence to its path through the HMM), a simpler scenario of *supervised training* can be applied. However, if there is no annotation, or only a partial annotation is available for the sequences in the training set T, we need to apply *unsupervised training* algorithms. Supervised training is the most common scenario connected to sequence annotation. However, we review both scenarios for completeness.

---

the posterior probability $\Pr(\pi_i = v|s) = \Pr(\pi_i = v, s)/\Pr(s)$, and $\Pr(s)$ can be easily obtained as a side product of the forward algorithm.

1.1.3.1 Supervised Training

This method is applied if we can determine the target path (sequence of states) through the model for each sequence in the training set. This is the case, for example, if the annotation is known for each sequence in the training set, and there is a one-to-one correspondence between such an annotation and the state paths in the HMM.

In this case, it is sufficient to count the frequency of using each transition and emission to estimate the model parameters that maximize the likelihood of the training data (Durbin et al., 1998, Chapter 11.3). In particular, we estimate the transition probability from state u to state v as:

$$a_{u,v} = \frac{A_{u,v}}{\sum_{v' \in V} A_{u,v'}}, \tag{1.7}$$

where $A_{u,v}$ is the number of transitions from state u to v in the training set. Similarly, the emission probabilities are estimated as:

$$e_u(y,x) = \frac{E_u(y,x)}{\sum_{x' \in \Sigma} E_u(y,x')}, \tag{1.8}$$

where $E_u(y,x)$ is the number of emissions of symbol x in state u in the training set, immediately preceded by emission of the sequence y. We need to determine $e_u(y,x)$ independently for all possible strings y that are o_u or fewer symbols long, because in special cases (such as at the beginning of the sequence) we may need to use a Markov chain of lower order than o_u.

The closed formula correspondence between simple statistics of the training set T and the parameters of the HMM is a very attractive property of the supervised training scenario. If one-to-one correspondence between the labels of the sequence and the states of the HMM cannot be established, we can use a variety of computational means (such as computational prediction tools and heuristics) to complete the annotation. Still, such treatment of the data is often preferable to the unsupervised training described below.

1.1.3.2 Unsupervised Training

If the sequences in the training set T are not annotated, we need to apply more complex methods for training. The task is, as in the supervised case, to find the parameters of the HMM with a given topology that maximize the likelihood of the training set. Some modifications of the problem have been shown to be NP-hard (Abe and Warmuth, 1992; Gillman and Sipser, 1994), while for special cases there exist polynomial-time algorithms (Freund and Ron, 1995).

There is, however, no general exact algorithm known for solving this unsupervised training problem efficiently. The method most commonly used, the Baum-Welch algorithm (Baum, 1972), is an iterative heuristic and can be considered a special case of the general

EM algorithm for learning maximum likelihood models from incomplete data sets (Dempster et al., 1977).

The Baum-Welch algorithm starts from an initial set of model parameters θ_0. In each iteration, it changes the parameters as follows:

1. Calculate the expected number of times each transition and emission is used to generate the training set T in an HMM whose parameters are θ_k.

2. Use the frequencies obtained in step 1 to re-estimate the parameters of the model, resulting in a new set of parameters, θ_{k+1}.

The first step of the algorithm can be viewed as creating a new *annotated* training set $T^{(k)}$, where for each *unannotated* sequence $s \in T$, we add every possible pair (s, π) of the sequence s and any state path, weighted by the probability $\Pr(\pi \mid s, \theta_k)$ of the path π in the model with parameters θ_k. The second step then estimates new parameters θ_{k+1}, as in the supervised scenario based on the new training set $T^{(k)}$. This algorithm takes exponential running time in each iteration, while the Baum-Welch algorithm achieves the same result in $O(nm^2)$ time per iteration, using the forward and backward algorithms. Details can be found, for example, in Durbin et al. (1998, Chapter 3.3).

Theorem 8 (Baum (1972)). *An iteration of the Baum-Welch algorithm never worsens the model parameters. More formally, for any k,*

$$\Pr(T \mid \theta_{k+1}) \geq \Pr(T \mid \theta_k) \tag{1.9}$$

The above theorem does not guarantee that the Baum-Welch algorithm reaches optimal model parameters. In fact, this is often not true: the Baum-Welch algorithm may reach a local maximum or a saddle point in the parameter space (Dempster et al., 1977). Moreover, it is not clear how one would estimate the number of iterations needed to reach such a local maximum. In practice, the Baum-Welch algorithm performs well in some cases and very badly in other cases (Freund and Ron, 1995), with its performance often depending on the choice of initial parameters θ_0.

A modification of the Baum-Welch algorithm, called *Viterbi training*, is often also used in practice. In the first step of the algorithm, instead of considering all possible paths through the model, we only consider the most probable path. However, there is no clear optimization formulation behind this algorithm, and it is not even guaranteed to improve the parameters in each step (Durbin et al., 1998, Chapter 3.3).

The Baum-Welch algorithm can also be used in the semi-supervised scenario. For example, the annotation for some sequences in T may be missing, or we may decide to use a complex HMM that has richer structure than the available sequence annotation, and thus it is not possible to assign a single path through the model to each annotation. In such case, we modify step 1 of the algorithm again to include only paths that agree with such partial annotations.

1.1.3.3 Beyond Maximum Likelihood

In the previous section, we introduced algorithms for training HMMs by maximizing the likelihood of the training set. If we denote $T = (s, a)$, where s are the sequences, and a are their annotations, then the maximum likelihood method (ML) learns the model parameters that maximize the joint probability $\Pr(s, a)$.

A common criticism of the ML approach in the machine learning literature is that it maximizes the wrong objective (see for example Krogh (1997)). Our goal in the decoding algorithm is to retrieve the annotation a that maximizes $\Pr(a|s)$, sequence s being fixed. Therefore, instead of maximizing the joint probability during the training, we should concentrate on maximizing conditional probability $\Pr(a|s)$, since the probability of the sequence itself is fixed in the decoding phase, and therefore it does not matter whether this probability is low or high. This optimization criterion is known as *conditional maximum likelihood* (CML).

In context of hidden Markov models, CML was used in applications in bioinformatics (Krogh, 1997) and natural language processing (Klein and Manning, 2002). Even if the sequences in both of these applications are annotated, there is no known closed formula or EM algorithm that would estimate the parameters of the model to optimize the conditional maximum likelihood. Instead, numerical gradient descent methods are used to achieve local maximum. In these studies, slight (Klein and Manning, 2002) to significant (Krogh, 1997) improvement was observed compared to models trained by ML.

Theoretical analysis is available in context of the simpler classification problem, where similar dichotomy occurs between naive Bayes classifier (ML) and logistic regression (CML). In this context, Ng and Jordan (2002) have shown that even though using CML gives asymptotically lower error, ML requires significantly smaller number of training samples to converge to the best model: it requires only a logarithmic number of samples with respect to the number of parameters, compared to the linear number of samples for CML. Thus ML is appropriate if only a small number of samples is available, while it is better to use CML if the training set is large. It is not known whether these results extend to the case of more complex models, such as HMMs. Also, an interesting question is: if the amount of available data increases, is it better to switch from ML to CML or to increase the number of the model parameters? A larger family of models may potentially allow the choice of a model that is closer to reality, decreasing both training and testing error.

One major disadvantage of HMMs optimized for CML is that it is hard to interpret the resulting emission and transition probabilities. The generative process associated with the HMM no longer generates sequences that look like sequences from the training set. The probabilities no longer represent frequencies observed directly in the sequence, which makes it hard to incorporate prior knowledge about the problem into the probabilistic model by applying restrictions on parameters of the model, or by creating a custom model topology.

Consider, for example, the HMM in Figure 1.2. Such a model can be used to model amino acid sequences of transmembrane proteins. Transmembrane proteins are composed of parts located inside of the cell (represented by state A), outside of the cell (represented by state C), and the parts crossing the cell membrane (represented by states B and D). It

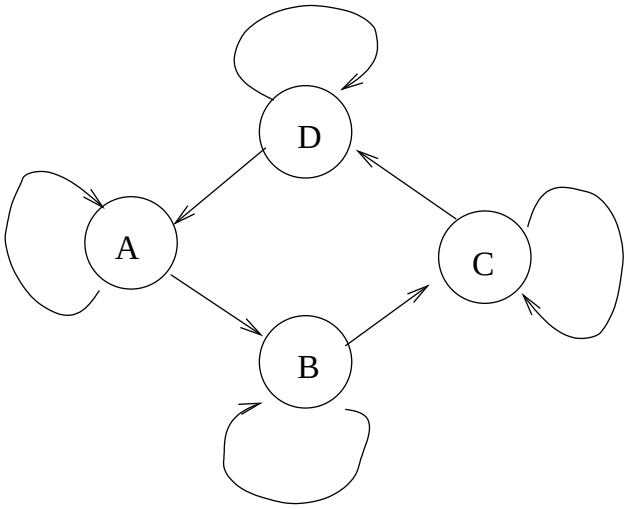

Figure 1.2: **A simple HMM topology for transmembrane protein topology prediction.** State A represents the parts of the protein located inside the cell, state D represents parts located outside the cell, and states B and D represent the parts crossing the cell membrane. In an ML model, states B and D would share the same set of emission probabilities.

may be reasonable to assume that since the sequences corresponding to states B and D are localized in the same part of the cell, and serve the same function (membrane transition), that in an ML model, both states should share the same emission probabilities. Based on this assumption, we can reduce the number of parameters (and thus the number of sequences required for training) by *tying* those parameters together. On the other hand, since in CML method the emission probabilities are set to maximize the conditional probability of the annotation given the sequence, rather than likelihood of the sequence, it is not clear that the emission probabilities in states B and D should be similar, even if the sequences attributed to these states are similar.

For another example, consider the smoke detector. A model of its readings would typically have two states, one representing normal readings, the other representing readings in case of fire. Potentially, we may also consider including a state representing readings at times when dinner is cooked (which are perhaps consistently higher than normal readings, for short periods of time). Adding such a new state into our model, representing dinner-time readings, could help to increase the likelihood in case of ML training. However, it is not clear that the same improvement would happen in CML, since raising the likelihood of the sequences is no longer the objective.

The above examples illustrate that in case of CML training, it is not clear whether common methods for incorporating background knowledge into HMM design (such as extending model topology or parameter tying) are still relevant or beneficial.[4]

[4]Conditional random fields (Lafferty et al., 2001) further continue in the direction of CML training, abandoning the probabilistic interpretation of emission and transition probabilities, replacing them with

Recent extensions abolish the probabilistic interpretation of HMMs altogether. Instead, they consider the following problem directly: set the parameters of the model (without normalization restrictions) so that the model discriminates well between correct and incorrect annotations. Such models (hidden Markov support vector machines (Altun et al., 2003), convex hidden Markov models (Xu et al., 2005)) are inspired by maximum margin training and kernel methods in support vector machines (Boser et al., 1992), a successful method for the classification problem.

1.2 Introduction to Gene Finding

A significant amount of resources is spent every year on DNA sequencing projects. The year 2001 saw the publication of the human genome project from both public (International Human Genome Sequencing Consortium, 2001) and private (Venter et al., 2001) initiatives. This was closely followed by the mouse genome (Mouse Genome Sequencing Consortium, 2002), and recently the rat (Rat Genome Sequencing Project Consortium, 2004), the chicken (International Chicken Genome Sequencing Consortium, 2004), and the chimpanzee genomes (Chimpanzee Sequencing and Analysis Consortium, 2005). According to the Genomes Online Database (Bernal et al., 2001), as of July 2005 there are close to 500 genome projects of *eukaryotic organisms*[5] in progress worldwide.

The main output of these genome projects are DNA sequences of a wide variety of organisms. DNA sequences can be viewed as long strings of *nucleotides* or *bases*—symbols from the alphabet $\{A, C, G, T\}$, encoding the biological information defining the organism. The length of the DNA sequence is specified by the number of bases, often referring to 10^3 bases as a *kilobase* (KB), 10^6 bases as a *megabase* (MB), and to 10^9 bases as a *gigabases* (GB).

The DNA sequence is a template for *protein sythesis*. Proteins are macromolecules that are essential to practically every function of a living cell. Thus, DNA sequences by themselves give us the information that is contained in living cells, but further understanding and analysis of this information is required in order to understand how living cells function.

The basics of the process by which proteins are synthesized in eukaryotic organisms are moderately well understood, and are described by the *central dogma of molecular biology*, illustrated in Figure 1.3. For each protein, there is a section of the DNA sequence called its *gene* that serves as a template for the synthesis of that protein. The gene is located on the DNA by the cell machinery and is transcribed into *primary mRNA*. RNA is a molecule very similar to DNA in structure, and for simplicity, we will envision the process of transcription as creating a simple copy of a subinterval of the DNA sequence.

In the mRNA maturation process, non-coding parts of the gene (*introns*) are removed by splicing. The remaining parts, called *exons*, are joined together in the original order as they appeared in the DNA sequence. The resulting sequence is then translated into a protein.

undirected potentials that do not need to be normalized to 1.

[5] *Eukaryotic organisms* are organisms with complex cells, where genetic material is organized in the nucleus and surrounded by a nuclear membrane. They include animals, plants, and fungi, but do not include simpler *prokaryotes*, such as bacteria.

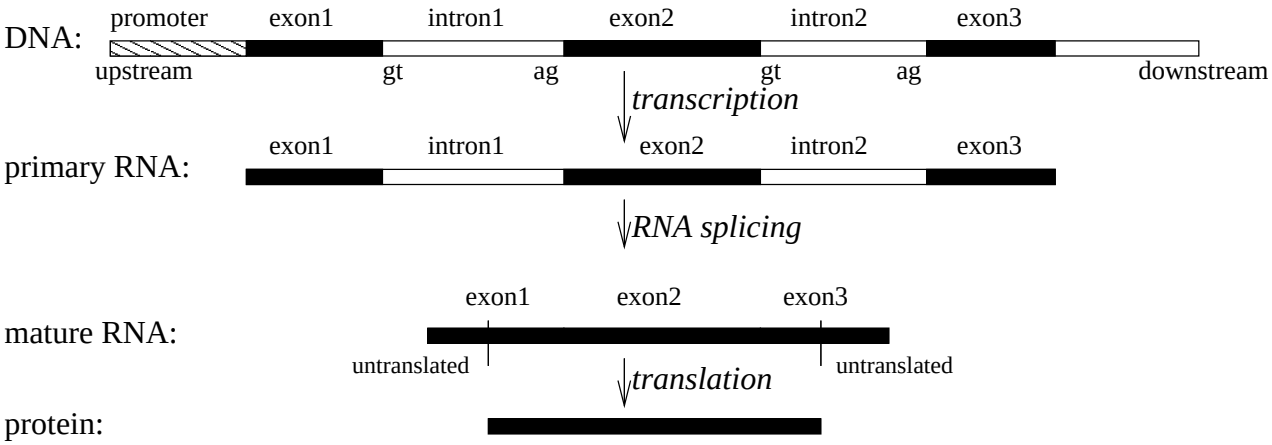

Figure 1.3: Central dogma of molecular biology

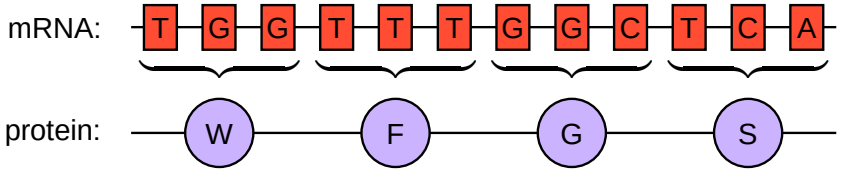

Figure 1.4: Translating nucleotide sequences to protein sequences

Proteins can also be seen as strings of simpler units, *amino acids*. We denote each amino acid by a symbol from a 20-letter alphabet. The nucleotides in the mRNA sequence are translated into amino acids of the protein sequence according to the genetic code (see Table 1.1). Except for short untranslated regions at the beginning and end of the gene, every triplet of nucleotides (called a *codon*) encodes a single amino acid (as illustrated in Figure 1.4). Consecutive codons of the mRNA encode consecutive amino acids of the protein.

It is important to know where the translated sequence begins. If we shift the codon locations by one or two positions, these will result in a completely different protein. We refer to the first nucleotide of a codon as being *in frame 0*, the second as being *in frame 1*, and the third as being *in frame 2*. If the first nucleotide of a triplet is in frame 0, we call the triplet an *in frame codon*. Every translated sequence starts with the codon ATG, encoding the amino acid methionine (M), and translation always stops upon encountering one of the stop codons, TAA, TGA, or TAG.

Figure 1.3 depicts a single gene of the genome. The genome itself is composed of several chromosomes. Many genes are located on each chromosome, corresponding to oriented subintervals of the chromosome. The subintervals need to be oriented, since the genes are sometimes on the *forward strand* (in direction from left to right), and sometimes on the *reverse strand* (in direction from right to left).

Without knowing where genes, exons, and introns are located in the DNA sequence, we

TTT → F	TTC → F	TTA → L	TTG → L
TCT → S	TCC → S	TCA → S	TCG → S
TAT → Y	TAC → Y	TAA → *	TAG → *
TGT → C	TGC → C	TGA → *	TGG → W
CTT → L	CTC → L	CTA → L	CTG → L
CCT → P	CCC → P	CCA → P	CCG → P
CAT → H	CAC → H	CAA → Q	CAG → Q
CGT → R	CGC → R	CGA → R	CGG → R
ATT → I	ATC → I	ATA → I	ATG → M
ACT → T	ACC → T	ACA → T	ACG → T
AAT → N	AAC → N	AAA → K	AAG → K
AGT → S	AGC → S	AGA → R	AGG → R
GTT → V	GTC → V	GTA → V	GTG → V
GCT → A	GCC → A	GCA → A	GCG → A
GAT → D	GAC → D	GAA → E	GAG → E
GGT → G	GGC → G	GGA → G	GGG → G

Table 1.1: Standard genetic code

do not know the proteins that are produced by the organism. Only partial information about location of genes and about the proteins directly can be acquired by experimental means. Locating these elements in a DNA sequence by computational means is called *gene finding*. Gene finding is a crucial step in the analysis of a genome. More formally, the problem can be defined as follows.

Definition 9 (Gene finding problem). *For a given DNA sequence (a string of symbols over the four letter alphabet $\{A, C, G, T\}$), assign to each symbol in the sequence one of the following labels:* intergenic region *(non-coding sequence located between genes),* exon on forward strand *or* exon on reverse strand *(coding sequence),* intron on forward strand *or* intron on reverse strand *(non-coding sequence located within genes that is removed during splicing).*[6]

Gene finding methods that only use the information contained in the analyzed DNA sequence itself are called *ab initio* methods. This thesis specifically focuses on *ab initio* gene finding. There are methods to incorporate additional information, not present in the DNA sequence, into gene finding. Such sources of information include similarities to the DNA sequence of other genomes, experimental evidence, or similarity to known proteins and expressed RNA sequences. We will briefly discuss these approaches in Section 1.2.3.

The gene finding problem presents many challenges. First, the problem is not defined as an exact optimization problem. As in many problems from computational biology, the prob-

[6]Some parts of the mature mRNA at the beginning and at the end of the sequence are not translated. These are called *untranslated regions* (or UTRs). For the purpose of this thesis, we will count these regions as intergenic regions.

lem can be formulated as several different optimization problems, each yielding potentially different solution. Thus the ultimate goal is not only to design efficient algorithms that solve a particular optimization problem, but also find an optimization formulation of the problem that corresponds well to underlying biological principles and at the same time can be solved reasonably efficiently.

Genes themselves are not mere intervals in the DNA sequence, but they also have complex internal intron-exon structure. Since our goal in gene finding is to predict the proteins produced by a particular organism, it is important that the whole structure of the gene is predicted accurately. Very small mistakes can result in practically useless predictions. For example, if boundary of an exon is shifted by a single position, the codon boundaries will shift, resulting in a completely different predicted amino acid sequence.

Another complication is that in spite of their importance, protein coding genes constitute only a small portion of many eukaryotic genomes. From the entire human genome length of approx. 3.7 GB, the estimated 20 000 to 25 000 genes cover only about 2% of the genome (International Human Genome Sequencing Consortium, 2004). This means that many machine learning techniques, which often assume that data sets contain approximately the same number of positive and negative samples, will over-predict genes in the human genome.

DNA sequences are generally long. Ideally, we would like gene finding algorithms to process whole chromosomes. In the human genome, chromosomes vary in length from 46 MB (chromosome 21) to 245 MB (chromosome 1). Even if we could separate the sequences corresponding to single genes, such sequences still can span hundreds of kilobases. With such a long sequences, it is important that the algorithms used for gene finding are efficient (ideally, linear) in both time and space.

Data sets of known genes are all based on indirect information. For some genes, evidence of their existence and their structure is solid, while for other genes the evidence is only circumstantial. It is generally impossible to verify experimentally that a given structure is *not* translated into a protein. This causes numerous problems in assembling adequate training sets and in the evaluation of gene finding methods.

Finally, in Figure 1.3, and in the problem definition, we assumed that a particular interval of the DNA sequence translates only into a single protein. This is often not the case. Different genes can overlap, and some genes can yield several proteins by *alternative splicing*: including or excluding some of the exons, shifting exon boundaries, etc. In this thesis, we ignore the problem of alternative splicing and overlapping genes—the problem remains open in the context of *ab initio* gene finding and is yet to be tackled by the gene finding community.

In the rest of this section, we give an introduction into common aspects of gene finding. First, we give a more detailed overview of the statistical properties of DNA sequences that are commonly used in gene finding. Next, we summarize the techniques that have been previously used for gene finding. We mention common measures for evaluating the accuracy of gene finders, and touch on the availability of experimental evidence for protein coding genes. Finally, in Section 1.3, we describe the use of hidden Markov models for gene finding in more detail.

1.2.1 Statistical Properties of Genes in DNA Sequences

There are two main sources of information upon which one relies when designing an algorithm for gene finding. First, the coding parts of the sequence (or exons) have distinctly different composition than the non-coding parts (so called *k-mer composition*). Second, the sequence is conserved around certain functional sites, such as the boundaries of exons. However, neither the difference in composition nor detection of signals is strong enough to allow reliable gene finding by itself (Lim and Burge, 2001). To design a successful gene finder, these sources of information must be combined.

In the following text, the *5' end* denotes the start of the gene, while *3' end* denotes the end of the gene. If the DNA sequence is displayed from left (its 5' end) to right (its 3' end), the genes on the forward strand have their 5' end on the left side, while the genes on the reverse strand have their 5' end on the right.

Consider a sequence window relative to the boundary of two regions, such as boundaries between the end of an exon and the beginning of an intron (called *donor splice sites*) or boundaries between the end of an intron and beginning of an exon (called *acceptor splice sites*). See Figure 1.5 for the illustration. We will often refer to such windows as intervals $[x, y]$. If x is a negative number, it means a relative position on the 5' side of the boundary, while if it is a positive number, it means a relative position on the 3' side. No position is marked as zero. For example, the interval $[-6, +3]$ relative to the donor site represents a window of nine nucleotides, six of them within the exon located on the 5' side of the donor site, and three of them within the intron located on the 3' side of the donor site.

1.2.1.1 Differences in k-mer Composition

Table 1.2 shows a comparison of the 3-mer composition in sequence elements that play a role in gene finding. We used annotated and masked sequence of half of human chromosome 22 (details of the data set can be found in Appendix A) and counted the frequencies of overlapping 3-mers occurring in each of these sequence elements: introns, exons, and intergenic regions. These frequencies were then compared using Pearson's correlation coefficient.

We can see that the 3-mer composition of exons does not correlate well with intronic or intergenic sequences, but that there is a high correlation between introns and intergenic regions. That suggests that coding sequences have significantly different composition from non-coding sequences. This is because coding sequences carry protein content information, while non-coding sequences are often considered to be mostly random sequences.

Comparing introns on the forward strand and introns on the reverse strand, one would expect the correlation to be close to one. However, the table shows that introns contain some information about their direction. It is also interesting that the average between the two directional intron distributions show high correlation with intergenic regions. This fact can be used when training a gene finder from annotated single-gene sequences, since in such a case, only a limited amount of intergenic sequence is available.

Interesting observations can also be made for untranslated regions, the regions that are transcribed, but not translated into proteins. Such a region on the 5' end of the gene is called

	exon	intergenic	intron (forward)	intron (average)	intron (reverse)	5'UTR	3'UTR
exon	1.00	0.57	0.42	0.46	0.49	0.84	0.92
intergenic	0.57	1.00	0.97	0.99	0.97	0.33	0.71
intron fwd.	0.42	0.97	1.00	0.97	0.90	0.24	0.57
intron avg.	0.46	0.99	0.97	1.00	0.97	0.22	0.61
intron rev.	0.49	0.97	0.90	0.97	1.00	0.19	0.61
5'UTR	0.84	0.33	0.24	0.22	0.19	1.00	0.85
3'UTR	0.92	0.71	0.57	0.61	0.61	0.85	1.00

Table 1.2: Correlation of 3-mer composition of sequence elements in gene finding

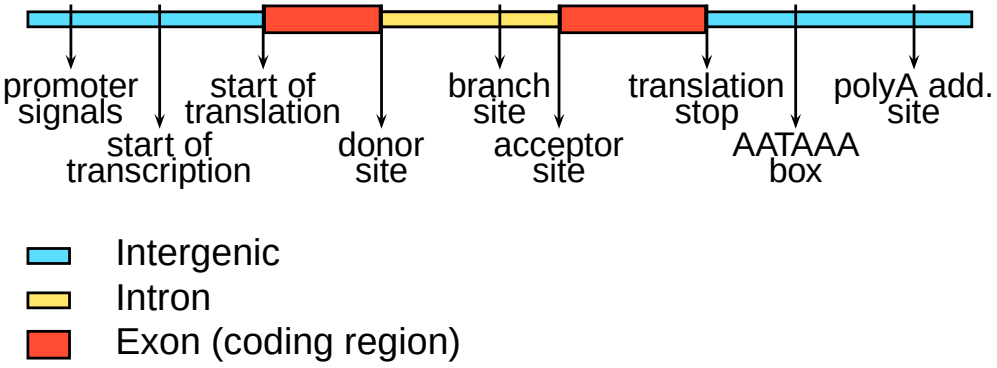

Figure 1.5: Summary of biological signals important for gene finding

a 5' UTR, and on the 3' end of the gene it is called a 3' UTR. UTRs are not annotated in our data set, so we made our observations in the region $[-75, -10]$ before the start of the coding sequence for 5' UTRs, and in the region $[+10, +75]$ after the end of the coding sequence for 3' UTRs. The composition of these regions seems to be much more similar to the coding exons than to the introns or intergenic regions (showing especially low similarity in case of 5' UTRs). Even though the differences seem to be even more pronounced than between exons and introns, we were not able to successfully exploit this observation in our gene finder.

1.2.1.2 Conserved Signal Sequences

Figure 1.5 summarizes the most common biological signals important for gene finding. Biological signals occur at the boundaries of sequence elements, as well as within them. Signal sequences are conserved because they are used by cell transcription, splicing, and translation machinery.

For example, three main functional elements are present inside introns: branch point, donor site (or 5' splice site) and acceptor site (or 3' splice site). During splicing, the donor

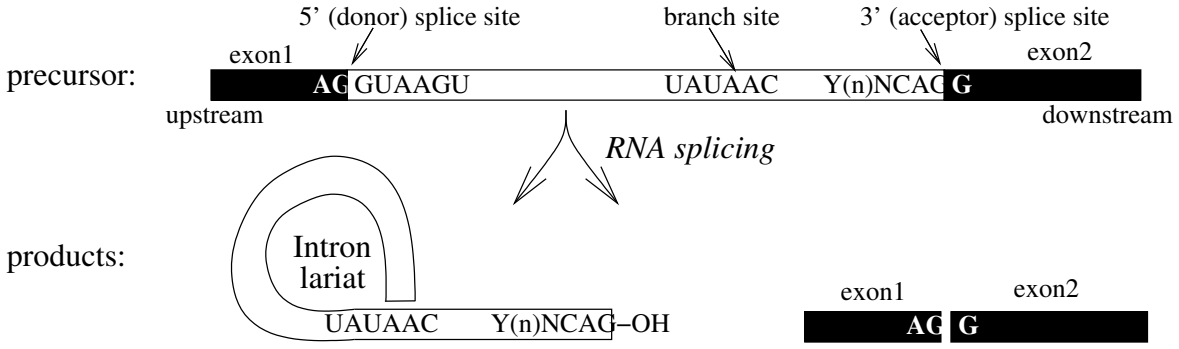

Figure 1.6: **Splicing mechanism.** Adapted from Gilbert (1997).

site separates from the neighboring exon and attaches to the branch site, located in the intron sequence near the acceptor site; the cut end of the introns sequence thereby becomes covalently linked to the branch site A nucleotide as shown in Figure 1.6. After this the two exons separated by this intron are joined together.

From our point of view, signals are short windows of the sequence around a particular functional site. Within this window, we will observe an unusually conserved distribution of nucleotide bases at each of the positions. Common properties of the signals are often displayed by *sequence logos* (Schneider and Stephens, 1990). For each position in the window corresponding to the signal, the sequence logo displays a stack of nucleotides. The nucleotides are ordered from top to bottom by their relative frequency, and their sizes correspond to the relative frequency of characters. The height of each stack represents the "importance" of each of the position. Positions for which the distribution of nucleotides is uniform do not help identifying signals, while positions where the conservation is perfect help the most. This can be measured by *number of bits* of information, $2 - H(x)$, where $H(x)$ is the Shannon entropy of the distribution of nucleotides at each position (Shannon, 1948). The logos in this section were created by the program WebLogo (Crooks et al., 2004), using the sequence annotation of half of human chromosome 22.

Figure 1.7 shows the logo of the donor splice site signal for the window $[-3, +6]$ (three nucleotides before the exon/intron boundary and six nucleotides after the boundary). The strongest signal is supplied by the consensus GT at the exon/intron boundary, with strong conservation at positions -2, -1, $+3$, $+4$, and $+5$. Similarly, acceptor sites (Figure 1.8) show the consensus sequence AG at the intron/exon boundary, but conservation is much weaker at the other positions within the window, with significant conservation observed only at position -3.[7] The acceptor signal is therefore very weak. It can be supplemented by the observation that introns tend to contain a pyrimidine (CT) rich tail towards the 3' end, as shown in Figure 1.9.

[7]We only considered so called *canonical splice sites* displaying the consensus GT/AG at intron/exon boundaries. Splice sites that do not follow this consensus exist, but they are uncommon (less than 2% of all splice sites).

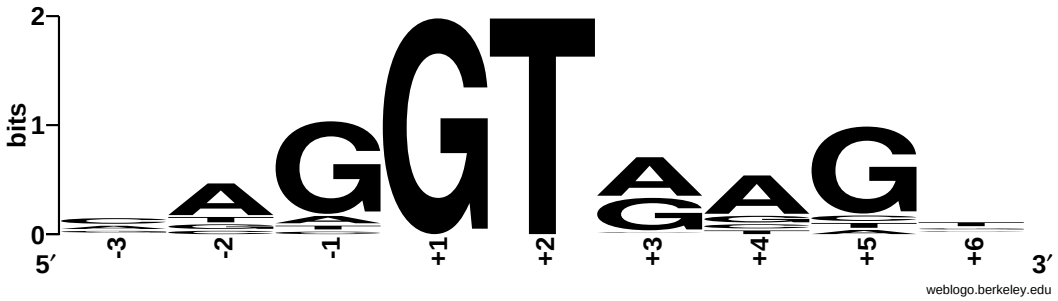

Figure 1.7: Logo of 5' (donor) splice site

Figure 1.8: Logo of 3' (acceptor) splice site

Finally, we also show logos for the start site (window $[-9, +4]$, Figure 1.10), and for the stop site (window $[-6, +9]$, Figure 1.11). Note that a coding sequence always starts with the start codon ATG. The logo of the stop site does not show all the information about the signal, in that the coding sequence always ends with one of the three possible stop codons: TAA, TGA, or TAG. The three stop codons will never occur inside the coding sequence in the coding frame.

The other signals mentioned in this section, the promoter, start of transcription, branch site, AATAAA box, and polyA site, are used in gene finding only infrequently. Since they are not associated with the boundary of an intron or exon, it is complicated to incorporate them into a gene finder. Another problem with their use is that they are not usually annotated, so it is hard to obtain a significant sample for training probabilistic models of these signals, especially for newly sequenced organisms.

1.2.2 Previous Work: Programs for *Ab Initio* Gene Finding

In general, programs for computational gene prediction from DNA sequences can be divided into two large groups (Stormo, 2000; Solovyev, 2002): dynamic programming, and probabilistic modeling. In this section we give a short overview of these two approaches.

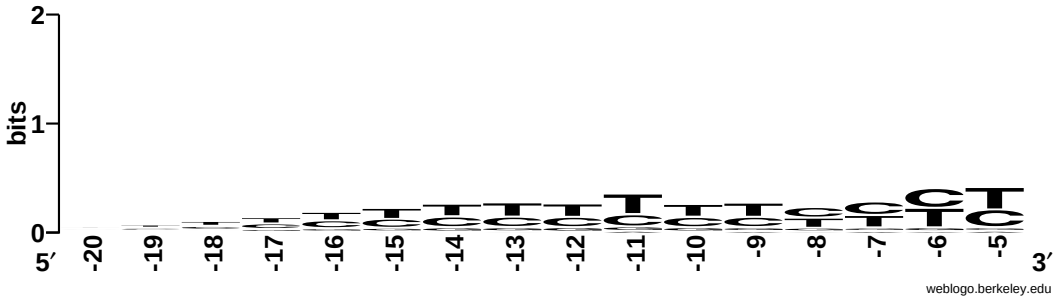

Figure 1.9: Logo of region $[-20, -5]$ before the acceptor splice site

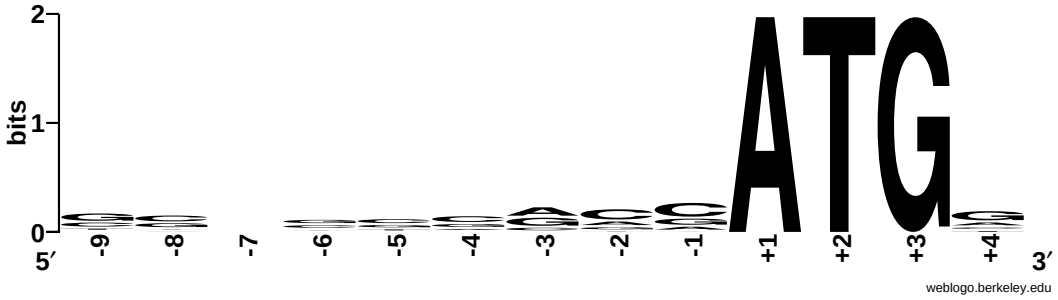

Figure 1.10: Logo of translation start signal

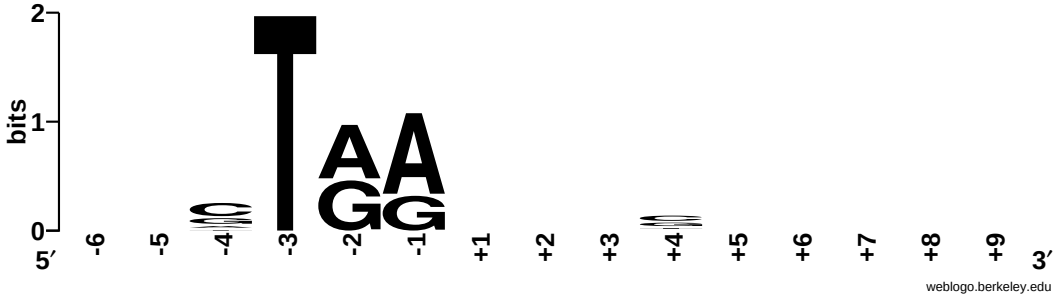

Figure 1.11: Logo of translation stop signal

1.2.2.1 Dynamic Programming

Dynamic programming methods are variations of the following general approach. First, they create a set of candidate exons. These candidates are scored based on their composition and surrounding signals. Then, from all biologically meaningful combinations, the best scoring combination is selected using dynamic programming.

In general, the number of potential exons may be quadratic in the length of the sequence. This may prevent dynamic programming algorithms from running in reasonable time. However, Burge (1997) has shown experimentally that if we apply reasonable filtering conditions, such as enforcing the presence of GT/AG consensus strings at the boundaries of candidate exons, and the rule that there are no in-frame stop codons, in real sequences the number of candidate exons grows roughly linearly with the length of the sequence.

Examples of such systems are GRAIL (Xu et al., 1994), FGENEH (Solovyev et al., 1995), geneid (Guigó, 1998), and recently DAGGER (Chuang and Roth, 2001). A major problem with these approaches is their use of *ad hoc* scoring schemes with many components whose interaction is not clearly defined. On the other hand, it is relatively easy to extend these methods when new sources of information become available.

1.2.2.2 Probabilistic Modeling

Probabilistic modeling methods formulate the problem of gene finding as the sequence annotation problem, as described earlier in this chapter. The commonly used probabilistic models are hidden Markov models, with a variety of topology restrictions to encode background knowledge about structure of the genes. We train these probabilistic models on a training set, and to find genes in the new sequences, we use standard decoding algorithms, such as the Viterbi algorithm. These models mostly differ in the topology of their underlying probabilistic models, and sometimes in their training and decoding methods.

The best known gene finder based on hidden Markov models is Genscan (Burge, 1997; Burge and Karlin, 1997). The Genscan model includes the basic intron/exon structure of genes (including sophisticated models of splice sites), simple single state models of 5' and 3' UTRs, and a simple model for intergenic regions. Genscan model also includes explicit length distribution models for exons, using generalized HMMs. Genes on both strands are modeled within the same model. The model is trained using maximum likelihood, and the decoding algorithm is the Viterbi algorithm.

The recently developed gene finding program Augustus by Stanke and Waack (2003) uses a model very similar to that of Genscan. Augustus, compared to Genscan, does not model UTRs, since these are not well annotated in available training sets. The authors also change the model used for introns to model length distributions more accurately, and they alter some of the signal models. Augustus and Genscan are the closest models in literature to our newly developed gene finder, ExonHunter. We describe differences and similarities of these three programs in more detail in Chapter 5.

Fgenesh (Salamov and Solovyev, 2000) uses a model very similar to Genscan. However, the influence of signals on prediction is artificially boosted, which is claimed to improve the

performance. The details are not documented in the authors' work.

HMM-gene by Krogh (1997) uses conditional maximum likelihood training, instead of maximum likelihood training. Krogh also devises a simple heuristic method for decoding to address some of the problems that we explore in Section 4. We cannot compare the model itself to that of Genscan or Augustus, since it is not documented in the literature.

Genie (Kulp et al., 1996; Reese et al., 2000) models exons, introns, and signals each by a single state in an HMM. However, each state can emit an arbitrarily long section of sequence. This is an instance of generalized HMMs, which are presented in Section 3.1. The states themselves use a variety of other models (for example, neural networks are used for the signals). The decoding algorithm first builds a graph representing all possible parses of the sequences, and this graph is then searched for the most probable annotation using the Viterbi algorithm.

GeneMark.hmm (Besemer and Borodovsky, 2005) is a hidden Markov model based gene finder, originally designed for the simpler problem of gene finding in prokaryotes, and later extended to eukaryotic genomes. Genezilla (formerly known as TIGRscan (Majoros et al., 2004)) is an open source software project of an HMM-based gene finder that is focusing on overcoming various software engineering challenges in building a high-performance gene finder. The authors modified traditionally used decoding algorithms to speed up sequence analysis in practice.

1.2.3 Beyond *Ab Initio* Gene Finding

In this section we briefly mention some of the programs that use other sources of information, in addition to the information contained in the DNA sequence, to improve gene prediction.

The DNA sequences of genomes are related by the process of evolution. Genomic sequence undergoes gradual changes over long periods of time, and natural selection favours the changes that are either advantageous or neutral. Naturally, the parts of the sequence that do not have any function often mutate more rapidly than the functional parts, since a random mutation in the functional gene may have large (often undesired) effects.

A special class of gene finders, *comparative gene finders*, use similarities of related genomes to their advantage. Because coding regions should be more conserved, comparison of human DNA sequence to that of mouse, chicken, or chimpanzee reveals higher levels of conservation in coding regions compared to non-coding regions. Examples of comparative gene finders include Twinscan (Korf et al., 2001), ExoniPhy (Siepel and Haussler, 2004), and N-SCAN (Gross and Brent, 2005). These gene finders extend regular HMMs for gene finding by introducing more complex alphabets that incorporate the alignment information between analyzed DNA sequence and sequences from other genomes. The emission probabilities for these alphabets take into account the evolutionary relationship of the genomes. Some comparative gene finders introduce innovations that might be useful in *ab initio* gene finding. For example, the authors of N-SCAN introduced perhaps the first realistic model of 5' UTR in their model (Brown et al., 2005), but the effect of this modification on *ab initio* gene finding was not evaluated separately.

Other sequence elements related to gene finding were explored separately from general gene finding problem. Ohler et al. (2001) developed a hidden Markov model for recognition of promoters and transcription start sites as a separate program McPromoter. Hajarnavis et al. (2004) developed a probabilistic model of the polyA signal and of the composition of 3' UTRs. Incorporation of such additions to an *ab initio* gene finder might improve accuracy, especially on long sequences.

Complex systems for introducing various sources of additional information into HMM based gene finding have been introduced (see, for example, GenomeScan (Yeh et al., 2001), Augustus++ (Stanke et al., 2005), and our own extension of ExonHunter (Brejová et al., 2005)). However, the architecture and evaluation of these systems are beyond the scope of this thesis.

1.2.4 Evaluation Measures

Because we are interested in studying how genes are translated into proteins, we need measures that evaluate complex gene structures, rather than just simple labels at particular positions. In this section we describe the accuracy measures that are commonly used for gene finding.

For simplicity, let us first assume that both the prediction and the correct annotation consists of non-overlapping genes without alternative splicing. In such case, we may consider three levels of accuracy: *nucleotide accuracy*, measuring how well we predict which nucleotides are coding, *exon accuracy*, measuring how well we predict exons, including their exact boundaries, and *gene accuracy*, measuring how many genes we predict completely correctly, starting with the correct start codon, including all exons and correct splice sites, and ending with the correct stop codon.

At all of these levels, we compute two measures: *sensitivity* and *specificity*. If we classify each object (coding nucleotide, exon, or gene structure) as either true positive (TP), true negative (TN), false positive (FP), or false negative (FN) as shown in Table 1.3, we can define the *sensitivity*, measuring the percentage of correct objects found:

$$\mathrm{Sn} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{1.10}$$

and *specificity*, measuring percentage of correct objects in all predicted objects:

$$\mathrm{Sp} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{1.11}$$

Note that in the case of nucleotide statistics, it is easy to achieve high sensitivity and low specificity (predict all nucleotides as coding). Achieving high sensitivity is not trivial in the case of exons and genes, especially if we do not allow predictions with overlapping genes. If sensitivity is high, but specificity is low, it means that the algorithm is over predicting the number of objects. Similarly, if the sensitivity is low, but specificity is high, the algorithm is too conservative. If the sensitivity and specifity are the same, then the number of objects predicted is on the right scale, as the number of true negatives and false positives are equal.

	correct objects	incorrect objects
predicted objects	true positives TP	false positives FP
not predicted objects	false negatives FN	true negatives TN

Table 1.3: Classification of objects in union of predicted and correct objects

In many scenarios (such as the classification problem), it is possible to trade sensitivity for specificity and vice versa. However, this is not easy in gene finding (especially if we formulate it as the sequence annotation problem for hidden Markov models).

Supplementary statistics are sometimes observed for other types of objects, such as introns, in the same way as for exons. Intron sensitivity and specificity combines the accuracy of prediction of splice sites with the ability to determine proper sequence of exons in genes.

We will often compare different programs or variants of the same algorithm by their sensitivity and specificity, in most cases at the exon level. If the approach A has both sensitivity and specificity higher than the other approach B, we will generally say that the results of A are better than results of B. On the other hand, if one approach has higher sensitivity, and the other approach has higher specificity, we will say that A and B are not comparable.

Such comparison is not completely fair, since these measures are taken on a particular data set. Especially when the number of samples in such a data set is small, the differences in performance between A and B may be result of a chance, rather than result of superiority of approach A over approach B.

In such cases, we would like to test the *statistical significance* of the results. However, most statistical significance tests assume that the samples in the data sets are independent, which is an assumption that is hard to meet in case of the elements of gene prediction. For example, a small change in prediction of one exon of a gene may lead to a completely different predictions of other exons within the same gene, or small extension of a gene on one end may critically affect prediction of neighbouring genes. To the best of our knowledge, we are not aware of any standard tests of statistical significance that would be commonly accepted and used in the gene finding community. Therefore, we do not compute the statistical significance of our results, and some of the results presented in this thesis may, in fact, not be statistically significant.

Finally, we need to consider the case when predictions and correct annotations may contain alternatively spliced forms of the same gene, or overlapping genes. In this case, nucleotide and exon statistics are computed in the same way, though in all sets, we count each object only once (i.e., if there are two instances of the same coding nucleotide in two splicing forms, we always count them as one nucleotide).

In gene statistics, the overlapping splicing forms are considered as a single gene, and the two genes are considered the same if they share at least one transcript. This allows us to

correctly define notions of true/false positives/negatives.

1.2.5 Experimental Verification of Gene Predictions

In the previous section, we assumed that we already have correct annotations of the sequences. However, it is not easy to obtain correct annotations. Otherwise, there would be no need for computational gene finding. In general, it is often possible to collect positive evidence for the existence of a particular splicing form or a particular gene, but it is generally not possible to rule out the existence of a predicted splicing form. In this section, we give a short overview of the experimental techniques that could be used to obtain experimental evidence on existence of particular genes or their splicing forms. Such experimental protocols can be generally divided into three categories: methods based on *random sampling*, *genome-wide analysis*, and *prediction driven* methods.

Common to all of these methods is that to discover or verify a particular splicing form, we must obtain a biological sample wherein the splicing form was transcribed into RNA, or *expressed*. Since some genes are transcribed only under very specific conditions, we may never be able to discover these genes or some of their splicing forms experimentally. Standardized samples containing mRNAs transcribed and spliced under specific conditions are called *libraries*.

1.2.5.1 Methods Based on Random Sampling

Sequencing of *expressed sequence tags* (ESTs) is a typical random sampling method (Adams et al., 1991). ESTs come from randomly chosen short pieces of spliced mRNA sequences. The likelihood of sequencing an EST corresponding to a particular splicing form depends on many factors, the most important being the abundance of the transcript in the particular library. In later stages of EST sequencing projects, the yield of novel ESTs decreases, and more and more samples start to reoccur. Eventually, it is no longer economical to continue the EST project. Thus, some mRNAs, even if they are expressed in a library, may not be sequenced. Due to technological restrictions, ESTs are usually 400–600 bp long (Pontius et al., 2002). These bases often come from either the 5' end or 3' end of the transcript. Since many transcripts are longer then 1000bp, sequences from the middle of such long transcripts are under-represented in EST databases. EST libraries may also be contaminated by incompletely spliced pre-mRNA transcripts.

It is also possible to sequence proteins directly, such as by *mass spectrometry* (Henzel et al., 1993). Mass spectrometry experiments rely heavily on computational post-processing (see for example Perkins et al. (1999)) to recover the amino acid sequence. The same restrictions with respect to choosing the proteins for sequencing apply as in the case of ESTs.

1.2.5.2 Genome-Wide Analysis

Genome-wide analysis is based on asking the following query: How much mRNA that contains a specific short sequence s is contained in a particular library? The length of the sequence s is usually large enough so that the sequence s occurs only once in the whole genome. Many such questions can be asked in parallel using *expression arrays*. In fact, recently such arrays tiling 10 human chromosomes, with one query every 5 nucleotides, were built by Affymetrix (Cheng et al., 2005). This allows building high-density transcriptional maps of human chromosomes.

However, analysis of such transcriptional maps is hard, and the results vary depending on the computational method used for the analysis. Due to expression array technology, it is not straightforward to distinguish signal from noise. Moreover, the signal obtained from such an expression array is a mixture of signals from all of the alternative transcripts of the same gene, and it is difficult to distinguish individual transcripts. Complex methods (not dissimilar to gene finding) were developed to interpret the data from such tiling arrays.

1.2.5.3 Prediction Driven Methods

While the previous two methods work independently of gene predictions, methods in this category are designed to confirm already predicted transcripts. Some aspects of a candidate gene structure can be confirmed with *reverse transcription-polymerase chain reaction* (RT-PCR). For example, it is possible to confirm the correctness of the boundaries of predicted introns, and sometimes (coupled with sequencing), it is possible to detect and correct small errors in gene predictions, such as skipping short exons, or a short-range mistake in the prediction of a splice sites. Special protocols using RT-PCR can also be used to extend or verify predictions at the 5' and 3' ends of the genes. Eyras et al. (2005) recently performed such large scale experiment using RT-PCR on a random sample of predicted genes in the chicken genome.

1.3 Hidden Markov Models for Gene Finding

In this section we show a simple HMM for gene finding upon which we have built our gene finding program ExonHunter. The topology of the HMM is based on basic knowledge of the properties of DNA sequences presented in Section 1.2.1. The model that we present in this section contains many parameters (such as order of various states and lengths of regions corresponding to signals) that were chosen somewhat arbitrarily according to prevailing practice in the field of gene finding. Most of these choices are not backed by systematic experiments and there may be better choices for these parameters. In Chapter 5, we present more detailed comparison of these parameters for three gene finders: Genscan (Burge, 1997), Augustus (Stanke and Waack, 2003), and our gene finder ExonHunter (see Table 5.1 on page 132).

As we noted above, eukaryotic DNA is composed of three types of regions. The *exons* are the coding parts, which are used to encode proteins. The *introns* are the non-coding

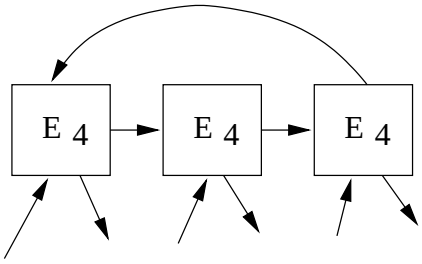

Figure 1.12: **Example of exon model.** States are shown as boxes with the label E for exons, the number associated with the boxes indicate order of the corresponding Markov chain.

sequences inserted between exons in the gene. Finally, the *intergenic regions* are non-coding regions separating genes. See Figure 1.3 on page 14. We will first show how to model exons, introns, and intergenic regions, and then how to place these small models together to form a model of whole genes and chromosomes.

In all figures in this section, we show states as boxes with the label E (exons/coding sequence), I (introns) or X (intergenic region); the numbers associated with these boxes indicate the order of the corresponding Markov chain in that state.

1.3.1 Exon Model

Every in-frame triple of bases in an exon (called a *codon*) encode one amino acid in the resulting protein. Because significant information is encoded in exons, it is natural that such coding sequences show strong compositional bias. Moreover, neighboring codons in the resulting sequence also show strong dependence (Zhang, 1998).

An example of an exon model is shown in Figure 1.12. We generate the nucleotides of exons using a three-periodic Markov chain, where each state is of fourth order. Note that introns may split some codons between neighboring exons. Thus, when modeling internal exons, we may enter the exon model in any state and leave it in any state. (This is not true for the first exon, which must be entered in the first nucleotide of the codon or for the last exon, which must be left after emitting the last nucleotide of a codon.) Note also that the first positions of exons depend on the last positions of preceding introns.

1.3.2 Intron Model

In our basic model, we represent the donor and acceptor signals in the intron structure by a chain of states of order 2, though we will show better methods for modeling signals in Chapter 2. We also include a pyrimidine tail section with 20 states. Between the two signals, we represent the rest of the intron by a single fourth-order state as shown in Figure 1.13. Alternately, one could represent the pyrimidine tail by a single state with a self-loop instead of the 20 states. That would result in model closer to reality—the length of the pyrimidine

tail varies in different introns. However, there are disadvantages to such an arrangement, which we discuss in Chapter 4.

1.3.3 Start and Stop Sites

The coding region of a gene's first exon starts with the start codon ATG. This start signal is encoded straightforwardly by a chain of states as shown in Figure 1.14. Note that compared to the donor and acceptor site signals, we use first-order states instead of second order states. There are many more donor and acceptor sites than start sites, since a gene contains on average 8–10 exons (International Human Genome Sequencing Consortium, 2004). Thus we are unlikely to have enough information to train the higher order Markov chain without overfitting.

Similarly, the stop codon (TAA, TAG, or TGA) is represented by the chain of states modeling stop codon and its surroundings. For the last nucleotide of the stop codon, we use a first-order state, to make sure that the stop codon is TAA, TAG, or TGA, and not TGG.

1.3.4 Untranslated Regions and Intergenic Region

The region immediately upstream of the start site is called the 5' untranslated region (5' UTR). There is a complex promoter signal near the transcription start site consisting of three functional elements, some of which may be omitted: cap-site, TATA-box and CAAT-box. Similarly, the region downstream from the stop site is called the 3' untranslated region (3' UTR). It contains a strong signal, called the AATAA-box and Poly(A)-site, at its end. Details of the composition of both kinds of UTRs can be found in Zhang (1998). However, we chose not to model untranslated regions, since they are not reliably annotated in the training sets, and thus it is hard to train the parameters of such models.

Intergenic regions are modeled by a single state fourth order Markov chain. It is common for gene finders to be trained on single gene sequences that do not include much of intergenic regions. Therefore we instead set the emission probabilities of the intergenic regions as an average of the values trained for introns on the forward and on the reverse strand. This is a reasonable approximation, as observed in Section 1.2.1.1.

1.3.5 Putting the Pieces Together

We are ready to put together the whole model. The model for a single gene sequence will require four copies of the exon model, and three copies of the intron model. The gene model is shown in Figure 1.16.

The four copies of exon models are used for different purposes. One copy is used for the first exon always starting in frame 0 after the start codon. One copy is used for the last exon that always ends in frame 2, before the stop codon. One copy is used for internal exons, which may be entered and left in any frame. Finally, the last one is used as a representation of single exon gene, which must be entered in frame 0 and left in frame 2. The three intron models are exactly the same, but they are used to preserve coding frame. For example, if

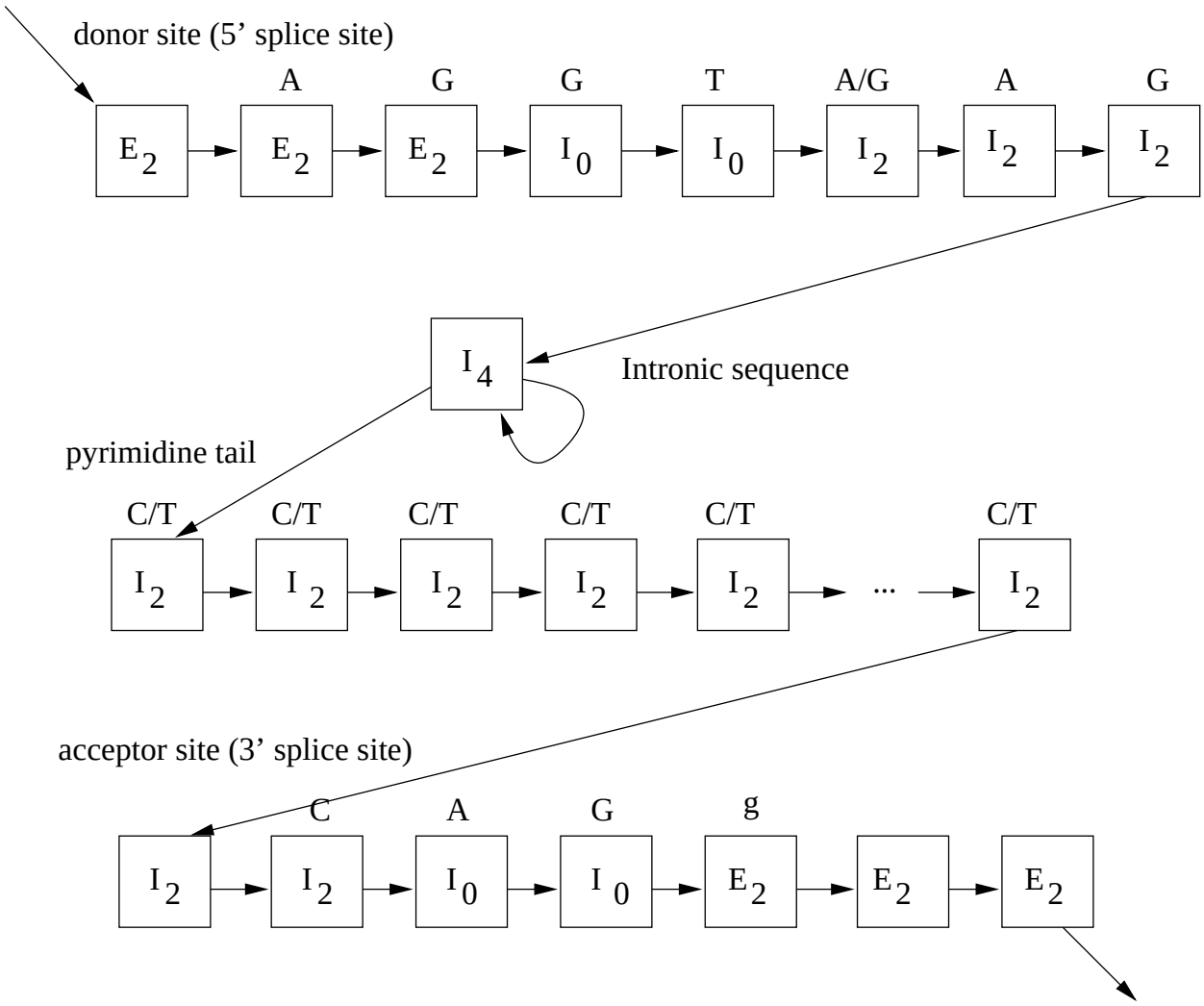

Figure 1.13: **Example of an intron model.** The consensus sequences corresponding to each of the signals are shown above each of the states representing the signal. Lower-case characters represent a weak consensus, upper-case characters represent a strong consensus.

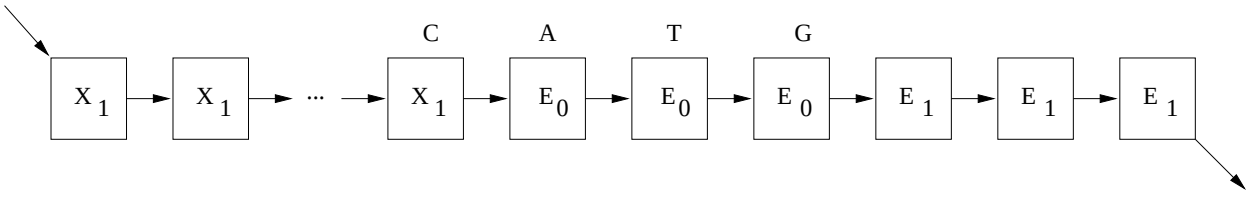

Figure 1.14: Start site model

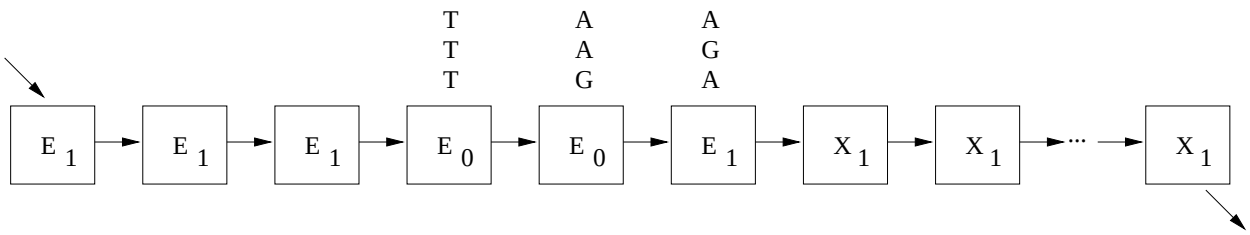

Figure 1.15: Stop site model

we leave an exon in frame 1, we must enter the next exon in frame 2. The three copies allow the model to maintain in which frame it left the last exon. The placement of other elements (such as start signals, stop signals) is straightforward.

By naively copying models of certain sequence elements, we could greatly increase the number of parameters of the model. This increases the amount of data needed to train its parameters. However, we can deal with the problem by using *parameter tying*. For example, we can assume that the four first nucleotides in all exon model copies have the same emission probabilities. Similarly, we can assume that all intron models have the same parameters.

To create a model of multi-gene sequences instead of just a single gene, we need to be able to model genes on both strands. This is easily achieved by copying the original model, and reversing all the transitions. In addition, when the forward model would omit a nucleotide, we need to ensure that the reverse model emits its complement.

Finally, we separate the genes using a single-state model of an intergenic region. From this state, we can either follow a transition to the beginning of a gene on the forward strand, or to the end of a gene on the reverse strand. The resulting model is shown in Figure 1.17.

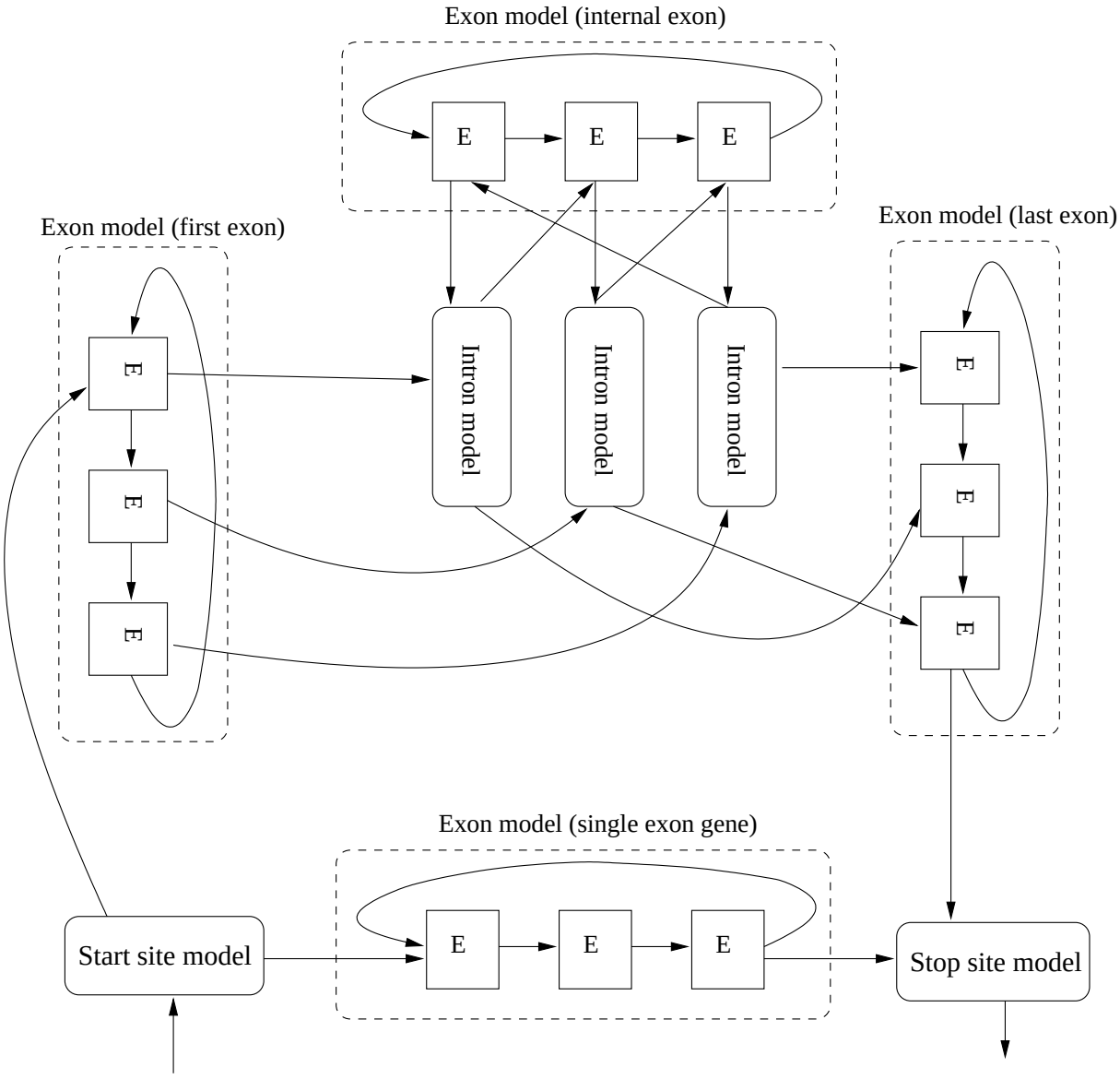

Figure 1.16: HMM for a sequence with a single gene on the forward strand

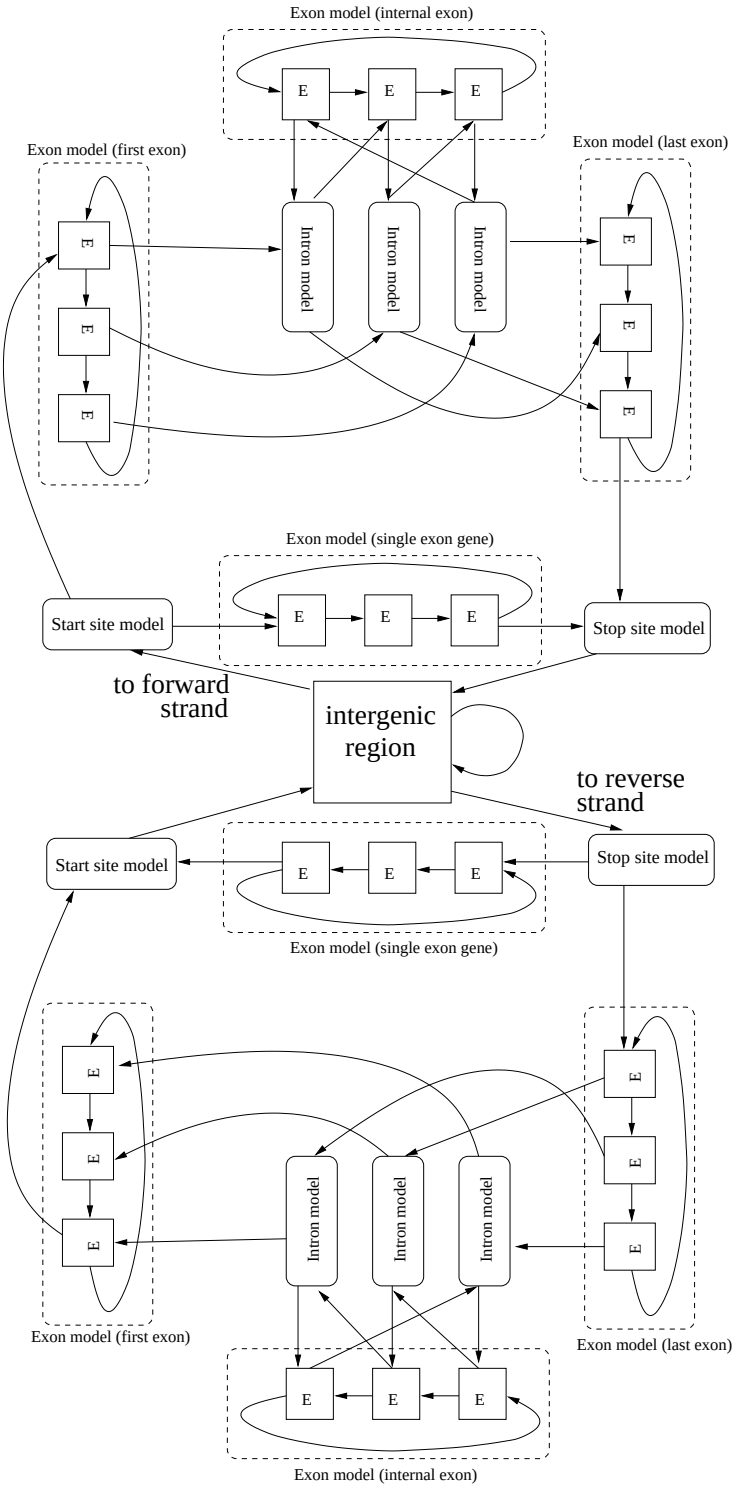

Figure 1.17: HMM for a multi gene sequence with genes on both strands

Chapter 2

Higher Order Tree Models for Signal Recognition

In this chapter, we examine the problem of signal recognition. As we have seen earlier, recognition of biological signals (such as donor, acceptor, start, and stop sites) is an important component of gene finding. Lim and Burge (2001) estimated that more than half of the information required for successful gene finding is contained in such biological signals. In this chapter, we propose a new family of *higher order tree* (HOT) models based on optimal directed spanning hypertrees in hypergraphs. These models incorporate a limited amount of intra-signal dependencies, chosen optimally from all dependencies of bounded cardinality among positions of a signal.

In Section 1.3, we suggested that signals can be modeled as stretches of states in hidden Markov models, with each state representing the probability distribution of a single nucleotide (see Figure 1.13 on page 30). If the Markov chains associated with the states are zeroth order, we obtain what are often called *position weight matrices* (PWMs) (Staden, 1984; Stormo et al., 1982). If we instead use Markov chains of order k, we will call such a model a *k-th order PWM*. Alternative names of the same model in literature include *position specific score matrix* or *weight matrix model* for PWMs of order 0, and *weight array model* for PWMs of order 1.

PWMs can easily model dependencies between adjacent positions in the signal, but they cannot model long-range dependencies within signals. Yet, such dependencies exist, as demonstrated in Figure 2.1. Thus the faithfulness of the signal model would improve if the model could capture these non-adjacent dependencies. Such improvement can potentially translate into higher accuracy of gene prediction.

To study this problem, we make the following modification to hidden Markov models. Instead of modeling signals as stretches of ℓ states in the model, we replace them with a single composite state, which generates strings of a fixed length ℓ in a single step. This can be easily incorporated in all algorithms used for training and decoding HMMs. In this way, we can formulate the problem of finding probabilistic models of signals separately from the problem of gene finding. In particular, we are looking for generative models that generate signal-like sequences in a fixed window of length ℓ around a given functional site in the DNA

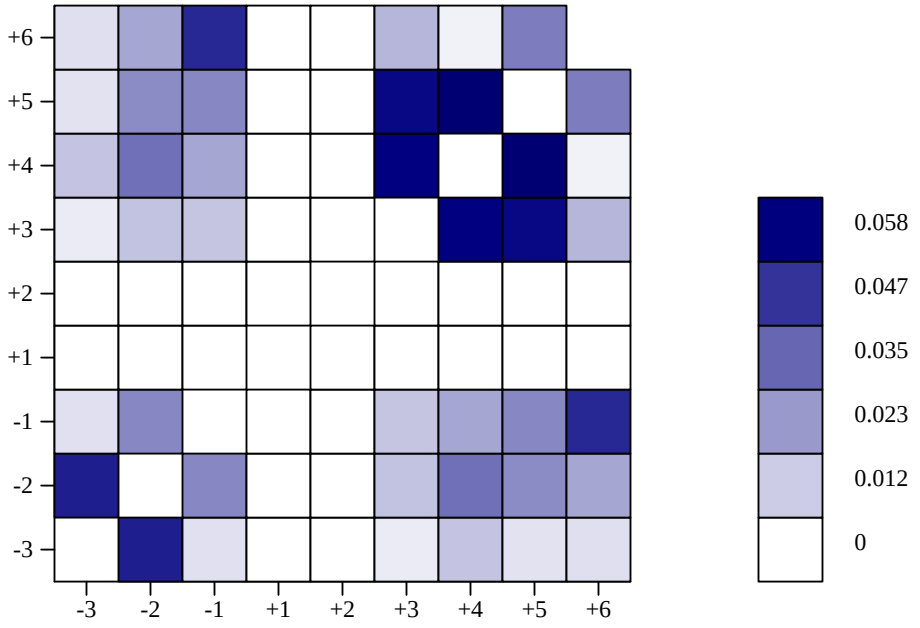

Figure 2.1: **Pairwise dependencies in human donor splice site.** These dependencies were observed in the human chromosome 22 training set. The color intensity represents the amount of dependency between two positions. In particular, the intensity of a cell (i, j) in the table is the differential entropy $H(i) + H(j) - H(i, j)$, where $H(i)$ is the entropy of position i, and $H(i, j)$ is the joint entropy of positions i and j. We will show in Section 2.2.2 that this is indeed a reasonable measure.

sequence.

The simplest generative model would give a separate probability to all 4^ℓ possible sequences of length ℓ. This model would implicitly account for all intra-signal dependencies. However, such a model would have $4^\ell - 1$ parameters that would need to be estimated from the training data. For example, to model donor splice sites, which is commonly done with a window of length 9, this would mean 262 143 parameters. Only rarely would we have enough data to train a model with so many independent parameters without overfitting.

In this chapter, we explore a new class of generative signal models, *higher order trees* (HOTs). The two extreme examples of these models are PWMs of order 0, which require only 3ℓ parameters, and the model suggested in the previous paragraph, with $4^\ell - 1$ parameters. Choosing HOT models of increasingly higher order will capture increasingly more intra-signal dependencies that can be observed in the data. However, the number of parameters is exponential in the order, and therefore increasing the order also requires larger training data set. Thus, we introduce a tradeoff between model faithfulness and available training data.

The rest of this chapter explores use of these models. First, we introduce HOTs, and we compare them to other models commonly used for capturing intra-signal dependencies in signal recognition. The HOT models are based on an underlying hypertree structure. Then we show how to find an optimal HOT structure from the training data so as to maximize the likelihood of the training data. Even though the problem is hard in general, we can formulate it as an integer programming problem easily solved by common software tools for integer programming such as CPLEX (ILOG Inc., 2003). We conclude this chapter with a series of experiments, evaluating the performance of our new models on human splice site recognition. Our new models offer a modest improvement over existing techniques, most notably in improving the accuracy of scores given by the probabilistic model, rather than in increasing its usefulness as a classifier.

In gene finding, there are many factors that work together in non-trivial combination to achieve good prediction accuracy. Our experiments show that improving the faithfulness of the donor and acceptor site models does not directly increase sensitivity and specificity of predicting splice sites in the final gene prediction. Surprisingly, this change improves other indicators, such as exon level statistics, and the discovery of start and stop sites. This result confirms that it is important to test improved models for signal recognition in the context of the hidden Markov models for which they were designed.

Most of the work presented in this chapter was published in Brejová et al. (2003). Parallel work, suggesting similar models to ours appeared in Barash et al. (2003), Yeo and Burge (2003), and Castelo and Guigó (2004). We discuss the similarities and differences among these publications at the end of the chapter.

2.1 Intra-signal Dependencies and HOT Models

The simplest generative model for signals is the position weight matrix (PWM) (Staden, 1984; Stormo et al., 1982). A PWM gives a separate probability distribution of nucleotides

	-3	**-2**	**-1**	**+1**	**+2**	**+3**	**+4**	**+5**	**+6**
A	.30	**.64**	.09	0	0	**.51**	**.67**	.06	.14
C	**.41**	.10	.04	0	0	.04	.10	.07	.20
G	.19	.12	**.82**	**1**	0	.44	.14	**.81**	.23
T	.10	.14	.05	0	**1**	.01	.09	.06	**.43**

Table 2.1: **Position weight matrix for donor site.** The nucleotide frequencies for positions $[-3, +6]$ around donor splice site were observed from the chromosome 22 training set annotations. We can observe the consensus sequence nAGGTnAGT.

to each position of the signal, and assumes that there are no other dependencies among the positions in the model. If $r(i, b)$ is the probability that the ith position in the signal is the base b, then the probability that the model generates a given sequence s of length ℓ is $\prod_{i=1}^{\ell} r(i, s_i)$.

Zhang (1998) characterized many signals used in gene finding in the form of PWMs. An example PWM that characterizes donor sites in the window $[-3, +6]$ around the donor splice junction is shown in Table 2.1.

The basic assumption of PWMs is that the probability of generating a character depends only on its position within the signal. Many researchers (e.g., Burge (1998); Zhang (1998)) have demonstrated that this assumption is false. In fact, there are significant dependencies within signals between positions located even several bases apart. Higher order PWMs (Zhang, 1998) incorporate dependencies between adjacent positions in the sequence. For example, in a first order PWM, the probabilities at each position depend on the immediate predecessor in the sequence. To generate a signal, one starts with its first position, and then each subsequent character is picked using a probability distribution that is conditional on the character generated as its predecessor.

Here, we investigate an extension of PWMs to allow for multiple dependencies among non-adjacent signal positions. In our model, each position in the signal depends on a fixed set of other positions, and there are no cyclic dependencies. We can view such a model as a directed acyclic graph (DAG), where nodes represent positions in the signal, and edges represent dependencies between the positions. Each node is assigned a probability distribution of bases depending only on the bases at positions that are its immediate predecessors in the DAG (see examples in Figure 2.2). We call the underlying DAG the *signal model topology*, and the maximum in-degree of a node in such a graph is the model's *order*. Such models are also called *Bayesian networks* and are extensively used in machine learning (Friedman et al., 1997).

The acyclicity of the graph assures straightforward interpretation of the parameters of this generative model as conditional probabilities. Some graphical models, such as undirected *Markov random fields* (Chellappa and Jain, 1993), allow cycles, but the probabilistic interpretation of these models is not straightforward, and we do not use them in this chapter.

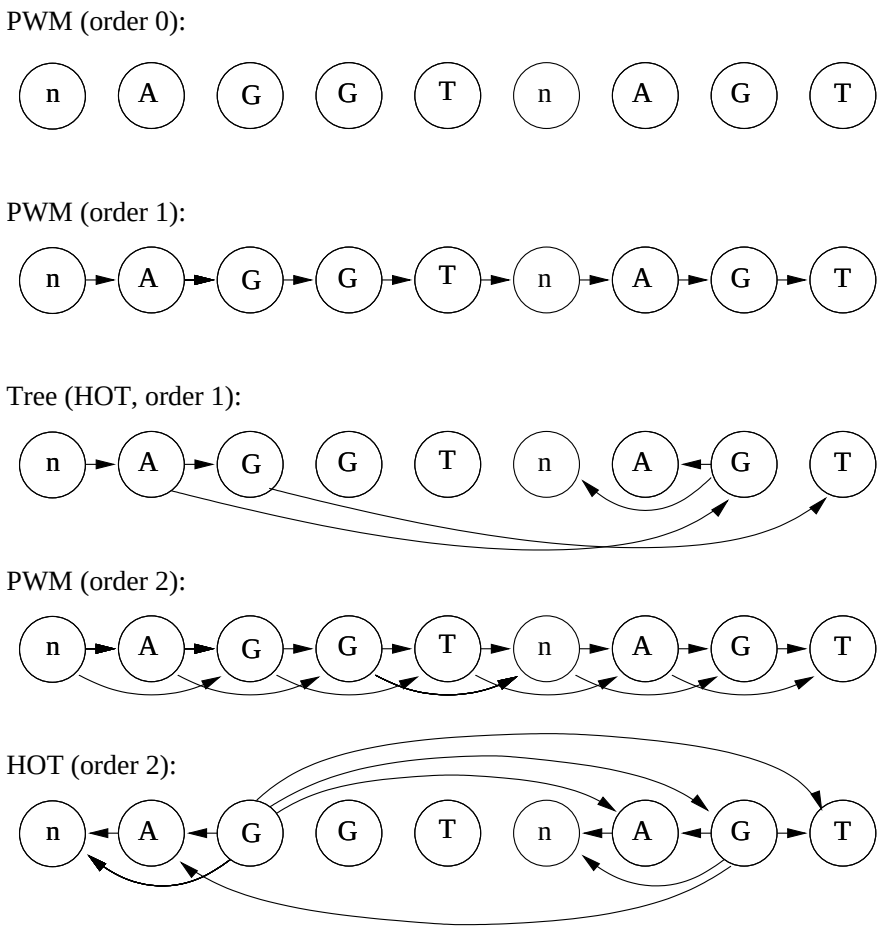

Figure 2.2: Examples of different model topologies for donor signal

To generate a signal in one of these models, one iteratively finds positions whose direct predecessors have all been fixed, and then chooses the symbol at that position from the conditional probability distribution dependent on the predecessors. This can of course be done in $O(k\ell)$ time, if the signal is of length ℓ and k is the order of the model.

Similarly, one can compute the probability that model M generates a given sequence s by simply multiplying these conditional probabilities. If the ith position in the sequence is dependent on positions $d_{i,1} \dots d_{i,k_i}$, and we denote $r(i, b, x_1, \dots x_{k_i}) = \Pr[s_i = b \,|\, d_{i,1} = x_1, d_{i,2} = x_2, \dots, d_{i,k_i} = x_{k_i}]$, the probability $\Pr(s \,|\, M)$ of generating the sequence s by model M is:

$$\Pr(s \,|\, M) = \prod_{i=1}^{\ell} r(i, s_i, s_{d_{i,1}}, s_{d_{i,2}}, \dots, s_{d_{i,k_i}}). \tag{2.1}$$

If the topology of the HOT model is fixed, the number of parameters that need to be trained depends on two variables: it is linear in the length of the signal, and exponential in the order of the model. The amount of data available for training the probability distribution at any given position does not change with the order of other positions or length of the signal. Therefore the model's order is the most relevant parameter in the training, and the amount of data needed for training a k-th order HOT model is comparable to the amount of data needed for training a PWM of the same order. It may appear that HOT models have fewer parameters, since some nodes may be of lower order than k. However, we are searching for a model over a wider family of distributions (namely, all possible topologies). Therefore, it is possible for a HOT model to overfit during training, even if the PWM model of the same order does not overfit.

2.1.1 Previous work

HOT models of order zero are exactly PWMs, which as we have noted are extensively used to characterize signals and motifs in bioinformatics (see for example Zhang (1998) for a comprehensive study of signals related to genes and gene finding).

The underlying topology of HOT models of order one are directed forests (in fact, later we show that it is sufficient to consider directed trees). Tree models of this sort were first explored in the statistics literature by Chow and Liu (1968), and they are also called Chow-Liu trees. The optimal structure, maximizing the likelihood of a tree model, can be found in polynomial time, using standard graph algorithms. Tree models were used in bioinformatics for the classification of splice sites by Cai et al. (2000) and by Agarwal and Bafna (1998). Agarwal and Bafna (1998) used an alternative approach to measure the amount of dependencies between signal positions, based on chi-squared statistical test. In both cases, the authors found that tree models can capture significant non-adjacent dependencies within the donor splice site, but Cai et al. (2000) also conclude that this does not make significant difference when such models are applied to the classification task of distinguishing true signals from decoys.

Ellrott et al. (2002) proposed a special subclass of tree models, the path models. The path models are HOT models of order 1, but their structure must be a directed path instead of a directed tree. They use these models for the related problem of locating transcription factor binding sites. Unfortunately, finding the optimal path model is NP-hard, since the underlying problem is the travelling salesman problem. We do not see much advantage in using path models since tree models are more expressive and can be optimized in polynomial time. Recently, their work was extended by Zhao et al. (2004) by employing variable order Markov chains together with path models. In their models, every position i can depend on several other positions immediately preceding position i on the chosen path. The order varies depending on the amount of training data available for training the conditional probabilities.

A very popular generative model in the context of gene finding is maximum dependency decomposition (MDD) designed by Burge (1997) for Genscan. The MDD model consists of two components: a decision tree that first generates the nucleotides at "important" positions with large amount of influence on the other positions within the signal, and a PWM-0 model at each leaf of the tree. The decision tree is inferred by a greedy method, starting with the position that exhibits the largest amount of dependencies to other positions as measured by chi-squared test. Burge identifies four important positions within donor site signal: -1, -2, $+5$, and $+6$. These positions proved to be significant in our experiments as well. The MDD model used in Genscan can be expressed as a fourth order HOT model. However, this is not very practical; such a model would contain many parameters not used by the MDD model.

The ultimate HOT model for a signal with a window of length ℓ is HOT-$(\ell - 1)$. Such a model essentially contains a separate independent probability for each of the $|\Sigma|^\ell$ strings. The number of independent parameters of such model is high and it would require an enormous amount of data to train such model reasonably well. However, Stanke and Waack (2003) suggested *similarity-based sequence weighting* to overcome this problem. In this method, the authors expand the training data set by including additional samples with lower weight. If s is in the original training data set, the authors include all sequences that are within Hamming distance 1 with weight $w_1 < 1$, all sequences that are within Hamming distance 2 with weight $w_2 < w_1$, etc. In this way, authors create sufficiently large data set to train the model.

As we mentioned before, HOT models are a subclass of Bayesian networks, which are commonly used for classification in machine learning literature (Friedman et al., 1997). Learning the optimal structure of a Bayesian network is a problem closely related to training the structure of HOT models. Much of the work in learning the structure of a Bayesian network is focused on the design of a *scoring metric* that evaluates both how well the structure fits the training data set, and the complexity of the network structure. A penalty for the network complexity is required to prevent learning the network corresponding to the complete graph all the time. After the scoring metric is defined, the problem is usually reduced to a heuristic search for a structure with high score in the metric. For more detailed review of this technique, see Heckerman (1999).

In HOT models, we take a different approach. We specifically include the limits on the complexity of the structure in the definition of the class of HOT models, since training of

higher order Markov chains requires a higher amount of training data. Once the order of the model is fixed, we compute the optimal model structure maximizing the likelihood. Since our models are small, this can be done exactly by using integer programming, and even simple greedy heuristics achieve good results.

In this chapter, we study only generative models for signal recognition. The reason for this is that we want to incorporate these models into a larger probabilistic generative models for gene finding. However, the classification problem for signal recognition has also been studied separately from the problem of gene finding, and multiple methods for discrimination of signals from their decoys have been developed.

Such discriminative methods work in two steps. First, we collect information about signal site and its surrounding. Such information may include symbols at each position of the signal, amino acid composition in some window (usually much larger than ℓ) on the left side and right side of the signal, the number of stop codons which appear in such windows, and other sequence features. This information is transformed into a set of real-valued or discrete features. Second, linear discrimination (Solovyev et al., 1994), quadratic discrimination (Zhang, 1997), neural networks (Xu et al., 1994), support vector machines (Zien et al., 2000; Sonnenburg et al., 2005), or other discrimination methods (Chuang and Roth, 2001) are used to separate signals from decoys.

While these methods generally show better performance than generative models when used only to predict signals in DNA sequences, they are not very useful in the context of hidden Markov models for gene finding. First, they usually do not have a straightforward probabilistic interpretation, making it hard to incorporate their predictions to generative probabilistic models. Second, they usually use information from a long window of sequence around the functional site, which is used in other parts of the model as well. This could yield unpredictable dependencies between their prediction and the rest of the hidden Markov model.

2.2 Maximum Likelihood Training of HOT Models

In this section, we investigate methods for the estimation of parameters of HOT models to maximize likelihood of the training data set. Once the topology of a model is fixed, we only need to count the frequencies from the training data to produce the probability distributions that maximize the likelihood. Thus the crucial step in training HOT models is choosing the optimal topology given a training data set. We define the training problem in terms of finding optimal spanning trees of directed hypergraphs, and we show that this underlying problem is NP-hard. Given the hardness of the problem, we use integer programming to find the optimal hypertree topology. We also describe a simple greedy algorithm, which we use for some of our experiments. Thi greedy algorithm often produces the optimal solution as found by integer programming, so it is a reasonable heuristic to use.

2.2.1 HOT Models and Hypergraphs

To formalize the problem of finding the best topology, we formulate the problem in terms of hypergraphs.

Definition 10 (Hypergraphs). *A* directed hypergraph *is a pair $\mathcal{H} = (V, \mathcal{E})$, where V is a set of vertices, and $\mathcal{E}$ is a set of directed hyperedges. Each* directed hyperedge $E = (T, h)$ *has a tail T, which is a subset of V (potentially empty), and a head $h \notin T$, which is a single vertex.*[1] *Let the* order of a directed hyperedge *be the cardinality of its tail.*

Definition 11 (Complete hypergraphs). *A* complete hypergraph of order k *is a hypergraph that contains all possible hyperedges (T, h) of order at most k.*

Definition 12 (Spanning directed hypertrees). *A* directed hypertree *is a directed hypergraph $\mathcal{H}$, where each node is the head of at most one hyperedge, and the directed graph which can be obtained by replacing every hyperedge $(\{v_1, \ldots, v_k\}, v)$ with the k edges (v_1, v), (v_2, v),..., (v_k, v) is acyclic. A* spanning directed hypertree *is a directed hypertree where each vertex is a head of a hyperedge.*

Note that a hyperedge in a hypertree may have an empty tail. Therefore, spanning directed hypertrees are analogous to spanning forests in directed graphs.

There is an easy correspondence between spanning directed hypertrees and HOT models: for a given vertex, all incoming edges in the HOT model can be represented as a single hyperedge. Such hyperedges form a spanning directed hypertree.

We will show that training HOT models to maximize likelihood of a given training set $(S^{(1)}, S^{(2)}, \ldots, S^{(m)})$ is equivalent to finding the minimum spanning hypertree in an hypergraph with appropriate hyperedge weights. To define the hyperedge weights, we need to first introduce the notion of Shannon entropy $H(P)$.

Definition 13 (Shannon entropy). *For a set of signal positions P and a string x_P of length $|P|$, let $f_P(x_P)$ be the number of occurrences of the string x_P at the positions in P in the training set. Then the Shannon entropy of these positions $H(P)$ is defined as follows:*

$$H(P) = -\sum_{x_P} \frac{f_P(x_P)}{m} \cdot \log \frac{f_P(x_P)}{m}. \tag{2.2}$$

Note that Shannon entropy $H(P)$ is always non-negative. In the rest of this chapter, we will also use the following well-known properties of Shannon entropy.

Lemma 14 (Shannon (1948)). *Let P and Q be disjoint sets of signal positions. Then $H(P, Q) \geq H(P)$ and $H(P, Q) \leq H(P) + H(Q)$.*

[1] Sometimes, directed hyperedges are defined as a pair (T, H), where both tail T and head H are sets of vertices, rather than requiring that the head is always a single vertex (Gallo et al., 1993). However, we will use the simpler definition here.

Lemma 15 (Jelinek (1968, Chapter 4.5)). *Let P, Q, R be pairwise disjoint sets of positions. Then $H(P,Q,R) - H(Q,R) \leq H(P,Q) - H(Q)$.*

Using the definition of Shannon entropy, we can now formulate the following claim.

Theorem 16. *Let $\mathcal{H}$ be a complete directed hypergraph of order k on a set of vertices representing the positions in a signal. Let the weight of every hyperedge $(T,h) \in \mathcal{H}$ be $H(T \cup \{h\}) - H(T)$, where $H(P)$ is the entropy of the signal positions from the set P in the training set of signals $(S^{(1)}, \ldots, S^{(m)})$.*

Then the directed acyclic graph M^* corresponding to the minimum spanning hypertree $\mathcal{M}^*$ of the hypergraph $\mathcal{H}$ yields the topology of the HOT model of order k maximizing the likelihood of the training set $(S^{(1)}, \ldots, S^{(m)})$.

Proof. We have noted on page 40 that in the maximum likelihood HOT model with fixed topology M, the string S has the probability

$$\Pr(S \mid M) = \prod_{(T,h) \in \mathcal{E}(M)} \frac{f_{T \cup \{h\}}(S_{T \cup \{h\}})}{f_T(S_T)}, \tag{2.3}$$

where $\mathcal{E}(M)$ is the set of hyperedges in the corresponding spanning directed hypertree. In order to maximize the likelihood of generating the training set of independent samples $(S^{(1)}, \ldots, S^{(m)})$, we have to maximize the following probability over all possible model topologies M:

$$\begin{aligned} \Pr(S^{(1)}, \ldots, S^{(m)} \mid M) &= \prod_{i=1}^{m} \Pr(S^{(i)} \mid M) = \prod_{i=1}^{m} \prod_{(T,h) \in \mathcal{E}(M)} \frac{f_{T \cup \{h\}}(S^{(i)}_{T \cup \{h\}})}{f_T(S^{(i)}_T)} \\ &= \prod_{(T,h) \in \mathcal{E}(M)} \frac{\prod\limits_{x_{T \cup \{h\}}} f_{T \cup \{h\}}(x_{T \cup \{h\}})^{f_{T \cup \{h\}}(x_{T \cup \{h\}})}}{\prod\limits_{x_T} f_T(x_T)^{f_T(x_T)}}, \end{aligned} \tag{2.4}$$

where $x_{T \cup \{h\}}$ in the product takes as values all possible strings of length $|T \cup \{h\}|$, and similarly x_T takes as values all possible strings of length $|T|$. Each particular string $x_{T \cup \{h\}}$ occurs in $f_{T \cup \{h\}}(x_{T \cup \{h\}})$ samples from the training set, and therefore we can rewrite the product as above.

Maximizing the probability of generating the training set $\Pr(S^{(1)}, \ldots, S^{(m)} \mid M)$ is equivalent to minimizing $-(1/m) \log \Pr(S^{(1)}, \ldots, S^{(m)} \mid M)$:

$$\begin{aligned} &-\frac{1}{m} \log \Pr(S^{(1)}, \ldots, S^{(m)} \mid M) = \\ &\sum_{(T,h) \in \mathcal{E}(M)} \left[-\sum_{x_{T \cup \{h\}}} \frac{f_{T \cup \{h\}}(x_{T \cup \{h\}})}{m} \log f_{T \cup \{h\}}(x_{T \cup \{h\}}) \right] + \left[\sum_{x_T} \frac{f_T(x_T)}{m} \log f_T(x_T) \right] \end{aligned} \tag{2.5}$$

Note that

$$\sum_{x_T} \frac{f_T(x_T)}{m} \log f_T(x_T) = \sum_{x_T} \frac{f_T(x_T)}{m} \left(\log \frac{f_T(x_T)}{m} + \log m \right) = H(T) + \log m, \tag{2.6}$$

since $\sum_{x_T} \frac{f_T(x_T)}{m} = 1$. Similarly:

$$\sum_{x_{T\cup\{h\}}} \frac{f_{T\cup\{h\}}(x_{T\cup\{h\}})}{m} \log f_{T\cup\{h\}}(x_{T\cup\{h\}}) = H(T \cup \{h\}) + \log m, \tag{2.7}$$

and therefore

$$-\frac{1}{m} \log \Pr(S^{(1)}, \ldots, S^{(m)} \mid M) = \sum_{(T,h)\in\mathcal{E}(M)} H(T \cup \{h\}) - H(T), \tag{2.8}$$

which is exactly what we wanted to prove. □

Theorem 16 shows how to reformulate the problem of finding the graphical model maximizing the likelihood of our training data to the problem of finding the minimum spanning directed hypertree problem.

Finally, we have noted before that our definition of directed hypertrees is analogous to forests in directed graphs, rather than trees. The following lemma suggests that this distinction is not important.

Lemma 17. *Let $\mathcal{H}$ be a complete directed hypergraph of order k on a set of vertices representing the positions in a signal. Let the weight of every hyperedge $(T, h) \in \mathcal{H}$ be $H(T \cup \{h\}) - H(T)$, where $H(P)$ is the entropy of the signal positions from the set P in the training set of signals $(S^{(1)}, \ldots, S^{(m)})$.*

Then there exists a minimum spanning hypertree $\mathcal{M}^*$ of the hypergraph $\mathcal{H}$ that has at most one hyperedge of each of the orders $0, 1, \ldots, k-1$. All the other hyperedges have order k.

Proof. Consider any minimum spanning hypertree $\mathcal{M}$. Let $v_1, \ldots, v_\ell$ be the vertices of $\mathcal{M}$ in the topological order, i.e., for every hyperedge $E = (T, v_j)$ in $\mathcal{M}$, $j > i$ for all $v_i \in T$.

We can construct a new spanning hypertree $\mathcal{M}^*$ that will satisfy the hyperedge order restrictions in our claim. Consider vertex v_i, which is a head of a hyperedge (T, v_i). If $|T| < \min\{i-1, k\}$, we can add some of the vertices from $\{v_1, \ldots, v_{i-1}\}$ to T so that $\mathcal{M}$ remains a hypertree, and the size of the new tail T' is $\min\{i-1, k\}$. Due to Lemma 15,

$$H(T' \cup \{h\}) - H(T') \leq H(T \cup \{h\}) - H(T), \tag{2.9}$$

and therefore this operation will not increase the cost of the hypertree. In other words, cost of this new spanning hypertree $\mathcal{M}^*$ must be smaller or equal to the cost of the spanning hypertree $\mathcal{M}$. Therefore $\mathcal{M}^*$ must be a minimum spanning directed hypertree as well. □

In the rest of this section we provide methods to solve the minimum spanning directed hypertree problem. Depending on the order of a HOT model, we will distinguish two cases. If the order is 1, the problem can be solved in polynomial time by finding the minimum spanning tree in an undirected graph. However, for order at least 2 the problem of finding the minimum directed spanning hypertree is NP-hard. In practical cases we solve the problem by integer programming, or find a reasonable model topology by a simple heuristic.

2.2.2 Finding the Optimal Topology for Tree Models

To solve the problem of finding an optimal topology of HOT model, we first consider the special case of tree models, where $k = 1$.

Lemma 18 (Chow and Liu (1968)). *Consider an undirected graph, where every vertex represents a position in the signal, and each edge (u, v) has weight $H(u, v) - H(u) - H(v)$, where the entropies are computed from the training set. The minimum spanning tree of such a graph, rooted at any vertex, corresponds to the maximum likelihood topology of the HOT model of order 1.*

Proof. According to Theorem 16 and Lemma 17, we are looking for a directed tree with set of edges E and root s minimizing the cost function:

$$\text{cost}(E, s) = H(s) + \sum_{(u,v)\in E} H(u, v) - H(u). \tag{2.10}$$

We can rewrite the formula by subtracting $H(v)$ from the cost of every edge. Since every vertex except the root has exactly one incoming edge, we can compensate for the subtracted cost by adding the term $\sum_{v\in V\setminus\{s\}} H(v)$ as follows:

$$\text{cost}(E, s) = \underbrace{\sum_{v} H(v)}_{(*)} + \sum_{(u,v)\in E} H(u, v) - H(u) - H(v). \tag{2.11}$$

Since (*) is the same for every structure of the model, we can ignore it in the optimization, thus turning the problem into the undirected problem described in the lemma statement.

This proof also shows that every directed tree with the same set of undirected edges has the same cost, regardless of the rooting of the tree. □

Thus, the optimal topology for a HOT model of order 1 can be determined by finding the minimum spanning undirected tree, where the weight of the edge (u, v) is $H(u, v) - H(u) - H(v)$. This can be easily done by Prim's algorithm (Prim, 1957) in $O(\ell^2 \log \ell)$ time (or $O(\ell^2 + \ell \log \ell)$, if we use Fibonacci heaps—see for example Cormen et al. (2001)).

Note that the measure we used in Figure 2.1 on page 36 to visualize the strength of dependencies between the positions of the signal is justified by the above proof as well.

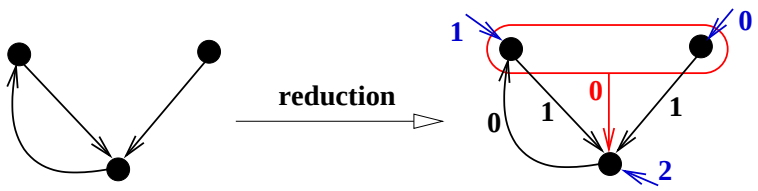

Figure 2.3: **Minimum spanning directed hypertree is NP-hard.** Example of the reduction.

2.2.3 Minimum Spanning Directed Hypertree is NP-hard

Unfortunately, the method introduced in the previous section cannot be extended to models of order $k \geq 2$. In this section we will show that the underlying problem of finding the minimum spanning directed hypertree is NP-hard, even for $k = 2$.

We will require the following result of Karp (1972).

Lemma 19 (Minimum feedback arc set, Karp (1972)). *Consider the* minimum feedback arc set *problem of finding the minimum edge set in a directed graph whose removal makes the graph acyclic. This problem is NP-hard, even if the graph has in-degree at most 2.*

Theorem 20. *Finding the minimum spanning directed hypertree in a hypergraph is NP-hard, even if all the edges are of order at most 2.*

Proof. We prove the NP-hardness by reduction from minimum feedback arc set. Consider an instance of the minimum feedback arc set problem, which is a directed graph G, with in-degree at most 2.

Create a hypergraph $\mathcal{H}$ on the same set of vertices. For every vertex v, we find the set X of tails of edges incoming to v in G. By our assumption, X has at most 2 elements. For each subset A of X, we create a hyperedge (A, v) with cost $|X| - |A|$. To complete the graph, we add all other possible hyperedges with tails of size at most 2, but with a large cost C, so that they will not be chosen. Figure 2.3 shows an example of the reduction.

As established earlier, each spanning hypertree of $\mathcal{H}$ corresponds to a directed acyclic graph M. This DAG is a subgraph of G, assuming it does not use one of the very high-cost edges. Conversely, if we remove a feedback arc set from G, we obtain a directed acyclic graph that correspond to some spanning hypertree of $\mathcal{H}$. Moreover, for every edge deleted from G, the cost of the corresponding hypertree increases by one. Therefore, graph G has a feedback arc set of size at most k if and only if $\mathcal{H}$ has a spanning hypertree of cost at most k (see the example in Figure 2.4). □

2.2.4 Finding the Optimal HOT Topology by Integer Programming

Because of Theorem 20, our hypertree formulation of the problem of finding the optimal HOT topology does not immediately result in an efficient algorithm for models of order

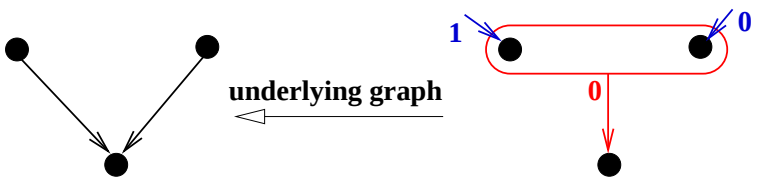

Figure 2.4: **Minimum spanning directed hypertree is NP-hard (cont.).** Solution of the minimum spanning hypertree indicates the minimum feedback arc set.

higher than one, though of course the instances that come from real data may not be hard to solve in general. However, we note that the length of the window ℓ of a signal is often small (and thus, the number of vertices in the underlying hypergraph problem is small). In this case, we can achieve a practical running time by formulating the problem as an integer linear program and solving the problem by a commercial package such as CPLEX (ILOG Inc., 2003). Integer linear programming is guaranteed to give us the optimal hypertree, yet it uses potentially exponential runtime (Schrijver, 1986).

Our IP model of the problem includes two kinds of variables. One family of variables models the acyclicity of the spanning hypergraph, by requiring that we order the nodes in the hypergraph, and that we place the heads of hyperedges later in the ordering than the nodes in their tails. The second family of variables models the hyperedges of the graph. We require that each node have only one incoming hyperedge, to again ensure the hypertree property.

In particular, we assign a decision variable $b_{i,j}$ to each pair of distinct positions, i and j. This variable is set to 1 exactly when i comes before j in the ordering of nodes, and 0 otherwise. There is a variable $a_{T,h}$ for each possible directed hyperedge $E = (T, h)$, where $|T| \leq k$, and where $h \notin T$. When $a_{T,h} = 1$, the hyperedge E is in the chosen spanning hypertree, while when it is 0, the hyperedge is not in the chosen hypertree.

We model the ordering constraint on the nodes by requiring that the order relationship is antisymmetric and that there are no 3-cycles in it. In particular, we require that $b_{i,j} + b_{j,i} = 1$ for all pairs i and j and that $b_{i,j} + b_{j,k} + b_{k,i} \leq 2$ for all triplets i, j and k of distinct nodes.

The requirement that chosen hyperedges are properly ordered is modeled by the constraint $a_{T,h} \leq b_{x,h}$ for all nodes x in T. This requires that all nodes in the tail of the hyperedge are ordered before the head node, for the chosen hyperedge.

Finally, we require that every node has exactly one incoming hyperedge (in the case of the tree's root, this will have no tail nodes). This is the constraint $\sum_{E:E=(T,h)} a_{T,h} = 1$ for all nodes h.

The cost of a chosen hypertree is $\sum_{E=(T,h)} a_{T,h} w_{T,h}$, where $w_{T,h}$ is the cost of including the hyperedge $E = (T, h)$ in the tree.

This description gives the following integer linear program:

Signal	Model	Runtime to find solution optimal	within 1%
Donor $[-3,+6]$	HOT-2	0.38s	0.38s
Acceptor $[-4,+3]$	HOT-2	5.97s	1.71s
Acceptor $[-20,+3]$	HOT-2	hours	91.38s

Table 2.2: **Finding optimal solution with CPLEX (running time).** The time was measured on a computer with 3Ghz Xeon processor and 4GB memory, running CPLEX 8.1 in Linux under normal load. Note that searching for acceptor models takes much more time than searching for donor models, even if we consider acceptor sites with windows shorter than donor sites. As shown in Figure 2.12, the intrasignal dependencies within acceptor sites are weaker than those in donor sites. This suggests that the branch and bound method used for solving the integer program cannot decrease the gap between upper and lower bounds quickly, which makes the instance of the integer program for acceptor sites harder than for donor sites.

$$\min \sum_{E=(T,h)} w_{T,h} a_{T,h}, \qquad \text{subject to:} \tag{2.12}$$

$$\begin{aligned}
b_{i,j} + b_{j,i} &= 1, \text{ for all pairs of nodes } i \text{ and } j, \\
b_{i,j} + b_{j,k} + b_{k,i} &\leq 2, \text{ for all triplets of nodes } i,\ j \text{ and } k, \\
a_{T,h} &\leq b_{x,h}, \text{ for all hyperedges } E = (T,h) \text{ and nodes } x \text{ in } T, \\
\sum_{E:E=(T,h)} a_{T,h} &= 1, \text{ for all nodes } h, \\
a_{T,h} &\in \{0,1\}, \text{ for all possible hyperedges } E = (T,h), \\
b_{i,j} &\in \{0,1\}, \text{ for all pairs of nodes } i \text{ and } j.
\end{aligned}$$

This linear program has $O((\ell+1)^{k+1})$ variables and $O((\ell+1)^{k+1})$ constraints, where ℓ is the size of the window, and k is the maximum order of a hyperedge. We used the integer programming solver CPLEX version 8.110 (ILOG Inc., 2003) to solve moderate-sized instances of these problems, where $k = 2$ (so all hyperedges have at most two head nodes). The runtime of the optimization procedure is summarized in Table 2.2. For signals with short window, the running time is negligible. For signals that require longer windows, finding the optimal solution requires a long time, but for our entropy based objective function it is quite easy to find a solution which is provably within 1% of the optimal solution. We did not experiment with integer programming for orders $k > 2$.

2.2.5 Greedy Heuristic for Finding a Good HOT Topology

For larger models, we have to resort to heuristic algorithms that do not guarantee to find the optimal structure. Fortunately, the resulting model topologies still perform well in practice.

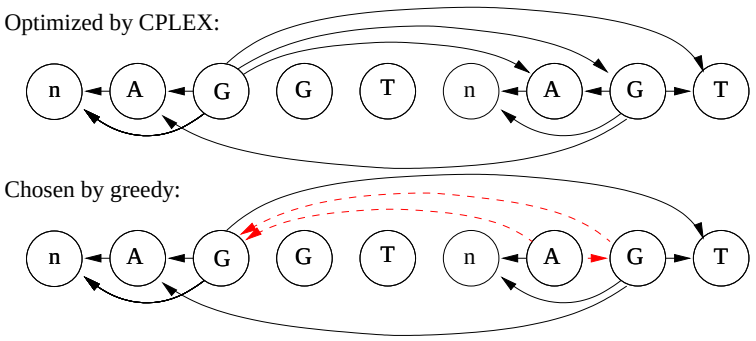

Figure 2.5: **Comparison of models inferred by integer programming and a greedy algorithm.** The dashed edges can be reversed without changing the distribution defined by the model, and therefore the two models are equivalent

We start with a single node in the spanning hypertree $\mathcal{T}$. In each iteration, we add one more vertex v into the hypertree $\mathcal{T}$, where v is the head of the shortest hyperedge (T, h) such that $T \subseteq \mathcal{T}$ and $h \notin \mathcal{T}$. This heuristic can be implemented in $O(kn^{k+1})$ time. Since the hypertrees generated in this way can differ depending on the chosen starting vertex, we run the algorithm for every vertex as a starting point and choose the best resulting hypertree. Note that this is just a simple variation of Prim's algorithm for maximum spanning trees (Prim, 1957).

The model inferred for donor site window $[-3, +6]$ is shown in Figure 2.5, together with the optimal model inferred by integer programming. To compare the two models, we need the following definition and theorem characterizing equivalence classes of probability distributions defined by underlying directed acyclic graphs of hypertrees.

Definition 21 (Covered edges). *An edge (u, v) of a directed acyclic graph is the* covered *if vertex u has the same set of parents as vertex v (aside from u being parent of v). Parents of vertex u are the tails of the edges entering u.*

Theorem 22 (Chickering (1995)). *When used as topologies for HOT models, two directed acyclic graphs G and G' define the same set of probability distributions if and only if graph G can be transformed to graph G' by a sequence of reversals of covered edges.*

The underlying undirected graph of both models in Figure 2.5 is the same, and the only difference is an orientation of three edges (highlighted and dashed). According to Theorem 22, these edges can be reversed, without changing the set of probability distributions that are represented by these models. In other words, the two models are equivalent.

2.3 Experiments

We tested the usefulness of higher order models at identifying splicing signals in the human genome. Our experiments show that there are some cases where allowing multiple non-

Name	Abbrev.	Length	Donor sites		Acceptor sites	
			True	False	True	False
SpliceDB	SpliceDB	N/A	14 558	N/A	14 558	N/A
Chromosome 22 (training)	chr22-training	17.9MB	1 566	1 040 152	1 576	2 656 849
Chromosome 22 (testing)	chr22-testing	16.7MB	1 676	1 693 151	1 688	2 515 278

Table 2.3: **Characteristics of data sets used for testing of signal models.** Only canonical donor (GT) and acceptor (AG) sites were considered as true sites. We list any occurrence of GT or AG that was not marked as a true splice site as a false donor or false acceptor site respectively. The difference in number of splice sites is caused by incomplete gene annotations

adjacent dependencies can increase specificity of the models, but the improvement is quite slight; this is similar to the results of Cai et al. (2000), who studied tree models.

However, one useful finding is that higher order tree models can be better in predicting the probability that a given position actually *is* a signal. This probability can be used in other programs, such as probabilistic gene finders. Perhaps our most interesting result is that using a higher order tree model allows a substantially better prediction of the probability that a position is a donor site than is available with other local sequence-based measures.

We used two training data sets in our experiments. First, we used a data set of gene annotations in chromosome 22, as described in Appendix A, which was split into training and testing sets. The chromosome 22 training set was mostly used for estimation of the background model, as well as to calibrate the frequency of splice sites. Second, we used a database of human splice sites SpliceDB (Burset et al., 2000) as the main training set. The dataset was cleaned for overlaps with the chromosome 22 testing set as described in Appendix A. The data sets are summarized in Table 2.3. In the experiments, we considered only canonical splice sites conforming to the GT/AG consensus at the splice junctions.

2.3.1 Using Generative Models as Classifiers

In classification, we want to assign each sequence s to one of the two classes: "signal" or "background". We can perform classification based on a score obtained from the signal model. By setting a threshold T, we can classify all the sequences s with score lower than T as "background", and all the sequences with score higher than T as "signal". By changing the threshold, we can control balance between sensitivity and specificity of such a method.

For a given sequence s of length ℓ, a generative probabilistic model M_+ computes the probability $\Pr(s|M_+)$ that the sequence s is generated by the signal model. For simplicity, we assume that non-signal positions in the sequence can be represented by a background model M_- and that the signal occurs with probability $\Pr(M_+)$. Under these assumptions, we can compute the probablity $\Pr(M_+|s)$ that a particular sequence S is an occurence of the

signal, using Bayes formula as follows:

$$\Pr(M_+|s) = \frac{\Pr(s|M_+) \cdot \Pr(M_+)}{\Pr(s|M_+) \cdot \Pr(M_+) + \Pr(s|M_-) \cdot (1 - \Pr(M_+))}. \tag{2.13}$$

The values of $\Pr(M_+|s)$ are more suitable for classification than $\Pr(s|M_+)$. The problem with $\Pr(s|M_+)$ is that a high probability of a sequence s being generated by the model does not necessarily mean that the sequence represents the signal. Some sequences may be common in non-signal regions, and thus have higher probability $\Pr(s|M_+)$ simply because they are more likely to occur in general.

To specify the background model M_-, we make the assumption that in contrast to the signal model, the background model should be positionally independent. We use a simple fifth order Markov chain trained with parameters estimated from the chromosome 22 training set chr22-training. Fifth order Markov chains are commonly used to model generic DNA sequence (Burge, 1997). The frequency of the signal occurrence is estimated from the chr22-training as well.

Note that in the chromosome 22 testing set, less than 0.1% of candidate sites are true donor/acceptor sites. Classifying such sites is extremely difficult. Many machine learning methods assume that 50% of all candidate sites in a testing data sets are true splice sites.

2.3.2 Accuracy Measures

We used two types of accuracy measures in our experiments. The first type is sensitivity and specificity, computed in the same way that we described in Section 1.2.4. In particular, if we denote the number of true positives as TP, false positives as FP, true negatives as TN, and false negatives FN, we measure classification accuracy with the sensitivity SN = TP/(TP + FN) and specificity SP = TP/(TP + FP). If we lower the threshold T, the sensitivity increases and the specificity decreases.

For a given model, we can plot a curve depicting the tradeoff between specificity (x-axis) and sensitivity (y-axis), and compare different models. In such plots, the curve of an ideal classifier would occupy the top and right boundary of the graph, passing through the top-right corner (100% sensitivity and specificity), while a random classifier would have specificity close to the prior frequency of the signal for almost all levels of sensitivity.

We can also display this information as a *receiver operating characteristic* (ROC) curve (Metz, 1978). In a ROC curve, we plot the percentage of false positives (x-axis) out of all negatives vs. sensitivity—percentage of true positives out of all positives (y-axis). The ROC curve of an ideal classifier would pass through the top-left corner (100% true positives and 0% false positives), while the ROC curve of a random classifier would be a straight line from $(0, 0)$ to $(1, 1)$. One of the advantages of ROC curves is that the results do not depend on the prior frequency of the signal in the testing set. However, sometimes important differences are more visible from sensitivity/specificity plots. An example of the sensitivity/specificity plot and the ROC curve for a PWM model is shown in Figure 2.6.

Our interest in gene finding and generative probabilistic models has inspired another type of accuracy measure. In these applications, signal models are not used as classifiers,

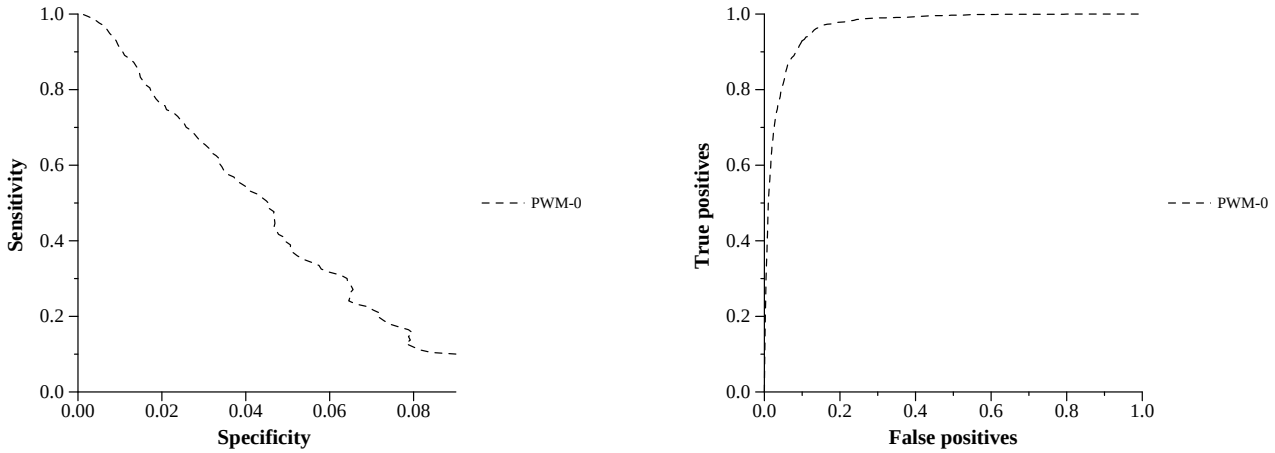

Figure 2.6: **Graphs comparing sensitivity and specificity.** Model PWM-0 for donor splice site was trained on SpliceDB set and evaluated on chr22-testing. Left: Sensitivity vs. specificity plot. Right: ROC curve

but rather to estimate $\Pr(M_+|s)$, the probability that the evaluated sequence is a signal. This estimate is used and combined with other parts of the model. Therefore, this estimate should be as close to the actual probability that the sequence is a signal as possible.

Let λ_s be the estimate of $\Pr(M_+|s)$ given by the model of a signal. Intuitively, λ_s should represent the fraction of true positives in a set of sequences that have similar probability $\Pr(M_+|s)$. To ascertain whether this assumption is true for our models, we first collect other sequences from the testing set, for which the model predicted a probability similar to λ_s. In particular, we choose the 1 500 samples from our test set whose estimated probability is closest to λ_s, but smaller, and analogously the 1 500 sample with closest probabilities, but higher. Then we compute the fraction of true positives λ'_s in this set and compare λ_s and λ'_s by computing the correlation coefficient of λ_s and λ'_s over all sequences s in the testing set. We call this measure a *reliability score*.

In what follows, we abbreviate the various chosen signal topology models as follows: PWM-k is the position weight matrix of kth order, TREE is the first order HOT model, and HOT-k are higher order models of order k. HOT models of order greater than one were optimized by the greedy heuristic method. We also confirmed that the HOT-2 models optimized by the heuristic and by integer programming were the same.

2.3.3 Donor Site Experiments

In this experiment, we want to determine whether our new structured HOT models increase the performance of donor site signal classification, and how well they compare to unstructured PWM models of the same order.

For our experiments, we represented the donor site by the window $[-6, +3]$ of length 9 around the site. In addition to the models presented in this chapter, we also implemented

Model	Sensitivity level 20%	40%	60%	80%	95%	Reliability score
Random	0.1%	0.1%	0.1%	0.1%	0.1%	N/A
PWM-0	7.2%	5.0%	3.4%	1.7%	0.7%	0.953
PWM-1	7.5%	5.5%	4.0%	2.6%	1.1%	0.959
TREE	7.5%	5.5%	4.0%	2.5%	0.9%	0.973
PWM-2	**7.9%**	5.7%	3.9%	2.5%	1.0%	0.967
HOT-2	7.6%	5.6%	**4.1%**	**2.7%**	**1.2%**	0.980
PWM-3	7.7%	5.7%	**4.1%**	2.6%	1.1%	0.971
HOT-3	**7.9%**	**6.0%**	3.9%	2.6%	**1.2%**	**0.986**
MDD	7.7%	5.3%	4.0%	2.6%	1.1%	0.982

Table 2.4: **Specificity at various sensitivity levels and reliability score of donor site models.** The donor site models were trained for window $[-6, +3]$ on the training set SpliceDB and tested on the testing set chr22-testing. For each model, the table shows the specificity at levels of sensitivity 20%, 40%, 60%, 80%, and 95%. The column labeled random shows the estimated specificity of classifier randomly ordering samples and then assigning increasing scores in this random order.

the maximal dependence decomposition (MDD) model of Burge (1997). MDD consists of a set of PWMs of order 0. The PWM to be used for a particular sequence is chosen using a decision tree. For our experiments with MDD, we used the decision tree presented by Burge (1997) that involves the four nucleotides at positions -1, -2, $+5$, and $+6$, but we retrained the PWMs.

Table 2.4 shows the summary of experiments with donor site models. The most significant improvement in accuracy is between zeroth order models and higher order models, with the PWM-0 model being clearly inferior to all others. In both unstructured PWM models and structured HOT models, there is a modest increase in specificity with increasing order.

We can also see that the accuracy is always slightly improved by considering structured models (HOTs) compared to the unstructured models (PWMs). Figure 2.7 shows the comparison of sensitivity and specificity of PWM-2 and HOT-2 models. However, such improvement is not consistent over the whole range of sensitivity levels. In some cases, the improvement is confined to mostly low sensitivity (TREE and HOT-3), in other cases improving high sensitivity range (HOT-2). The improvement can easily be seen on the ROC curves—Figure 2.8 shows a comparison of ROC curves of PWM-2 and HOT-2.

With respect to the reliability score, the structured models clearly win out over unstructured models. Figure 2.9 suggests that this is most likely because unstructured models tend to systematically under-value low scoring patterns, and over value high scoring patterns.

From the tested models, the best model to use in gene finding seems to be HOT-3, though we note that the differences between HOT-3 and some of the other models are very small. The MDD model, which was commonly used in many gene finding programs since it was suggested and implemented in Genscan (Burge, 1997), seems to be closest in performance

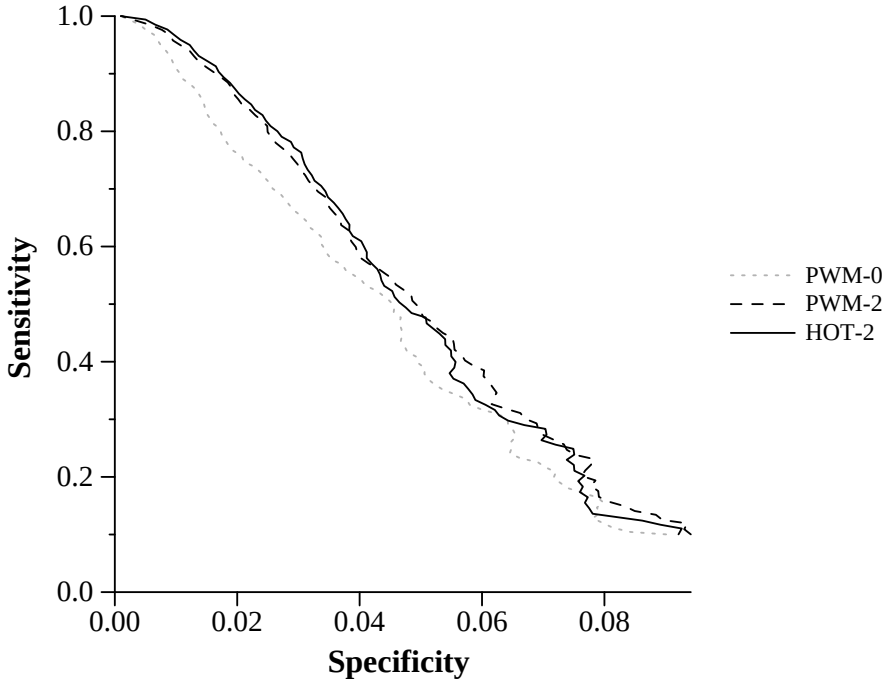

Figure 2.7: **Comparison of donor site prediction for PWM-2 and HOT-2.** The specificity slightly improves in the high-sensitivity range by switching from the unstructured model to the structured HOT model of the same order.

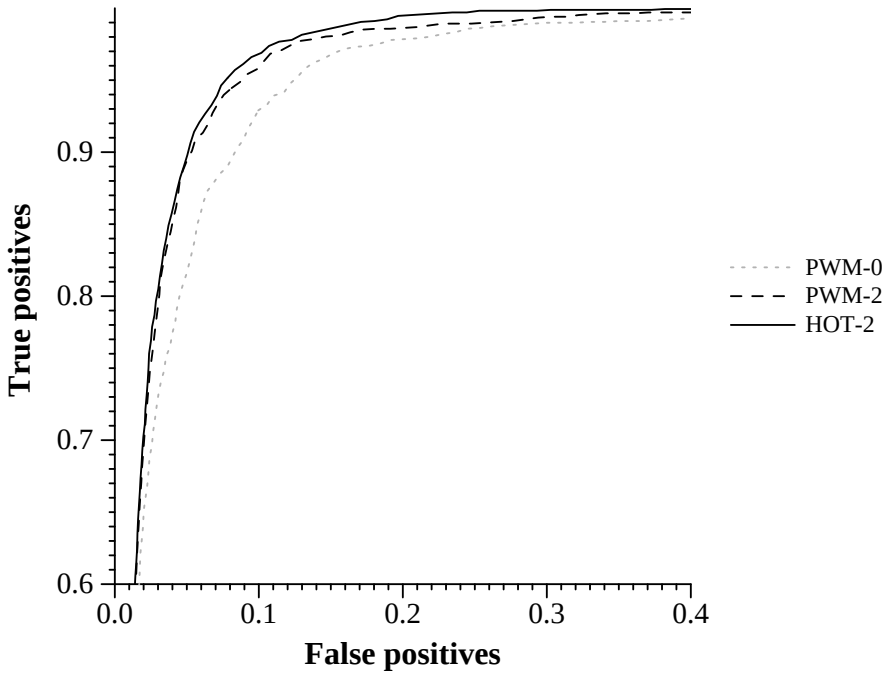

Figure 2.8: **Detail of ROC curve for second order models of donor site.** The ROC curve of HOT-2 dominates the ROC curve of PWM-2, showing the modest advantage of a structured model over its unstructured alternative. However, most of the improvement occurs for scores recalling more than 10% of false positives. Such a false positive rate is impractical if we are using the model as a classifier, since at this rate the absolute number of false positives will be much higher than the absolute number of true positives.

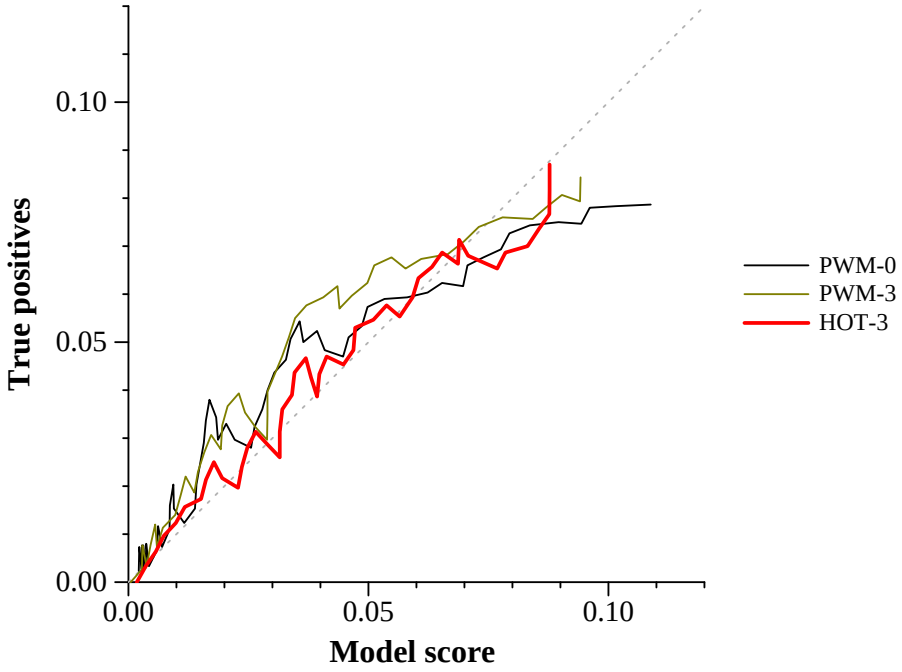

Figure 2.9: **Score vs. actual fraction of true positives.** The model's score, representing an estimate of the expected true positive rate is correlated with the actual fraction of true positives. For a sequence s with score λ_s, the actual number of true positives is computed by counting the fraction of true positives among the examples with scores closest to λ_s. Out of the three models, HOT-3 exhibits the best correlation of model scores with the actual fraction of true positives.

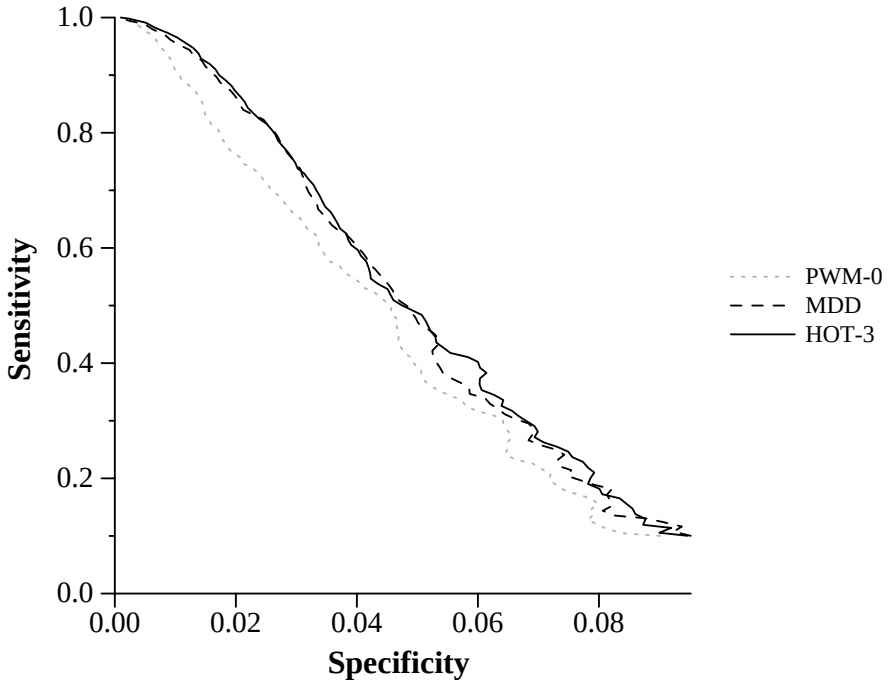

Figure 2.10: The HOT-3 model dominates MDD model of donor site

to HOT-2, with HOT-3 being moderately superior (Figure 2.10 and Table 2.4).

Table 2.5 shows the inferred dependencies in all structured models. It seems that the most important position determining the form of the donor splice site is position -1. In HOT-2, this position is a parent to all positions but $+3$, and in HOT-3 it is a parent to all other positions. This trend is not apparent from Figure 2.1 on page 36, from which the most significant position seems to be $+5$ (consequently, the position -1 does not appear much in the tree model). It seems that positions $+5$ and -1 in fact work together as a pair and their influence becomes significant only if considered as such.

2.3.4 Relationship Between Model Order and the Amount of Training Data

In this section, we study how the performance of the signal models change with increasing amount of training data. In particular, we were interested in how much training data is needed for models of higher order to avoid overfitting, and whether the structured HOT models are more prone to overfitting than their unstructured counterparts.

Figure 2.11 shows that the order of the models which should be used indeed depends on the amount of training data that is available. We have randomly created subsets of the training set of size 1 000, 4 000, 7 000, 10 000, and 13 000 splice sites, and compared the specificity of both unstructured and structured models for each of these training sets at sensitivity level 90%.

Position	Neighbors in TREE	Parents in HOT-2	Parents in HOT-3
-3	-2	$-2, -1$	$-2, -1, +6$
-2	$-3, -1, +6$	$-1, +5$	$-1, +5, +4$
-1	$-2, +5$		
$+3$	$+5$	$+4, +5$	$-1, +5$
$+4$	$+5$	$-1, +5$	$-1, +5, +3$
$+5$	$-2, +3, +4$	-1	-1
$+6$	-1	$-1, +5$	$-1, +3, -2$

Table 2.5: **Structures inferred for structured HOT models.** The hypertree topologies were trained on the data set SpliceDB. For TREE, we show both parents and children of each position, since all the trees which differ only in rooting have the same performance. For HOT-2 and HOT-3 we only show immediate ancestors. The nucleotides at positions +1 and +2 are always GT, and they have no effect on the rest of the model, forming isolated vertices in the model topologies. Note that the positions -1, -2, $+5$, and $+6$, identified as important positions in the MDD model (Burge, 1997), are also identified by our models.

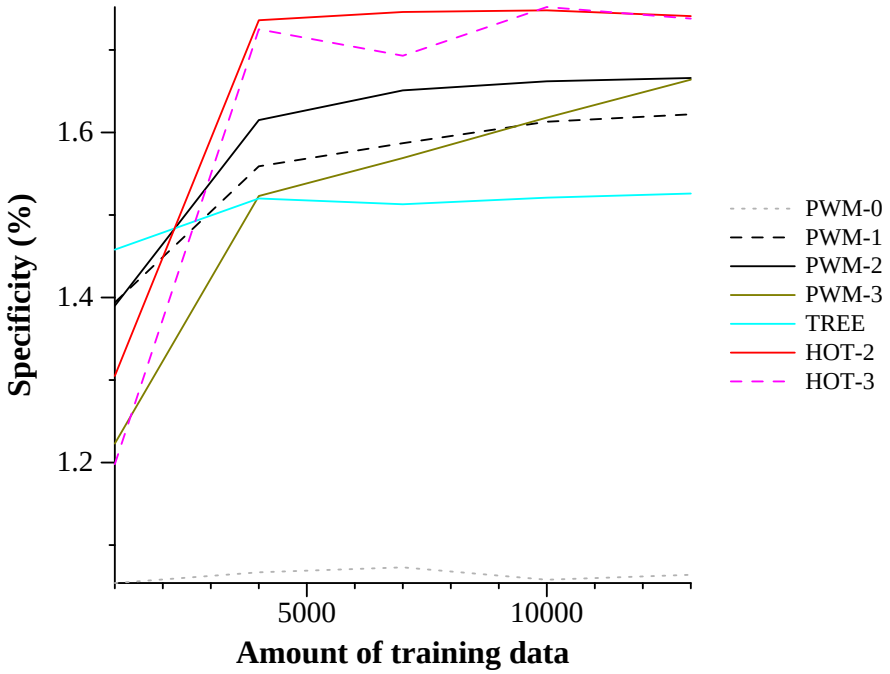

Figure 2.11: Specificity of donor models at 90% sensitivity with increasing amount of training data

We test the statistical significance of our results by using McNemar's test (McNemar, 1947) as outlined by Dietterich (1998). For two classifiers A and B and a test set of independent samples, this test compares the number of samples n_{10} that were classified correctly by A, but incorrectly by B, and number of samples n_{01} that were classified correctly by B, but incorrectly by A. The null hypothesis is that the two classifiers have the same error rate on the test set. The McNemar statistics

$$\chi^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}} \tag{2.14}$$

is distributed approximately as chi-squared distribution with 1 degree of freedom, and the probability that this quantity is greater than 3.841459 is less than 5%. In such a case, we may reject the null hypothesis.

It seems that it is always better to use at least the first order model. If we have only a small amount of data, TREE seems to be the best model to use, while the other models overfit to the training data. With an increasing amount of data, HOT-2, and later HOT-3 become the best models. Adding structure to the models does not seems to increase the chance of overfitting, except when we have very small amounts of data. All the results we mentioned are statistically significant, except for the differences between the HOT-2 and HOT-3 models for large testing set sizes (10 000 elements and more).

Thus in this case, we indeed experience a tradeoff between the amount of available data and the amount of intra-signal dependencies the signals can capture. Figure 2.11, together with Table 2.4, suggest that if we are using an unstructured PWM model of order $k \geq 2$ and we increase the amount of available training data, it is in general more profitable to change the model to the structured model of the same order k, rather than to the PWM model of order $k + 1$.

2.3.5 Acceptor Site Experiments

We also studied acceptor sites on the window $[-20, +3]$ around the splice site and used structured and unstructured models of order up to 3. The results are shown in Table 2.6.

In case of acceptor sites, the structured models do not bring any advantage compared to high order PWMs. Indeed, compared to dependencies between neighboring nucleotides, non-adjacent dependencies seem to be weak (see Figure 2.12). Large changes in specificity are associated with an increase of order up to two, with only a small advantage brought by the third-order model compared to the second-order model. The results show PWM-3 as the best model to use for acceptor sites.

2.3.6 Signal Models in Gene Finding

We also evaluated a variety of signal models in the context of gene finding. We used our gene finder ExonHunter described in Chapter 5, but we modified its signal models as follows. We only used start site, stop site, donor, and acceptor site models (removing all other signal models from the HMM). For all experiments, start sites and stop sites were always PWM-1.

Model	Sensitivity level 20%	40%	60%	80%	95%	Reliability score
Random	0.07%	0.07%	0.07%	0.07%	0.07%	N/A
PWM-0	2.7%	1.8%	1.3%	0.8%	0.4%	0.838
PWM-1	4.8%	2.8%	1.9%	1.1%	0.5%	0.943
TREE	4.7%	2.8%	2.0%	1.1%	0.5%	0.944
PWM-2	6.0%	3.5%	**2.4%**	1.3%	**0.6%**	0.962
HOT-2	5.8%	3.7%	2.2%	1.3%	**0.6%**	**0.964**
PWM-3	6.1%	**3.8%**	2.3%	**1.4%**	**0.6%**	**0.964**
HOT-3	**6.3%**	3.5%	2.1%	1.2%	0.5%	0.963

Table 2.6: **Specificity and reliability score of acceptor site models.** The models of acceptor site were trained for window $[-20, +3]$.

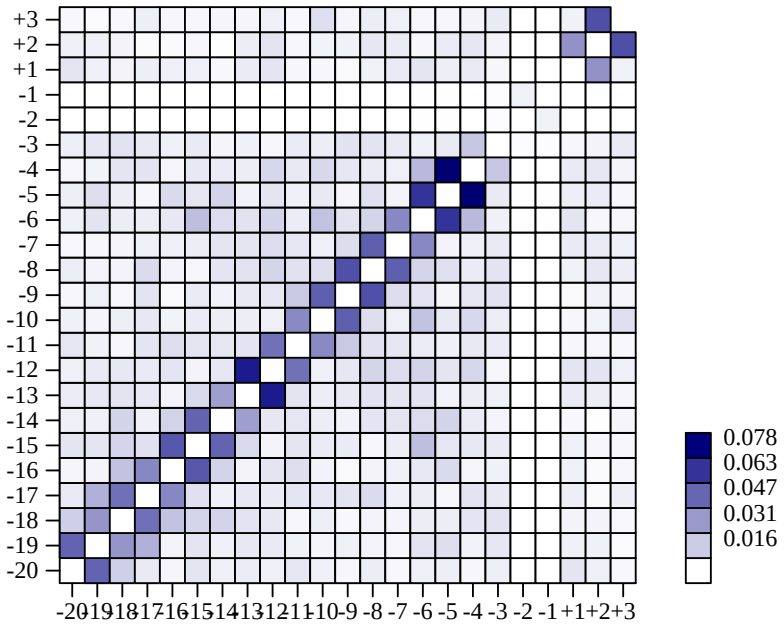

Figure 2.12: **Pairwise dependencies in human acceptor splice site.** Dependencies as observed in the human chromosome 22 training set. The color intensity represents the amount of dependency between the two positions, $H(i) + H(j) - H(i, j)$.

	(Donor model, Acceptor model)		
	(PWM-0,PWM-0)	(PWM-3,PWM-3)	(HOT-3,PWM-3)
Nucleotide Sensitivity	75%	76%	77%
Nucleotide Specificity	77%	77%	77%
Exon Sensitivity	60%	62%	63%
Exon Specificity	62%	62%	63%
Donor Sensitivity	70%	72%	72%
Donor Specificity	73%	72%	72%
Acceptor Sensitivity	68%	70%	70%
Acceptor Specificity	70%	69%	70%
Start Sensitivity	48%	58%	68%
Start Specificity	38%	37%	40%
Stop Sensitivity	48%	51%	63%
Stop Specificity	34%	35%	35%

Table 2.7: **Performance of signal models in gene finding.** Performance on the data set encode-small. The table heading shows combination of donor site and acceptor site model used. In all experiments, start site and stop site are modeled by PWM-1.

In the first experiment, we used PWM-0 for both donor and acceptor sites. In the second experiment, we increased order of both models to PWM-3. Finally, in the third experiment, we changed the donor model from unstructured PWM-3 to structured HOT-3.

All models were trained on the combination of training sets described in Chapter 5 and the repeat advisor was included for decoding (see Chapter 5 for details). Predictions were obtained on the data set encode-small, described in Appendix A. The data set contains 1.5MB of sequence with total of 41 genes with 110 transcripts (2.68 transcripts per gene), and 1228 exons (some overlapping).

We observed a modest improvement in both nucleotide and exon statistics, though the improvement is relatively small between unstructured and structured models. In fact, the statistics for donor and acceptor site sensitivity and specificity are almost identical in all three experiments. However, a very surprising effect of introducing the structured HOT-3 model for donor sites is a large improvement of the model's ability to correctly locate start sites and stop sites, even though the model for start site and stop site is the same first order PWM model in all experiments. This result suggests that in gene finding, there are many factors playing together in non-trivial combinations to obtain good gene structure predictions. The changes in a model of one element may cause surprising changes in predictions of other elements.

2.4 Parallel Work

Barash et al. (2003) suggest using Bayesian networks for the related problem of identification of protein binding sites. To optimize the structure of the model, they use BDeu metrics

(Heckerman et al., 1995), and a heuristic search procedure. They show that the models of binding sites that they learn achieve better false positive rates than PWM models of binding sites, using results of genome-wide yeast localization assays (Lee et al., 2002).

Castelo and Guigó (2004) uses a similar approach for identification of splice sites, with a different heuristic to search through the model parameter space. Our main criticism is that they use a testing set which contains the same number of positive sites and decoys, thus significantly simplifying the problem, and they only show results for 95% sensitivity.

In our work, we suggest using integer programming instead of a heuristic method to optimize the network structure, especially for models with small window size ℓ, where the resulting integer programs can be solved in negligible time. The integer program does not require the use of our entropy-based measure, which maximizes the likelihood of the model. Other common measures, such as KL-divergence or BDeu metrics could be used instead.

Yeo and Burge (2003) show a different generalization of position weight matrices and the maximal dependence decomposition model introduced by Burge (1997). They use the maximum entropy principle, where one seeks the probability distribution that maximizes the Shannon entropy subject to satisfying constraints that are based on marginalized observations from the training set. Different sets of constraints result in different probabilistic distributions. Yeo and Burge (2003) select the constraints by a greedy heuristics.

2.5 Summary

In this chapter, we explored the problem of probabilistic modeling of splice site signals in DNA sequences by modeling non-adjacent dependencies in the signals. We introduced a new class of models, higher order trees (HOTs), which are essentially Bayesian networks with limited in-degree of variables. The limited in-degree is needed to support the trade-off between the amount of data available for training and amount of dependencies captured by the model. If the in-degree were not limited, the models would overfit the training data.

We suggested a new technique, based on integer programming and the maximum likelihood principle, to find the optimal structure of such models. Integer programming can be used to quickly find the optimal solution for small instances of the problem, or a solution close to optimal for large instances of the problem.

Finally, we showed that introducting structured HOT models into modeling of donor splice sites improves accuracy, but has no have any impact when used to acceptor splice sites, as they do not exhibit strong non-adjacent dependencies. Using higher-order and structured models in gene finding increases the accuracy of gene finding. We also made an interesting observation that the changes in donor site do not seem to affect the accuracy of donor site prediction by themselves, but have a large impact on the accuracy in other parts of the model (such as prediction of start sites and stop sites).

Finally, the structure of HOT models is very intuitive. Thus analysis of the structure of learned HOT models, applied to gene finding or other similar problems in bioinformatics (such as location of transcription factor binding sites, or location of promoters) may reveal mechanisms by which proteins bind to the DNA sequence to perform particular functions.

Chapter 3

Length Distributions in HMMs

In this chapter, we examine a tradeoff between model faithfulness and the running time of an HMM's decoding algorithm in connectin with representing the length distributions of elements of biological sequences. The following example demonstrates the issue.

Assume that we model a sequence element (for example an intronic region) with a single state of an HMM. To allow such a region to be longer than 1, we include a self-loop transition, with transition probability p. The probability that the HMM stays in this state for exactly ℓ steps is $(1-p)p^{\ell-1}$. Therefore the length distribution of intronic regions generated by such a model will be geometric.

The length distributions of biological sequence elements are far from geometric. Figure 3.1 shows actual length distributions of various sequence elements necessary for gene finding, and Figure 3.2 on page 70 shows their best approximation by geometric distributions. If each such element is represented by a single state, this problem is traditionally solved by introducing *generalized HMMs* (Rabiner, 1989). In generalized HMMs, the self-loop transitions are replaced by explicit state durations. Upon entering a state, the generative model first chooses the number of symbols that will be generated in this state. The length distribution is defined explicitly for each state. This requires a change in the Viterbi algorithm, increasing the running time from $O(nm^2)$ to $O(n^2m^2)$, where n is the length of the sequence and m is the number of the states of the HMM. A running time that is quadratic in the length of the sequence is, however, impractical for the analysis of DNA sequences several megabases long.

We cannot offer a better solution for the general class of length distributions. However, we observe that the length distributions commonly occurring in gene finding are all well approximated by the class of distributions with *geometric tails.* In this class, the accuracy of the fit of the distribution to the training data is determined by a parameter t, balancing the accuracy and running time of sequence annotation.

In particular, we offer two modifications of the Viterbi algorithm to analyze such models: one with running time $O(ntm + nm^2)$ for the cases when t is small, and one with running time $O(n\sqrt{t}m^2)$ for the cases when t is large, which achieve more coarse approximation of the distribution. Our solution provides a range of modeling options to balance the faithfulness of the model with the decoding time.

Durbin et al. (1998) proposed a different solution, based on replacing a single state with

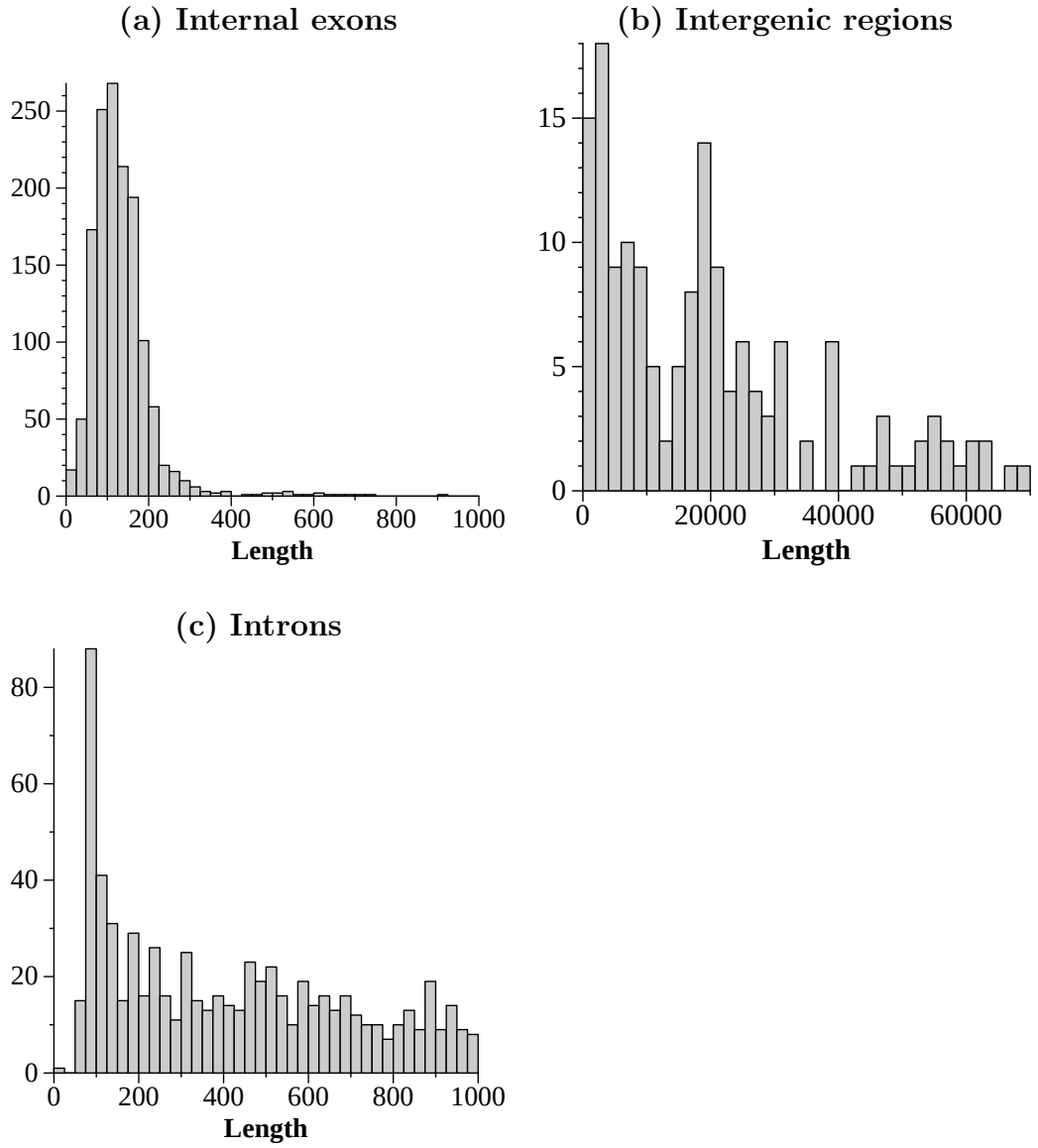

Figure 3.1: **Length distributions in Human chromosome 22.** Graphs were obtained from the data set chr22-training prepared from sequence annotation based on RefSeq collection (Pruitt et al., 2005) downloaded from UCSC genome browser (Karolchik et al., 2003) in July 2005. More details on the data set can be found in Appendix A

a group of states organized in a specific topology, yielding a non-geometric distribution. However, if such a model is decoded by the Viterbi algorithm, we show that the decoding result is equivalent to that which would arise from much simpler HMM with geometric length distributions only.

Finally, it is not always desirable to model a sequence element by a single state. For example, to model exonic sequences, we require a 3-periodic Markov chain, which is modeled by 3 states. In such a case, we would like to set the length distribution for a group of states representing the sequence element, instead of setting it independently for each state in the group. We solve this problem by introducing *boxed HMMs*—a simple hierarchical extension of HMMs, and giving an appropriate decoding algorithm for boxed HMMs.

Most of the results presented in this chapter were published in Brejová and Vinař (2002) (Sections 3.2 and 3.5) and Brejová et al. (2005) (Section 3.3).

3.1 Generalized HMMs with Explicit State Duration

To introduce the notion of *generalized HMMs*, we need to modify the definitions of a hidden Markov model, state path, and the probability of generating sequence/state path pairs as follows:

Definition 23 (Generalized HMMs). *A generalized hidden Markov model is a tuple* $(V, \Sigma, o, e, a, \sigma, \delta)$*, where the set of states* V*, the output alphabet* Σ*, the order* o *of every state, the emission probability table* e*, the transition probability table* a*, and the start state* σ *are defined as for an ordinary hidden Markov model (Definition 1 on page 4).*

The function δ_π*, for every state* π*, is a function from* $\{1, 2, \dots\}$ *to* $[0, 1]$ *defining the probability distribution of the length of time (duration) the model spends in state* π*.*

In ordinary HMMs, the generative process always emits in the current state a single symbol, followed by a transition to (potentially the same) state according to the transition probabilities. We now modify this process: upon entering state π, we first choose the *duration* d of the state from the length probability distribution δ_π. Then d symbols are generated according to the state's emission probabilities e_π. Only after d symbols are generated in π, the model follows a transition to a new state (which may be any state in the HMM including π), according to the transition probabilities a_π.

Ordinary HMMs can be viewed as a special case of generalized HMMs. We can simply set the duration of each of the states in the original HMM to one. On the other hand, the following slightly more complicated construction will help us to illustrate our modifications to generalized HMMs. In this construction, we will remove self-loops from the model as follows. States without self-loops will have length distribution $\delta_\pi(1) = 1$, $\delta_\pi(\ell) = 0$ for all $\ell > 1$. The length distribution of states with self-loops, where $a_{\pi,\pi} = p$, will be geometric, i.e. $\delta_\pi(\ell) = p^{\ell-1}(1-p)$. We also have to remove self-loops from the transition function, and renormalize the transition probabilities.

One problem with the above definition is that the length distribution is defined as a function over all positive integers. Thus, the description length of such a generative model

is potentially infinite. This is usually solved by either imposing an upper bound on the duration of a state (often used in speech recognition applications), or by defining the length distribution as a function with few parameters (as in the example above).

These changes to the generative process are captured in the following definition of a state path and the joint probability of generating a sequence and a state path.

Definition 24 (State path in a generalized HMM). *A state path for a generalized HMM is a sequence $\pi = \pi_1^{b_1:e_1}\pi_2^{b_2:e_2}\dots\pi_k^{b_k:e_k}$, where π_i is a state of the generalized HMM, π_1 is the start state, $b_1 = 1$, $e_i \geq b_i$ (for all $1 \leq i \leq k$), and $b_{i+1} = e_i + 1$ (for all $1 \leq i < k$).*

In this state path, the model emits symbols s_{b_i} to s_{e_i} (inclusively) in state π_i, and then transits to state π_{i+1}. More precisely, the probability of generating the sequence of symbols $s = s_1 \dots s_n$ by the state path π, where $e_k = n$, is defined as follows:

$$\begin{aligned}\Pr(\pi, s) &= \delta_{\pi_1}(e_1 - b_1 + 1) \cdot emit(\pi_1, b_1, e_1) \cdot \\ &\quad \prod_{i=2}^{k} a_{\pi_{i-1},\pi_i} \cdot \delta_{\pi_i}(e_i - b_i + 1) \cdot emit(\pi_i, b_i, e_i), \end{aligned} \tag{3.1}$$

where $emit(\pi, b, e) = \prod_{j=b}^{e} e_\pi(s_j)$ is the emission probability of the sequence $s_b \dots s_e$ in state π.

Note that emission probabilities may be of order higher than zero. Treatment of higher order emissions is automatically included in our definitions and algorithms by use of the shortened notation for emission probabilities as introduced on page 4.

To compute the most probable state path that generates a particular sequence of symbols, a modification is required to the Viterbi algorithm. In each step of the dynamic programming, in addition to examining all potential last transitions, we also have to consider all possible durations of the last state. If $P(i, v)$ is the probability of the most probable path generating the first i symbols of the sequence s and finishing in state v, assuming that in the next step the model will transit out of state v or finish, then the dynamic programming is characterized by the following recurrence:

$$P(i, v) = \max_{1 \leq j \leq i} [emit(v, j, i) \cdot \delta_v(i - j + 1) \cdot \max_{u \in V} P(j - 1, u) \cdot a(u, v)] \tag{3.2}$$

The base cases are the same as for the regular Viterbi algorithm. The straightforward implementation of this dynamic programming gives running time of $O(n^3m^2)$, where n is the length of the sequence and m is the number of the states, since the computation of $emit(v, j, i)$ takes $O(n)$ time in the worst case. However, it is possible to reduce the running time to $O(n^2m^2)$ using an $O(nm)$ pre-computation time, after which it is possible to compute $emit(v, j, i)$ in constant time for any i and j. The method we show below is analogous to the one given by Mitchell et al. (1995).

First, assume that all emission probabilities are non-zero. Then for each state v and each i, we pre-compute $L(v, i) = emit(v, 1, i)$. This can be computed in linear time, since $L(v, i) = L(v, i - 1) \cdot e_v(s_i)$, where $L(v, 0) = 1$. To compute $emit(v, i, j)$, it is enough to use a single division operation: $emit(v, i, j) = L(j)/L(i - 1)$.

If we allow zero emission probabilities, at some point i the value of $L(v,i)$ may become zero, and in such case all values of $L(v,j)$ for $j > i$ will also be zero. Then it is no longer possible to compute $emit(v,i,j)$ as outlined above. To overcome this problem, whenever the value of $L(v,i)$ becomes zero, we replace it with 1. We will also maintain an index $K(v,i)$, which is the last index when such a replacement occurred (i.e., $K(v,i) = i$ if $i = 0$, or $e_v(s_i) = 0$, and $K(v,i) = K(v,i-1)$ otherwise). To compute $emit(v,i,j)$, we distinguish two cases:

- if $K(v,j) < i$, then $emit(v,i,j) = L(j)/L(i-1)$,
- otherwise, $emit(v,i,j) = 0$.

In either case, we can now compute the function $emit(v,i,j)$ in constant time, and the running time of the Viterbi algorithm for generalized HMMs is $O(n^2m^2)$.

3.2 Distributions with Geometric Tails

Generalized hidden Markov models can be used to accurately model any length distribution. However, the running time $O(n^2m^2)$ of the modified Viterbi algorithm for generalized HMMs is not practical for the analysis of DNA sequences that can be several megabases long. On the other hand, in ordinary hidden Markov models we can employ the faster version of the Viterbi algorithm with running time $O(nm^2)$, but in this case, the length distributions are forced to be approximated by geometric distributions. An example of such an approximation can be seen in Figure 3.2.

A linear-time solution is needed for the analysis of long DNA sequences, yet using geometric length distributions will decrease the performance of the model. One way of decreasing the running time is to restrict the family of length distributions allowed in the generalized states. We propose geometric-tail distributions for this purpose. They can model length distributions of many elements of biological sequences more accurately, while not incurring the decoding slowness characteristic for generalized HMMs. A geometric-tail length distribution joins two distributions: the first part is an arbitrary length distribution, and the second part is a geometric tail (see the example in Figure 3.3).

Definition 25 (Geometric-tail distribution). *A* geometric-tail distribution δ *of order* t *is defined by a* t*-tuple* $(\delta_1, \delta_2, \ldots, \delta_{t-1}, q)$*, where* $0 \leq q < 1$ *is the coefficient of the geometric tail starting at length* t *of the distribution. We define* δ_t *so that* $\left(\sum_{i=1}^{t-1} \delta_i\right) + \frac{\delta_t}{1-q} = 1$*. In this distribution,*

$$\delta(x) = Pr[X = x] = \begin{cases} \delta_x, & \text{if } x \in \{1, \ldots, t\}, \\ \delta_t \cdot q^{x-t}, & \text{if } x > t. \end{cases} \tag{3.3}$$

The geometric tail is motivated by the availability of an efficient decoding algorithm for such distributions, as we will see in Section 3.2.2. The running time of the decoding

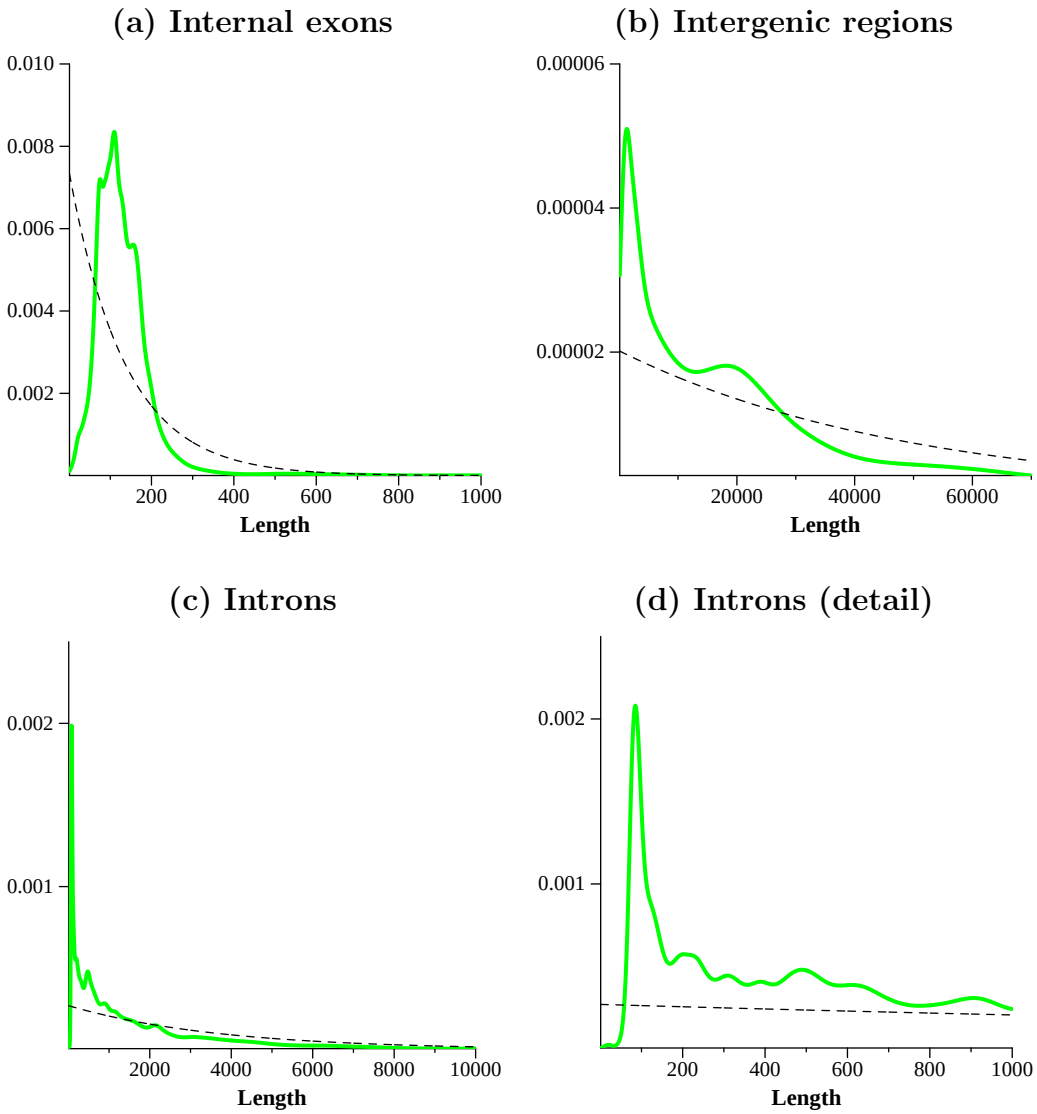

Figure 3.2: **Approximation of length distributions by geometric distributions.** Solid line represents the smoothed distribution of sequence elements on chromosome 22. Dashed line represents the maximum likelihood approximation by geometric distribution.

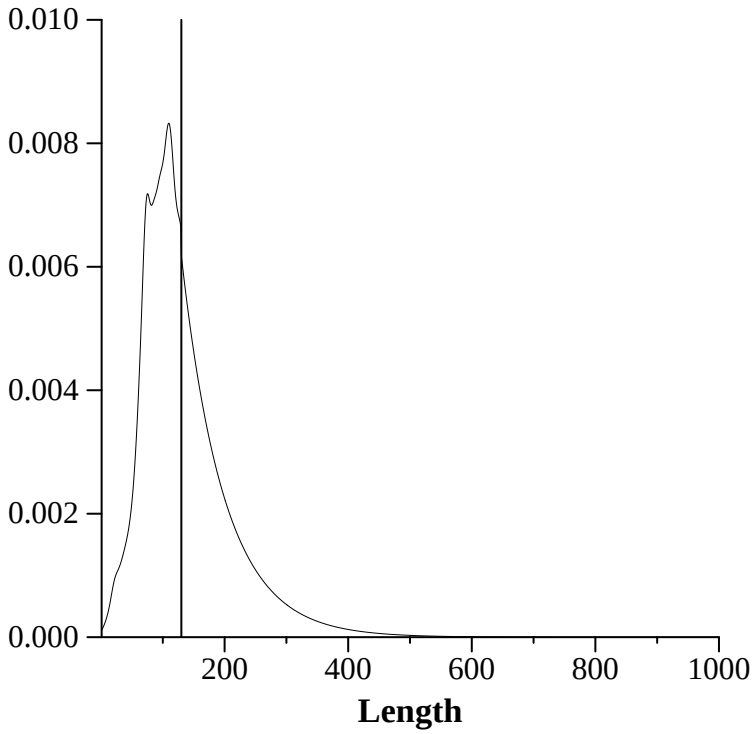

Figure 3.3: **Example of a geometric-tail distribution.** A geometric-tail distribution is composed of two parts: the first part is an arbitrary length distribution, the second part is geometric. The vertical line represents the boundary of the two parts, at $t = 130$.

algorithm is $O(nmt + nm^2)$, where t is the start of geometric tail, n is the length of the sequence, and m is the number of states in the HMM. This is practical even for long DNA sequence, as long as we use distributions with moderate values of t.

Moreover, this variety of distributions (including the tail section) is well suited for modeling the lengths of many biological sequence elements that have most of the probability mass concentrated in relatively short lengths, but where much longer sequences do sporadically occur. The geometric tail assigns a non-zero probability to a region of any length, so we do not exclude any possible feature by setting the upper bound too small. Also, the geometric tail has only one parameter, q, which measures how quickly the tail decays. This is advantageous, since the probability distribution with only a single parameter is much easier to estimate from sparse data without risk of overfitting the training data. On the other hand, the densely populated core region of the distribution, where the δ_i values are explicitly enumerated, can be estimated simply by the smoothed empirical distribution of the training data. If appropriate, we can also stipulate a lower bound on the length of an element by assigning zero probability to all smaller lengths.

3.2.1 Maximum Likelihood Training

Assume we have a training sample of lengths in which the length i occurs f_i times. In this section, we derive a maximum likelihood training method for geometric-tail distributions, with a fixed start of geometric tail t. The tail t is then set manually, based on the properties of a particular distribution, and the running time requirements for the decoding algorithm. In general, by increasing the value of t, we achieve better approximation.

We want to use the training sample to estimate the parameters $(\delta_1, \ldots, \delta_{t-1})$ and q of the geometric-tail distribution δ of order t. The parameters of the head of the distribution $\delta_1, \ldots, \delta_{t-1}$ and the parameters of the geometric tail δ_t and q can be optimized separately, as long as we fix the probability mass $p = \sum_{i=1}^{t-1} \delta_i$.

Lemma 26 (Maximizing likelihood of the distribution head). *The likelihood of the sample $f_1, \ldots, f_{t-1}$ corresponding to the head of the geometric-tail distribution, subject to $\sum_{i=1}^{t-1} \delta_i = p$, is maximized by setting $\delta_i = p \cdot f_i / M$, where $M = \sum_{i=1}^{t-1} f_i$.*

Proof. The logarithm of the likelihood of the training data set for a given set of parameters of the distribution head is:

$$\log \Pr(f_1, \ldots, f_{t-1} \mid \delta_1, \ldots, \delta_{t-1}) = \log \prod_{i=1}^{t-1} \delta_i^{f_i} = \sum_{i=1}^{t-1} f_i \log \delta_i \tag{3.4}$$

Set $\delta_i = p \cdot f_i / M$, and let us consider any other set of parameters $\delta'_1, \ldots, \delta'_{t-1}$. Then, since

$\log x \leq x - 1$ and p/M is a positive constant, we have:

$$\begin{aligned}
\sum_{i=1}^{t-1} \delta_i \log \delta'_i/\delta_i &\leq \sum_{i=1}^{t-1} \delta_i(\delta'_i/\delta_i - 1) = \sum_{i=1}^{t-1} \delta'_i - \sum_{i=1}^{t-1} \delta_i = 0 \\
\sum_{i=1}^{t-1} f_i \log \delta'_i/\delta_i &\leq 0 \\
\sum_{i=1}^{t-1} f_i \log \delta'_i &\leq \sum_{i=1}^{t-1} f_i \log \delta_i
\end{aligned}$$

Thus our set of parameters $\delta_1, \ldots, \delta_{t-1}$ maximizes the likelihood of the training set, which is what we wanted to prove. □

Lemma 27 (Maximizing likelihood of the distribution tail). *The likelihood of the sample $f_t, f_{t+1}, \ldots$ corresponding to the geometric tail of the geometric-tail distribution, subject to $\sum_{i=t}^{\infty} \delta_i = (1-p)$, is maximized by setting:*

$$q = \frac{\sum_{i=t}^{\infty} (i-t) f_i}{\sum_{i=t}^{\infty} (i-t+1) f_i} \tag{3.5}$$

$$\delta_t = (1-p)(1-q) \tag{3.6}$$

Proof. Since the probability of length $i \geq t$ in the geometric tail is $\delta_t \cdot q^{i-t}$, to normalize the distribution to overall mass of $(1-p)$, we have to set $\delta_t = (1-p)(1-q)$. Thus the logarithm of the likelihood of the training set in the distribution tail is:

$$\begin{aligned}
\log \Pr(f_t, f_{t+1}, \ldots \mid \delta_t, q) &= \sum_{i=t}^{\infty} \log((1-p)(1-q)q^{i-t})^{f_i} \\
&= \sum_{i=t}^{\infty} f_i \cdot (\log(1-p) + \log(1-q) + (i-t)\log q) \qquad (3.7)
\end{aligned}$$

Since $\log(1-p)$ is constant, we have to maximize

$$\sum_{i=t}^{\infty} f_i \log(1-q) + f_i \cdot (i-t) \log q \tag{3.8}$$

This function is strictly concave, and by taking a derivative with respect to q, we find that it is maximized when

$$q = \frac{\sum_{i=t}^{\infty} (i-t) f_i}{\sum_{i=t}^{\infty} (i-t+1) f_i}, \tag{3.9}$$

which is what we wanted to prove. □

Now it remains only to assign a proper mass to both head and tail sections of the geometric-tail distribution.

Theorem 28 (Training geometric-tail distributions). *To maximize the likelihood of the sample $f_1, f_2, \ldots$ by a geometric-tail distribution with tail starting at position t, we set the probability of head $p = M/M'$, where M is the number of samples of length less than t, and M' is the number of all samples.*

The maximum likelihood can be then maximized separately in the head section by using Lemma 26 and in the tail section by using Lemma 27.

Proof. For a given value of p, we can use Lemma 26 and Lemma 27 to estimate the parameter of head an tail of the geometric-tail distribution. Thus, the logarithm of likelihood of the training set in the geometric-tail distribution is:

$$\log \Pr(f_1, \ldots \mid p) = \sum_{i=1}^{t-1} f_i \cdot \log\left(p \cdot \frac{f_i}{M}\right) + \sum_{i=t}^{\infty} f_i \cdot [\log(1-p)(1-q) + (i-t)\log q] \quad (3.10)$$

Note that according to the result of Lemma 27, the value of q does not depend on value of p. Therefore, the values of p and q can be optimized independently, where the value of q is determined by Lemma 27. Therefore, we can remove all the elements that do not depend on p, and our task is to maximize the following formula:

$$\sum_{i=1}^{t-1} f_i \log p + \sum_{i=t}^{\infty} f_i \log(1-p) = M \log p + (M' - M)\log(1-p). \quad (3.11)$$

Since this function is strictly concave, it is maximized for $p = M'/M$, which is what we wanted to prove. □

In practice, we used a smoothing procedure introduced in Genscan (Burge, 1997). In this method, every point k observed in the training data set n_k times is replaced by a normal distribution with mean $\mu = k$ and standard deviation $\sigma^2 = 2c/n_k$, rescaled to a mass of n_k/N, where N is the total number of observations. The resulting smooth distribution is obtained as a sum of these normal distributions. Constant c is selected by hand so that the resulting curve reasonably describes the underlying data. It is also desirable to employ additional smoothing before the start of geometric tail; otherwise a large gap may occur between values of δ_{t-1} and δ_t, introducing bias not observed in the original data set.

The resulting geometric-tail length distributions approximating lengths of exons, introns, and intergenic regions are depicted in Figure 3.4. Note that additional smoothing was required to reduce the gap between values of δ_{t-1} and δ_t, as suggested above. The geometric-tail distributions approximate lengths of exons well even for small values of t. Introns are harder to approximate, since the tail of the intron distribution has a slower than geometric decay. However, a reasonable approximation can be achieved with moderate values of t. For

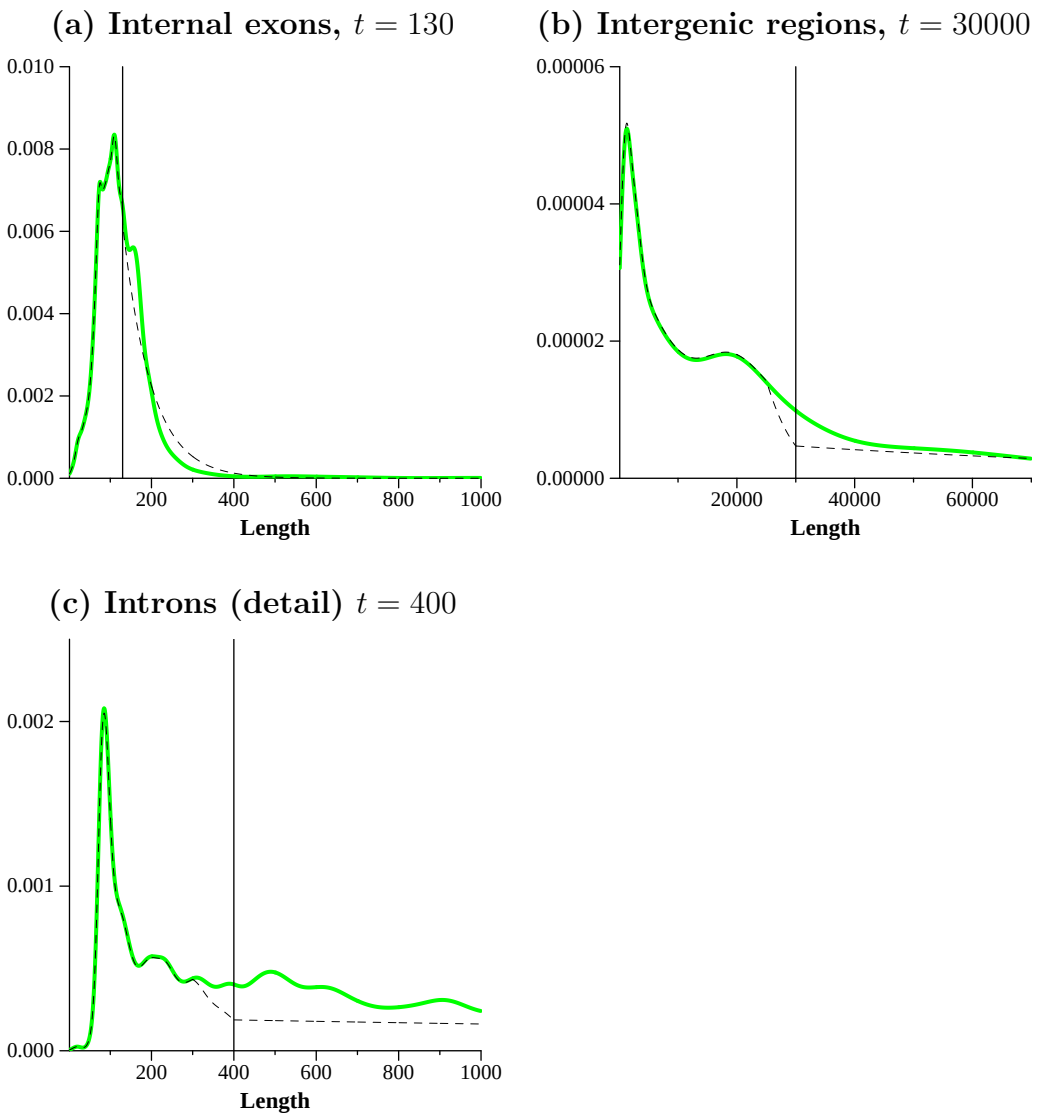

Figure 3.4: **Approximation by geometric-tail distributions.** Solid line represents smoothed distribution of sequence elements on chromosome 22. Dashed line shows approximation by geometric-tail distribution. Vertical line indicates start of geometric tail. Both internal exons and introns can be well approximated with small values of t. For intergenic regions, large value of t is required. We applied additional smoothing around the joining of the head and tail of the distribution to reduce the size of the gap between δ_{t-1} and δ_t.

intergenic regions, much higher values of t are required. However, we will show an alternative approach to decoding geometric-tail length distributions for large values of t in Section 3.3.

We did not design any automated method for choosing good values of t. As we will see in the following section, the running time of the decoding algorithm increases with the value of t. On the other hand, higher values of t often help to increase faithfulness of the model, as demonstrated in Section 3.2.3. For our applications, we pick the values of t manually to achieve a good approximation of the real distribution in a feasible running time. In practice, even small values of t often help to improve over geometric length distributions.

3.2.2 Decoding HMMs with Geometric-Tail Lengths

Generalized HMMs allow any distribution with finite description. Therefore, we can easily use geometric-tail distributions with generalized HMMs. Such models may even demonstrate improved accuracy due to the fact that the tail of the distribution (where training data is usually sparse) is now parametrized with only a single parameter, which may reduce the chance of overfitting.

However, our main goal in introducing geometric-tail distributions is to reduce the decoding time, so that they can be used in the analysis of long biological sequences. To achieve this goal, we need to modify the Viterbi algorithm.

Consider a state v with geometric-tail distribution, where the start of the geometric tail is at the length t_v and the geometric coefficient of the tail is q_v. Recall that in the generalized Viterbi algorithm, we define $P(i, v)$ as the probability of the most probable path generating the first i symbols of sequence s and finishing in state v. To compute $P(i, v)$ using recurrence 3.2, we had to consider all possible durations of state v explicitly.

For geometric-tail distributions, we can reduce the running time by distinguishing between two cases: durations less than or equal to t_v, and durations longer than t_v. Let $Q(i, v)$ be the probability of the most probable path generating the first i symbols of the sequence, and spending at least last t_v steps in state v.

To compute the value of $Q(i, v)$, we consider two cases. Either the i-th character extends the duration of the state v, which was already at least t_v, or generating the i-th character brings the duration of state v to exactly t_v steps. In the first case, $Q(i, v)$ can be computed by multiplying $Q(i - 1, v)$ by the geometric coefficient q_v and the corresponding emission probability, as in the case of self-loops in the ordinary hidden Markov models. In the second case, the probability $Q(i, v)$ is computed from $P(i - t_v, u)$ for some state u, multiplied by the appropriate transition probability to state v and the emission probability for the last t_v characters.

To compute the value $P(i, v)$, we only need to check the $t_v - 1$ previously computed values of P (for when the duration of state v is less than t), and the value of $Q(i, v)$ (which covers all other lengths). This is more precisely expressed in the following recurrence defining a dynamic programming algorithm:

$$P(i,v) = \max \begin{cases} Q(i,v), & \text{(duration at least } t_v\text{)} \\ \max\limits_{1 \le k \le t_v} [emit(v, i-k+1, i) \cdot \delta_v(k) \cdot \max\limits_{u \in V} P(i-k,u) \cdot a(u,v)] & \\ \qquad \text{(duration less than } t_v\text{)} & \end{cases} \tag{3.12}$$

$$Q(i,v) = \max \begin{cases} Q(i-1,v) \cdot q_v \cdot e_v(i) & \text{(duration more than } t_v\text{)} \\ emit(v, i-t_v+1, i) \cdot \delta_v(t_v) \cdot \max\limits_{u \in V} P(i-t_v,u) \cdot a(u,v) & \\ \qquad \text{(duration exactly } t_v\text{)} & \end{cases} \tag{3.13}$$

A straightforward dynamic programming algorithm implemented based on this recurrence would take $O(ntm^2)$ time, where t is the average of the values of t_u over all states u, where t_u equals to one for ordinary states emitting only a single character. This can be further reduced by pre-computing the value of $\max_{u \in V} P(i,u) \cdot a(u,v)$ whenever the computation of the i-th column of the dynamic programming matrix has been completed. This pre-computation costs $O(nm^2)$, and reduces the overall running time to $O(nmt + nm^2)$.

This algorithm is practical even for the analysis of long sequences, provided that small values of t give reasonable approximation of the length distributions of modeled elements. As is illustrated in Figure 3.4, this is true for exons and introns. However the required value of t is too large for intergenic regions. We will address this problem in the next section.

Stanke and Waack (2003) observed that the same effect as geometric-tail distributions can be achieved by replacing a generalized state with a group of three generalized states with upper bound t on the duration, as depicted in Figure 3.5. This immediately gives a running time of $O(ntm^2)$, using the standard algorithm from Section 3.1. However, this approach increases the conceptual complexity of the model. In Section 3.5, we introduce further extensions to this basic framework that allow us to explicitly model length distributions not only of a single state, but also of sequence elements represented by a group of states. Such extensions are not feasible with the approach of Stanke and Waack (2003).

3.2.3 Generalization Properties

In Figure 3.1, we have seen examples of the length distributions needed for the sequence elements in gene finding. These distributions can be approximated well with geometric-tail distributions with small values of t, as shown in Figure 3.4.

The parameters of a geometric-tail distributions are estimated by optimizing the fit to the training data. However, even if the resulting distribution fits the training data well, it does not mean that this would generalize to the testing data. Figure 3.6 shows the log likelihood of testing data per data sample for increasing values of t with the optimal geometric tail distribution fit to the training data. We used the training set chr22-training and the testing set chr22-testing described in Appendix A.

For comparison, we have also computed the likelihood for the "ideal fit" distribution ob-

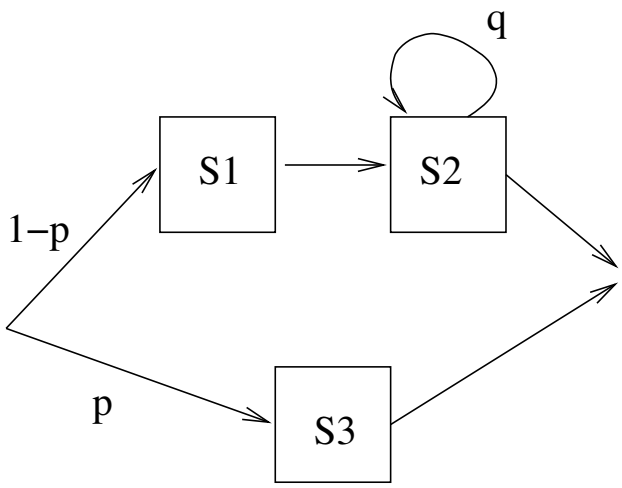

Figure 3.5: **Alternative implementation of geometric-tail distributions.** For a geometric-tail distribution of order t with parameters $(\delta_1, \ldots, \delta_{t-1}, q)$, state $S3$ represents durations shorter than t, and the combination of states $S1$ and $S2$ represents durations of at least t. State $S3$ is a generalized state with maximum duration $t-1$ and the duration probabilities $(\delta_1, \ldots, \delta_{t-1})$. State $S1$ is a generalized state with fixed duration $t-1$. State $S2$ is a regular state with self-loop of probability q. The probability of entering state $S3$ representing short sequences is $p = \sum_{i=1}^{t-1} \delta_i$; otherwise the model emits a long sequence in states $S1$ and $S2$.

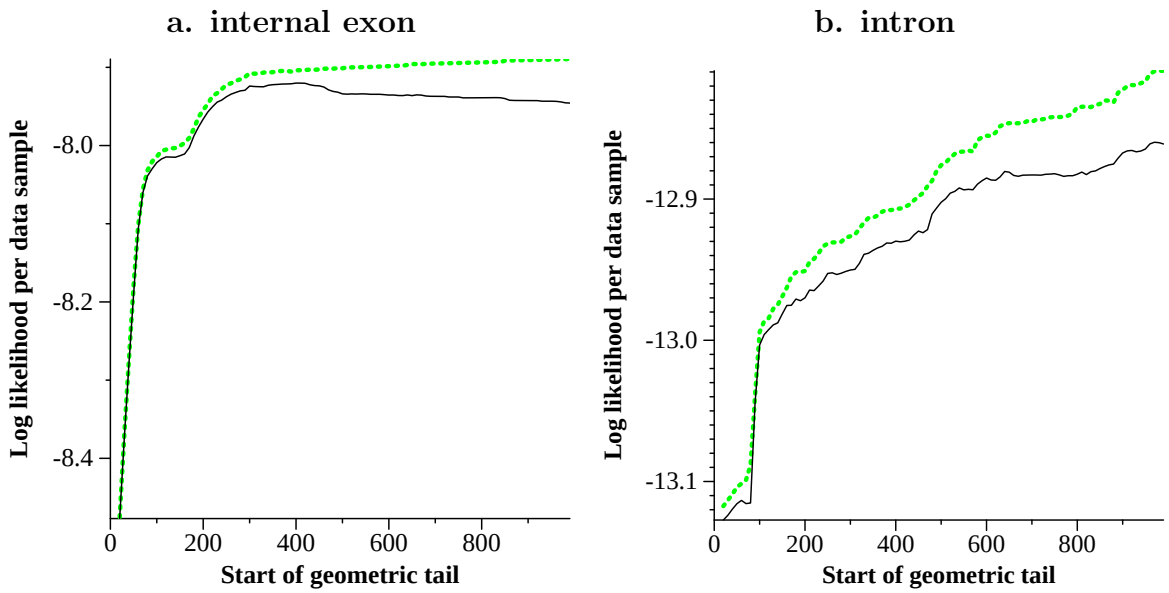

Figure 3.6: **Generalization capacity of geometric-tail distributions.** The $\log_2$ likelihood per data sample increases with increasing values of the parameter t. Geometric distribution corresponds to $t = 1$ with log likelihood well below the bottom of the y-axis scale. Solid line corresponds to the training set chr22-training and the testing set chr22-testing. Data set chr22-testing was used as both training and testing set to produce dotted line.

tained by using the same data set for both training and testing. This is the "best case" upper bound, estimating how well the algorithm could do if the testing data exactly corresponded to the training data for a given family of length distributions.

For both exon and intron lengths, the likelihood of the testing data grows rapidly, even for small values of t. A geometric distribution corresponds to the value $t = 1$. We can see that even small values of t give much better results than does a geometric distribution. For internal exons, we achieve the best performance for values of t between 300 and 400. For larger values, the distribution starts overfitting the training data, as demonstrated by the slightly decreased performance for larger values of t. For intron length distribution, the likelihood of testing data grows steadily even for large values of t (several thousands, the data is not shown). In this case we need to balance running time versus model faithfulness.

We see that even small values of t help significantly compared to a geometric distribution ($t = 1$). Moderate values of t allow fast running time, while yielding good results.

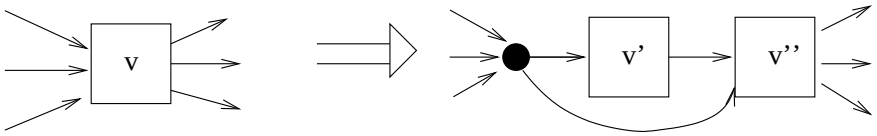

Figure 3.7: **Geometric-tail distribution gadget for large values of t.** The state v' is assigned a geometric-tail distribution, in each step emitting $\sqrt{t}$ symbols. The state v'' is a generalized state with limit on duration $\sqrt{t}$ and uniform length distribution.

3.3 Decoding Geometric-Tail Distributions with Large Values of t

In our discussion so far, we have given a new model for length distributions. We achieved a better running time of decoding algorithm by approximating the length distributions by geometric-tail distributions. In particular, the running time of the decoding algorithm for this solution is $O(nmt + nm^2)$, where the parameter t is the length at which the geometric tail starts. This is practical for distributions which are well approximated with small values of t (such as those in Figure 3.4 (a) and (c)). However, this is not always the case in practice. For example, Figure 3.4(b) shows the lengths of intergenic regions, which need to be modeled in the context of gene finding. This distribution cannot be approximated well with values of t below 10 000.

Let the state v correspond to a sequence element for which a length distribution with a large value of t is required. We replace this state with two states: a special state v' with all of the incoming transitions of state v, and state v'' with all outgoing transitions of state v, and a single transition from v' to v'' (see Figure 3.7). Both states v' and v'' have the same emission probabilities as the state v. The state v' has a geometric-tail length distribution with the tail starting at $t' = \sqrt{t}$, and length distribution of this state corresponds to the lengths for the original state v divided by $\sqrt{t}$. The difference between v' and a regular generalized state is that it emits symbols in multiples of $\sqrt{t}$. When we enter the state, we decide how many blocks of $\sqrt{t}$ symbols the state will emit. The length distribution of the second state v'' is uniform with upper bound of $\sqrt{t}$.[1] A transition is also added to allow skipping state v' altogether. This transition represents all lengths that are shorter than $\sqrt{t}$. Such a gadget effectively replaces the original length distribution by a step-function approximation—see the example in Figure 3.8.

For simplicity, this construction assumes that the values of t are perfect squares. In practice, we choose values of t that can be factored into a product of two reasonably close numbers. All the methods are easily extended to such case.

Extending the algorithm from Section 3.2.2 to handle state v' is straightforward. For these states, we modify the formulas 3.12 and 3.13 as follows:

[1] It is possible to use non-uniform distributions for state v''. However, we did not experiment with this option.

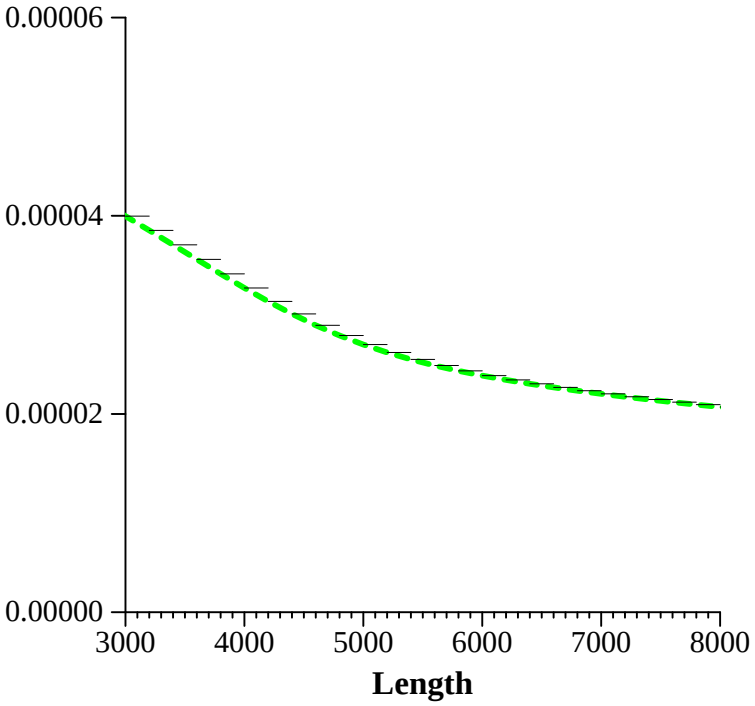

Figure 3.8: **Step-function approximation of length distribution.** Dashed line: detail of length distribution of intergenic regions from Figure 3.4(b). Solid line: step function approximation.

$$P(i,v') = \max \begin{cases} Q(i,v'), & \text{(at least } t' \text{ blocks)} \\ \max\limits_{1\le k<t'} [emit(v', i-kt'+1, i) \cdot \delta_{v'}(k) \cdot \max\limits_{u\in V} P(i-kt', u) \cdot a(u,v')] & \\ \qquad \text{(less than } t' \text{ blocks)} \end{cases} \tag{3.14}$$

$$Q(i,v') = \max \begin{cases} Q(i-t', v') \cdot q_v \cdot emit(v', i-t'+1, i) & \text{(more than } t' \text{ blocks)} \\ emit(v', i-t'^2+1, i) \cdot \delta_{v'}(t') \cdot \max\limits_{u\in V} P(i-t'^2, u) \cdot a(u,v') & \\ \qquad \text{(exactly } t' \text{ blocks)} \end{cases} \tag{3.15}$$

Note that we have introduced stepping by blocks of t' characters into Formula 3.14 and replaced the single emission probability in Formula 3.15 with a block emission of length t'. Since $emit()$ is computed in linear time, the algorithm will run in $O(nmt' + nm^2)$ time, even though it explicitly operates over regions of lengths up to $t = t'^2$.

Thus by replacing the length distribution with the step-function approximation, and by modifying the Viterbi algorithm as stated above, we achieve a running time of $O(nm\sqrt{t} + nm^2)$. This running time is practical even for values of t as large as tens of thousands; a good value of t for human intergenic regions is around 30 000.

3.4 Gadgets of States

An alternative way of avoiding the geometric character of length distributions generated by hidden Markov models is to model a sequence element by multiple states instead of a single state. Durbin et al. (1998) (recently also re-examined by Johnson (2005)) discuss several ways to model non-geometric length distributions by replacing a single state with a group of states that share the same set of emission probabilities. Transitions are added inside this group so that the probability of staying within the group for ℓ steps is close to the probability that the modeled feature has length ℓ. In this section we explore this technique and show that its usability is limited when the Viterbi algorithm is used for decoding.[2]

3.4.1 Phase-type Distributions

Consider an example gadget in Figure 3.9. Such a gadget can represent a single sequence element within a larger HMM. The left-most transition is an entry point to such a sub-model, and the right-most transition is the exit. If the gadget consists of n states, the probability of generating a string of length $\ell > n$ is

$$f(\ell) = \binom{\ell-1}{n-1} p^{\ell-n}(1-p)^n, \tag{3.16}$$

[2]To the best of our knowledge, this approach has not been used in applications of hidden Markov models in bioinformatics.

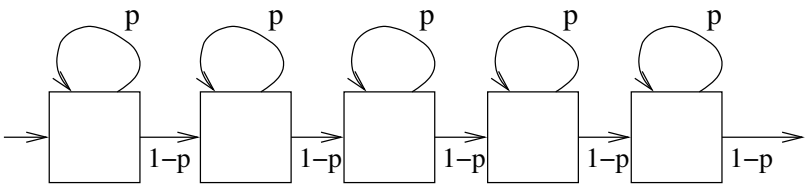

Figure 3.9: Gadget generating non-geometric length distribution in HMM

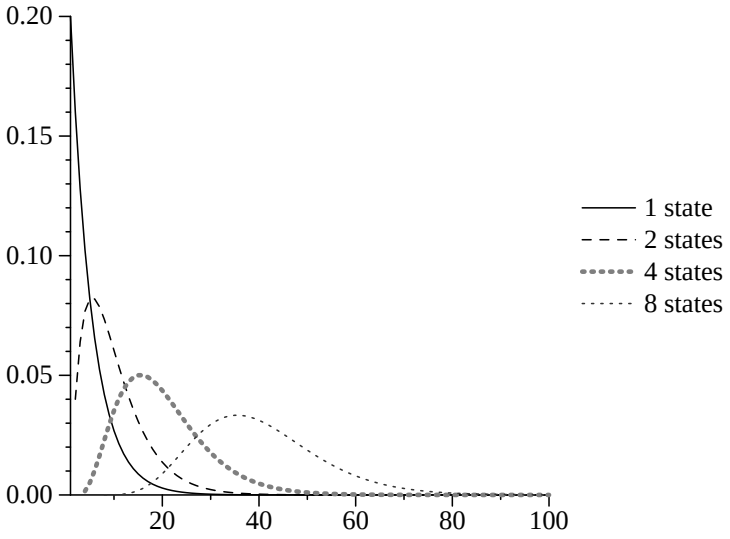

Figure 3.10: **Family of distributions generated by the gadget from Figure 3.9.** We set $p = 0.8$ and varied the number of states in the chain. Many distributions with a single mode can be generated using this approach. The distributions generated by this particular gadget of states are a subclass of the discrete gamma distributions $\Gamma(p\ell, 1)$.

which can be used to model a wide variety of distributions with a single mode (see Figure 3.10). Using such gadgets does not require modifying the algorithms for HMMs, though it will slow down the decoding due to the number of states used to model the length distribution; this slowdown is not significant if the number of such states is small. Of course, there is no reason to limit ourselves to gadgets with the structure in Figure 3.9; one can use any topology of states.

Definition 29 (Gadget of states). A Gadget of states *is a group of states in an HMM with the same emission probabilities. All transitions entering the gadget come from a single state outside the gadget, or they enter a single state inside the gadget. Similarly, all transitions leaving the gadget leave a single state inside the gadget, or enter a single state outside the gadget.* The duration of the gadget *is the number of symbols generated by the HMM in the states within that gadget.*

Definition 30 (Phase-type distributions). *The family of length distributions that can*

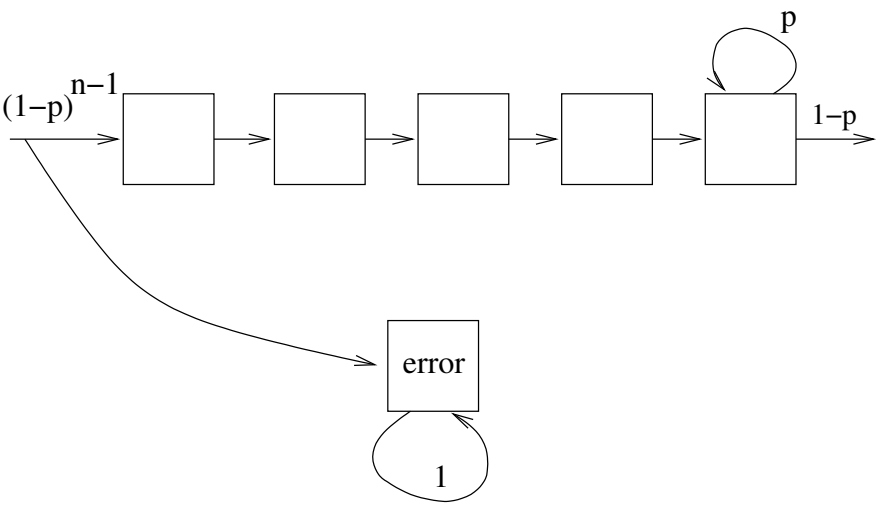

Figure 3.11: **Gadget with geometric length distribution replaces gadget from Figure 3.9.** This gadget can be used to replace the gadget from Figure 3.9, achieving the same results as if the Viterbi algorithm is used for decoding.

be represented by a gadget of states is called phase-type distributions. *The number of states required to represent a particular phase-type distribution is the* order of the distribution.

The family of phase-type distributions, and their graphical characterization by gadgets of states, plays an important role in queuing theory and system theory. A recent overview of the results and open problems can be found in Commault and Mocanu (2003). In fact, any length distribution can be approximated arbitrarily well by a phase-type distribution, even though better approximations often require high order. Maximum likelihood approximation of a given sample by a phase-type distribution has been explored by Asmussen et al. (1996) using the EM algorithm.

This approach of using phase-type distributions seems like an ideal framework for modeling general length distributions in HMMs: depending on the desired running time, we fix the order of the phase-type distribution, and then we find the best approximation of a given sample from training data. Such a generative model indeed generates the sequences of the desired (non-geometric) length distribution, which closer and closer represent the real length distributions.

Note that this modeling technique significantly departs from the principles we used so far. In our previous models, for each pair of the sequence and its annotation, there was always a unique path through the model that generated them. Instead, gadgets of states introduce many paths that represent the same sequence/annotation pair, to achieve the non-geometric character of the length distribution. For example, in Figure 3.9, each individual path of a given length ℓ has the same (small) probability. The probability of generating path of length ℓ is now the product of the number of such paths with the probability of each path. In fact, the non-monotonic non-geometric length distribution is achieved because up to some point

the number of paths grows faster with length ℓ than the probability of each individual path decays with length ℓ.

3.4.2 Gadgets of States and the Viterbi Algorithm

Since the length distribution is now created by multiple paths, each with a small probability, it is not clear whether decoding such model by the Viterbi algorithm for finding only a single *most probable path* is still a good method. In this section, we show that in fact the Viterbi algorithm does not decode such models correctly.

Let us compare the gadget A in Figure 3.9 with the gadget B in Figure 3.11. Gadget B has n states corresponding to the states in gadget A. However, only one of those states includes a self-loop and thus the length distribution generated by this gadget is geometric (except that lengths shorter than n have zero probability). Moreover, there is an additional absorbing *error state* in gadget B to help to balance length probabilities. The probability of entering this state will be $1-(1-p)^{n-1}$. The error state emits only a special *error symbol* that is not part of the original alphabet Σ of the sequences to be decoded. For this reason, no non-zero probability path will include the error state for an input sequence of symbols over Σ^*.

For every length $\ell > n$, there are many paths in gadget A of length ℓ, all of which have the same probability: $emit_\ell \cdot p^{\ell-n}(1-p)^n$, where $emit_\ell$ is the emission probability of the corresponding part of the sequence. In gadget B, there is only one path of length ℓ that does not include the error state, and its probability is also $emit_\ell \cdot p^{\ell-n}(1-p)^n$. Therefore, from the point of view of the Viterbi algorithm, there is *no difference between gadgets A and B:* the maximum probability path in the HMM including model A will be the same with the same probability as if we replace model A with model B.

Therefore, even though gadget A seems to model non-geometric length distributions, the same effect (with respect to the Viterbi decoding) is achieved if we use gadget B instead. However, gadget B generates only geometric distributions. Moreover, a large amount of probability mass is lost to the error state of gadget B.[3] Since this state cannot be used in any state path generating a real sequence, this incorporates a multiplicative penalty of $(1-p)^{n-1}$ for entering gadget B (and therefore gadget A since they are equivalent).

For example, if we use 5 states and probability $p = 0.9$, then 0.9999 of the probability mass will end in the error state, or, entering the sequence element represented by the gadget is now approximately 10000 times less likely, than estimated from the training data. Thus introducing the gadget in Figure 3.9 into an HMM not only does not give any improvement in accuracy of the prediction (since the model is equivalent to one with geometric distribution), but also causes the model to "avoid" the part of the model represented by the gadget.

Thus, finding the most probable path, which is solved by the Viterbi algorithm, is inappropriate for models that use gadgets to model complex length distributions.

[3]The error state in gadget B is necessary to ensure that a path of length ℓ has the same probability in both gadgets A and B. If we renormalized the probabilities in gadget B instead, the probabilities in gadget B would be higher than in gadget A, and the two gadgets would not be equivalent any more.

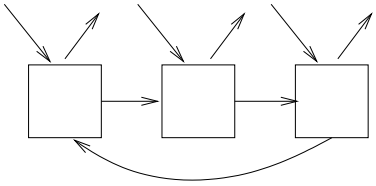

Figure 3.12: **3-periodic Markov chains used for modeling exons.** Each of the states in a periodic Markov chain has different set of emission probabilities. The incoming and outgoing edges represent different connection points with the rest of the model (introns and intergenic regions).

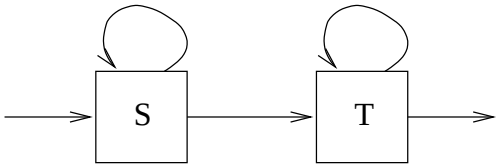

Figure 3.13: **Alternative model of intron.** State S models most of the intronic sequence, with emission probabilities similar to those in intergenic regions. A relatively short, pyrimidine-rich tail of varying length occurs towards the end of introns, with emission probabilities heavily biased towards C/T. This region is modeled by state T. In this model, both S and T have geometric length distributions.

A logical suggestion would be to replace the Viterbi algorithm and the most probable path problem with some other formulation of decoding. We have discussed the disadvantages of *a posteriori* decoding (forward-backward algorithm) in the context of biological applications such as gene finding in Section 1.1.2.2. Another option is to aim for finding the most probable annotation instead of the most probable path—we will defer discussion on this to Chapter 4.

3.5 Length Distributions of Complex Sub-models

One of the disadvantages of the length distribution modeling methods presented in this chapter so far is that they are applicable only to sequence elements modeled by a single state. There are examples where one may want to employ more complex sub-models for particular sequence elements. In gene finding, exons in the gene are best modeled by 3-periodic Markov chains of the type shown in Figure 3.12. The sequence characteristics of intronic regions change dramatically toward their 3' end, thus we may want to employ an intron model like the one in Figure 3.13. At the same time, we would still like to be able to model the length distributions of exons and introns as a whole, not state-by-state.

We introduce boxed HMMs to address this problem. States of a boxed HMM are organized in *boxes*. As in ordinary HMMs, each state emits a sequence according to a Markov

chain of some order. A box is a subset of states, where each state belongs to at most one box (some states may be unboxed). We will denote by B_u the box containing state u. Each box of a boxed HMM is assigned an arbitrary length distribution. Let δ_B be the length distribution associated with box B.

Each state located in a box has two sets of outgoing transitions: *internal* transitions which lead only into states grouped in the same box, and *external* transitions which can lead to any state that is not part of the same box. The probability of an internal transition from state u to state v is denoted by $a'(u, v)$, and the probability of an external transition is denoted $a(u, v)$. For every state u, $\sum_v a(u, v) = 1$ and for every boxed state u, $\sum_{v \in B(u)} a'(u, v) = 1$ and $\sum_{v \notin B(u)} a(u, v) = 1$. Unboxed states have only external transitions (and self-loop transitions are allowed in this case). A simple example of a boxed HMM is given in Figure 3.14.

The boxed HMM as a generative model works as follows. Unboxed states are treated as in ordinary HMMs. When a transition to a box B occurs, the length m is generated according to the length distribution δ_B. For the next $m - 1$ steps, internal transitions are used in, and the HMM will remain in box B for m steps. In the m-th step, an external transition is used to take the model to a new state. Each state inside the box must have at least one external transition.[4]

Boxes of a boxed HMM correspond to states in generalized HMMs. However, each state inside a box can have a different set of emission probabilities, and a different set of external transitions, and similarly, the transitions entering the box specify which state of the box is to be used as the first. This feature is very useful, for example for modeling frame consistency between exons in genes.

It might seem that the actual length distribution is a combination of the explicit distribution δ_B associated with a box B and the implicit geometric-like distribution induced by the internal transitions inside the box. However, for a given length m and start state in box B, the sum of probabilities over all strings of length m generated by box B is 1, and the internal transitions do not constitute any implicit distribution of lengths.

Boxed HMMs, coupled with geometric-tail distributions as introduced above form a convenient modeling tool for many biological applications such as gene finding. We first discuss the modifications to the Viterbi algorithm to allow computation of the most probable path in boxed HMMs for general length distributions. Then we follow a similar path as for generalized HMMs, to improve decoding times for the case of length distributions with geometric tails.

3.5.1 A Viterbi Algorithm for Boxed HMMs

Decoding of a boxed HMM is very similar to decoding of a regular HMM. For a given sequence s, and a state path π, the HMM defines their joint probability $\Pr(s, \pi)$. In the process of decoding, we are looking for the *most probable path*, the state path π that maximizes the

[4] In some cases, this restriction can be relaxed and the algorithms can be easily modified to handle such extensions. For example, if the length distribution is defined so that all the lengths are multiple of k, states that can never be reached by a number of steps that is multiple of k do not need to have external transitions.

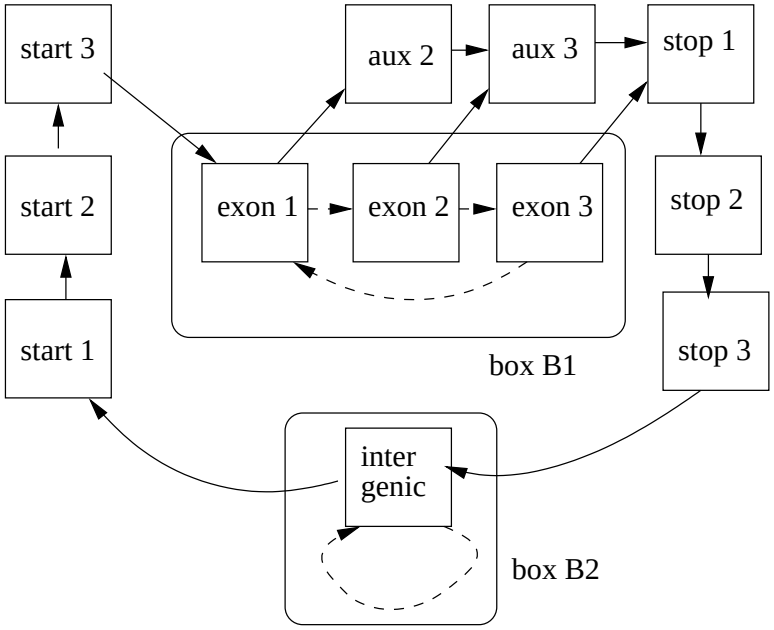

Figure 3.14: **Example of boxed HMM.** A simple boxed HMM model for DNA sequences consisting of single-exon genes on the forward strand only. Internal transitions are marked by dashed lines. All transitions have probability 1. Box B_1 models the exonic region between the start and stop codons, box B_2 models intergenic regions. Both boxes can use non-geometric length distributions. Here a boxed HMM is a more accurate model than the corresponding pure HMM because a boxed HMM can model non-geometric length distributions of exons and intergenic regions.

probability $\Pr(s, \pi)$.

To compute the most probable path in a boxed HMM, we modify Equation 3.1 on page 68, which gives the joint probability of sequence and state path for generalized HMMs. Let $emit(v', v, j, i)$ be defined for all pairs of states v and v' belonging to the same box B_v as the *probability of the most probable path* emitting the sequence $s_j \dots s_i$, using only internal transitions within box B_v, starting in state v', and finishing in state v (as in the case of generalized HMMs, we assume that in the next step the model will either transit out of the box or finish). For the purpose of dynamic programming, instead of considering all possible durations of the last state v, we need to consider the duration within the box B_v associated with state v, as well as all possible states v' where we could have entered the box. Thus Formula 3.1 to compute $P(i, v)$, the probability of the most probable path generating the first i symbols of the sequence, and ending in state v, changes for boxed states as follows (the formula remains unchanged for unboxed states):

$$P(i, v) = \max_{\substack{1 \le j \le i \\ v' \in B_v}} [emit(v', v, j, i) \cdot \delta_{B_v}(i - j + 1) \cdot \max_{u \in V - B_v} P(j - 1, u) \cdot a(u, v')] \tag{3.17}$$

The pre-computation trick which we used to compute values of $emit()$ in constant time, cannot be applied in this scenario. The information about maximum path probabilities for the first i symbols of the sequence and for the first $j-1$ symbols of the sequence do not help us to compute the maximum path probabilities for generating the symbols $s_j \dots s_i$. Instead, we can use the traditional Viterbi algorithm to compute the probability $emit(v', v, j, i)$, as the states within the box and the internal transitions of the box implicitly form an ordinary HMM.

Let Δ be the number of states in the largest box. For a given i and j, the running time of computing $emit(v', v, j, i)$ for all possible combinations of v', v is $O((j-i)m\Delta)$. In a straightforward implementation, this would mean a running time $O(n^3 m^2 \Delta)$. To make the algorithm more efficient, we can reorder the computation as follows:

initialize P
for $i = 1 \dots n$, **for** all states u
| **if** v is a boxed state
| | **for** $\ell = 1 \dots i$, **for** all states $v' \in B_v$
| | | compute $emit(v', v, i - \ell + 1, i)$ (*)
| compute $P(v, i)$
| values computed in (*) can now be discarded

To compute $emit(v', v, j, i)$ in step (*), we can reuse previously computed values of $emit(*, v, j + 1, i)$, using a recurrence analogous to the Viterbi algorithm:

$$emit(v', v, j, i) = \max_{w \in B_v} e_{v'}(j) \cdot a'_{v',w} \cdot emit(w, v, j + 1, i), \tag{3.18}$$

where $e_v(j)$ is the emission probability of symbol x_j in state v. Thus to compute any value of $emit(v', v, j, i)$, we need time $O(\Delta)$ only, since in this order of computation, all values

needed in recurrence 3.18 were already computed in previous steps. The total decoding time is $O(n^2m^2\Delta)$, which is a factor of Δ slower than the decoding of generalized HMMs.

The algorithm can be even more efficient if the topology of states in each box satisfies further restrictions. For example, if the states in each box are organized as in Figure 3.14, Box 1 (a k-periodic Markov chain), we can reduce the running time to $O(n^2m^2)$, because for a fixed i, j, and u, only one of the values $emit(*, u, i, j)$ will be non-zero.

3.5.2 Boxed HMMs with Geometric-Tail Distributions

The modifications introduced to the Viterbi algorithm in Section 3.2.2 can now be applied in a straightforward way to the case of boxed HMMs. Analogously to recurrences 3.12 and 3.13, we obtain:

$$P(i,v) = \max \begin{cases} Q(i,v), & \text{(duration at least } t_v\text{)} \\ \max\limits_{\substack{1\le k\le t_v \\ w\in B_v}} [emit(w,v,i-k+1,i)\cdot\delta_v(k)\cdot\max\limits_{u\in V} P(i-k,u)\cdot a(u,w)] & \\ & \text{(duration less than } t_v\text{)} \end{cases} \tag{3.19}$$

$$Q(i,v) = \max \begin{cases} \max\limits_{w\in B_v} Q(i-1,w)\cdot a'(w,v)\cdot q_v\cdot e_v(i) & \text{(duration more than } t_v\text{)} \\ \max\limits_{w\in B_v} [emit(w,v,i-t_v+1,i)\cdot\delta_v(t_v)\cdot\max\limits_{u\in V} P(i-t_v,u)\cdot a(u,w)] & \\ & \text{(duration exactly } t_v\text{)} \end{cases} \tag{3.20}$$

The order of computation can be reorganized as in the previous section, and thus we can decode the boxed HMMs with geometric-tail length distributions in $O(ntm\Delta^2 + nm^2)$ running time.

On the other hand, the algorithm in Section 3.3 for geometric-tail distribution with large values of t approximated by a step-function cannot be immediately extended to the boxed HMMs. The recurrences 3.14 and 3.15 again extend easily to the case of boxed HMMs:

$$P(i,v') = \max \begin{cases} Q(i,v'), & \text{(at least } t' \text{ blocks)} \\ \max\limits_{\substack{1\le k<t' \\ w\in B_{v'}}} [emit(w,v',i-kt'+1,i)\cdot \delta_{v'}(k)\cdot \max\limits_{u\in V} P(j-1,u)\cdot a(u,w)] & \\ \text{(less than } t' \text{ blocks)} \end{cases} \quad (3.21)$$

$$Q(i,v') = \max \begin{cases} \max\limits_{w,w'\in B_{v'}} Q(i-t',w')\cdot a'(w',w)\cdot q_v \cdot emit(w,v',i-t'+1,i) \\ \text{(more than } t' \text{ blocks)} \\ \max\limits_{w\in B_{v'}} [emit(w,v,i-t'^2+1,i)\cdot \delta_{v'}(t')\cdot \max\limits_{u\in V} P(i-t'^2,u)\cdot a(u,w)] \\ \text{(exactly } t' \text{ blocks)} \end{cases} \quad (3.22)$$

The resulting running time is $O(n\sqrt{t}mC + nmC\Delta^2 + nm^2)$, where C is the running time required for a single *emit*() query. However, in this case the reordering of computation does not help us to compute needed values of the function *emit*() in constant time. Our previous algorithms relied on the fact that whenever we needed to compute value $emit(u,v,i,j)$ (for $i<j$), we also required the computation of values $emit(*,v,i+1,j)$. However, this is not the case for recurrences 3.21 and 3.22.

In the rest of this section, we introduce a different scheme for computing values of *emit*() that does not depend on the order of computation. The scheme requires pre-computation time of $O(nm\Delta^3)$, after which each *emit*() query can be answered in $C = O(\Delta^3\alpha(nm))$ time, where $\alpha(n)$ is the inverse Ackerman function. This function grows very slowly, and for all practical cases can be considered constant (Cormen et al., 2001). This gives a modification of the Viterbi algorithm for boxed HMMs for the case when we are using the step-function approximation of geometric-tail distributions, with running time of $O(n\sqrt{t}mC + nmC\Delta^2 + nm^2\Delta^2) \approx O(n\sqrt{t}m\Delta^3 + nm\Delta^5 + nm^2\Delta^2)$. This is still a reasonable running time if the size of the largest box Δ is quite small.

To solve this problem, we formulate the computation of *emit*() as a graph problem. For a sequence of length n and an HMM with m states, we create a layered graph with $n+1$ layers, and with m vertices in each layer. Let $[i,j]$ denote the jth vertex in the ith layer. There is an edge between vertices $[i,j]$ and $[i+1,j']$ if and only if there is an *internal transition* between vertices j and j' in HMM. The weight of such edge will be $-\log e_i(j)\cdot a'(j,j')$. In such a graph, the negative logarithm of the probability defined by $emit(u,v,i,j)$ corresponds exactly to the shortest distance between vertices $[u,i]$ and $[v,j+1]$.

Definition 31 (Tree decomposition). *A* tree decomposition *of a graph $G=(V,E)$ is a pair (X,T), where $X=\{X_1,\dots X_k\}$ is a family of subsets of V, T is a tree on a set of vertices $\{X_1,\dots,X_k\}$, and the following conditions hold:*

- Edge mapping: *For each edge $(v,w)\in V$, there is a set X_i for which both v and w belong to X_i.*

- Connectivity: *For each vertex v, all sets X_i that contain vertex v form a connected subgraph of tree T.*

The treewidth *of the tree decomposition is* $\max_i(|X_i| - 1)$.

Lemma 32. *The graph constructed to compute values of emit() has a tree decomposition with treewidth 2Δ, where Δ is the size of the largest box of the HMM.*

Proof. For every box B in the HMM, create n sets $X_{B,1}, \ldots, X_{B,n}$, set $X_{B,i}$ containing vertices $[i, v]$ and $[i + 1, v]$ for all $v \in B$. Let the tree T over the set of vertices $X_{B,i}$ for all boxes B and $1 \leq i \leq n$ contain edges $(X_{B,i}, X_{B,i+1})$. This tree clearly satisfies both conditions of the tree decomposition, and the treewidth of such a decomposition is $2\Delta - 1$.[5] □

Chaudhuri and Zaroliagis (2000) presented an indexing scheme where shortest path distance queries in a graph can be answered in running time independent of the size of the graph, provided that the graph has small treewidth. This result is formulated more precisely in the following theorem.

Theorem 33 (Chaudhuri and Zaroliagis (2000)). *Let G be a weighted digraph with n vertices and treewidth at most t with a known tree decomposition. Then after $O(t^3 n)$ preprocessing time, shortest path distance queries in G can be answered in $O(t^3 \alpha(n))$ time, where $\alpha(n)$ is the inverse Ackerman function.*

This immediately yields an indexing scheme for computing *emit*() in time $C = O(\Delta^3 \alpha(mn))$ with $O(\Delta^3 mn)$ preprocessing time.

3.6 Summary and Experiments

In this chapter, we have introduced an efficient method of modeling non-geometric length distributions in hidden Markov models. The solution has two components: an approximation of the real length distribution by a geometric-tail length distribution, and a modification to the Viterbi algorithm that allows efficient decoding in $O(nmt + nm^2)$ time, where t is a parameter of the distribution. Some length distributions require large values of t to be reasonably well approximated. For these cases, we extend the method by replacing the geometric-tail distribution with its step-function approximation, and extend the decoding algorithm to this context with runtime of $O(nm\sqrt{t}+nm^2)$, which is practical even with large values of t. We also extended these methods to more general class of HMMs which we call boxed HMMs. In boxed HMMs, we can specify the length distributions of whole sub-models instead of individual states. This is an advantage if we model a particular sequence element as a group of states instead of a single state. The decoding times remain linear in the length of the sequence, and are practical even for large values of t, if the size of the largest box in the model is small.

[5] Note, that T is, in fact, a path. However, it is not clear whether this difference would bring any additional improvement in algorithm of Chaudhuri and Zaroliagis (2000).

Method	Running time	Applicability in gene finding
Generalized HMMs	$O(n^2m^2)$	not applicable (running time)
Limited duration d	$O(ndm^2)$	not applicable (modeling)
Gadget methods w/ k added states	$O(n(k+m)^2)$	improperly decoded by Viterbi
Genscan	$O(n^2m^2)$ worst case $O(nm^2)$ expected	exons
Augustus	$O(ntm^2)$	exons, introns
Geometric-tail w/ tail start at t	$O(ntm + nm^2)$	exons, introns
Geometric-tail+step-function	$O(n\sqrt{t}m + nm^2)$	intergenic regions

Table 3.1: **Overview of methods for modeling length distributions.** Our methods introduced in the previous sections are highlighted in bold typeface. Different states in an HMM can use different methods for representing length distributions.

Table 3.1 summarizes available methods for modeling length distributions in HMMs and their suitability for application in gene finding. We have described generalized HMMs in detail and their disadvantages for modeling long DNA sequences in Section 3.1.

HMMs with limited state duration are essentially generalized HMMs, where the duration of each state is bounded by a small constant d. This restriction significantly reduces the running time from $O(n^2m^2)$ to $O(ndm^2)$ and the method is often used in speech recognition applications. For biological sequence element lengths, no reasonable upper bound on the state duration exist. Hence, in general this method is not suitable for applications such as gene finding. We also explored the method of replacing a single state with a gadget of states to modify the length distribution. This method works well for analysis of sequences by posterior decoding. However, the Viterbi algorithm produces inferior results with this method, as we demonstrated in Section 3.4.

Two methods specifically tailored to gene finding have emerged to address the problem of length modeling. Genscan (Burge, 1997) uses generalized states to model the length distribution of exons. The use of generalized states leads in general to a quadratic running time in the length of the sequence. However, in the case of exons, it is possible to use the boundaries observed in DNA sequences: no exon can extend beyond the closest in-frame stop codon. Since 3 out of 64 possible codons code for stop codons, in a random sequence one would expect a stop codon to occur approximately every 20 codons or 60 bases. Thus at most positions of the DNA sequence, only few previous positions need to be inspected in the dynamic programming, and the expected running time thus grows linearly with the length of the sequence (Burge, 1997).

Second, in a method very similar to our geometric-tail distributions presented in Brejová and Vinař (2002), Stanke and Waack (2003) introduced a new model for intron lengths, as described in Figure 3.5 on page 78. In their program Augustus, they use this method together with Genscan's method for modeling exon lengths.

Compared to these two methods, our geometric-tail distributions have several advantages. First, the Genscan method does not extend to modeling lengths of introns or intergenic re-

exon lengths	**intron lengths**	**intergenic lengths**	**exon sensitivity**	**exon specificity**
geom.	geom.	geom.	60%	61%
$t = 120$	geom.	geom.	61%	63%
geom.	$t = 150$	geom.	61%	63%
$t = 120$	$t = 150$	geom.	63%	64%
$t = 120$	$t = 150$	$t = 30\,000$	63%	64%

Table 3.2: **Performance of non-geometric length distributions on gene finding in human.** The results for testing set encode-small described in Appendix A.

exon lengths	**intron lengths**	**intergenic lengths**	**exon sensitivity**	**exon specificity**
geom.	geom.	geom.	70%	66%
$t = 250$	geom.	geom.	69%	66%
geom.	$t = 150$	geom.	74%	69%
$t = 250$	$t = 150$	geom.	74%	70%
$t = 250$	$t = 150$	$t = 3600$	74%	70%

Table 3.3: **Performance of non-geometric length distributions on gene finding in fruit fly.** The results for testing set drome-small described in Appendix A.

gions. This is because for these sequence elements there are no natural boundaries in DNA sequences such as in-frame stop codons. Second, geometric-tail distributions and corresponding algorithms can be easily extended to simple hierarchical models, as we demonstrated in Section 3.5. This gives us flexibility to model sequence elements by complex sub-models instead of a single state. This is not easily done for the methods introduced in Augustus or Genscan. Finally, coupled with the step-function approximation, it is possible to use geometric-tail distributions to model intergenic region lengths with reasonable decoding time. This is not possible with any of the other methods described above.

For evaluation, we have implemented our methods to our gene finding program ExonHunter (see Chapter 5 for more detailed description of the implementation). The training and testing data sets for this experiment are described in Appendix A.

Tables 3.2 (human) and 3.3 (fruit fly) show that the use of non-geometric length distributions contribute to the increased accuracy of gene prediction.

In human, we observed the greatest increase in performance in combination of exon and intron non-geometric length distributions. This observation supports our hypothesis that most augmentations to HMMs for gene finding do not bring significant increase in performance by themselves. Instead, they work in concert and the improved overall likelihood of the model then helps to improve the performance of predictions. We can also conclude that use of non-geometric exon lengths or use of non-geometric intron lengths improves the

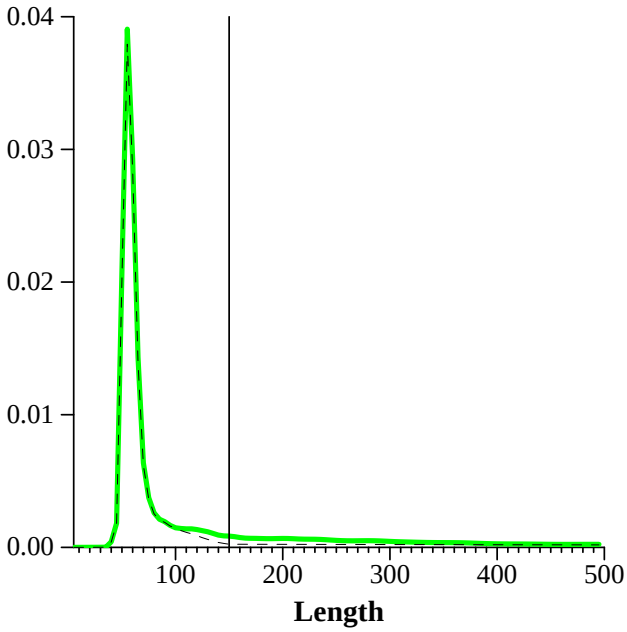

Figure 3.15: **Intron lengths of fruit fly.** A geometric-tail distribution (dashed line) approximates well the distribution of intron lengths (solid line) in fruit fly genome chromosome 3L/3R.

results of gene prediction about the same in human.

In contrast to these results for human sequences, in fruit fly the effect of non-geometric length distributions can be attributed almost exclusively to the intron length distributions. Fruit fly genes have short introns whose length can be approximated very well by geometric-tail distributions (Figure 3.15). Perhaps for this reason, the improvement in fruit fly gene finding is more pronounced than the effect in human gene finding.

Chapter 4

Finding the Most Probable Annotation

In the previous two chapters, we explored two different trade-offs related to modeling accuracy in HMMs. In models of biological signals (Chapter 2), the trade-off was between the amount of available training data and the faithfulness of a model structure. In the second trade-off, explored in Chapter 3, we exchanged the speed of the decoding algorithm for the increased faithfulness of modeling length distributions.

Sometimes we attempt to increase the faithfulness of an HMM by introducing complex model topologies in an attempt to capture the properties of a particular sequence element better. For example, Section 3.4 introduces a method for modeling length distributions where a single state in the HMM is replaced with a group of states (a gadget), and the correct probability is obtained only by adding the probabilities of multiple paths through such model.

We demonstrated in Section 3.4 that in this scenario, the Viterbi algorithm that finds only the single *most probable state path* in an HMM is no longer an appropriate method for decoding such models. This observation suggests an alternative definition of HMM decoding.

Definition 34 (Most probable annotation decoding). *Suppose we are given a hidden Markov model with set of states V and a mapping $\lambda : V \to \Lambda$ from the set of states V to a set of labels Λ. For a given sequence of symbols $s = s_1 \dots s_n$, the* probability of the annotation *$L = \lambda_1 \dots \lambda_n$ is defined as the sum of probabilities of all state paths whose annotation in L.*

More precisely, let Π_L be the set of all state paths π, where for all i, $\lambda(\pi_i) = \lambda_i$. Then the probability of annotation L is

$$\Pr(L, s) = \sum_{\pi \in \Pi_L} \Pr(\pi, s) \tag{4.1}$$

The most probable annotation problem *is the problem of finding the annotation L that maximizes the probability* $\Pr(L, s)$.

The labels in this definition represent the features of the sequence which we are interested in. There may be several state paths that correspond to the same sequence of labels. These

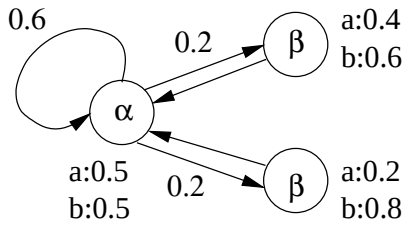

Figure 4.1: **The most probable path is different than the most probable annotation.** The most probable path for the string "ababa" is "$\alpha\alpha\alpha\alpha\alpha$," with probability 0.004, while the most probable labeling is "$\alpha\beta\alpha\beta\alpha$," with probability 0.01. The highest probability path with the same labeling has probability only 0.003.

represent "alternative explanations" of the same annotation, and we want to consider all possible explanations of a particular annotation together. This notion of a single annotation corresponding to multiple paths is consistent with the length distribution example cited above: all states within the same gadget have the same label, and thus when looking for the most probable annotation, we consider all the paths through the gadget together, as long as they all enter and leave the gadget at the same positions in the sequence.

The choice of the labels is an important step in creating models that are to be decoded using the most probable annotation. For example, in gene finding, we may decide to have separate sub-models for different classes of genes with different statistical properties, such as those encoding transmembrane proteins and globular proteins. If we assign the same sets of labels to all classes of genes, the most probable annotation decoding will find the most likely exon/intron structure. On the other hand, we can also assign separate sets of labels to each class, in which case the most probable annotation will also distinguish between these classes. Most probable annotation decoding gives us the flexibility to adjust the level of detail in the annotation simply by changing the label set.

In this chapter we will show that for some hidden Markov models, this new formulation of decoding is NP-hard, while for other models, the most probable annotation can be still found in time linear in the length of the sequence. Thus we introduce a different type of trade-off: increasing the faithfulness of a model by using a complex topology and the new formulation of decoding may make decoding intractable.

Consider a simple example in Figure 4.1. The example consists of an HMM with one state labeled α, and two states labeled β. The most probable state path for the string $(ab)^n a$ is always α^{2n+1} with the probability of $0.09^n \cdot 0.5$. However, the most probable annotation $\alpha(\beta\alpha)^n$ has higher probability, $0.14^n \cdot 0.5$, even though the highest probability path with the same annotation has probability of only $0.08^n \cdot 0.5$. Differences between the most probable path and the most probable annotation are surely a cause for concern. Moreover, as n grows in our example, the number of paths forming the most probable annotation increases exponentially. Thus, in this example, the probability of each single path of the most probable annotation is very low compared to the probability of the most probable path.

In these examples, a problem was caused by the existence of multiple paths corresponding

to the same annotation. We say that such an HMM has the *multiple path problem.*

The discrepancies between most probable state paths and most probable annotations in HMMs with the multiple path problem, as well as desirability of being able to retrieve the most probable annotation in case of such an HMM, have been recognized before (Burge, 1997; Krogh, 1997), and various heuristics have been suggested.

A common (but rarely implemented) idea is to compute the k most probable paths, which provide k candidate annotations (Durbin et al., 1998). These paths can be found efficiently with an algorithm for finding the k shortest paths in a directed acyclic graph by Eppstein (1998), and the probabilities of annotations corresponding to these candidate paths can be evaluated by a simple modification of the forward algorithm, in time linear in the length of the sequence. Finally, the best candidate annotation would be chosen. However, this approach will often fail (as in the case above), since the probability of each path in the most probable annotation may be small.

A different heuristic, called the N-best algorithm, was introduced by Schwartz and Chow (1990) and used in the context of biological sequence analysis by Krogh (1997). The algorithm, similar to the Viterbi algorithm, maintains a pool of several candidate annotations. The algorithm guarantees only that the probability of the chosen annotation is at least as high as the probability of the most probable state path, not that it is the most probable annotation.

Finally, one can apply *a posteriori* decoding—using the forward-backward algorithm to compute the most probable label at each sequence position. However, no state path may correspond to this annotation, so we cannot guarantee that it is consistent with the biological constraints of the model. To complete the heuristic, a second step is required to modify such a labeling to obtain a plausible annotation (see, *e.g.*, Martelli et al. (2002)), and the approach still does not guarantee that the most probable annotation is found.

In fact, it is unlikely that a polynomial-time algorithm computing the most probable annotation exists: Lyngsø and Pedersen (2002) showed that the problem is NP-hard. There is one loophole that can be explored. The proof of Lyngsø and Pedersen (2002) assumed that both an HMM and a sequence were part of the input. This does not match our decoding scenario precisely: in our application, the HMM is fixed and the input consists only of the sequence. Perhaps, for each particular HMM, an algorithm that is polynomial-time in the length of the sequence, but exponential in the size of HMM can be found.

In this chapter, we show that this is not the case, unless $P = NP$. We present a proof that there exists an HMM of modest size, for which finding the most probable annotation of an input sequence is NP-hard. This, however, does not mean that the most probable annotation problem is hard to solve for *all* HMMs. To that end, we present a range of algorithms with increasing running time that can compute the most probable annotation for increasingly larger classes of HMMs (the fastest one being the original Viterbi algorithm). However, the problem of deciding whether an HMM with a particular topology is NP-hard to decode, or whether there exist a polynomial-time decoding algorithm for it, is still open.

Most of the results presented in this chapter were published in Brejová et al. (2004a).

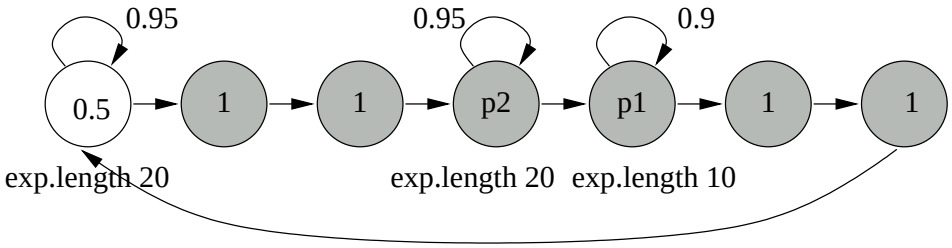

Figure 4.2: **HMM A: An HMM with the multiple path problem.** This HMM is inspired by the structure of introns (gray regions) whose composition changes towards their end. Colors (white or gray) represent state labels. The HMM emits symbols over the alphabet $\{0, 1\}$ and the numbers inside states represent the emission probability of symbol 1. The values of p_1 and p_2 are the parameters of the experiment.

4.1 Comparing Decoding by the Most Probable Path and by the Most Probable Annotation

We have designed a simple experiment to test if computing the most probable labeling instead of the most probable state path increases accuracy. We used the HMM A in Figure 4.2 to generate 5 000 sequences of mean length about 500 for various combinations of the parameters p_1 and p_2. This HMM outputs alternating white regions of mean length 20 and gray regions of mean length 34. The composition of white regions is constant, while the composition of gray regions changes towards their right ends. The gray regions are bounded by the signal 11 on both sides.

To analyze these sequences, we used three decoding algorithms: standard Viterbi, an algorithm for computing the most probable annotation, which will be introduced later in Section 4.3, and the Viterbi algorithm on a simplified model B in Figure 4.3.

Note that if an HMM does not have the multiple path problem, so that for every labeling, there exist only one path with that labeling, the most probable labeling corresponds to the most probable state path, and thus we can use the Viterbi algorithm to find the most probable labeling.

Therefore, in the simplified model, we replaced the two gray-labeled states with probabilities p_1 and p_2 by a single state and set the parameters to maximize the likelihood (probability of self-loop ≈ 0.97, probability of emission of 1 equal to $(2p_2 + p_1)/3$), as shown in Figure 4.3. This new HMM does not have the multiple path problem and therefore the Viterbi algorithm yields the most probable labeling.

We evaluated the error rate (percentage of the positions that were mislabeled compared to the labels on the state path that generated each sequence) for each decoding algorithm. Figure 4.4 shows our results.

We have observed two trends in the data. First, computing the most probable annotation in model A increases the accuracy compared to applying the Viterbi algorithm to model A. Second, the Viterbi algorithm applied to a simplified model B, which does not have the multiple path problem, often performs better than the Viterbi algorithm on the full model

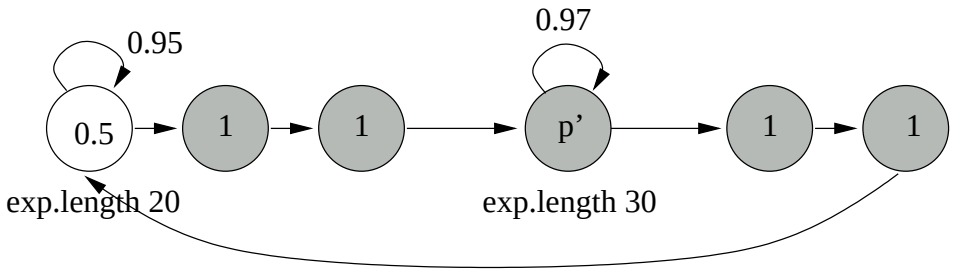

Figure 4.3: **HMM B: Simplified model of HMM A.** The two states in the HMM from Figure 4.2 with self-loops were replaced by a single self-loop state, where the emission probability of symbol 1 is $p' = (2p_2 + p_1)/3$. At most one state path corresponds to every labeling in this simplified HMM.

A. This behavior is paradoxical: since the data set was actually generated by model A, one would expect that decoding using the same model would give the best results. However, if the Viterbi algorithm is used for decoding, using a model B that is further from reality, but does not have the multiple path problem, gives better results.

We also encountered the same problem in Section 3.4. In that section, we replaced a single state with a gadget of several states to introduce a non-geometric length distribution. The probability of generating a sequence of length ℓ within such gadget was a sum of probabilities of many low-probability paths through the gadget and by this trick we achieved non-geometric length distribution. Such an HMM has a multiple path problem, and applying the Viterbi algorithm to decode it yielded surprising and paradoxical results. In particular, we observed in Section 3.4.2 that significant probability mass was "lost" upon entering such a gadget, and thus the most probable state path would enter such gadget less often. As suggested at the beginning of this chapter, this effect would disappear if we used the most probable annotation definition of decoding.

4.2 Finding the Most Probable Annotation is NP-hard

In our discussion so far, we concentrated on justifying the new definition of decoding by the most probable annotation. We demonstrated that such a decoding achieves better results in HMMs with the multiple path problem. In this section, we show that the problem of finding the most probable annotation is NP-hard. First, we review the NP-hardness proof of Lyngsø and Pedersen (2002), which used reduction from the maximum clique problem. However, in their reduction, they construct both an HMM and a sequence for every instance of the maximum clique problem. This does not correspond to our scenario, since in the annotation problem, the HMM is always fixed, and only the sequence changes from instance to instance. Therefore we give a new NP-hardness proof that shows that finding the most probable annotation is NP-hard, even if the HMM is fixed. Then we proceed to construct a small HMM that is NP-hard to decode by the most probable annotation.

For convenience of notation, we need to introduce *silent states* into hidden Markov mod-

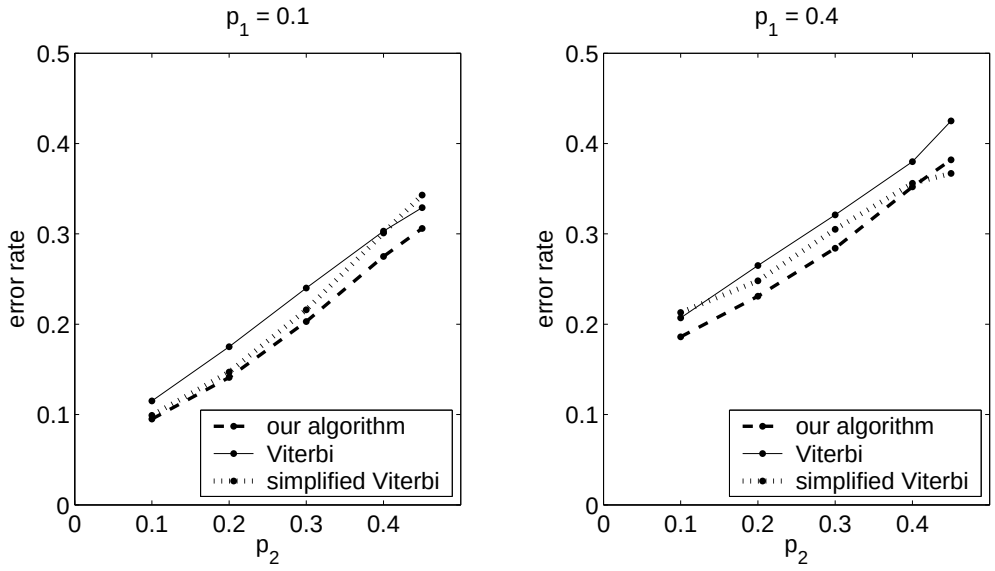

Figure 4.4: **Comparison of different decoding methods.** The graphs show the error rate using three decoding algorithms: Viterbi on model A, the algorithm for the most probable annotation on model A, and Viterbi on the simplified model B.

els. Silent states are regular states, except they do not emit any symbols. They are used as a convenient extension to the modeling language (see (Durbin et al., 1998, Section 3.4)), or to simply increase readability of the HMM diagrams. Obviously, the HMM is well-defined only if there is no directed cycle composed of silent states. In diagrams, we will display silent states as small black circles.

Silent states can be easily removed from the HMM. Every path π from a non-silent state u to a non-silent state v, going through only silent states can be replaced by a direct transition (u, v), where the transition probability will be the product of the transition probabilities on path π.

4.2.1 Proof of Lyngsø and Pedersen

We first review the proof of Lyngsø and Pedersen (2002), to clearly identify the differences between our work and theirs.

Theorem 35 (Lyngsø and Pedersen (2002)). *Finding the most probable annotation in hidden Markov models is NP-hard.*

Proof. We prove the claim by reduction from the maximum clique problem. Consider a graph $G = (V, E)$ over the set of vertices $V = \{1, 2, \ldots, n\}$. We will construct an HMM H and a sequence s such that the most probable annotation of the sequence s in HMM H can be used to identify the maximum clique in the graph G.

The HMM will emit sequences over a unary alphabet $\Sigma = \{*\}$. Clearly then, every state in the HMM will emit $*$ with probability 1. There will be $n + 2$ labels $\{1, \ldots, n, \#, !\}$. The

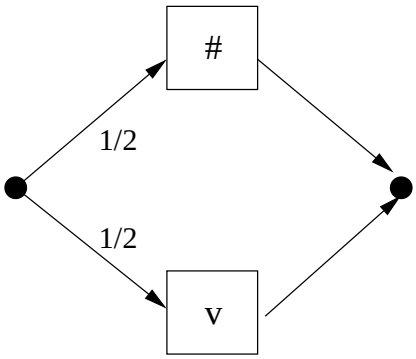

Figure 4.5: **NP hardness of the most probable labeling.** Gadget for vertex v in chain of vertex u, where (u, v) is an edge in graph G.

HMM will be composed of n chains of states, each corresponding to one vertex in V. In particular, the chain corresponding to vertex $u \in V$ will contain one block for each state $v \in V$ in the order $\{1, 2, \ldots, n\}$, where the block is:

- a single state with label v, if $u = v$
- the sub-model shown in Figure 4.5, if (u, v) is an edge in E,
- a single state with label #, otherwise.

Each chain will connect to the start state at the beginning and to the final state at the end. The transition from the start state to the beginning of the chain of a vertex u has probability $2^{\deg(u)}/\gamma$, where $\gamma = \sum_{v \in V} 2^{\deg(v)}$, and $\deg(v)$ is the degree of vertex v in graph G. Figure 4.6 shows an example of the construction.

Obviously, the only string generated by such a hidden Markov model is $*^{n+2}$. Any path in this model has the same probability $1/\gamma$. Consider any annotation $S = !s_1 s_2 \ldots s_n!$ of the sequence. Each s_i is either # or $s_i = i$. Let K_S be the set of all indices i for which $s_i = i$.

Each annotation S with non-zero probability is generated by several chains, with each of these chains contributing the same probability $1/\gamma$. If S is generated by a chain corresponding to vertex u, then all $v \in K_S$ must be connected to u, and also vertex u belongs to K_S. Thus the annotation S generated by k chains has probability k/γ, and the vertices corresponding to these k chains form a k-clique in G. Therefore, if the probability of the most probable annotation is k/γ, then there exists a clique of size k in graph G.

On the other hand, consider a clique K of size k in graph G. Take labeling $S = *s_1 s_2 \ldots s_n*$, where $s_i = i$, if $i \in K$, and $s_i = \#$ otherwise. This labeling is generated by each of the chains corresponding to vertices in K. Therefore, this labeling has probability at least k/γ. □

In this reduction, the construction of the HMM from a given graph is an essential part of the NP-hardness proof. However, this does not correspond to a typical application, where

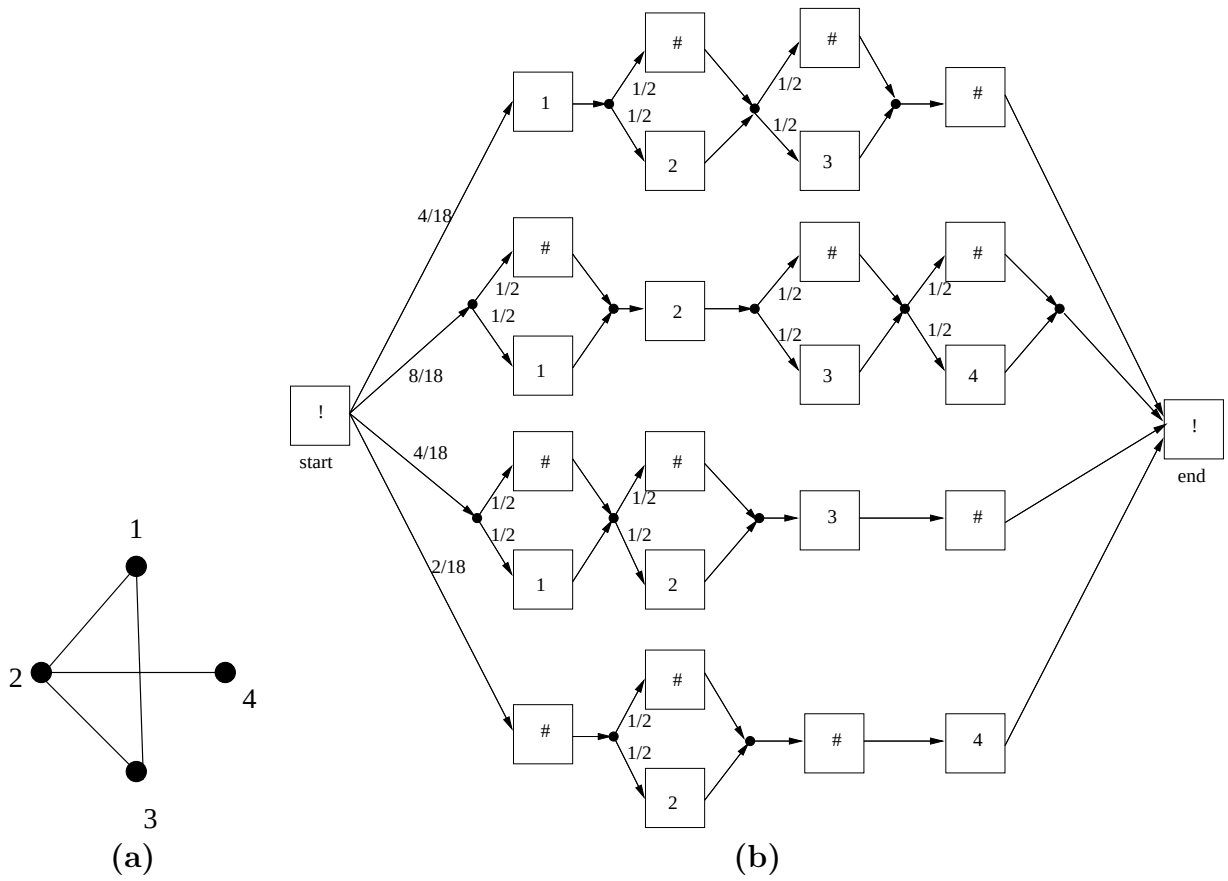

Figure 4.6: **Example of the construction of Lyngsø and Pedersen (2002).** HMM (b) is an example of the HMM constructed in the NP-hardness proof for the graph (a).

the HMM is given beforehand, and the task is to provide the decoding algorithm for that particular HMM. The HMM of Lyngsø and Pedersen (2002) has $\Theta(n^2)$ states, where n is the length of the sequence, while in bioinformatics applications, the HMM is of constant size.

In the rest of this section, we show a stronger result: there exists a specific constant-size HMM in which finding the most probable annotation of a sequence is NP-hard. This requires us to move much of the complexity of the reduction into the sequence itself, as the sequence will encode a particular instance of an NP-hard problem; in our case, a logical formula for the satisfiability problem.

4.2.2 Layered Graphs and the Best-Layer-Coloring Problem

Before we proceed with the proof itself, we introduce a simpler path counting problem on layered directed acyclic graphs, which we show is NP-complete. Then we show a reduction from this problem to the most probable annotation problem for HMMs, and so demonstrate that for SAT instances, we will have a specific HMM of constant (though large) size for which decoding is hard.

Definition 36 (Layered digraphs). *A* colored proper layered digraph *is a directed graph, with its vertices arranged in layers $L_1, L_2, \ldots, L_w$. Each edge connects a vertex in some layer L_i to a vertex in layer L_{i+1}. Each vertex is colored white or black.*

A layer coloring *is an assignment of a color (white or black) to each of the layers. A directed path from layer L_1 to layer L_w is* consistent with a layer coloring *if the colors of the vertices on the path match the colors of the layer coloring.*

Figure 4.7 shows an example of a layer coloring. The layered graph has four layers. There are no paths consistent with the layer coloring $\circ\ \circ\ \circ\ \circ$, but there are six different paths consistent with the coloring $\circ\ \bullet\ \circ\ \circ$. The Best-Layer-Coloring problem asks the following natural question.

Definition 37 (Best-Layer-Coloring problem). *Given a colored proper layered digraph G and a threshold T, is there a layer coloring which has at least T paths consistent with it?*

Theorem 38. Best-Layer-Coloring *is NP-complete, even for graphs where each layer has at most a constant number of vertices.*

Proof. Best-Layer-Coloring is in NP: for a given layer coloring, the number of consistent paths is at most exponential in the number of layers and can be computed by simple dynamic programming.

To prove NP-hardness, we reduce SAT to Best-Layer-Coloring. Consider an instance of SAT, which is a logical formula in conjunctive normal form with n variables $u_1, u_2, \ldots, u_n$ and m clauses $c_1, c_2, \ldots, c_m$. We give an overview of the construction in Figure 4.8.

The graph consists of $m+1$ blocks $0, 1, 2, \ldots, m$, each with $2n$ layers. The layer coloring of each block represents a truth assignment of the variables $u_1, \ldots, u_n$. The truth assignment of each variable is encoded by two consecutive layer colors: $\circ\ \circ$ (false) or $\circ\ \bullet$ (true). Layer

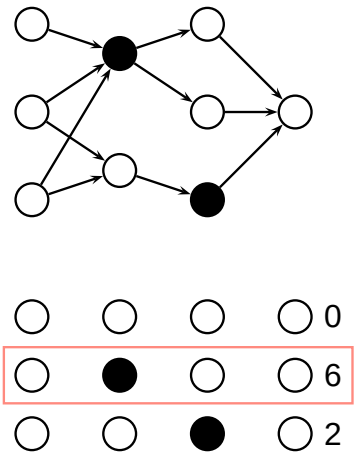

Figure 4.7: **Illustration of the** Best-Layer-Coloring **problem.** An example of a layered graph and of its layer colorings showing the number of paths consistent with each coloring. The coloring ∘ • ∘ ∘ is the best coloring, with six consistent paths.

colorings that are not of this form will have no corresponding paths, so we do not need to consider them. Thus, we can use the terms "layer coloring of a block" and "truth assignment of a block" interchangeably. Let x be the truth assignment of block 0 and let $y_1, \ldots, y_m$ be the truth assignments of blocks $1, \ldots, m$.

In a "yes" instance of SAT, we want truth assignments $x, y_1, \ldots, y_m$ to be the *same* satisfying truth assignment of the SAT formula.

We will decompose the structure of the graph into several components, each of them having several inputs and outputs. An input of a component is the number of consistent paths ending in a designated vertex on the left-most layer of the component. Similarly, an output of a component is the number of consistent paths ending in a designated vertex on the right-most layer of the component. Let $A \xrightarrow{x} B$ denote a component that transforms a vector of inputs A to a vector of outputs B when corresponding layers have coloring x.

The component encode(x) in Figure 4.8 encodes the truth assignment x as a vector of three integers $v(x)$ on its output. In each of the blocks $1, 2, \ldots, m$, we enforce the truth assignment to be the same as x with the component eq(x, y_i). The input of this component is the encoding of the truth assignment x, vector $v(x)$, and the output is the number $2K(n)$, where $K(n) = 4^n - 2^{n+1} + 1$, if the truth assignments x and y_i are the same, or a number smaller than $2K(n)$ otherwise. Finally, component $SAT(c_i, y_i)$ outputs its input if truth assignment y_i satisfies clause c_i, or 0 otherwise. The input to $SAT(c_1, y_1)$ is a single path.

There is an additional layer before the first block and after the last block to ensure the proper start and end of each consistent path.

The path counting threshold for the Best-Layer-Coloring instance T is $2m \cdot K(n) + 1$. For the number of consistent paths to reach this threshold, all of the block colorings must

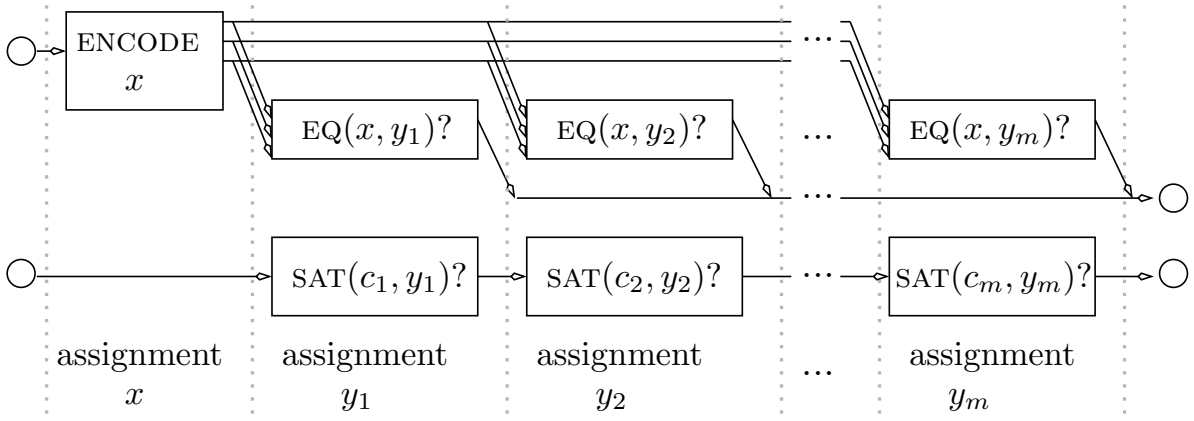

Figure 4.8: **Overview of NP-completeness proof of** Best-Layer-Coloring**.** The boxes represent components of the graph construction. Lines connecting the components represent layered subgraphs that propagate the number of paths from left to right regardless of the layer coloring (this is achieved by using one black and one white vertex in each layer).

represent the same truth assignment and the assignment must satisfy all clauses. Otherwise the number of consistent paths will be smaller.

What remains to show is that the components we have described exist. Lemma 39 shows how to create the component $SAT(c_i, y_i)$, while Lemma 40 shows the construction of encode(x) and eq(x, y_i) The total number of vertices in each layer of this construction is at most 29, which is sufficient to show that an instance of SAT can be reduced to Best-Layer-Coloring with a constant number of nodes per layer. □

Lemma 39. *For a given clause c, there exists a component $SAT(c, y)$ with a constant number of vertices in each layer that outputs its input, if the truth assignment y satisfies clause c, or 0 otherwise.*

Proof. The component has two parallel lanes, one corresponding to the clause being satisfied, the other to it being unsatisfied. There is one 2-layer section for each variable of the truth assignment. The structure of the ith section depends on whether variable u_i is present in the clause c_i as the positive literal u_i, the negative literal $\neg u_i$ or not at all, as shown in Figure 4.9. If the variable is present and its assignment satisfies the clause, the path switches from the "unsatisfied" lane to the "satisfied" lane. The input is the first vertex of the "unsatisfied" lane and the output is the last vertex of the "satisfied" lane. Figure 4.10 shows an example of a chain of the SAT components assembled as in Figure 4.8.

□

Lemma 40. *There exist components* encode(x) *and* eq(x, y) *with a constant number of vertices in each layer, such that the output of* eq(x, y) *is $2K(n)$, if $x = y$, and smaller otherwise.*

Proof. Let $b(x)$ be the number whose binary representation encodes the truth assignment x of variables $u_1, \ldots, u_n$ (with u_1 as the highest-order bit and u_n as the lowest-order bit). Our

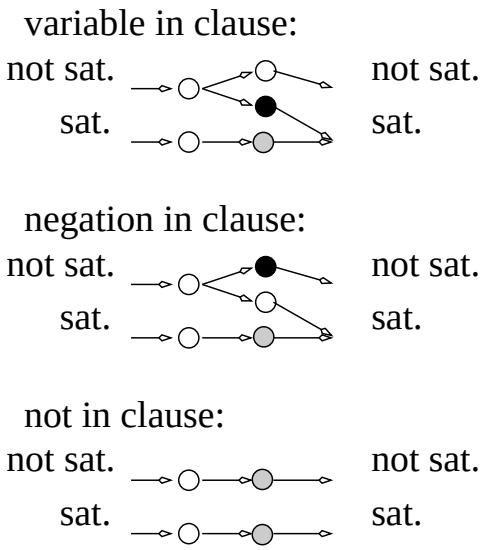

Figure 4.9: **Part of $SAT(c, y)$ component corresponding to one variable.** Gray vertices represent 'uncolored' vertices that can be used in a path for either layer color. These vertices are used only to simplify the drawings: each 'uncolored' vertex can be replaced by a white and a black vertex with the same incoming and outgoing edges. In layer a coloring, ∘ ∘ encodes false, ∘ • encodes true.

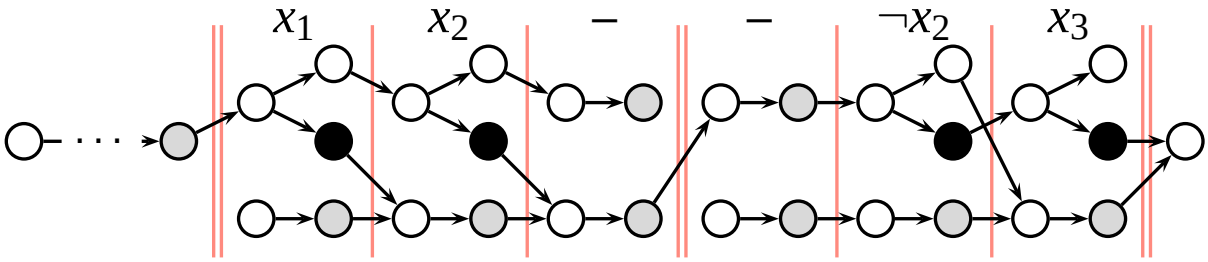

Figure 4.10: **Example of assembly of** SAT **components.** Assembly for the formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. The assembly results in one path, if the formula is satisfied, or in zero paths, if the formula is not satisfied by the coloring of layers, assuming that the same assignment is repeated three times.

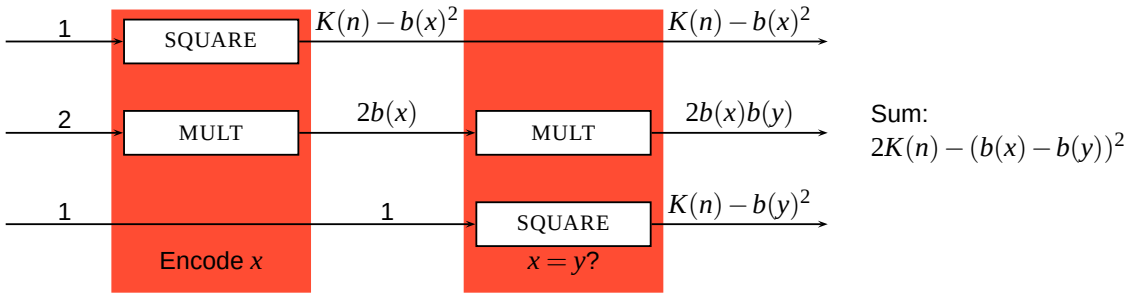

Figure 4.11: **Overview of** ENCODE **and** EQ**.** The number of paths generated by the component ENCODE (left side) and the component EQ (right side) together is maximized if the two assignments x and y are equal. In a layer coloring, $\circ\,\circ$ encodes false, $\circ\,\bullet$ encodes true.

goal is to encode $b(x)$ as a vector with the ENCODE(x) component and use this encoding to compute $2K(n) - (b(x) - b(y_i))^2$ as the output of the EQ(x, y) module. To this end, the ENCODE component computes ENCODE$(x) : 1 \xrightarrow{x} (1, K(n) - b(x)^2, 2b(x))$ and the EQ component computes EQ$(x, y) : (1, \alpha, \beta) \xrightarrow{y} K(n) - b(y)^2 + \beta \cdot b(y) + \alpha$. An overview of these two components is shown in Figure 4.11. If the output of ENCODE(x) is used as the input of EQ(x, y), the output of EQ(x, y) will be $2K(n) - (b(x) - b(y))^2$, which is equal to $2K(n)$ if $x = y$ and is less than $2K(n)$ otherwise.

These two required components can be constructed as a combination of two subcomponents, MULT$(x) : \alpha \xrightarrow{x} \alpha b(x)$ and SQUARE$(x) : 1 \xrightarrow{x} K(n) - b(x)^2$, as shown in Figure 4.11. Both MULT(x) and SQUARE(x) consist of identical 2-layer sections, each processing one bit of $b(x)$.

Consider the section of component MULT(x) processing the k-th bit of the truth assignment x. Let w be the binary representation of truth assignment of the first $k-1$ variables, t be the truth assignment of the k-th variable, and $z = 2w + t$ be the truth assignment of the first k variables.

The section has two inputs and outputs: $(\alpha, \alpha w) \xrightarrow{t} (\alpha, \alpha z)$. The value αz can be computed from the values αw, α, and t by the following equation:

$$\alpha z = \begin{cases} 2\alpha w, & \text{if } t = 0 \\ 2\alpha w + \alpha, & \text{if } t = 1 \end{cases} \tag{4.2}$$

This computation can implemented by the layered graph shown in Figure 4.12. Assembling n copies of this section into the component MULT(x) is straightforward, as illustrated in Figure 4.13.

Similarly, we can design the component SQUARE(x). Section k of this component has four inputs and outputs $(1, B(k-1), C(w, k-1), D(w, k-1)) \xrightarrow{t} (1, B(k), C(z, k), D(z, k))$,

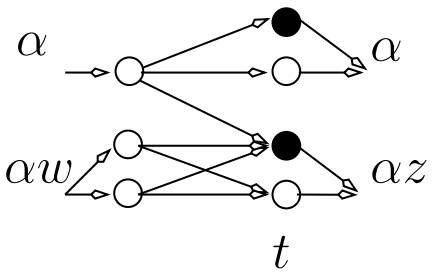

Figure 4.12: **One section of component** $\textsc{mult}(x) : \alpha \xrightarrow{x} \alpha b(x)$**.** In layer coloring, $\circ\ \circ$ encodes false (or $t = 0$), $\circ\ \bullet$ encodes true (or $t = 1$). In the bottom level, the number of paths is always doubled between the first and second layer, and an additional α paths are added if $t = 1$. This structure implements Equation 4.2.

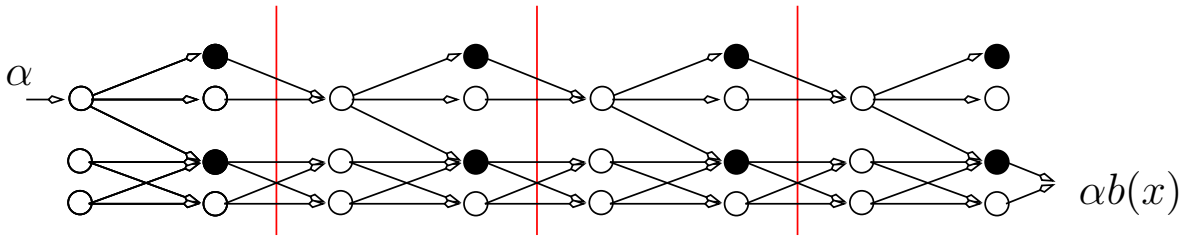

Figure 4.13: **Component** $\textsc{mult}(x) : \alpha \xrightarrow{x} \alpha b(x)$**.** Assembly for a formula with four variables. If α paths are consistent with the coloring in the left part of the graph, $\alpha b(x)$ paths are consistent with the layering on the output of this component.

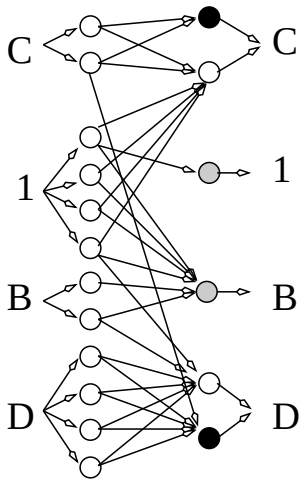

Figure 4.14: One section of component SQUARE$(x) : 1 \xrightarrow{x} K(n) - b(x)^2$

where functions B, C, D are defined as follows:

$$B(k) = 2^{k+2} - 4, \tag{4.3}$$
$$C(z,k) = B(k) - 4z, \tag{4.4}$$
$$D(z,k) = 4^k - 2^{k+1} + 1 - z^2. \tag{4.5}$$

The values of B, C, D can be computed bit by bit using the following recurrence relations:

$$B(k) = 2B(k-1) + 4 \tag{4.6}$$
$$C(z,k) = \begin{cases} 2C(w,k-1) + 4, & \text{if } t = 0 \\ 2C(w,k-1), & \text{if } t = 1 \end{cases} \tag{4.7}$$
$$D(z,k) = \begin{cases} 4D(w,k-1) + B(k-1) + 1, & \text{if } t = 0 \\ 4D(w,k-1) + C(w,k-1), & \text{if } t = 1 \end{cases} \tag{4.8}$$

The graph for each section, based on the recurrences (4.6), (4.7), (4.8), is depicted in Figure 4.14. Graphs of n such identical sections can be assembled into the component SQUARE(x). Since $B(0)$, $C(0,0)$, and $D(0,0)$ are all zeroes, there are no paths entering these inputs in the first section. The output D of the last section is the output of the whole component, and it has value $D(b(x), n) = K(n) - b(x)^2$, as desired.

□

Now, with all the required components, our proof is complete. We have given a layered directed graph for the SAT instance. The layer with the largest number of vertices has four lines carrying previously computed values (with two vertices in each of them), one component EQ (consisting of MULT with at most four vertices per layer, SQUARE with at most 12 vertices per layer, and a line carrying a previously computed value with two vertices), and one component SAT (with at most four vertices per layer). Therefore each layer has at

most 30 vertices, and wherein if there exist a layer coloring with $2K(n)+1$ consistent paths, the formula is satisfiable.

4.2.3 From Layer Colorings to HMMs

In the previous section we have shown the NP-completeness of the Best-Layer-Coloring. Here we use the Best-Layer-Coloring problem to prove that most probable annotation in HMMs is NP-hard, even for a fixed HMM.

Theorem 41. *There exists a constant-size HMM for which it is NP-hard to find the most probable labeling of an input binary string.*

Proof. Theorem 38 shows that there exists a constant k such that finding the best layer coloring is NP-hard, even if each layer has size at most k. Here we show that Best-Layer-Coloring for a graph with at most k vertices per layer can be reduced to finding the most probable labeling of a binary string in a fixed HMM, the size of which depends only on k.

In particular, consider an instance of Best-Layer-Coloring: a proper layered digraph G with at most k vertices in each layer. We will show that there exists a constant-size HMM M (depending only on the constant k) and a binary string S of a polynomial size in the number of layers of G such that the most probable annotation of string S in M gives the best layer coloring of G.

First, we modify the digraph G by adding unconnected vertices so that each layer has exactly k white and k black vertices, and we number the vertices in each layer so that the white vertices have numbers $1, \ldots, k$ and black vertices $k+1, \ldots, 2k$. Denote $V = \{1, \ldots, 2k\}$.

Now, we construct an HMM M', which has all the properties we require except that the alphabet Σ will be huge, but constant. Such an HMM is easily transformed into an HMM M using only a binary alphabet: each character in Σ is replaced by a binary string of length $\lceil \log_2 |\Sigma| \rceil$, and each state is replaced by a binary tree of states with $|\Sigma|$ leaves.

Our HMM M' has $2k \cdot 2^{2k}$ states of the form (i, V'), for each $i \in V$ and $V' \subseteq V$, and special "error" and "start" states. The alphabet contains $(2^{2k})^{2k}$ symbols of the form $(V'_1, \ldots, V'_{2k})$, where $V'_i \subseteq V$, and a special symbol \$.

The alphabet symbols encode G's structure: each symbol encodes the configuration of edges between two layers of G (for each vertex i, V'_i is the set of its neighbors in the next layer). Each state corresponds to a vertex and the set of its neighbors in the next layer.

We will set the label of each state (i, V') to white if $i \leq k$, or to black otherwise. Vertex (i, V') can emit all symbols of form $(V'_1, V'_2, \ldots, V'_{i-1}, V', V'_{i+1}, \ldots, V'_n)$ with equal probability p_1. State (i, V') will have transitions to all states (j, V''), where $j \in V'$ and $V'' \subseteq V$, with equal probability $p_2 = 1/(2k \cdot 2^{2k})$. There will also be a transition from each state (i, V') to the 'error' state with probability $1 - |V'|/2k$.

The start state does not emit any characters and has a transition to each state (i, V'), with equal probability p_2. The error state emits the special character \$ and has a transition only to itself.

Consider string $S = s_1 \ldots s_w$, where $w+1$ is the number of layers of G, and s_i corresponds to the configuration of edges between layers i and $i+1$. Any state path that emits this string

has equal probability $p_1^w \cdot p_2^w$. Moreover, there is a one-to-one correspondence between the state paths and paths through G, and a labeling of a state path corresponds to a coloring of the corresponding path in G. Therefore, the most probable labeling corresponds to the best layer coloring in G. □

4.2.4 Constructing a Small HMM that is NP-hard to Decode

The HMM obtained in the proof of Theorem 41 has $O(k \cdot 2^{2k(2k+1)})$ states, which is very large, considering the number of layers required for the proof of Theorem 38 is $k = 30$.

In this section we will use ideas from the proof of Theorem 38 to reduce SAT to the most probable labeling problem directly, obtaining a much smaller HMM.

The SAT instance will be encoded in the sequence. If the SAT formula contains m clauses with n variables, then the sequence will consist of $(m+1)$ blocks of $(n+1)$ symbols, terminated by a special symbol ! as follows.

The first block is a string $0^n\#$, where each of the zeroes represents one variable. Each of the next m blocks encodes one clause of the formula. The ith symbol is 1, if the clause contains the positive literal of the ith variable, 0 for the negative literal, and $-$, if the clause does not contain the ith variable. Each block is terminated by a special symbol \$. For example, formula $x_1 \wedge (x_2 \vee \neg x_3)$ will be encoded as $000\#1--\$-10\$!$.

The HMM will have two labels: white and gray. We can imagine that the most probable annotation of the HMM is simply a coloring of the original sequence. We expect the most probable annotation to represent the satisfying assignment, where special characters are colored white, each variable with value "true" is colored gray, and each variable with value "false" is colored white. In the most probable annotation representing a satisfying assignment, the same pattern will repeat $m+1$ times, and in each clause at least one symbol "0" will be labeled white, or one symbol "1" will be labeled gray. Figure 4.15 shows an example of a satisfying assignment as a labeling.

Figure 4.16 shows the schema of the HMM. The labels of the states are represented by the color of each state (white or gray). Each state emits only the symbols depicted inside the state with equal probability. Multiple edges join some pairs of states; the multiplicity of an edge is noted at the tail of the edge (if not explicitly stated, it is 1). Silent states are inserted to increase the readability of the diagram.

We want all non-zero probability paths in this HMM to have the same probability. To achieve this goal, we need to transform the diagram in the figure to an HMM by removing silent states and introducing an error symbol, and an error state (not shown in the figure). The error state emits only the error symbol and has only one outgoing transition, leading back to the error state. The error symbol is also emitted in each of the states. Emissions of the error symbol and transitions to the error state are used to normalize probabilities, so that every ordinary symbol emission, and every transition between two ordinary states, have the same probabilities.

Once we have accomplished this transformation, every path has the same probability, we only need to count the number of paths produced by each particular annotation.

Formula:	$x_1 \wedge (x_2 \vee \neg x_3)$
Satisfying assignment:	x_1: true, x_2: true, x_3: false
Encoding of the formula:	`0 0 0 # 1 - - $ - 1 0 $ !`
Labeling corresponding to the assignment:	`0 0 0 # 1 - - $ - 1 0 $ !`

Figure 4.15: Encoding formulas and assignments for HMM solving SAT

To understand the HMM, we will relate it to the diagram of a layered graph in Figure 4.8 on page 107. Each path in this layered graph will be represented by a path in the HMM and the coloring of the path in the layered graph will be represented by the annotation (labels of the states) in the HMM. For a given annotation and a given formula with n variables and m clauses, we will denote by x the truth assignment implied by the annotation of the first block, and y_i the truth assignment implied by the annotation of block $i+1$, representing the i-th clause. In Section 4.2.2, we defined $K(n) = 4^n - 2^{n+1} + 1$, and we also defined $b(x)$ to be the number whose binary representation encodes the truth assignment x.

At each point of time, sub-model A carries exactly one path. Block A corresponds to the top-most level in the layered graph (the top-most level always carries one path as well). From time to time, one or more paths are split from the path carried by sub-model A; these paths are propagated to the other blocks. All non-zero probability paths must end in the state "stop" after $(m+1)(n+1)+1$ steps. This is because the analyzed sequence always ends with a symbol !, which is emitted only by the "stop" state. The number of paths that end in this state is proportional to the probability of a particular annotation. Thus the annotation generating the largest number of such paths will be the most probable annotation.

There are two parts of the HMM. Sub-models B, C, D contribute one path to the final annotation if all block annotations represent satisfying assignments (i.e., for each block corresponding to a clause, there must be at least one symbol 1 colored gray, or one symbol 0 colored white). At the beginning of each clause, this path is in sub-model C, and it is transferred to sub-model D once a variable is found that is satisfied by the truth assignment represented by the annotation. After the clause is terminated with symbol "$", this path is transferred back to the sub-model B, if there was a satisfied variable, or it is discontinued in sub-model C, if the clause is not satisfied. Thus, the function of sub-model B is characterized by the following lemma.

Lemma 42. *Upon emission of the symbol "$", terminating block $i+1$ representing the i-th clause, block B contains one path if the first i clauses are satisfied by their assignments $y_1, \ldots, y_i$, or zero paths otherwise.*

Sub-models E, F, G, H, J, L, and M enforce that all clause blocks of the input sequence are annotated with the same value assignment of the variables. The combination of these sub-models mimics the function of the component ENCODE(x) in Figure 4.8. The following lemma characterizes the intermediate number of paths in each of the sub-models G, E, F, J

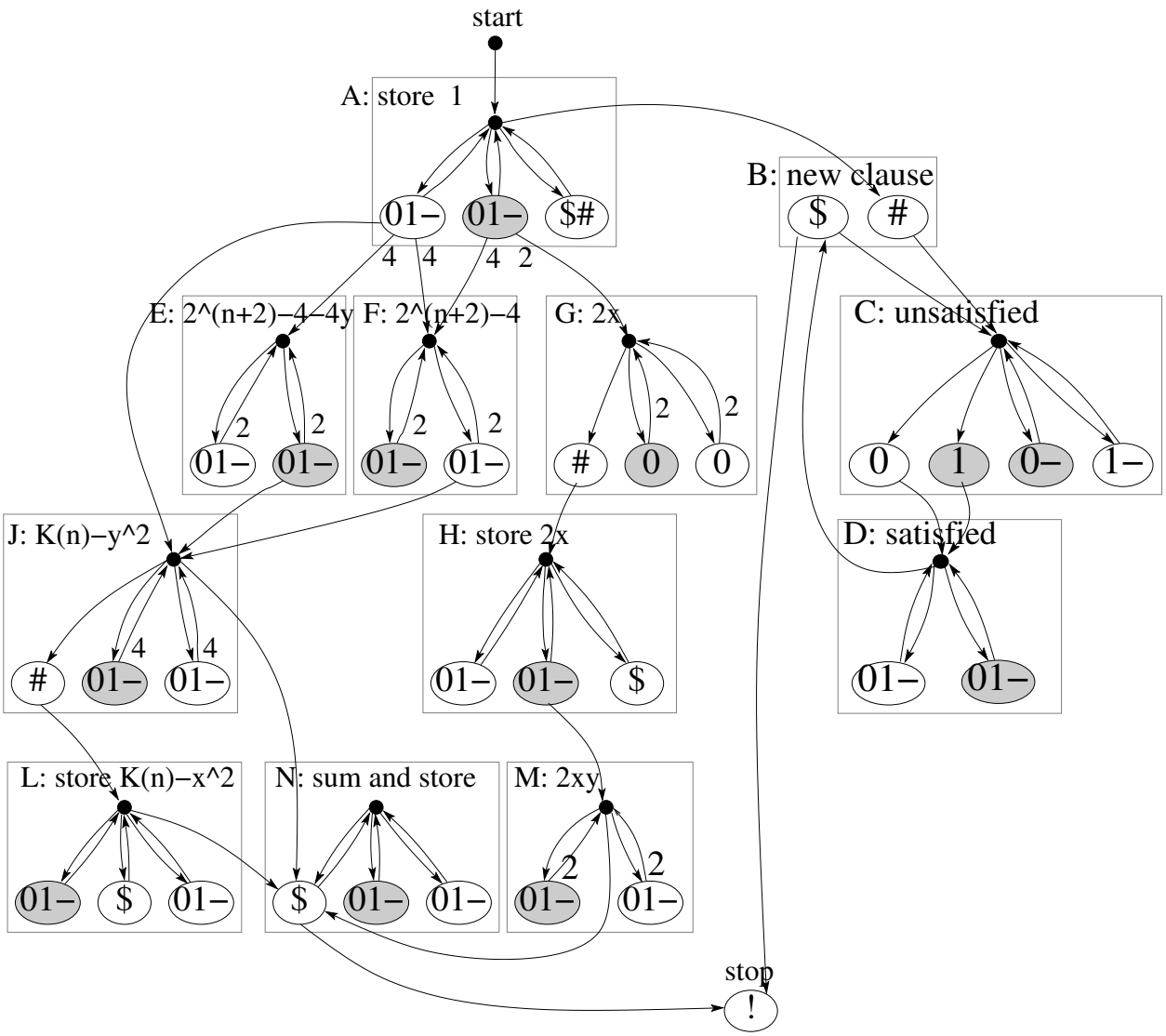

Figure 4.16: **HMM solving** SAT**.** A small HMM for which it is NP-hard to find the most probable annotation of a sequence. The input sequence encodes the formula in blocks, where each block except the first encodes one clause of the formula. The most probable labeling represents a satisfying assignment, if such an assignment exists. Gray labels represent value true, and white labels represent value false. The first block is terminated by #, blocks corresponding to clauses are terminated by $, and the formula is terminated by !.

in the process of emitting the first block of the sequence. The correctness of the lemma follows directly from the Recurrences 4.6–4.8 on page 111.

Lemma 43. *Let z be the binary representation of the first k variables in the truth assignment x. After emitting the first k symbols of the first block of the sequence,*

a) sub-model G contains $2z$ paths,

b) sub-model E contains $C(z,k)$ paths,

c) sub-model F contains $B(k)$ paths,

d) sub-model J contains $D(z,k)$ paths,

where $B(k) = 2^{k+2} - 4$, $C(z,k) = B(k) - 4z$, and $D(z,k) = 4^k - 2^{k+1} + 1 - z^2$ are defined as in Section 4.2.2.

The sub-models characterized in the previous lemma are used to compute the values of $K(n) - b(x)^2$ and $2x$, which are then stored in sub-models L and H for the rest of the execution.

Lemma 44. *After emitting the symbol "#" terminating the first block of the sequence,*

a) sub-model L contains $K(n) - b(x)^2$ paths,

b) sub-model H contains $2b(x)$ paths.

The number of paths contained in sub-model L is added to the number of paths in sub-model N upon terminating each clause-block of the sequence. The paths in block H are used for further computation to obtain value of $2xy_i$ for each clause i.

After finishing the first block, for each of the subsequent blocks of the input sequence corresponding to i-th clause, $K(n) - b(y_i)^2$ and $2b(x)b(y_i)$ paths are created in sub-models J and M. This function is analogous to the component $\textsc{eq}(x, y_i)$ in Figure 4.8. Sub-models M, E, F, and J are used for computing intermediate results, again according to Recurrences 4.6–4.8.

Lemma 45. *Let z be the binary representation the first k variables in the truth assignment y_i. After emitting the first k symbols of block $i+1$ of the sequence representing the i-th clause of the formula,*

a) sub-model M contains $2xz$ paths,

b) sub-model E contains $C(z,k)$ paths,

c) sub-model F contains $B(k)$ paths,

d) sub-model J contains $D(z,k)$ paths,

Whenever the model reaches the end of a clause, marked by the \$ symbol, the paths from sub-models J, L, and M are added into sub-model N, contributing altogether $2K(n) - (b(x) - b(y))^2$ paths.

Lemma 46. *After emitting the symbol "\$" terminating block $i+1$ of the sequence corresponding to the i-th clause of the formula, the number of paths contained in block N is*

$$\sum_{j=1}^{i} 2K(n) - (b(x) - b(y_i))^2 \tag{4.9}$$

Upon emitting the symbol "!" terminating the sequence, the final "stop" state will receive all the paths accumulated in sub-model N. This number is maximized when $x = y$ (i.e., if the annotation is the same for every clause). If all clauses have the same assignment, sub-model N contributes $2mK(n)$ paths; otherwise, it contributes some smaller number. The "stop" state also receives one path from sub-model B if all clauses are satisfied by their assignments.

Therefore, a satisfying annotation that is consistent over all the clauses will yield $2mK(n)+1$ paths. Any other annotation will yield a smaller number of paths, and thus a satisfying consistent annotation is the most probable annotation, if such an annotation exists. Therefore, if we can solve the most probable annotation problem for the HMM in Figure 4.16, we can use it to solve SAT in polynomial time.

After removing the silent states and introducing the error state, the resulting HMM has 34 states. As we will see in the following section, this HMM is hard to decode because states with different labels are distributed all over the HMM, with many edges between states with different labels. For HMMs with only a few edges leading between states of different labels, we will introduce polynomial-time algorithms.

4.3 Computing the Most Probable Annotation

In the previous section, we have shown that in general, it is NP-hard to compute the most probable annotation for a given HMM. However, we can characterize special classes of HMMs for which the most probable annotation can be computed efficiently.

For example, consider an HMM where the annotation is uniquely determined by the state path, so any two state paths have different annotations. Then the most probable state path corresponds exactly to the most probable annotation, and it is thus easily computable by the Viterbi algorithm in $O(nm^2)$ time, where n is the length of the sequence, and m is the number of states in the HMM.

In this section, we give a range of algorithms with increasing running time (the Viterbi algorithm being the fastest of them) that can decode increasingly wider classes of HMMs. For each algorithm, we give a sufficient condition that characterizes the class of HMM topologies that can be decoded using the algorithm. Moreover, each of the algorithms is also guaranteed to return an annotation with probability at least as high as the probability of the most

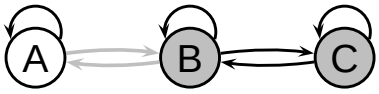

Figure 4.17: **HMM with critical edges.** The labels of the states are determined by the colors (white or gray).

probable state path, even if the input HMM does not belong to the class of HMMs that are guaranteed to be decoded correctly by the algorithm.

4.3.1 Most Probable Extended Annotation

We first introduce a notion of *extended annotation* and an algorithm that computes the *most probable extended annotation* in polynomial time. Then we characterize a class of HMMs for which an extended annotation uniquely determines the annotation. For this class, our algorithm finds the most probable annotation. In the following definitions we assume that the HMM has a designated start state s and a designated final state f. However, the definitions and algorithms are easily extended to models without such states. Let $\text{in}(u)$ for any state u be the set of states that have a transition into state u.

Definition 47 (Extended annotation). *A* critical edge *is a transition between two states of different label. The* extended annotation *of a state path $\pi_1\pi_2\ldots\pi_n$ is the pair (L, C), where $L = \lambda_1, \lambda_2, \ldots, \lambda_n$ is the sequence of labels of each state in the path and $C = c_1, c_2, \ldots, c_k$ is the sequence of all critical edges followed on the path.*

For example, the HMM in Figure 4.17 has three states and two different labels. The edges between states A and B are critical edges, since states A and B have different labels. State path $ABCB$ and state path $ABBB$ both yield the same extended annotation ($\square\blacksquare\blacksquare\blacksquare, A \to B$). The following extended Viterbi algorithm (EVA) computes the most probable extended annotation.

Theorem 48 (Extended Viterbi algorithm). *For a given sequence $S = x_1 \ldots x_n$ and an HMM with m states, it is possible to compute the most probable extended annotation in time $O(n^2m^3)$.*

Proof. We modify the Viterbi algorithm for computing the most probable state path. The Viterbi algorithm computes by dynamic programming values of $V[u, i]$, where $V[u, i] = \max \Pr(x_1 \ldots x_i, \pi_1 \ldots \pi_i)$, where the maximum is taken over all state paths $\pi_1 \ldots \pi_i$ starting in state s and ending in state u.

To compute a particular value of $V[u, i]$, the dynamic programming uses the following recurrence, examining all possible options for the second to last state:

$$V[u, i] = \max_{v \in \text{in}(u)} V[v, i-1] \cdot a(v, u)e_u(x_i). \tag{4.10}$$

The extended Viterbi algorithm (EVA) is a modification of the Viterbi algorithm to compute the most probable extended annotation. Instead of computing the values $V[u, i]$, we compute values $L[u, i]$ defined as follows:

$$L[u, i] = \max \Pr(x_1 \dots x_i, (L, C), \pi_i = u), \tag{4.11}$$

where the maximum is taken over all extended annotations (L, C) of sequence $x_1 \dots x_i$, where the generating process ends in state u. The Viterbi algorithm computes the most probable paths through the model using dynamic programming, at each step considering all possible options for the second last state. Instead, we will examine all possible durations of the last segment with the same label and instead of the single most probable path in that segment, we will compute the sum of all possible state paths in this segment. If the segment starts at position $j \leq i$ of the sequence, let $P[v, u, j, i]$ be this sum; it is the probability of generating the sequence $x_j \dots x_i$, starting in state v, and ending in state u, using only states with label $\lambda(u) = \lambda(v)$. We get the following recurrence:

$$L[u, i] = \max_{j \leq i} \max_{v:\lambda(v)=\lambda(u)} \max_{w \in \text{in}(v):\lambda(w)\neq\lambda(v)} L[w, j-1] \cdot a(w, v) \cdot P[v, u, j, i] \tag{4.12}$$

We compute values of L in order of increasing i. For each i, we compute all relevant values of $P(v, u, j, i)$ in order of decreasing j by an algorithm similar to the backward algorithm, using the following recurrence:

$$P[v, u, j, i] = \sum_{w:v\in\text{in}(w),\lambda(v)=\lambda(w)} e_v(x_j) \cdot a(v, w) \cdot P[w, u, j+1, i] \tag{4.13}$$

When the computation of L is finished, the most probable extended annotation can be reconstructed by tracing back the labels and critical edges used to obtain the value of $L[f, n]$, as for the Viterbi algorithm. □

Note that the probability of the extended annotation returned by the algorithm is always at least as high as the probability of the most probable state path Π found by the Viterbi algorithm. This is because the probability of the extended annotation corresponding to Π must be at least as high as the probability of Π itself.

4.3.2 Critical Edge Condition

The algorithm defined above is guaranteed to compute the most probable annotation for a much wider class of HMMs than the Viterbi algorithm. Here is a sufficient condition for this class.

Definition 49. *An HMM* satisfies the critical edge condition *for an input sequence s, if any two paths for s with the same annotation have the same sequence of critical edges. An HMM* satisfied the critical edge condition *in general if for all input sequences s, the critical edge condition is satisfied.*

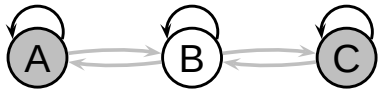

Figure 4.18: An HMM violating critical edge condition

The hidden Markov model in Figure 4.17 clearly satisfies the critical edge condition, since for each transfer between the two labels there is only one possible critical edge which can be used. A more complicated example of a model satisfying the critical edge condition is shown in Figure 4.21. On the other hand, the HMM in Figure 4.18 does not satisfy the critical edge condition, since for the annotation □■□ there are two possible extended annotations: $(\square\blacksquare\square, B \to A, A \to B)$ and $(\square\blacksquare\square, B \to C, C \to B)$. The significance of the critical edge condition is shown by the following claim.

Corollary 50. *If an HMM satisfies the critical edge condition for a sequence s, then the EVA computes the most probable annotation of sequence s.*

Proof. We call an annotation (or extended annotation, or state path) L *possible* with respect to sequence s if $\Pr(L \mid s) > 0$.

The EVA computes the most probable extended annotation. Therefore for the statement to be false, the most probable annotation and the most probable extended annotation must be different.

This happens only if there exist at least two different possible extended annotations L_1 and L_2, that correspond to the most probable annotation. Let π_1 be a possible state path corresponding to L_1, and π_2 be a possible state path corresponding to L_2. Since L_1 and L_2 are different, the paths π_1 and π_2 must differ in at least one critical edge; yet both π_1 and π_2 produce the same annotation. Therefore the HMM cannot satisfy the critical edge condition. □

We can test algorithmically whether a given HMM topology (not considering emission probabilities) satisfies the critical edge condition for every input sequence. We first use depth-first search to build a set S_s of all pairs of states that are reachable from the start state by the same annotation. We start from the pair $(s, s) \in S_s$, and in each iteration we add a new pair (u, v) if $\lambda(u) = \lambda(v)$, and there exists $(u', v') \in S_s$ such that $u' \in \text{in}(u)$ and $v' \in \text{in}(v)$. The search is completed in $O(m^2)$ iterations, each requiring $O(m^2)$ running time. Similarly, we also build a set S_f of all pairs of states from which the final state can be reached by the same annotation. For the critical condition to be violated, there must exist a pair $(u, v) \in S_s$ and $(u', v') \in S_f$ such that $\lambda(u) \neq \lambda(u')$, and (u, u') and (v, v') are two different transitions. The algorithm takes $O(m^4)$ time.

It is possible to modify this verification algorithm to verify the critical edge condition in $O(m^4|\Sigma|^2)$ time, if emission probabilities are given. Note that this test may yield a different result, since some states may not produce some of the alphabet symbols, making it

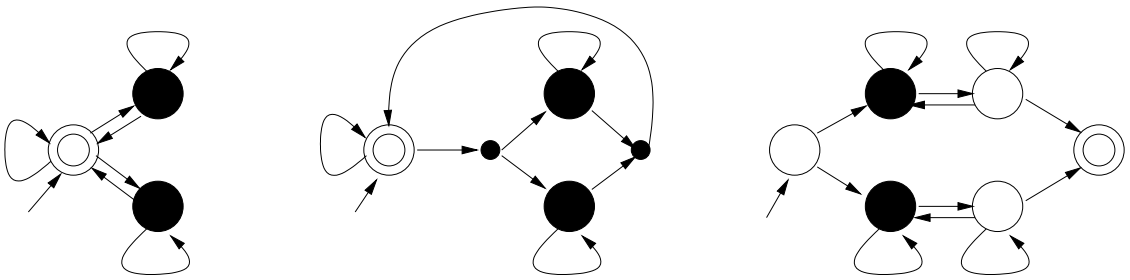

Figure 4.19: **Usefulness of silent states.** The color of each state corresponds to the label. Silent states are represented as smaller circles. The HMM (a) violates the critical edge condition and cannot be decoded by our algorithm. There is no equivalent topology without silent states satisfying the condition. Using silent states, we are able to construct an equivalent HMM (b) that satisfies the critical edge condition. However, the technique is not universal: HMM (c) cannot be transformed to comply with the condition.

impossible for two different paths with the same extended annotation to generate the same string; hence, this extended algorithm may find even more HMMs that satisfy the condition.

And finally, we can also verify the condition for a given HMM and input string in $O(nm^4)$ time. In that case, we will build a set of state pairs that can be reached by the same annotation for each position in the sequence.

4.3.3 Silent States and the Critical Edge Condition

For purposes of decoding algorithms such as Viterbi, forward, or backward algorithm, there are two ways of dealing with the silent states: either modify the algorithms to account for their presence or construct an equivalent HMM with silent states removed.

It is possible to modify the EVA to account for the presence of silent states as well, using an approach analogous to one shown in the book of Durbin et al. (1998, Section 3.4). For the purpose of evaluating the critical edge condition, we need to assign labels to silent states, even though the silent states do not emit any symbols.

The second method, removing the silent states, cannot be used because the transformation of the HMM topology can result in an HMM violating the critical edge condition—see Figure 4.19. Conversely, some HMMs can be transformed to equivalent HMMs that satisfy the critical edge condition by addition of silent states. Thus, in our case, the silent states are a crucial modeling tool.

4.3.4 Applications of the EVA

The EVA solves the most probable annotation problem on a much wider variety of HMM topologies than does the Viterbi algorithm. In this section we show several biological applications, where the EVA can be used, and which could consequently benefit from increased

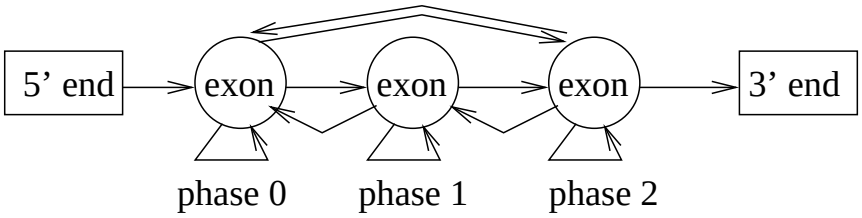

Figure 4.20: Simplified model of ESTScan

accuracy.

The simplified model of ESTScan (Iseli et al., 1999) shown in Figure 4.20 has the multiple path problem. ESTScan uses an HMM to predict the coding part of an EST. Compared to a typical coding region predictor, ESTScan needs to handle insertions and deletions within the coding sequence, which are caused by the low quality of EST sequencing. The exact place of a sequencing error cannot be easily identified, so to simply distinguish coding parts of ESTs from non-coding, we assign the same label to all states corresponding to coding sequence. Each path in this model corresponds to some combination of insertions and deletions, but many such paths can yield the same annotation. The actual model used in ESTScan has a more complicated topology, ensuring for example that only one insertion or deletion can occur within the same codon.

Because the model has the multiple path problem, the Viterbi algorithm is not appropriate for decoding it. However, the model satisfies the critical edge condition and thus the EVA can be used to find the most probable annotation. The condition is satisfied because the states labeled "exon" are grouped in a subgraph with only one incoming and one outgoing edge.

A more complicated example is the simple model of exon/intron structure of eukaryotic genes in Figure 4.21. Multiple copies of the same intron model preserve the three-periodicity of coding regions. Intronic sequence in DNA contains a pyrimidine-rich tail close to the acceptor site. Its composition is very different from the rest of the intron, and provide strong support for a possible neighboring acceptor site. The tail has variable length, and it does not have a clear boundary. This creates a multiple-path problem because there are always several high-probability alternatives for the transfer from "intron" state to the "tail" state. Even though there are multiple edges for transitions between "exon" and "intron" labels, the model does not violate the critical edge condition, since the length of the exonic sequence uniquely determines which critical edge will be used.

TMHMM (Figure 4.22) is an HMM for prediction of topology of transmembrane proteins (Krogh et al., 2001). The task is to predict positions of transmembrane helices, cytoplasmic, and non-cytoplasmic loops in a protein sequence. The two different models of non-cytoplasmic loops create the multiple path problem, potentially decreasing prediction accuracy if the Viterbi algorithm is used. We introduce silent states to ensure that the critical edge condition is satisfied.

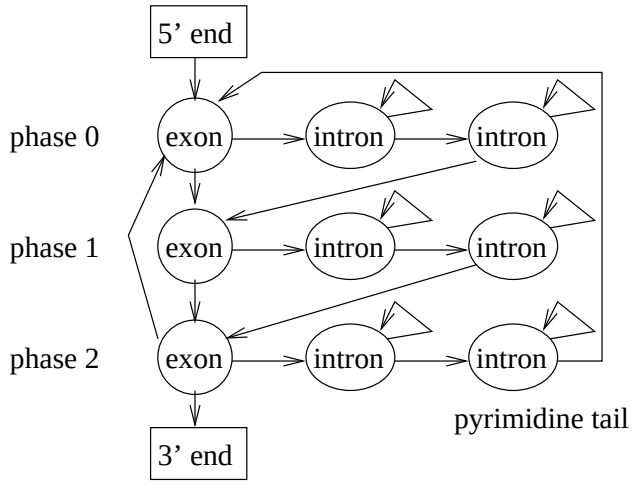

Figure 4.21: Simple model of exon/intron structure

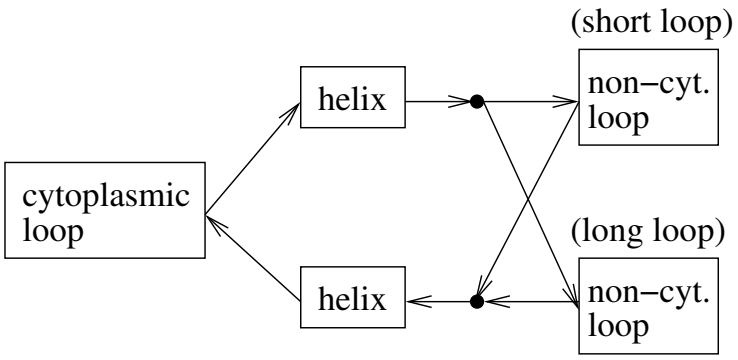

Figure 4.22: **TMHMM: prediction of topology of transmembrane proteins.** Boxes in the figure represent groups of states with the same label.

4.3.5 Generalizing the EVA and the Critical Edge Condition

In previous sections we introduced two scenarios under which it was possible to find the most probable annotation in HMM. First, if every annotation can be uniquely mapped to a single path in HMM, the Viterbi algorithm with running time $O(nm^2)$ can be used to find the most probable annotation, which is the same as the most probable path in this case. Second, if the HMM satisfies the critical edge condition (which is satisfied by a much larger class of HMMs), then the extended Viterbi algorithm (EVA) with running time $O(n^2m^3)$ can be used to find the most probable annotation.

In this section we extend the notion of critical edge condition and EVA to an even larger class of HMMs. As the configuration of critical edges in HMM gets more complex, the running time of the decoding algorithm progressively increases.

Definition 51 (Generalized extended annotation). *A* generalized extended annotation *is a pair (L, C), where $L = \lambda_1\lambda_2 \dots \lambda_n$ is a sequence of labels, and $C = c_1, c_2, \dots, c_k$ contains for every transition between different labels in L either a critical edge that can be used for such a transition, or the symbol "*" (a* masked critical edge*).*

We say that a generalized extended annotation matches *a state path $\pi_1\pi_2 \dots \pi_n$ if the labeling L corresponds to the labels of the states, and if every critical edge in C corresponds to the critical edges used on the path (while critical edges, which are masked by "*" in C, can be arbitrary).*

We say that a generalized extended annotation is of order d *if from any d consecutive critical edges of C at least one is not masked.*

Note that one path can be matched by several generalized extended annotations.

Theorem 52 (Generalized EVA). *For a given sequence $S = x_1 \dots x_n$ and an HMM with m states, it is possible to compute the most probable generalized extended annotation of order d in time $O(n^{d+1}m^{d+2})$. We call the corresponding algorithm* generalized EVA of order d.

Note that Theorem 48 is a special case of generalized EVA of order 1.

Proof. In Theorem 48 we were computing values of $L[u, i]$—the probability of the most probable extended annotation of the first i symbols of the sequence ending at state u—by dynamic programming. The recurrence decomposed the problem of computing $L[u, i]$ into subproblems depending on the last critical edge used.

We can further modify the algorithm to account for masked critical edges in generalized extended annotations. The dynamic programming algorithm will compute values of $L_d[u, i]$: the probability of the most probable *generalized* extended annotation of *order at most d* of the first i symbols ending in state u. This will be done by decomposing the problem into subproblems depending on the position of the *last unmasked critical edge*. If the last unmasked critical edge was used at position $j < i$, then we need to consider all possible annotations of the region of the sequence $x_j \dots x_i$ with at most $d - 1$ transitions between different labels. In particular:

$$L_d[u, i] = \max_{j \le i, (v,w): \lambda(v) \neq \lambda(w)} L[w, j-1] \cdot a(w, v) \cdot \max_{\lambda_j, \dots, \lambda_i \in \mathcal{L}_d(j-i+1)} P_d(v, j, u, i, \lambda_j \dots \lambda_i), \quad (4.14)$$

where $\mathcal{L}_d(k)$ is the set of labelings of length k with at most $d-1$ transitions between different labels, and $P_d(v, j, u, i, \lambda_j \dots \lambda_i)$ is a probability of labeling $\lambda_j \dots \lambda_i$ of sequence $x_j \dots x_i$, if the model starts in state v and finishes in state u. This probability for a given labeling can be computed in $O(nm^2)$ by the backward algorithm. For each tuple (v, j, u, i) there are $O(n^{d-1}m^{d-2})$ possible labelings $\lambda_j \dots \lambda_i$, and for each pair v, i there are $O(nm^2)$ values of v, w and j. Therefore, to compute the $O(nm)$ values of $L_d[u, i]$, we need a running time of $O(n^{d+2}m^{d+3})$.

This running time can be further reduced by careful organization of how the values of $P_d(v, j, u, i, \lambda_j \dots \lambda_i)$ are computed. For a given pair (u, i), all labelings that will need to be explored can be organized in a tree. The root of the tree is composed of labeling containing only a single label $\lambda(u)$. Children of each node either extend this labeling backward with the same label as the parent node, or change the label to a different label; such extensions are done until either the beginning of the sequence is reached, or the threshold $d-1$ of allowed label changes is exceeded. If the computation of the values of P_d is organized along this tree, only $O(m)$ time is needed to compute the desired probabilities for each labeling from the probabilities of the parent labeling. Using this method, we can decrease the running time to $O(n^{d+1}m^{d+2})$. □

Analogously to the critical edge condition for EVA, we now introduce the generalized critical edge condition which will characterize the class of HMM topologies that can be decoded by the generalized EVA.

Definition 53 (Consensus generalized extended annotation). The consensus generalized extended annotation *of a set of extended annotations* $\{(L, C_1), (L, C_2), \dots, (L, C_\ell)\}$ *sharing the same labeling* L *is a generalized extended annotation* $(L, (c_1, c_2, \dots, c_k))$ *such that* $c_i = c$, *if the* i*th critical edge of all of* $C_1, C_2, \dots, C_\ell$ *is the same edge* c, *and* $c_i = *$ *otherwise.*

Definition 54 (Generalized critical edge condition). *An HMM* satisfies the generalized critical edge condition of order d, *if for any sequence* s, *and a non-zero probability annotation* L *of* s, *the consensus generalized extended annotation of all non-zero probability extended annotations* (L, C) *of* s *is of order at most* d.

The HMM in Figure 4.23 does not satisfy the critical edge condition. This is because there are two critical edges leading from white label to gray label: $B \to C$ and $D \to E$. Note that the critical edge condition is a special case of the generalized critical edge condition for $d = 1$.

On the other hand, the HMM satisfies the generalized critical edge condition for $d = 2$. For example, annotation ■□□■■■■□□□ has only two possible combinations of critical edges in an extended annotation: $(F \to A, B \to C, F \to A)$ and $(F \to A, D \to E, F \to A)$. This yields a consensus annotation: $(F \to A, *, F \to A)$, which is consistent with the generalized critical edge condition of order $d = 2$. The following corollary shows that the most probable labeling for this HMM can be found by the generalized EVA in running time $O(n^3m^4)$.

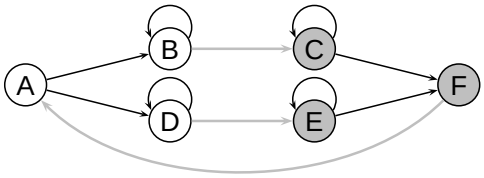

Figure 4.23: HMM requiring generalized EVA algorithm

Corollary 55. *If an HMM satisfies the generalized critical edge condition of order d, then the generalized EVA of order d finds the most probable annotation in HMM.*

Proof. Consider the most probable annotation L. Its consensus generalized extended annotation (L, C) is of order at most d, since the HMM satisfies the generalized critical edge condition of order d. Thus (L, C) must be one of the options considered by the generalized EVA, and thus the generalized extended annotation (L', C') returned by generalized EVA will have a probability at least that of L.

Now assume that L' is not equal to L. All paths included in generalized extended annotation (L', C') are also included in annotation L', and therefore the probability of annotation L' is at least that of L (and, in fact, it must be equal, since L is the most probable annotation). □

4.4 Summary

In this section we have investigated the most probable annotation problem in HMMs. We showed that the problem is NP-hard, even for a fixed HMM constructed in the proof, in contrast to the previous NP-hardness proof by Lyngsø and Pedersen (2002), where the HMM constructed depended on the input instance. In most biological applications, the HMM is fixed.

Even though the problem is NP-hard in general, it is possible to compute the most probable labeling for some HMMs. First of all, if there is only a single feasible state path for every possible annotation, the problem can be solved by the Viterbi algorithm. Otherwise, the HMMs have the multiple path problem, and we need to use different methods to decode them.

We provided an $O(n^2m^3)$ time extended Viterbi algorithm (EVA) and characterized a wide class of HMMs (those satisfying the critical edge condition) for which we can find the most probable annotation using it. Its run time may cause problems in applications with long input sequences, such as gene finding. Still, it is acceptable in other cases, such as analysis of protein sequences or ESTs. In practice, the running time may be further decreased by application of biological constraints (such as location of open reading frames) and various stopping conditions.

The model topologies that can be decoded by the EVA include those for transmembrane protein topology prediction (TMHMM), distinguishing coding regions in ESTs (ESTScan),

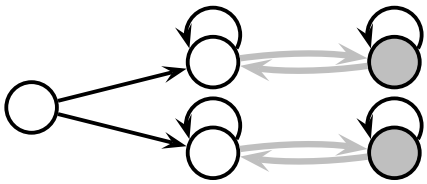

Figure 4.24: **An HMM with unknown decoding algorithm.** Is there a polynomial algorithm that finds the most probable annotation in this HMM, or is the problem NP-hard?

or a better intron model used in gene finding. We also noted that the use of the Viterbi algorithm instead of the most probable labeling may lead to paradoxical behavior, where a more accurate model will yield worse results.

We can further generalize the EVA and the critical edge condition and use increasingly slower decoding algorithms to decode increasingly wider classes of HMMs. Even if a particular HMM topology cannot be proved to be always decoded correctly with a particular algorithm, increasing the running time helps to increase the accuracy. The probability of the annotation corresponding to the most probable path found by the Viterbi algorithm in $O(nm^2)$ time is at most as large as the probability of the annotation corresponding to the most probable extended annotation found by EVA in $O(n^2m^3)$. This trend extends to the generalized EVA, where the probability increases with the value of the parameter d, though at the cost of an increasing running time $O(n^{d+1}m^{d+2})$.

Several problems remain open. First, we do not know at present any polynomial algorithm for finding the most probable annotation for the model shown in Figure 4.24. Is decoding of this simple model NP-hard? Similar topologies are useful in various applications for providing alternative models for multi-label structures (such as different types of genes). More generally, can we provide a complete characterization of the models that are NP-hard to decode?

Second, are there HMM topologies (other than ones without the multiple path problem) that can be decoded in sub-quadratic time? Such models may be useful in applications where the input sequence is long.

Chapter 5

Implementing ExonHunter

To facilitate the evaluation of our ideas from the previous sections, we implemented an *ab initio* gene finder, which we named ExonHunter. ExonHunter is based on hidden Markov models, as described in Section 1.3. It incorporates structured HOT signals as outlined in Chapter 2, and advanced methods for length distribution modeling described in Chapter 3. In the design of the model topology we avoided topologies that have the multiple path problem described in Chapter 4.

The main purpose of this chapter is to desribe the implementation which we used to evaluate our ideas in Chapters 2 and 3. ExonHunter's performance is comparable to other commonly used *ab initio* gene finders, and therefore it is a reasonable base line for our experiments.

Ab initio gene finders only use information that is contained in the DNA sequence itself. We also implemented extensions to ExonHunter that allow consideration of information beyond the DNA sequence. Such information includes genome-to-genome comparisons, databases of expressed sequence tags and proteins, etc. The description and evaluation of this extension is beyond the scope of this thesis, and is described in Brejová et al. (2005), and in detail in Brejová (2005). However, in this chapter, we briefly describe how we incorporate information about common sequence repeats into our gene finder, since using this information is crucial for the performance of the gene finder.

5.1 Hidden Markov Model of ExonHunter

We use the HMM shown in Figure 1.17 on page 33. This model finds genes on both strands of DNA at the same time.

We model exon states, intron states, and intergenic states by fourth order Markov chains. The intron states of all three models on the forward strand have the same parameters. Similarly, the intron states of all three models on the reverse strand have the same parameters. The parameters of the exon states which are in the same frame on the same strand are also tied together. Finally, the parameters of the fourth order Markov chain representing intergenic regions are computed as an average of the parameters for introns on forward strand

and introns on reverse strand as we cannot rely on accurate annotation of intergenic regions; in fact, the best available training sequences contain single genes, with only a small amount of their flanking intergenic sequence.

Each basic signal is represented by a single state generating the whole length of the signal, as described in Chapter 2. The donor signal uses the window $[-3, +6]$ around the donor splice site, and is represented as a HOT-2 model. The acceptor signal uses the window $[-4, +3]$ around the acceptor splice site, and is represented as a HOT-2 model as well. The start site signal uses the window $[-9, +4]$, and is represented as a TREE model. Finally, the stop site signal uses the window $[-6, +9]$, and is represented as a TREE model as well.

We have used two additional signals in our model to enhance detection of start sites and acceptor sites. First, we added a model of the *signal peptide*. The signal peptide is a chain of several amino acids close to the start of proteins, serving as a targeting signal for the cell transport machinery. Having a signal peptide is characteristic of proteins directed to the endoplasmatic reticulum, transmembrane proteins, and of proteins that are transported outside the cell. The signal peptide is a special signal because it is at the same time a signal and part of the resulting protein.

To build a model of the signal peptide, we translated the DNA sequences annotated in the training set to protein sequences. Zhang and Henzel (2004) presented a profile HMM built from a data set of experimentally confirmed occurrences of the signal peptide. We used this model and scored the likelihood of occurrences of the signal peptide in sequences in our training set with the program HMMER (Eddy, 1998). Then we chose the top 30% of identified sequences, and trained a DNA signal model for the window $[+5, +20]$ relative to the start site. We chose 30% of sequences because the signal peptide occurs in approximately 30% of all proteins. For that reason, we use a mixture of 30% of this signal and 70% of regular coding sequence for this window in the model.

The second signal is a *pyrimidine (C/T) rich tail* that occurs at the end of each intron. The pyrimidine tail is of variable length, therefore the model suggested in Figure 4.21 on page 123 would be a proper model of this phenomenon. However, such a model would introduce the multiple path problem into the HMM, and in consideration of the results of Chapter 4, we decided to replace it with a signal model of fixed length on the window $[-20, -5]$ relative to the acceptor splice site.

Both signal peptide and pyrimidine rich tail are signals that have only vague positional dependence. Significant positions important for recognition of the signal may be shifted by several positions within the signal. Therefore, we use a *windowed position weight matrices* (WPWM) for these signals. In windowed models, the statistics for position i are collected not only from position i, but also from positions in interval $[i - w, i + w]$, where w is the size of the window. Such models were suggested in Genscan for some of the signals (Burge, 1997). For the signal peptide, we use a three-periodic WPWM of order 2 with window size 4. The model of pyrimidine rich tail is a WPWM of order 3 with window size 10.

We modeled length distributions as suggested in Chapter 3. We trained the length distributions of exons separately for first exons (tail starts at position 100), internal exons (tail starts at position 120), last exons (tail starts at position 290), and single exon genes

(tail starts at position 370). The length distributions of introns were trained with a tail starting at position 150. In case of intergenic regions, we applied the faster algorithm for step-function approximation, and the geometric tail starts at 30000.

We subjected all conditional probability distributions estimated from the training data (both for Markov chains and for the signals) to *interpolation* to prevent overfitting the training data. This interpolation was introduced to gene finding by Salzberg et al. (1998) in their gene finding program GLIMMER for the simpler task of prokaryotic gene finding (prokaryotic genes do not have introns).

Consider a Markov chain of order 2 and an *interpolation threshold* D. Suppose we are estimating the probability $\Pr(X_i = x_i \,|\, X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2})$ of generating the symbol x_i given that the symbols x_{i-1} and x_{i-2} were generated at the preceding two positions. Let the number of all samples in the training set for which $X_{i-1} = x_{i-1}$ and $X_{i-2} = x_{i-2}$ be D', out of which d' have $X_i = x_i$.

If D' is at least D, we estimate the probability simply by the frequency count as d'/D'. Otherwise, we consider these statistics unreliable, and supplement the training set by lower order statistics, computing the probability using the following formula:

$$\frac{d' + (D - D') \cdot \Pr(X_i = x_i \,|\, X_{i-1} = x_{i-1})}{D}. \tag{5.1}$$

Intuitively, this formula replaces the "missing" data points by using a sample generated with lower order statistics. This is done recursively if the number of samples is not sufficient to train the lower order statistics. We have set $D = 100$ in ExonHunter.

Finally, many researchers have noticed before that various statistics used in gene finding change significantly with the *GC content* of the DNA sequence (Burge, 1997; Stanke and Waack, 2003). The GC content is the percentage of the nucleotides G and C in the sequence. For example, GC-rich sequences are more likely to contain genes than AT-rich sequences. We use four GC content categories, set so that each category has roughly the same amount of the DNA sequence in the training set (for human sequences, the boundaries between categories are 41%, 48%, and 55%). We compute the GC content based on a window of 1000 nucleotide around each sequence position, and change parameter sets according to the category. This is different from other gene finding programs that use a single GC category computed over the whole sequence. Such an approach is not suitable for long sequences, such as chromosomes.

Table 5.1 shows comparison of common features of three gene finders: Genscan (Burge, 1997), Augustus (Stanke and Waack, 2003), and our new gene finder, ExonHunter. Genscan is the only of the three gene finders that includes a model of untranslated regions, even though the model is simplistic. Other gene finders do not contain models of untranslated regions due to lack of dependable training data.

Novel features of Augustus include similarity-based sequence weighting model for donor splice sites (see the description in Section 2.1.1), and a high number of GC content levels. The high number of GC content levels is achieved by parameters at a particular level being a linear combination of all training samples, where those samples that do not belong to the particular level are weighted with lower weight.

	Genscan	**Augustus**	**ExonHunter**
Composition:			
– Non-coding	variable (base for log-odds)	MC-4	MC-4
– Coding	3xMC-5	3xMC-4	3xMC-4
Lengths:			
– Exons	exact	exact	geometric tail ($t = 150 \ldots 370$)
– Introns	geometric	short/long introns	geometric tail ($t = 150$)
– Intergenic	geometric	geometric	geometric tail/step function ($t = 30000$)
Signals:			
– Donor	$[-3, +6]$ MDD	$[-8, -4]$ MC-4 $[-3, +6]$ SSW	$[-3, +6]$ HOT-2
– Acceptor	$[-38, -5]$ WPWM-3 $[-4, +3]$ PWM-1	$[-37, -6]$ WPWM-3 $[-5, +1]$ PWM-0	$[-20, -5]$ WPWM-3 $[-4, +3]$ HOT-2
– Start site	$[-6, +6]$ PWM-0	$[-20, -1]$ WPWM-3 $[+1, +7]$ PWM-0 $[+8, +23]$ 3xMC-4	$[-9, +4]$ TREE $[+5, +20]$ 3xWPWM-2
– Stop site	$[-3, +3]$ PWM-0	$[-3, -1]$ PWM-1	$[-6, +9]$ TREE
GC-content:	4 levels whole sequence	10 levels with smoothing whole sequence	4 levels sliding window
UTRs:	simple model	within intergenic	within intergenic

Legend:
– kxMC-o: k-periodic Markov of order o
– MDD: maximum dependence decomposition (Burge, 1997)
– SSW: similarity-based sequence weighting (Stanke and Waack, 2003)

Table 5.1: **Feature comparison of gene finders.** Basic features of the gene finders Genscan (Burge, 1997), Augustus (Stanke and Waack, 2003) and our gene finder, ExonHunter. The table was assembled based on our best understanding of the descriptions in the papers presenting the programs.

Finally, our gene finder includes the novel length distribution models described in Chapter 3, as well as the new structured signal models that were introduced in Chapter 2.

5.2 Common Sequence Repeats

About 50% of human DNA sequence is composed of sequence repeats (International Human Genome Sequencing Consortium, 2004). These may be short simple sequences of 1-5 nucleotides that are repeated with minor variations many times, or they may be long complex regions copied several times in various places of the genome. These repeats are only rarely part of the protein coding genes we are looking for. Yet, often their statistical properties resemble real genes.

Gene finding programs usually either mask the original sequence for repeats, or ignore the issue of repeats altogether. The program RepeatMasker (Smit et al., 2002) can be used to find such repeats and replace them in the sequences by symbol N—a symbol for unknown nucleotides. However, this requires gene finding programs to systematically and explicitly deal with potentially long stretches of such unknown nucleotides.

Instead, we use the framework we have developed for incorporating additional information into ExonHunter (Brejová et al., 2005). Such information is expressed in form of *advisors*—probabilistic statements about the likelihood of labeling a particular position as being from intron, exon, or intergenic region. In our case, we base an advisor on a list of likely repeats produced by RepeatMasker. In regions covered by the repeats, the advisor identifies a high probability of the region being part of an intron or intergenic region. This has an effect of significantly increasing the probability of all state paths that identify introns or intergenic regions in repetitive parts of the sequence. The details of the technique can be found in Brejová et al. (2005) and are beyond the scope of this thesis.

5.3 Performance of ExonHunter on Human Sequences

We trained ExonHunter for human DNA sequence, using a mixture of data sets as outlined in Table 5.2. The training was conducted on sequences with repeats identified by RepeatMasker (Smit et al., 2002) and replaced by the "unknown" nucleotide N.

Then we evaluated ExonHunter on the testing set of the ENCODE gene prediction workshop (see more detailed description of the dataset in Appendix A). The data set contains 21 MB of sequence and includes 296 genes and 2782 unique exons. Some exons overlap due to alternative transcripts of the same gene. We compared two versions of ExonHunter (one without the repeat advisor, and one with the repeat advisor) with other *ab initio* programs submitted to the ENCODE workshop (an overview of these programs can be found in Section 1.2.2), as well as with Genscan. The predictions were evaluated using the program Eval (Keibler and Brent, 2003). The results are shown in Table 5.3.

In the first group, we considered programs that did not use repeat information. This includes GeneMark.hmm (Besemer and Borodovsky, 2005), Genscan (Burge, 1997), and

Feature	Training sets
Intron/exon composition	encode-train, augustus-train
Intron/exon lengths	encode-train, augustus-train
Number of exons	encode-train
Intergenic lengths	chr22-training
Donor/acceptor signals	encode-train, augustus-train, SpliceDB
Start/stop signals	encode-train, augustus-train
Signal peptide	augustus-train-signalpeptide

Table 5.2: **Training sets for human sequences.** More detailed description of the data sets used for training can be used in Appendix A.

	Exon		**Intron**		**Nucleotide**	
Gene finder	**Sn**	**Sp**	**Sn**	**Sp**	**Sn**	**Sp**
Genscan	59%	37%	63%	39%	85%	44%
GeneMark	45%	26%	60%	38%	77%	37%
ExonHunter (no repeats)	55%	39%	75%	51%	79%	52%
geneid	47%	59%	77%	62%	74%	78%
Genezilla	62%	50%	55%	61%	86%	50%
Augustus	52%	63%	34%	82%	78%	75%
ExonHunter (repeats)	57%	51%	75%	55%	79%	72%

Table 5.3: **Comparison of gene finding programs on ENCODE testing set with ExonHunter.** The exon, intron, and nucleotide statistics were computed from the files submitted by program authors to EGASP workshop, except for Genscan and ExonHunter. The top of the table shows programs that did not use information about masked repeats, the bottom of the table shows programs that work on masked sequences or take the repeat information into account.

Gene finder	Acceptor		Donor	
	Sn	**Sp**	**Sn**	**Sp**
Genscan	72%	45%	72%	46%
GeneMark	62%	32%	61%	29%
ExonHunter (no repeats)	66%	49%	68%	50%
geneid	63%	69%	62%	64%
Genezilla	73%	53%	73%	54%
Augustus	62%	68%	62%	69%
ExonHunter (repeats)	67%	63%	69%	66%

Table 5.4: Comparison of accuracy of signal predictions in ENCODE testing set

ExonHunter without the use of the repeat advisor. The second group consists of the programs geneid (Parra et al., 2000), Genezilla (Majoros et al., 2004), and Augustus (Stanke and Waack, 2003).

From the results at the exon and nucleotide level it seems that the three programs, Genezilla, Augustus, and ExonHunter have comparable performance, each reaching a different balance between sensitivity and specificity. Genezilla shows high sensitivity and low specificity, while Augustus covers the other end of the spectrum. An interesting observation can be made at the intron level, where ExonHunter shows very high sensitivity, together with geneid. This means that ExonHunter performs very well in correctly chaining exons together, while its performance lacks in predicting correct exons. This can be explained by ExonHunter's tendency to split genes: ExonHunter predicts on average 4.65 exons per gene, while the correct annotation has average 8.11 exons per gene. When a gene is split, there are at least two exons where the splice sites were replaced by incorrectly predicted start sites and stop sites, thus lowering both exon sensitivity and specificity. Still, ExonHunter has a very good performance in predicting both donor and acceptor splice sites (see Table 5.4), so the predictions of start sites and stop sites likely need to be improved.

5.4 Performance of ExonHunter on Fruit Fly Sequences

We also trained ExonHunter for sequences of *Drosophila Melanogaster* (fruit fly), using annotation and the masked sequence of chromosome 3L as a training set. Then we evaluated ExonHunter on the sequence of chromosome 2L and compared the results to predictions of Genscan and Augustus. The predictions of Augustus for the fruit fly genome are available from the author's web site (Stanke, 2005). The authors of Genscan did not train their program on the fruit fly genome, and they recommend using human genome parameters instead. This results in low performance for Genscan on this data set, and we include these results for reference only. Predictions of other programs mentioned in previous section cannot be easily obtained for this data set.

More information on the data sets we used can be found in Appendix A. Table 5.5 shows

	Gene		Exon		Intron		Nucleotide	
Gene finder	**Sn**	**Sp**	**Sn**	**Sp**	**Sn**	**Sp**	**Sn**	**Sp**
Genscan	24%	19%	60%	42%	60%	35%	95%	69%
ExonHunter	39%	36%	73%	66%	59%	59%	96%	92%
Augustus	34%	43%	64%	74%	67%	66%	86%	97%

Table 5.5: Accuracy comparison on fruit fly chromosome 2L

	Acceptor		Donor		Start		Stop	
Gene finder	**Sn**	**Sp**	**Sn**	**Sp**	**Sn**	**Sp**	**Sn**	**Sp**
Genscan	78%	53%	74%	54%	54%	40%	65%	38%
ExonHunter	89%	79%	82%	78%	66%	57%	84%	56%
Augustus	78%	88%	73%	88%	56%	66%	77%	71%

Table 5.6: Comparison of signal predictions on fruit fly chromosome 2L

the results of the experiment.

Again, we can see that the predictions of ExonHunter are comparable to Augustus, with the predictions of Augustus achieving high specificity, and the predictions of ExonHunter achieving high sensitivity. This is also an apparent trend in the prediction of signals (Table 5.6).

5.5 Summary

In this chapter, we provided the implementation details of our *ab initio* gene finding program ExonHunter and compared its features to two other gene finding programs: Genscan (Burge, 1997) and Augustus (Stanke and Waack, 2003). We used ExonHunter in our experiments in Chapters 2 and 3 to evaluate the effect of our new models for signal recognition and for length distributions on gene finding accuracy. In this chapter, we compared the performance of ExonHunter to other gene finders on the testing sets from human and fruit fly. ExonHunter's performance is comparable to the performance of the best *ab initio* gene finders. Compared to Augustus (Stanke and Waack, 2003), ExonHunter provides higher sensitivity, but lower specificity. The situation is reversed with respect to the gene finder Genezilla (Majoros et al., 2004).

Chapter 6

Conclusion

In this thesis we explored several methods for improving hidden Markov models for biological sequence analysis, with focus on the problem of gene finding. We demonstrated on three problems related to gene finding that it is possible to create better models of biological sequences by extending the probabilistic modeling techniques available within the framework of hidden Markov models to better reflect specific properties of biological sequences. In these problems, we concentrated on exploring the trade-off between increased model faithfulness introduced by a particular extension of HMMs and a variety of limiting factors.

First, we explored the modeling of biological signals. We introduced a new class of generative models, called higher order trees (HOT), that can capture dependencies between non-adjacent positions within these signals. Such models are particularly useful for modeling donor splice sites in gene finding, and incorporating them into a gene finder helps to improve not only the prediction of donor sites, but of start sites and stop sites as well. By increasing the complexity of HOT models, we can design models that are more faithful to reality. However, such an increase in model complexity also requires more training data. This introduces a trade-off between the model faithfulness and the amount of available training data, which in our case can be adjusted by changing the order of the HOT models.

In the second problem of modeling length distributions, we introduced new methods that are tailored specifically to biological sequences and gene finding. A common method of generalizing hidden Markov models to incorporate non-geometric length distributions is not usable in the case of gene finding, since such increased modeling ability is accompanied by an increase in the running time of the decoding algorithm from linear to quadratic in the length of the analyzed sequence. Quadratic running time is not practical for long DNA sequences. We noticed that elements in biological sequences are well approximated by geometric-tail length distributions, for which we also designed a more efficient decoding algorithm that is linear in the length of the sequences. A new parameter t of the distribution introduces a trade-off between how well the geometric-tail distributions fits the real data, and the running time of the decoding algorithm.

Third, we explored the problem of decoding complex HMM topologies. The common method for decoding hidden Markov models is to find the most probable state path through the model with the Viterbi algorithm. However, we argued that this method of decoding

is not appropriate for complex topologies, where multiple paths through the model may correspond to the same sequence annotation. We suggest to replace the most probable path decoding approach with a different formulation of the decoding problem, where we seek the most probable annotation. We proved that for some HMMs, finding the most probable annotation is NP-hard. However, on the positive side, we designed an algorithm that can decode many HMMs relevant for biological applications in quadratic time. We showed that we can use increasingly slower decoding algorithms to find the most probable annotations for increasingly wider classes of HMMs, though we cannot give such a scheme for all HMMs. These slower decoding algorithms are not practical for gene finding. However, the analysis of this problem helped us to reject some modifications that would introduce the multiple path problem into our gene finder. The most probable annotation problem introduces a trade-off between model faithfulness achieved by more complex models and tractability of underlying optimization problems.

Finally, we also designed a new gene finder, ExonHunter, that we used to test our ideas. ExonHunter's performance is competitive compared to other gene finders developed in parallel with our work. In our study of gene finding and hidden Markov models, we found that it is important to extend established methods in directions that take into account specifics of a particular problem and data. We believe that our extensions to hidden Markov models will be useful in their other applications both in and outside of bioinformatics.

Appendix A

Datasets and Their Preparation

In this Appendix we describe the sources of the training and testing data sets used in the thesis, as well as the steps we have taken in their preparation.

A.1 ENCODE Gene Prediction Workshop

The ENCODE project (Encyclopedia of DNA Elements) is a two-phase project with the aim of producing a high quality annotations of all functional elements in the human genome (ENCODE Project Consortium, 2004). In the first phase of the project, approximately 1% of the human genome (44 regions selected partly manually and partly at random) was selected for detailed analysis. In an attempt to evaluate the current state of gene prediction, the HAVANA annotation team from the Sanger Institute prepared a high-quality manual annotation of these 44 regions and solicited gene predictions from research groups working on the problem of gene finding (Guigó and Reese, 2005). The data can be downloaded from the workshop web page (Ashurst et al., 2005).

The annotation of the 44 regions was prepared and released in two phases. First, the *ENCODE training set*, which we refer to as **encode-training**, includes 13 regions of total length 9 MB, containing 137 genes and 1 597 unique coding exons. Since this set contains alternatively spliced genes, for the purpose of the training we reduce the set, selecting non-overlapping genes that cover the largest portion of the sequences.

Second, the *ENCODE testing set* (**encode-testing**) contains the remaining 31 regions of total length 21 MB, with 296 genes and 2 782 unique coding exons.

We also created a smaller test set (**encode-small**) for the evaluation of partial implementations of ExonHunter. This set contains three of the randomly selected encode-testing regions (ENr131, ENr233, ENr332) of total length 1.5 MB, with 41 genes and 533 unique exons.

A.2 Chromosome 22 Annotated with RefSeq

We downloaded the sequence of chromosome 22 (hg17, May 2004 assembly) and its annotation, based on the manually curated RefSeq database (Pruitt et al., 2005) from the UCSC human genome browser (Karolchik et al., 2003), using the version current to July 2005. We split the chromosome 22 sequence into two halves, with the first half designated as *chromosome 22 training set* (**chr22-training**), and the second half designated as the *chromosome 22 testing set* (**chr22-testing**).

The training set was further reduced to contain only non-overlapping genes chosen for the maximum coverage of the sequence. The length of the training set is 18 MB, with 196 genes, and 1 786 coding exons. The testing set has length 17 MB, with 220 genes, and 1 913 coding exons.

A.3 Augustus Training Set

The Augustus training set is a set of 1 284 single gene sequences, assembled by Stanke and Waack (2003). The data set was used to train their gene finder Augustus. We excluded 81 genes from the data set which overlapped with the ROSETTA testing set (Alexandersson et al., 2003), which we used in the preliminary evaluation of our gene finder (Brejová et al., 2005). The resulting data set has 1 203 genes, with 6 151 exons in 11 MB of sequence.

A.4 SpliceDB Collection of Splice Site Signals

SpliceDB is a database of mammalian splice sites (Burset et al., 2000). We downloaded 15 263 donor/acceptor site pairs supported by human ESTs on May 26, 2005. For the purpose of signal experiments, we further excluded 705 splice site sequences that showed similarity to the chr22-testing dataset.

A.5 Fruit Fly Datasets

We downloaded fruit fly genome sequences and their RefSeq annotations from the UCSC genome browser (Karolchik et al., 2003) on July 11, 2005 (sequence assembly April 2004). We used chromosome 3L and 3R, with its annotation reduced to non-overlapping genes as a training set (**drome-training**) and chromosome 2L as a testing set (**drome-testing**). The training set has 52 MB with 5 835 genes and 22 578 exons. The testing set has 22 MB with 3 380 genes and 10 021 exons. Finally, for experiments with length distributions, we also created a small subset of the drome-testing set (**drome-small**), which has 2 MB with 357 genes and 1 187 exons.

Bibliography

Abe, N. and Warmuth, M. K. (1992). On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9(2-3):205–260.

Adams, M. D., Kelley, J. M., Gocayne, J. D., Dubnick, M., Polymeropoulos, M. H., Xiao, H., Merril, C. R., Wu, A., Olde, B., and Moreno, R. F. (1991). Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–1656.

Agarwal, P. and Bafna, V. (1998). Detecting non-adjoining correlations within signals in DNA. In *RECOMB 1998: Proceedings of the 2nd Annual International Conference on Research in Computational Molecular Biology (RECOMB 1998)*, pages 2–8. ACM Press.

Alexandersson, M., Cawley, S., and Pachter, L. (2003). SLAM: cross-species gene finding and alignment with a generalized pair hidden markov model. *Genome Research*, 13(3):496–502.

Altun, Y., Tsochantaridis, I., and Hofmann, T. (2003). Hidden Markov support vector machines. In *ICML 2003: 20th International Conference on Machine Learning*, pages 3–10. AAAI Press.

Ashurst, J., Birney, E., Good, P., Guigó, R., and Hubbard, T. (2005). ENCODE gene prediction workshop. `http://genome.imim.es/gencode/workshop2005.html`.

Asmussen, S., Nerman, O., and Olsson, M. (1996). Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics*, 23(4):419–441.

Barash, Y., Elidan, G., Friedman, N., and Kaplan, T. (2003). Modeling dependencies in protein-DNA binding sites. In *RECOMB 2003: Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 28–37, New York, NY, USA. ACM Press.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In *Inequalities III, Proceeding of the Third Symposium*, pages 1–8. Academic Press, New York.

Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73:360–363.

Bernal, A., Ear, U., and Kyrpides, N. (2001). Genomes online database (GOLD): a monitor of genome projects world-wide. *Nucleic Acids Research*, 29(1):126–127.

Besemer, J. and Borodovsky, M. (2005). GeneMark: web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Research*, 33(Web Server issue):W451–454.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *COLT 1992: Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press.

Brejová, B. (2005). *Evidence Combination in Hidden Markov Models for Gene Prediction*. PhD thesis, School of Computer Science, University of Waterloo.

Brejová, B., Brown, D. G., Li, M., and Vinař, T. (2005). ExonHunter: a comprehensive approach to gene finding. *Bioinformatics*, 21(Suppl 1):i57–i65. Intelligent Systems for Molecular Biology (ISMB 2005).

Brejová, B., Brown, D. G., and Vinař, T. (2003). Optimal DNA signal recognition models with a fixed amount of intrasignal dependency. In Benson, G. and Page, R., editors, *WABI 2003: Algorithms and Bioinformatics: 3rd International Workshop*, volume 2812 of *Lecture Notes in Bioinformatics*, pages 78–94, Budapest, Hungary. Springer.

Brejová, B., Brown, D. G., and Vinař, T. (2004a). The most probable labeling problem in HMMs and its applications to bioinformatics. In Jonassen, I. and Kim, J., editors, *WABI 2004: Algorithms in Bioinformatics*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 426–437, Bergen, Norway. Springer.

Brejová, B., Brown, D. G., and Vinař, T. (2004b). Optimal spaced seeds for homologous coding regions. *Journal of Bioinformatics and Computational Biology*, 1(4):595–610. Early version appeared in CPM 2003.

Brejová, B. and Vinař, T. (2002). A better method for length distribution modeling in HMMs and its application to gene finding. In Apostolico, A. and Takeda, M., editors, *CPM 2002: Combinatorial Pattern Matching, 13th Annual Symposium*, volume 2373 of *Lecture Notes in Computer Science*, pages 190–202, Fukuoka, Japan. Springer.

Brown, R. H., Gross, S. S., and Brent, M. R. (2005). Begin at the beginning: predicting genes with 5' UTRs. *Genome Research*, 15(5):742–747.

Burge, C. and Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78–94.

Burge, C. B. (1997). *Identification of Genes in Human Genomic DNA*. PhD thesis, Department of Mathematics, Stanford University.

Burge, C. B. (1998). Modeling dependencies in pre-mRNA splicing signals. In Salzberg, S. L., Searls, D. B., and Kasif, S., editors, *Computational Methods in Molecular Biology*, pages 129–164. Elsevier, Amsterdam.

Burset, M., Seledtsov, I. A., and Solovyev, V. V. (2000). Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic Acids Research*, 28(21):4364–4365.

Bystroff, C., Thorsson, V., and Baker, D. (2000). HMMSTR: a hidden Markov model for local sequence-structure correlations in proteins. *Journal of Molecular Biology*, 301(1):173–180.

Cai, D., Delcher, A., Kao, B., and Kasif, S. (2000). Modeling splice sites with Bayes networks. *Bioinformatics*, 16(2):152–158.

Castelo, R. and Guigó, R. (2004). Splice site identification by idlBNs. *Bioinformatics*, 20(S1):i69–i76. Intelligent Systems for Molecular Biology (ISMB 2004).

Chaudhuri, S. and Zaroliagis, C. D. (2000). Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27(3):212–226.

Chellappa, R. and Jain, A. (1993). Academic Press.

Cheng, J. et al. (2005). Transcriptional maps of 10 human chromosomes at 5-nucleotide resolution. *Science*, 308(5725):1149–1154.

Chickering, D. M. (1995). A transformational characterization of equivalent Bayesian network structures. In *UAI 1995: Proceedings of 11th Conference on Uncertainty in Artificial Intelligence*, pages 87–98. Morgan Kaufmann.

Chimpanzee Sequencing and Analysis Consortium (2005). Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437(7055):69–87.

Chow, C. K. and Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467.

Chuang, J. S. and Roth, D. (2001). Gene recognition based on DAG shortest paths. *Bioinformatics*, 17(S1):S56–S64.

Commault, C. and Mocanu, S. (2003). Phase-type distributions and representations: some resuts and open problems for system theory. *International Journal of Control*, 76(6):566–580.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press.

Crooks, G. E., Hon, G., Chandonia, J.-M., and Brenner, S. E. (2004). WebLogo: a sequence logo generator. *Genome Research*, 14(6):1188–1190.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1–38.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.

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

Eddy, S. R. (1998). Profile hidden Markov models. *Bioinformatics*, 14(9):755–763.

Ellrott, K., Yang, C., Sladek, F. M., and Jiang, T. (2002). Identifying transcription factor binding sites through Markov chain optimization. In *ECCB 2002: Proceedings of the European Conference on Computational Biology*, pages 100–109.

ENCODE Project Consortium (2004). The ENCODE (ENCyclopedia Of DNA Elements) project. *Science*, 306(5696):636–640.

Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673.

Eyras, E., Reymond, A., Castelo, R., Bye, J. M., Camara, F., Flicek, P., Huckle, E. J., Parra, G., Shteynberg, D. D., Wyss, C., Rogers, J., Antonarakis, S. E., Birney, E., Guigó, R., and Brent, M. R. (2005). Gene finding in the chicken genome. *BMC Bioinformatics*, 6(1):131.

Fariselli, P., Martelli, P. L., and Casadio, R. (2005). The posterior-Viterbi: a new decoding algorithm for hidden Markov models. Technical Report q-bio.BM/0501006, arXiv Quantitative Biology. `http://arxiv.org/abs/q-bio.BM/0501006`.

Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278.

Freund, Y. and Ron, D. (1995). Learning to model sequences generated by switching distributions. In *COLT 1995: Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 41–50. ACM Press.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29:131–163.

Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201.

Gilbert, S. F. (1997). The biochemisty of pre-mRNA splicing. `http://zygote.swarthmore.edu/rna2.html`.

Gillman, D. and Sipser, M. (1994). Inference and minimization of hidden Markov chains. In *COLT 1994: Proceedings of the 7th Annual Conference on Computational Learning Theory*, pages 147–158. ACM Press.

Gross, S. S. and Brent, M. R. (2005). Using multiple alignments to improve gene prediction. In *RECOMB 2005: 9th Annual International Conference on Research in Computational Molecular Biology*, volume 3500 of *Lecture Notes in Bioinformatics*, pages 374–388. Springer.

Guigó, R. (1998). Assembling genes from predicted exons in linear time with dynamic programming. *Journal of Computational Biology*, 5(4):681–702.

Guigó, R. and Reese, M. G. (2005). EGASP: collaboration through competition to find human genes. *Nature Methods*, 2(8):575–577.

Hajarnavis, A., Korf, I., and Durbin, R. (2004). A probabilistic model of 3' end formation in *Caenorhabditis elegans*. *Nucleic Acids Research*, 32(11):3392–3399.

Heckerman, D. (1999). A tutorial on learning with Bayesian networks. In Jordan, M., editor, *Learning in Graphical Models*. MIT Press.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):192–243.

Henzel, W. J., Billeci, T. M., Stults, J. T., Wong, S. C., Grimley, C., and Watanabe, C. (1993). Identifying proteins from two-dimensional gels by molecular mass searching of peptide fragments in protein sequence databases. *Proceedings of the National Academy of Sciences of the USA*, 90(11):5011–5015.

ILOG Inc. (2003). CPLEX optimizer version 8.1. Computer software.

International Chicken Genome Sequencing Consortium (2004). Sequence and comparative analysis of the chicken genome provide unique perspectives on vertebrate evolution. *Nature*, 432(7018):695–716.

International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.

International Human Genome Sequencing Consortium (2004). Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–935.

Iseli, C., Jongeneel, C. V., and Bucher, P. (1999). ESTScan: a program for detecting, evaluating, and reconstructing potential coding regions in EST sequences. In *ISMB 1999: Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 138–148.

Jelinek, F. (1968). *Probabilistic Information Theory*. Mcgraw-Hill.

Johnson, M. T. (2005). Capacity and complexity of HMM duration modeling techniques. *IEEE Signal Processing Letters*, 12(5):407–410.

Kall, L., Krogh, A., and Sonnhammer, E. L. L. (2005). An HMM posterior decoder for sequence feature prediction that includes homology information. *Bioinformatics*, 21(S1):i251–i257. Intelligent Systems for Molecular Biology (ISMB 2005).

Karolchik, D., Baertsch, R., Diekhans, M., Furey, T. S., Hinrichs, A., Lu, Y. T., Roskin, K. M., Schwartz, M., Sugnet, C. W., Thomas, D. J., Weber, R. J., Haussler, D., and Kent, W. J. (2003). The UCSC genome browser database. *Nucleic Acids Research*, 31(1):51–54.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103, New York. Plenum Press.

Keibler, E. and Brent, M. R. (2003). Eval: a software package for analysis of genome annotations. *BMC Bioinformatics*, 4:50.

Klein, D. and Manning, C. D. (2002). Conditional structure versus conditional estimation in NLP models. In *EMNLP 2002: Conference on Empirical Methods in Natural Language Processing*, pages 9–16. Association for Computational Linguistics.

Korf, I., Flicek, P., Duan, D., and Brent, M. R. (2001). Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17(S1):S140–S148.

Krogh, A. (1997). Two methods for improving performance of an HMM and their application for gene finding. In *ISMB 1997: Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology*, pages 179–186.

Krogh, A., Larsson, B., von Heijne, G., and Sonnhammer, E. L. (2001). Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *Journal of Molecular Biology*, 305(3):567–570.

Kulp, D., Haussler, D., Reese, M. G., and Eeckman, F. H. (1996). A generalized hidden Markov model for the recognition of human genes in DNA. In *ISMB 1996: Proceedings of the 4th International Conference on Intelligent Systems for Molecular Biology*, pages 134–142.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001: 18th International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

Lee, T. I. et al. (2002). Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science*, 298(5594):799–804.

Lim, L. P. and Burge, C. B. (2001). A computational analysis of sequence features involved in recognition of short introns. *Proceedings of the National Academy of Sciences USA*, 98(20):11193–11198.

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

Majoros, W. H., Pertea, M., and Salzberg, S. L. (2004). TigrScan and GlimmerHMM: two open source *ab initio* eukaryotic gene-finders. *Bioinformatics*, 20(16):2878–2879.

Martelli, P. L., Fariselli, P., Krogh, A., and Casadio, R. (2002). A sequence-profile-based HMM for predicting and discriminating beta barrel membrane proteins. *Bioinformatics*, 18(S1):S46–53.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157.

Metz, C. E. (1978). Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283–288.

Mitchell, C., Harper, M., and Jamieson, L. (1995). On the complexity of explicit duration HMMs. *IEEE Transactions on Speech and Audio Processing*, 3(3):213–217.

Mouse Genome Sequencing Consortium (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–522.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *NIPS 2002: Advances in Neural Information Processing Systems*, pages 841–848, Cambridge, MA. MIT Press.

Ohler, U., Niemann, H., and Rubin, G. M. (2001). Joint modeling of DNA sequence and physical properties to improve eukaryotic promoter recognition. *Bioinformatics*, 17(S1):S199–206.

Parra, G., Blanco, E., and Guigó, R. (2000). GeneID in *Drosophila*. *Genome Research*, 10(4):511–515.

Perkins, D. N., Pappin, D. J., Creasy, D. M., and Cottrell, J. S. (1999). Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3557.

Pontius, J. U., Wagner, L., and Schuler, G. D. (2002). UniGene: A unified view of the transcriptome. In *NCBI Handbook*, chapter 21. The National Library of Medicine.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401.

Pruitt, K. D., Tatusova, T., and Maglott, D. R. (2005). NCBI reference sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, 33(Database issue):D501–4.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285.

Rat Genome Sequencing Project Consortium (2004). Genome sequence of the Brown Norway rat yields insights into mammalian evolution. *Nature*, 428(6982):493–521.

Reese, M. G., Kulp, D., Tammana, H., and Haussler, D. (2000). Genie—gene finding in *Drosophila melanogaster*. *Genome Research*, 10(4):529–538.

Salamov, A. A. and Solovyev, V. V. (2000). *Ab initio* gene finding in *Drosophila* genomic DNA. *Genome Research*, 10(4):516–522.

Salzberg, S. L., Delcher, A. L., Kasif, S., and White, O. (1998). Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26(2):544–548.

Schneider, T. D. and Stephens, R. M. (1990). Sequence logos: a new way to display consensus sequences. *Nucleic Acids Research*, 18(20):6097–6100.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley and sons.

Schwartz, R. and Chow, Y.-L. (1990). The N-best algorithms: an efficient and exact procedure for finding the N most likely sentence hypotheses. In *ICASSP 1990: Acoustics, Speech, and Signal Processing*, pages 81–84, vol. 1.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423,623–656.

Siepel, A. and Haussler, D. (2004). Computational identification of evolutionarily conserved exons. In *RECOMB 2004: 8th Annual International Conference on Research in Computational Molecular Biology*, pages 177–186, New York, NY, USA. ACM Press.

Smit, A. F. A., Hubley, R., and Green, P. (2002). RepeatMasker. `http://www.repeatmasker.org`.

Solovyev, V. V. (2002). Finding genes by computer: Probabilistic and discriminative approaches. In Jiang, T., Xu, Y., and Zhang, M. Q., editors, *Current Topics in Molecular Biology*, pages 201–248. The MIT Press.

Solovyev, V. V., Salamov, A. A., and Lawrence, C. B. (1994). Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Research*, 22(24):5156–5163.

Solovyev, V. V., Salamov, A. A., and Lawrence, C. B. (1995). Identification of human gene structure using linear discriminant functions and dynamic programming. In *ISMB 1995: Proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology*, volume 3, pages 367–375.

Sonnenburg, S., Rätsch, G., and Schäfer, C. (2005). Learning interpretable SVMs for biological sequence classification. In *RECOMB 2005: Research in Computational Molecular Biology, 9th Annual International Conference*, volume 3500 of *Lecture Notes in Computer Science*, pages 389–407. Springer.

Staden, R. (1984). Computer methods to aid the determination and analysis of DNA sequences. *Biochemical Society Transactions*, 12(6):1005–1008.

Stanke, M. (2005). Augustus gene prediction web site. `http://augustus.gobics.de/`.

Stanke, M., Schöffmann, O., Morgenstern, B., and Waack, S. (2005). Gene prediction in eukaryotes with a generalized hidden Markov model that takes hints. Submitted for publication.

Stanke, M. and Waack, S. (2003). Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*, 19(S2):II215–II225.

Stormo, G. D. (2000). Gene-finding approaches for eukaryotes. *Genome Research*, 10(4):394–397.

Stormo, G. D., Schneider, T. D., Gold, L., and Ehrenfeucht, A. (1982). Use of the perceptron algorithm to distinguish translational initiation sites in *E. coli*. *Nucleic Acids Research*, 10(9):2997–3011.

Venter, J. C. et al. (2001). The sequence of the human genome. *Science*, 291(5507):1304–1311.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267.

Xu, L., Cheng, L., Wang, T., and Schuurmans, D. (2005). Convex hidden Markov models. Submitted.

Xu, Y., Einstein, J. R., Mural, R. J., Shah, M., and Uberbacher, E. C. (1994). An improved system for exon recognition and gene modeling in human DNA sequences. In *ISMB 1994: Proceeding of the 2nd International Conference on Intelligent Systems for Molecular Biology*, pages 376–384.

Yeh, R. F., Lim, L. P., and Burge, C. B. (2001). Computational inference of homologous gene structures in the human genome. *Genome Research*, 11(5):803–806.

Yeo, G. and Burge, C. B. (2003). Maximum entropy modeling of short sequence motifs with applications to RNA splicing signals. In *RECOMB 2003: Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 322–331, New York, NY, USA. ACM Press.

Zhang, M. Q. (1997). Identification of protein coding regions in the human genome by quadratic discriminant analysis. *Proceedings of the National Academy of Sciences of the USA*, 94(2):565–568.

Zhang, M. Q. (1998). Statistical features of human exons and their flanking regions. *Human Molecular Genetics*, 7(5):919–922.

Zhang, Z. and Henzel, W. J. (2004). Signal peptide prediction based on analysis of experimentally verified cleavage sites. *Protein Science*, 13(10):2819–2824.

Zhao, X., Huang, H., and Speed, T. P. (2004). Finding short DNA motifs using permuted markov models. In *RECOMB 2004: Proceedings of the 8th Annual International Conference on Resaerch in Computational Molecular Biology*, pages 68–75, New York, NY, USA. ACM Press.

Zien, A., Rätsch, G., Mika, S., Scholkopf, B., Lengauer, T., and Muller, K. R. (2000). Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807.