Evidence Combination in Hidden Markov Models for Gene Prediction

by

Bronislava Brejová

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

Abstract

This thesis introduces new techniques for finding genes in genomic sequences. Genes are regions of a genome encoding proteins of an organism. Identification of genes in a genome is an important step in the annotation process after a new genome is sequenced. The prediction accuracy of gene finding can be greatly improved by using experimental evidence. This evidence includes homologies between the genome and databases of known proteins, or evolutionary conservation of genomic sequence in different species.

We propose a flexible framework to incorporate several different sources of such evidence into a gene finder based on a hidden Markov model. Various sources of evidence are expressed as partial probabilistic statements about the annotation of positions in the sequence, and these are combined with the hidden Markov model to obtain the final gene prediction. The opportunity to use partial statements allows us to handle missing information transparently and to cope with the heterogeneous character of individual sources of evidence. On the other hand, this feature makes the combination step more difficult. We present a new method for combining partial probabilistic statements and prove that it is an extension of existing methods for combining complete probability statements. We evaluate the performance of our system and its individual components on data from the human and fruit fly genomes.

The use of sequence evolutionary conservation as a source of evidence in gene finding requires efficient and sensitive tools for finding similar regions in very long sequences. We present a method for improving the sensitivity of existing tools for this task by careful modeling of sequence properties. In particular, we build a hidden Markov model representing a typical homology between two protein coding regions and then use this model to optimize a component of a heuristic algorithm called a spaced seed. The seeds that we discover significantly improve the accuracy and running time of similarity search in protein coding regions, and are directly applicable to our gene finder.

Acknowledgements

I would like to thank my supervisors Ming Li and Dan Brown for their support, encouragement, and guidance. Dan Brown carefully read many drafts of this thesis and his comments have greatly improved the presentation. Thanks to my spouse Tomáš Vinař for collaborating with me on this research project and for his love, care, and support.

I would also like to thank members of my committee Ian Munro, Dale Schuurmans, Mary Thompson, and Franco Preparata for their time. Special thanks to Dale Schuurmans for asking difficult questions and for many helpful discussions about machine learning. Therese Biedl also read the thesis and provided useful comments.

Thanks to many people at the University of Waterloo for inspiration, advice, encouragement and a great open atmosphere. Therese Biedl was a coauthor of my first research paper and taught me a lot in the process. Together with Erik Demaine they organized problem solving sessions that spread contagious enthusiasm for research. Jonathan Buss and Paul Kearney have provided support during the absence of my supervisor. Also thanks to Jianwei Niu, Mike Hu, Alex Hudek, and Mirela Andronescu for being great office mates.

Finally, I would like to thank my parents for their love and for encouraging my interest in science and mathematics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

New high-throughput technologies allowed scientists to produce the first draft of the whole human genome in 2001 [51]. This result was preceded by the sequencing of genomes of many viruses and bacteria, as well as higher organisms such as the fruit fly *Drosophila melanogaster* [2] and flowering plant *Arabidopsis thaliana* [10]. The sequencing technology developed in these landmark achievements is today used in hundreds of other sequencing projects.

The sequence of the human genome contains over three billion building blocks called nucleotides, which are typically represented by letters A, C, G, and T. Due to its sheer size, it cannot be manually analyzed by humans. Instead, biologists rely on an annotation of the genomic sequence that highlights functionally important regions. In this thesis, we study two important components of the genome annotation process.

The first component is gene finding. Here, the goal is to identify the regions of the genome that encode the proteins produced by the cells of an organism. Once we know the exact boundaries of a gene, we can infer the sequence of its protein and study its function in more depth. We can also search the regions surrounding the gene for the special sequences involved in the regulation of the protein's production.

The second component aims at identifying the similarities between different parts of one genome or between two different genomes. Such similar sequences have often evolved from the same ancestral sequence by a series of mutations. Therefore, by studying sequence similarity, we can explore evolutionary relationships between organisms and processes that shape them. Also, similar sequences often have similar biological function, and thus we can transfer functional knowledge from one genome to another.

The contributions of this thesis lie at the intersection of these two important problems. We study how to use sequence similarity information and other external sources of evidence to improve the accuracy of gene finding. Although many gene finding systems successfully use one source of such information, the systematic combination of many diverse evidence sources is still an open problem. In Chapter 2, we present a new flexible framework for expressing and combining diverse sources of information that are appropriate for the problem of gene finding. Our framework allows us to express missing and heterogeneous information in a very natural way, as partial probabilistic statements. We develop a new method for combining such statements and resolving conflicts among them. We show that our combination method generalizes existing methods for combining complete probability statements to the case of partial statements.

In Chapter 3, we study the problem of finding similar sequences in large sequence databases. We improve a popular heuristic algorithm by building realistic probabilistic models of target sequence similarities and using them to adjust a component of the algorithm called its spaced seed. Our new spaced seeds allow us to find more sequence similarities inside protein coding regions, which in turn provide more useful evidence for gene finding.

Finally in Chapter 4, we describe our gene finder, ExonHunter. It uses the evidence representation framework introduced in Chapter 2 to find genes in the genomic sequences of higher organisms. We evaluate its performance on real genomic sequences from the human and fruit fly genomes. Throughout the thesis, we attempt to develop methods closely tailored to the biological properties of the sequences we are modeling.

In the rest of this chapter, we introduce the problem of gene finding and survey existing work in this area. We also introduce hidden Markov models, which form the basis of our gene finder and are also used to model the properties of sequence similarities in Chapter 3.

1.1 The problem of eukaryotic gene finding

In this thesis, we consider the prediction of protein-coding genes in eukaryotic organisms. A *gene* is a region of a DNA sequence encoding a protein. For our purposes, a DNA molecule can be represented as a sequence whose symbols come from the four different nucleotides $\{A, C, G, T\}$; a protein can be represented as a sequence whose symbols come from the 20 different amino acids. *Eukaryotic organisms* are higher organisms, such as animals, plants, and fungi. Their genes have a different and more complicated structure than the genes of prokaryotic organisms, such as bacteria.

The process of protein synthesis in eukaryotic cells works as follows (see also Figure 1.1). First, a gene is copied (transcribed) to a molecule of *messenger RNA* (mRNA). The mRNA is then modified by a process called splicing, in which some regions of the mRNA sequence, called *introns*, are removed and the remaining parts, called *exons*, are joined together. In the final phase, the mRNA is used as a template for protein synthesis. Protein is encoded in mRNA using the genetic code (shown in Figure 1.2), which translates triplets of nucleotides (called *codons*) to individual amino acids. Many amino acids are encoded by several different codons, as there are $4^3 = 64$ codons, but only 20 amino acids. Three of the 64 codons do not encode any amino acid and instead mark the end of the protein. Each protein sequence starts with the amino acid methionine, encoded by the codon ATG. Unlike stop codons, this codon may also occur inside a gene. Also note that regions at both ends of the spliced mRNA, called *untranslated regions* (UTRs), are not used for translation.

Thus, when we map the regions that are used as codons in the translation process back to the original DNA, they form several consecutive regions, which we call *coding regions*. These are separated by introns. Note that coding regions are sometimes also called exons in the context of gene finding, although technically some exons include portions of UTRs and are thus not entirely coding. The goal of gene prediction is to detect the coding regions of each gene in a query DNA sequence, which may contain more than one gene.

The DNA molecule has two complementary strands going in opposite directions. Gene prediction software has the sequence of one strand as its input, but genes can be located on the other strand as well. Genes on the reverse strand appear in the opposite direction with complementary symbols, as shown in Figure 1.3. In order to completely specify a coding region, we need to give its position, strand and reading frame. The *reading frame* specifies which positions, modulo 3, are the start positions of codons. Reading frame cannot be determined solely from the position of a

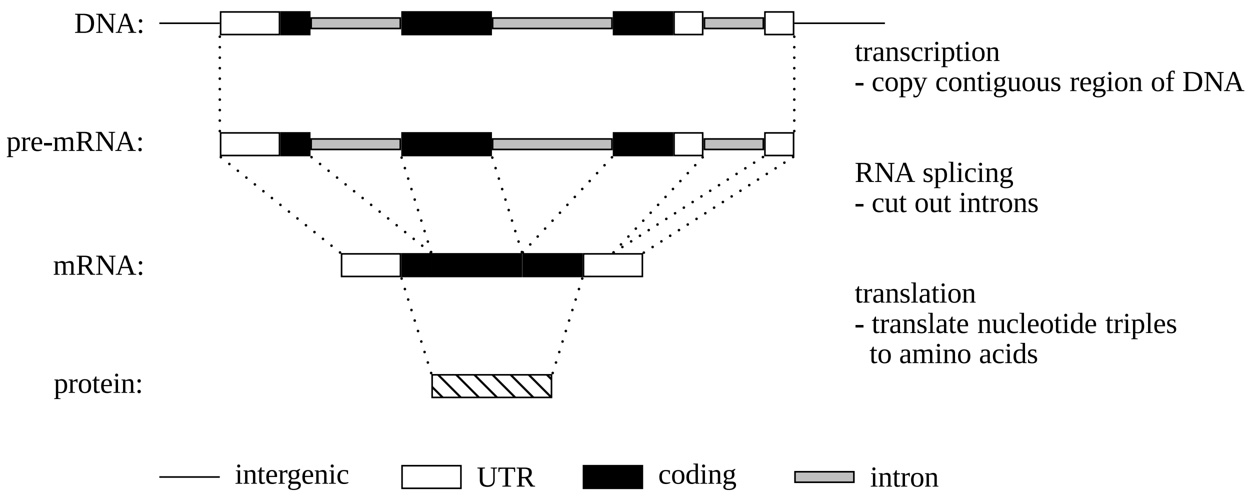


Figure 1.1: Translation of a gene to a protein.

Alanine (A)
`GC*`

Cysteine (C)
`TGC`
`TGT`

Aspartic acid (D)
`GAC`
`GAT`

Glutamic acid (E)
`GAA`
`GAG`

Phenylalanine (F)
`TTC`
`TTT`

Glycine (G)
`GG*`

Histidine (H)
`CAC`
`CAT`

Isoleucine (I)
`ATA`
`ATC`
`ATT`

Lysine (K)
`AAA`
`AAG`

Leucine (L)
`CT*`
`TTA`
`TTG`

Methionine (M)
`ATG`

Asparagine (N)
`AAC`
`AAT`

Proline (P)
`CC*`

Glutamine (Q)
`CAA`
`CAG`

Arginine (R)
`CG*`
`AGA`
`AGG`

Serine (S)
`TC*`
`AGT`
`AGC`

Threonine (T)
`AC*`

Valine (V)
`GT*`

Tryptophan (W)
`TGG`

Tyrosine (Y)
`TAC`
`TAT`

Stop codon (*)
`TAA`
`TAG`
`TGA`

Figure 1.2: The standard genetic code. Each amino acid is listed with its one-letter code and the list of the codons encoding it. When a codon is shown with a star in the third position, it indicates that all four codons that begin with the first two letters and end with either A, C, G, or T will encode that amino acid.

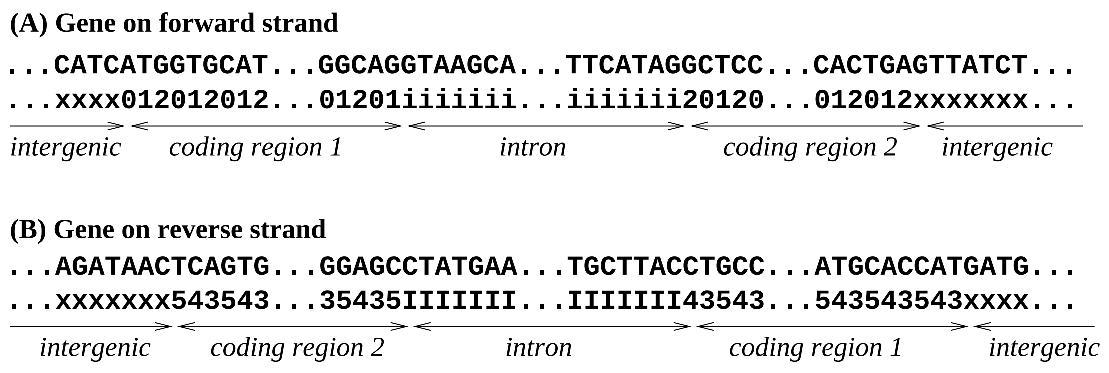

(A) Gene on forward strand
...CATCATGGTGCAT...GGCAGGTAAGCA...TTCATAGGCTCC...CACTGAGTTATCT...
...xxxx012012012...01201iiiiiii...iiiiiii20120...012012xxxxxxx...
intergenic
coding region 1
intron
coding region 2
intergenic
(B) Gene on reverse strand
...AGATAACTCAGTG...GGAGCCTATGAA...TGCTTACCTGCC...ATGCACCATGATG...
...xxxxxxx543543...35435IIIIIII...IIIIIII43543...543543543xxxx...
intergenic
coding region 2
intron
coding region 1
intergenic

Figure 1.3: (A) A two-exon gene on forward strand. The top line shows the pieces of DNA sequence surrounding coding region boundaries. The bottom line shows labeling distinguishing intergenic regions (label x), introns (label i), and positions within codon in coding regions (labels $0, 1, 2$). Note that in this labeling, we consider untranslated regions (UTRs) as parts of intergenic regions. (B) The same gene, as it would appear if it were located on the reverse strand. Note that the gene ends with the reverse complement, CAT, of the start codon ATG. Introns on reverse strand are labeled I and coding regions $3, 4, 5$.

coding region in the DNA sequence, because an intron can separate one codon to two different exons. Therefore, a coding region does not always start with a full codon. However, if all coding regions of a gene are known, reading frames can be determined based on the lengths of all previous coding regions, since every third base in the translated portion of a mature mRNA begins a codon.

Gene prediction is even more complicated. Many genes are spliced differently under different conditions (*e.g.*, in different tissues). Thus, they may not have a unique structure, but several overlapping ones. Current tools for predicting alternatively spliced variants are able to predict only those gene structures that are very well supported by high-quality evidence [65, 56]. We will take the approach of most gene predicting programs (*e.g.*, [7, 37, 95, 173]), and predict only one structure per gene. The goal is for the predicted structure to be either one of the alternative structures or at least a collection of coding regions from different structures.

Another difficulty is that two different genes may come from overlapping regions of a genome. Gene overlaps are especially prevalent in the short genomes of viruses, bacteria, and organelles, but recent reports have discovered hundreds of overlapping genes even in the human genome [163]. Still, eukaryotic gene prediction programs typically assume that genes do not overlap [136]. Some of the overlaps involve only untranslated regions and thus can be predicted correctly if we only focus on finding coding regions of the genes. Genes embedded in an intron of another gene also occur frequently. While this case cannot be successfully recognized by most gene finders, it can be solved by a post-processing step in which the sequences of all introns with any evidence of containing coding regions are submitted to the gene finder separately.

If we predict only one splicing variant per gene and ignore overlapping genes, the result of gene finding can be expressed as a labeling of the DNA sequence. Labels distinguish amongst intergenic regions, six combinations of frame and strand in coding regions, and introns on both strands. An example of such labeling is given in Figure 1.3. Under these simplifying assumptions, we can formalize the gene prediction problem as follows:

Problem 1 (Gene prediction). *Given is the query DNA sequence $X = x_1, x_2, \ldots, x_n$, and option-*

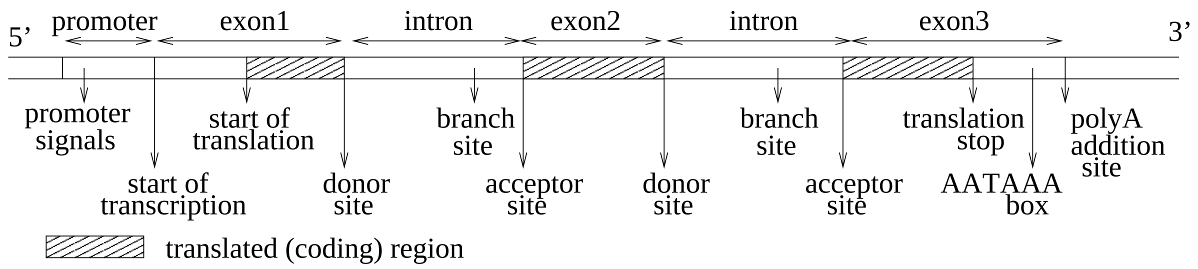


Figure 1.4: Typical signals in a multi-exon gene.

Table 1.1: Nucleotide frequencies in human donor splice site signals, estimated from the ENCODE training set described in Section 4.2. Each column shows the frequencies at one position within the signal window for donor splice sites, which separate exons and introns. The first three positions of the signal are inside the exon, the next two form the consensus string GT, and the last four positions are found inside the intron.

	End of exon			Splice site		Beginning of intron			
A	31%	**63%**	9%	0%	0%	**53%**	**69%**	7%	15%
C	**38%**	12%	3%	0%	0%	3%	8%	5%	18%
G	19%	11%	**81%**	**100%**	0%	41%	13%	**82%**	21%
T	11%	14%	7%	0%	**100%**	2%	10%	6%	**46%**

ally, sources of additional evidence. The task is to find the labeling $L = \ell_1, \ell_2, \ldots, \ell_n$ corresponding to the correct gene structure of all genes in X.

1.1.1 Properties of protein coding genes that aid gene prediction

There is no single reliable feature that would distinguish coding and non-coding regions. Instead, gene finders need to combine information from multiple clues that suggest possible locations of protein coding regions and their boundaries.

The boundaries of individual coding regions are marked with signals. Signals are short conserved sequences recognized by the protein or protein/RNA complexes involved in transcription, splicing, and translation. However, the signals involved in gene structure are usually rather weak, and sequences that are identical or very similar to real signals commonly occur at other places in the genome as well. As such, they cannot be used as the sole predictor of gene structure.

The location of the most important known classes of signals is shown in Figure 1.4 and an example of such a signal is shown in Table 1.1. An intron starts with a donor site signal and ends with an acceptor site signal. A branch point signal often occurs close to the end of introns, but its distance to the acceptor site is variable, and the signal is weak. The first coding region of a gene starts with an ATG codon, which codes for the amino acid methionine, and there is a translation start signal surrounding the ATG codon. The last coding region ends with a stop codon (one of TAA, TGA, and TAG) and a weak signal surrounds it.

The signals mentioned so far indicate the boundaries of coding regions and are used by the splicing and translation mechanisms. Additional signals, guiding the transcription mechanism, are located in the promoter region preceding the start of translation and in the untranslated region beyond the end of the last coding region of a gene.

Coding and non-coding sequences differ in their composition. By composition, we mean the distribution of all possible words of a given length k in a sequence. In addition, coding regions have three-periodic structure caused by the genetic code. This code translates triplets of nucleotides to individual amino acids. We can characterize the composition of coding regions by three different distributions, one for each reading frame. Words included in one of these three distributions start at the same position modulo 3. These three distributions differ from one another and from the distribution observed in non-coding regions.

Perhaps the most prominent feature of sequence composition inside coding regions is the absence of stop codons (triples TAA, TAG, TGA). These mark the end of the coding region and do not occur as codons inside them.[1] These triples may appear at the boundaries of two codons. For example, one codon may end with TA and the next one may start with A. Long regions without stop codons, called *open reading frames*, are good candidates for possible coding regions. For example a stretch of 100 codons without any stop codon has only $(61/64)^{100} \approx 0.008$ probability of happening by chance, if we assume that each triple is equally likely to occur. Although open reading frames are used for finding candidate genes in bacterial genomes, exons of eukaryotic genes are usually not sufficiently long to be located by such a simple method; the average human exon is less than 50 codons long [51].

Other statistical properties useful in gene prediction include the length distribution of the individual sequence features, such as coding regions or introns, and the distribution of the number of exons in a gene.

Lim and Burge [112] investigated the contribution of different features to the accuracy of intron recognition. Signals contribute over 75% of the information in simpler eukaryotic organisms. In the human genome and in the genome of the flowering plant *Arabidopsis thaliana*, signals provide only roughly 50% of the information, and thus sequence composition becomes an important factor in intron recognition.

1.2 Hidden Markov models and their algorithms

A hidden Markov model (HMM) is a generative probabilistic model for modeling sequence data over a given finite alphabet. In this thesis, we use hidden Markov models both as a basis of our gene finder and to characterize the conservation patterns of similar coding regions. In this section, we define hidden Markov models and explain how they are used for sequence annotation tasks such as gene finding. We also describe the Viterbi algorithm [165], which is commonly used to annotate sequences with HMMs, and two extensions of the basic HMM framework that are useful in gene finding.

An HMM consists of a finite set of states and three sets of parameters, called the initial, emission, and transition probabilities. The initial probability s_k is defined for each state k of the model. The transition probability $a_{k,l}$ is defined for each pair of states (k, l), and the emission probability $e_{k,b}$ is defined for each state k and each character b of the output alphabet. The initial probabilities form a probability distribution, as do the transition probabilities $a_{k,l}$ for each state k, and the emission probabilities $e_{k,b}$ for each k.

[1]As is true for most rules in biology, this one has an exception. The stop codon TGA can also code for a rare amino acid, selenocysteine [100]. This phenomenon occurs only in several dozen cases in the human genes.

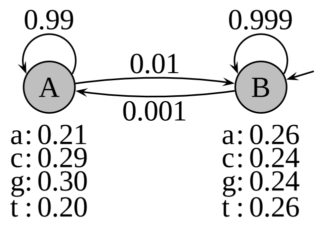


Figure 1.5: A toy hidden Markov model for gene finding (see Figure 1.6 for a more realistic example). Human coding regions are richer in the nucleotides C and G than are other sequences. State A represents coding regions and has 59% of generating 'C' or 'G'; state B represents non-coding regions and has only 48% probability of generating 'C' or "G'. The expected length of a region generated in state A is 100, while the expected length of a region generated in state B is 1000.

An HMM generates a sequence from left to right, one character in each step. First, a start state is randomly generated according to the initial probabilities. Then, in each step, the model randomly generates one character and then moves to a new state. Both the current character and the next state depend only on the current state. If the current state is k, the character b will be generated with probability $e_{k,b}$, and the next state will be l with probability $a_{k,l}$.

In n steps, the HMM generates a sequence $X = x_1, \ldots, x_n$ and traverses a sequence of states (or *state path*) $H = h_1, \ldots, h_n$. For a fixed length n, the probability that the model will traverse the state path H and generate the sequence X is the following product of the model parameters:

$$\Pr(H, X) = s_{h_1} \left(\prod_{i=1}^{n-1} e_{h_i,x_i} \cdot a_{h_i,h_{i+1}} \right) e_{h_n,x_n}. \tag{1.1}$$

1.2.1 Hidden Markov models for sequence annotation

Hidden Markov models are frequently used in bioinformatics to annotate biological sequences. Here, the task is to label each character of the input sequence with a label designating its function. In the context of gene finding, we will have different labels for protein coding regions, introns, and intergenic regions, as we noted in Section 1.1. Other examples of sequence annotation tasks include the prediction of protein secondary structure (*e.g.*, [113]) and prediction of transmembrane protein topology (*e.g.*, [98]).

Each state in an HMM for sequence annotation is associated with one of the labels. All states corresponding to one label generate sequence regions whose function corresponds to the label (see Figure 1.5 for a small example). The topology of a model (that is, the number of states, their labels, and allowed transitions) is usually designed manually by a researcher, based on domain knowledge. Model parameters are set so that the statistical properties of the generated sequences are similar to those of observed sequences. They can be estimated automatically from a training set of known sequences and their annotations. More details on training algorithms can be found, for example, in the book by Durbin *et al.* [60].

Once the HMM is completely specified, we can use it to annotate input sequences. As we have discussed, an HMM defines the joint probability $\Pr(H, X)$ for a state path H and a sequence X. Every state path $H = h_1, \ldots, h_n$ uniquely determines a labeling $L_H = \ell_1, \ldots, \ell_n$, where ℓ_i is the label of state h_i. Thus, the joint probability $\Pr(L, X)$ can be expressed as the following sum:

$$\Pr(L, X) = \sum_{H : L_H = L} \Pr(H, X).$$

One reasonable goal is to annotate sequence X by the most probable labeling L^*, given this sequence, that is, $L^* = \arg\max_L \Pr(L|X)$.

The problem of finding the most probable labeling for a given HMM and sequence is NP-hard [115]. If the HMM is fixed, and only the sequence is given as input, the computational complexity depends on the model topology [28]. While for some models the problem is still NP-hard, for others it can be solved in polynomial time. In particular, if every labeling corresponds to at most one state path, the problem of finding the most probable labeling is equivalent to finding the most probable state path:

$$H^* = \arg\max_H \Pr(H|X) = \arg\max_H \Pr(H, X).$$

The most probable state path can be found in time linear in the sequence length by the Viterbi algorithm [165], described in the next section.

In fact, even if some state paths correspond to more than one labeling, we can still use the Viterbi algorithm to find the labeling L_{H^*}, as a heuristic approximation to the most probable labeling L^*. On some examples the approximation ratio of this heuristic grows exponentially with the length of the sequence. Still, the algorithm sometimes performs better in practice than the Viterbi algorithm used with a simplified model with only one state per label [28].

1.2.2 The Viterbi algorithm for HMM decoding

The Viterbi algorithm [165] computes the most probable state path H^*:

$$H^* = \arg\max_H \Pr(H|X) = \arg\max_H \Pr(H, X).$$

It is a simple dynamic programming algorithm that, for every position i in the sequence and every state k, finds the most probable state path $h_1 \dots h_i$ for generating the first i characters $x_1 \dots x_i$, provided that $h_i = k$. The value $V[i, k]$ stores the joint probability $\Pr(h_1 \dots h_i, x_1 \dots x_i)$ of this optimal state path. Notice that if $h_1 \dots h_i$ is the most probable state path generating $x_1 \dots x_i$ and ending in state h_i, then $h_1 \dots h_{i-1}$ must be the most probable state path generating $x_1 \dots x_{i-1}$ and ending in state h_{i-1}. To compute $V[i, k]$, we consider all possible states as candidates for the second-to-last state h_{i-1}, and select the one that leads to the most probable state path, as expressed in the following recurrence:

$$V[i, k] = \begin{cases} s_k \cdot e_{k,x_1} & \text{if } i = 1, \\ \max_l V[i-1, l] \cdot a_{l,k} \cdot e_{k,x_i} & \text{otherwise.} \end{cases} \tag{1.2}$$

The probability $\Pr(H^*, X)$ is then the maximum over all states k of $V[n, k]$, and the most probable state path H^* can be traced back through the dynamic programming table by standard techniques. The running time of the algorithm is $O(nm^2)$, where n is the length of the sequence and m is the number of HMM states.

1.2.3 Generalized hidden Markov models

It is often appropriate to use states of higher order in HMMs. In a state of order o, the probability of generating the character b is a function of the o previously generated characters (a standard HMM has all states of order zero). The emission table has the form $e_{k,b_1,\dots,b_o,b}$, where $\sum_b e_{k,b_1,\dots,b_o,b} = 1$, for a fixed state k and characters $b_1, \dots, b_o$. In an HMM with all states of order o, Formula (1.1) generalizes as follows (we ignore the special case of the first o characters):

$$\Pr(H, X) = s_{h_1} \left(\prod_{i=1}^{n-1} e_{h_i,x_{i-o},\dots,x_i} \cdot a_{h_i h_{i+1}} \right) e_{h_n,x_{n-o},\dots,x_n}.$$

A state of order o represents the distribution of $(o+1)$-tuples of characters in the sequence. The Viterbi algorithm for finding the most probable state path can be adapted easily to handle higher order states with the same running time [60].

Another useful generalization is to incorporate explicit state duration into the model. An ordinary HMM may remain in the same state for several steps, provided that the state has a self-loop transition with some non-zero probability p. The probability distribution of the number of steps the model remains in this state is geometric, with parameter p. That is, the probability of staying in the state exactly k steps is $p^{k-1}(1-p)$. However, the length of the sequence feature represented by the state may not be in reality geometrically distributed. To solve this problem, states with explicit duration are given an arbitrary probability distribution of lengths. Upon entering such a state, we first draw a length ℓ from the distribution and stay in the state for exactly ℓ steps, emitting symbols according to the state's emission distribution, then follow one of the outgoing transitions to arrive to a different state. The length distribution associated with a state can have a parametric form, or it can be simply the empirical distribution estimated from the training set, possibly regularized by smoothing. Unfortunately, the running time of the Viterbi algorithm for HMMs with explicit duration states is quadratic in the sequence length, which makes it impractical for gene finding. The running time can be significantly reduced by certain restrictions on the length distributions [135, 38, 31].

1.3 *Ab initio* gene finding

Ab initio gene finding is the task of predicting the genes in a sequence, based solely on sequence features of the query DNA, without using any additional evidence. As discussed in Section 1.1.1, several sequence features help to recognize genes, but none of them is sufficiently reliable by itself. Therefore, gene finding methods typically score individual sequence features and combine scores to an overall score for each potential gene structure. Although the focus of this thesis is on approaches that use additional evidence to predict genes, many of them are based on the *ab initio* methods briefly described in this section.

1.3.1 Dynamic programming algorithms

In 1990, Gelfand designed one of the first methods for predicting the exon-intron structure of a whole gene [71]. The algorithm first identifies high scoring exon boundaries (donor and acceptor sites). Then all possible gene structures that use these boundaries are exhaustively enumerated and each is scored by a combination of coding potential and average splice site strength. The highest scoring gene structure is then reported. As there can be exponentially many gene structures, such approach is feasible only for very short sequences.

To overcome this problem, later programs have designed scoring schemes that can be efficiently optimized by dynamic programming. Gene prediction programs based on dynamic programming, such as GRAIL by Xu *et al.* [171] and GeneID by Guigó [78], first find candidate coding regions, score them and then report the gene structure maximizing the sum of the exon scores. These exon scores combine several components based on sequence composition and strength of signals at both ends of the exon. Both GRAIL and GeneParser, by Snyder and Stormo [154], combine these components into a single exon score using a neural network. The program Fgenes by Salamov and Solovyev [143] uses linear discriminant functions. GeneParser also gives scores to the introns that separate adjacent exons.

In the final optimization step, only valid gene structures are considered, as consecutive coding regions need to have compatible reading frames. The problem of finding the gene structure with highest score is equivalent to finding the maximum weight path in a directed acyclic graph in which the vertices correspond to candidate exons and the edges connect pairs of candidate exons that can be consecutive in a valid gene structure.

Dynamic programming approaches have much flexibility in choosing the scoring function. However, the scoring function is usually optimized solely for recognizing individual exons. In the final step we choose the compatible set of exons maximizing the sum of such scores, and it may happen, for example, that several short candidate exons with lower scores may be chosen instead of one true exon with high score. Another problem of the dynamic programming approach is its running time, which grows quadratically with the number of candidate exons, if intron scores are used. To keep the running time acceptable, candidates are filtered using a threshold on their score. Low threshold leads to high running time, while high threshold may leave out some true exons and prevent the program from recovering the whole gene structure correctly. Some of the problems of the dynamic programming approach are addressed by using hidden Markov models, described in the following section.

1.3.2 The use of hidden Markov models for gene finding

HMMs and generalized HMMs are the dominant paradigm in *ab initio* gene finding, thanks to the success of some early gene finders such as Genscan [37]. HMMs allow the scoring of exons and whole gene structures in a systematic way on a sound probabilistic basis. Methods for parameter estimation and inference with HMMs are also well developed [60].

Gene finding is a classic sequence annotation task, so construction and use of an HMM for gene finding follows the general pattern described in Section 1.2.1. An HMM for gene finding has states for different features found in genomic sequences, such as intergenic regions, introns, and coding regions. We can model the signals at the boundaries of coding regions by groups of states with one state for each signal position. We can constrain the model so that transitions between states allow only biologically meaningful gene structures. An example of the topology of an HMM for gene finding is shown in Figure 1.6. This model contains three copies of the submodel for introns to ensure reading frame consistency between adjacent exons. Also, the whole gene structure is duplicated with its transitions reversed to represent genes on the reverse strand. Once the topology of the model is fixed, we can use standard algorithms for parameter estimation and sequence annotation.

Gene finders typically use generalized HMMs, as described in Section 1.2.3. Higher order states

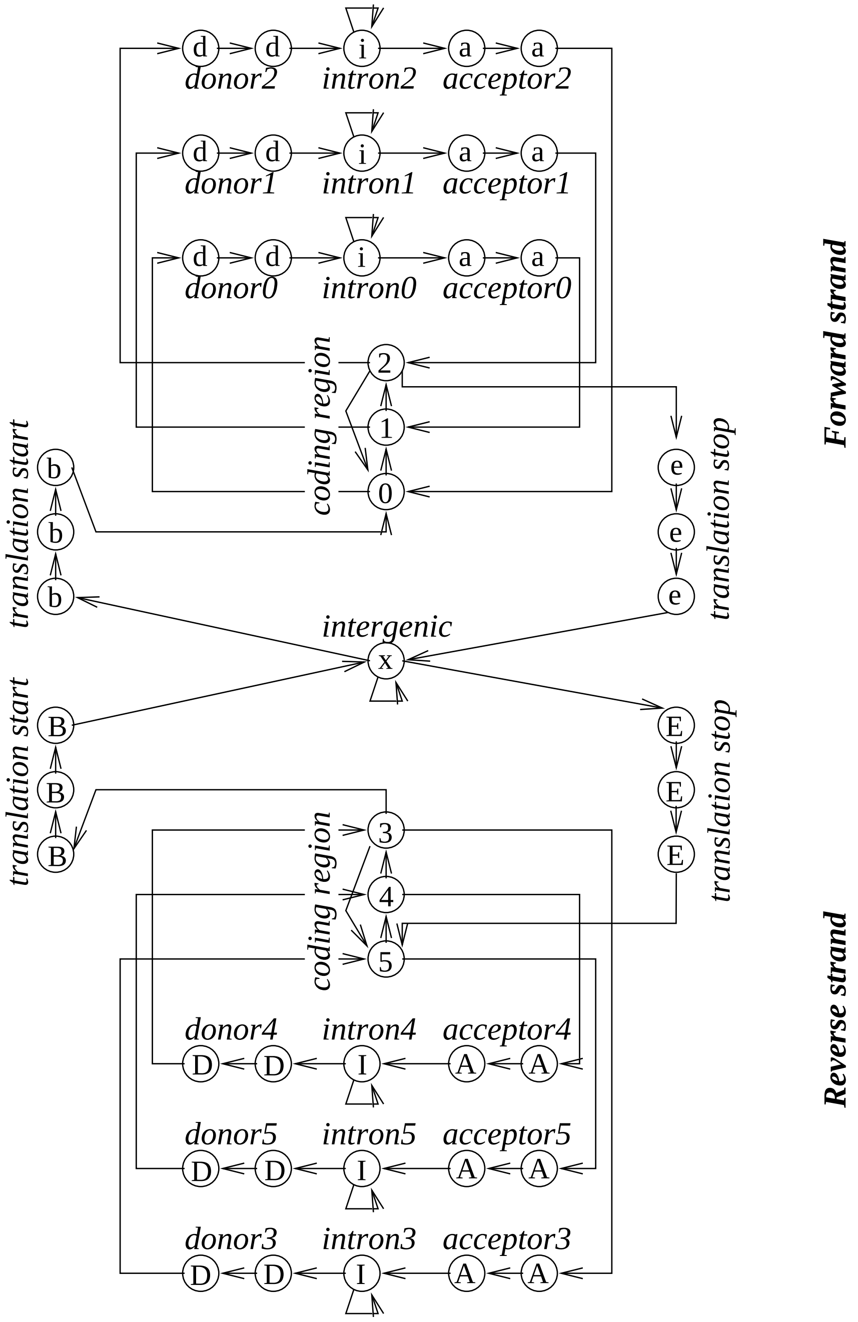


Figure 1.6: Example of the topology of an HMM for gene finding. Inside each state is its label. Only transitions with non-zero probability are shown. The model has two parts, one for each strand. On each strand there are three copies of the submodel for introns, donor sites, and acceptor sites, to keep track of the position within the codon at which the last coding region ended.

represent the distributions of k-tuples in individual sequence features, and states with explicit duration represent length distributions of coding regions and introns. Some statistical properties cannot be conveniently expressed in HMMs: The number of exons generated in one gene is forced to be geometrically distributed and the distribution of the overall length of a gene cannot be independently characterized, but is the convolution of the distributions of individual elements.

GeneMark, by Borodovsky and McIninch [22], is a precursor of HMMs for gene finding. In this program, the authors use a higher order three-periodic model, similar to the three states for coding regions in the HMM of Figure 1.6, to score a sliding window of a sequence. The probability is then compared to a probability in a background model of non-coding region, to identify likely coding regions. Later, Krogh *et al.* [99] constructed an HMM for gene finding in the genome of bacterium *E. coli.* Generalized hidden Markov models form the basis of many successful eukaryotic gene finders, such as Genie [105], Genscan [37, 38], VEIL [83], HMMGene [96], GeneMark.hmm [114], and Fgenesh [143]. More recent work includes Augustus [156] and GeneZilla [117].

1.4 Sources of additional evidence in gene finding

Ab initio gene finders predict gene structure using only the information contained in the input DNA sequence. In theory, it should be possible to predict genes in this way by emulating the cellular processes involved in the transcription and translation of genes. However, our understanding of these processes is incomplete. Moreover, the transcription and splicing mechanisms of the cell are themselves error-prone; they often create aberrant mRNAs that are detected and degraded by a cellular mechanism called nonsense mediated degradation [167]. Since we cannot reliably predict genes from the genomic sequence alone, many gene finders use additional sources of information, perhaps in the form of experimental evidence that a particular gene is indeed transcribed, to achieve higher prediction accuracy. In this section, we discuss the properties of various sources of information. In Section 1.5, we show how *ab initio* methods have been extended to include such evidence.

Most of the external evidence used in gene finding is in the form of sequence databases. These databases may contain the sequences of mRNAs extracted from cells, of known proteins, or of other genomes. Great amounts of sequence data are publicly available in databases such as GenBank [18]. The first step in using sequence data as a source of evidence is to identify *local alignments*, or regions with high sequence similarity, between the query DNA sequence and sequences from a database (see Figure 1.7). Still, some similarities may occur simply by chance, particularly if the sequence database is big. Therefore, each potential alignment is scored and only alignments with statistically significant scores are reported to the user. Alignment scoring schemes in general assign positive scores to matching nucleotides and negative scores to mismatches and gaps (a gap is a region in one sequence identified as missing from the other sequence). Sequences with significant local alignments usually either come from the same genomic region, with mismatches caused by sequencing errors, or have evolved from the same ancestral sequence by a series of mutations, deletions and insertions. Sequences sharing the same ancestral sequence are called *homologous.*

There are numerous programs for finding local alignments. The most popular is the BLAST family of programs [8]. We consider the algorithmic issues of homology search in Chapter 3. In the rest of this section, we discuss how to interpret local alignments depending on the type of sequence database used.

```
GGCAGGGGCTGCAGGGAGGGCAGGGCGTCC----CCGGAGGGCAGCAGCGTGAAGGCCT
||| ||||||||||||| |||||||  |||       |  |||||||| ||||||||||
GGCTGGGGCTGCAGGGAAGGCAGGGGTTCCATCTGGTGGAGGCAGCAGAGTGAAGGCCT
```

Figure 1.7: Example of a local alignment of two DNA sequences. Matching letters are highlighted with vertical bars. The top sequence contains a gap of length four, possibly caused by a deletion or insertion that has occurred in the course of evolution.

1.4.1 EST databases

Large-scale genome sequencing projects are usually complemented by the sequencing of mRNA sequences expressed in different tissues [3]. A random sample of mRNA molecules present in a tissue is extracted and reverse-transcribed into DNA; the resultant molecule is called a cDNA. Fragments of these cDNA molecules are then sequenced, and the sequences obtained in this way are called expressed sequence tags (ESTs). It is also possible to specifically sequence the ends of mRNA molecules by a technique called RACE (rapid amplification of cDNA ends) [69]. Finally, entire cDNA molecules can also be sequenced [137].

The sequence of an EST is essentially the concatenation of the sequences of several exons of a gene (both coding and untranslated). If we use a sequence alignment program to align an EST to the gene from which the corresponding mRNA molecule was produced, we are often able to identify exons as regions of the genomic DNA matching an EST and introns as regions of the genomic DNA between two alignments that are consecutive in an EST. Also, since ESTs often come from ends of genes, they may help to determine gene boundaries.

The use of ESTs in gene finding is complicated by several factors. First of all, it is not always trivial to align EST sequence correctly to their genome. ESTs are typically only draft sequences with error rate ranging from 1.5% to 4% [169]. Sequencing errors may cause incorrect identification of exact splice site positions. Occasionally, it may be also difficult to distinguish between an alignment of a low-quality EST to its true corresponding gene and its alignment to a related gene. Furthermore, an mRNA sample may be contaminated with unspliced mRNA; hence it may contain intronic sequence.

The use of ESTs in coding region prediction is further complicated because ESTs contain coding regions as well as UTRs. Thus, they do not help to identify translation start sites and stop sites. And, since the average length of an EST is 400 nucleotides [56], a single EST typically does not cover all exons of a gene. Finally, multiple ESTs for one gene can originate from different splicing variants of the gene, which can suggest contradictory intron-exon structure. The sequences of full length cDNAs help to solve some of these issues, since they cover the whole length of the gene, and thus it is easier to map them to the correct place in the genome and determine which parts are coding.

Schiex *et al.* [145] and Krogh [97] observe that the use of ESTs in a gene finder may actually decrease its specificity due to numerous spurious alignments. On the other hand, carefully aligned full length cDNAs are considered the most reliable way to verify gene predictions [11].

1.4.2 Expression data

Instead of sequencing random mRNAs extracted from tissue samples, we can test the presence of specific mRNAs. Such approach is used to confirm or refine existing gene predictions.

One option is to use expression arrays. For example, Shoemaker *et al.* [149] have chosen a string of length 50-60 from each predicted exon on human chromosome 22 and fabricated a probe, a short DNA sequence, complementary to this string. Such probes are attached in a regular grid to a glass chip and washed in a solution containing cDNAs obtained from a specific tissue sample. Molecules of cDNA bind to probes complementary to their sequence. The amount of cDNA attached to each probe, its expression level, is then measured. Probes that do not come from real exons ideally have very low expression level, and probes from the same gene have similar expression levels. However, expression arrays are prone to experimental errors, so the experiment has to be repeated many times and analyzed by complex statistical methods [149, 68].

Instead of expression arrays, we can also use RT-PCR (reverse transcriptase polymerase chain reaction). PCR requires two short DNA probes and a solution of different DNA molecules. If some DNA sequence in the solution contains both probes closer to each other than some distance threshold, PCR will create many copies of the region of the DNA sequence between the two probes. In the context of gene finding, Das *et al.* [57] suggest to create probes from pairs of adjacent exons in a gene structure. These probes are used in a PCR reaction applied to a solution of cDNAs obtained from a particular tissue. If the two exons are part of the same mRNA molecule, PCR will create many copies of the region between the two probes. This sequence then can be extracted and sequenced, to verify if it indeed comes from the gene in question.

1.4.3 Protein databases

The sequences of proteins tend to change only slowly in the course of evolution. For example, at least half of fruit fly proteins have a high-scoring alignment to some mammalian protein [142]. Once we determine and experimentally verify a protein of one species, we may find sequences coding for a similar protein in the genome of other species. A local alignment between a known protein and a DNA sequence that could encode a similar protein suggests the presence of a coding region. Similarly, two alignments to regions adjacent in the protein, but separated by a gap in the query DNA, suggest the presence of an intron. An alignment that includes one of the ends of a protein sequence suggests the location of a translation start or stop signals close to the alignment in the query DNA. Gene finders using protein alignments include PROCRUSTES, by Gelfand *et al.* [72]; AAT, by Huang *et al.* [87]; GenomeScan, by Yeh *et al.* [173]; and GeneWise, by Birney *et al.* [20].

The main obstacle to using protein alignments for gene finding are pseudogenes. These are copies of genes that do not produce full-length functional proteins [81, 160]. Since they are not functional, they are not constrained by natural selection, so over time they accumulate mutations. Some of these mutations will disrupt reading frame or introduce in-frame stop codons into the copies of exons. Still, recently copied pseudogenes often align quite well with functional proteins and may cause false positives in gene finding. For example, the Mouse Genome Sequencing Consortium [53] reports a gene in the mouse genome that has a a single active copy and 400 pseudogenes. More than a quarter of these pseudogenes retain enough similarity to be predicted as genes in the initial annotation process.

Instead of aligning individual proteins to the query DNA, we can first construct a multiple alignment of several homologous proteins and then align this multiple alignment to the query sequence. The multiple alignment of a family highlights positions most conserved by evolution and might increase the sensitivity of similarity search [9]. A similar approach uses information from databases of protein domains such as BLOCKS or Pfam [85, 14]. A protein domain is a region of a protein that folds independently of the rest of the protein. Similar domains are clustered in these databases and represented by a characteristic profile. A profile can then be aligned to the query DNA sequence. Multiple alignments and profiles were used in gene finding in GENIE by Kulp *et al.* [106] and in Aln by Gotoh [75].

1.4.4 Genome comparisons

If genomic sequences of two species are available, it is possible to exploit typical patterns of evolution to annotate genes. Coding regions usually evolve much more slowly and are well conserved even between relatively distant species. On the other hand, non-coding regions have fewer functional constraints and random mutations often accumulate more quickly in them.

If we compare two distantly related species, significant sequence similarity between their sequences occurs usually only inside coding regions. For example, the program TAP, by Novichkov *et al.* [125] uses the fruit fly or frog genome to predict human genes, and Exofish, by Crollius *et al.* [138], uses the pufferfish genome for the same task.

On the other hand, if we use two species that are closely related, such as human and mouse, many non-coding regions have high sequence similarity as well. Therefore, sequence similarity does not, by itself, identify coding regions. SGP2, by Parra *et al.* [127], uses alignments between the human and mouse genomes to indicate possible coding regions in human, but to avoid many false positives, its authors use a special alignment scoring scheme and assign higher weight to the *ab initio* component than to the genomic alignment evidence. TwinScan, by Korf *et al.* [95], finds genes in local alignment of human and mouse by detecting specific mutation patterns typical for coding regions. Recall that coding regions have three-periodic structure in which every triple encodes one amino acid. As we discuss in more detail in Section 3.3, the third position of the codon is much more likely to mutate than the first two. Such regular patterns of matches and mismatches can be detected and used in gene finding.

Batzoglou *et al.* [15] observe that homologous genes of human and mouse have usually the same number of exons and exon lengths are well preserved. This information can be used to predict gene structures of two organisms at the same time, by favoring those that have the matching numbers and lengths of exons. This approach has been used, for example, in ROSETTA, by Batzoglou *et al.* [15], and SLAM, by Alexandersson *et al.* [5].

Instead of pairwise alignments of genomic sequences, we can use multiple sequence alignments. Several such techniques have been introduced only recently, as genomic sequences have become more abundant. For example, McAuliffe *et al.* [118] observe that the pairwise alignment of human genome and a genome of another primate species does not contain enough information to be useful in gene finding, since most positions are conserved. In contrast, multiple sequence alignments of several primate species provide a much stronger signal. Multiple genome alignments were also used by Pedersen and Hein [130], Siepel and Haussler [151], Chatterji and Pachter [44], and Gross and Brent [76].

1.4.5 Other sources of information

While sequence similarity is the main source of additional information used in gene prediction, other evidence might be used as well. Some gene prediction programs (*e.g.*, EUGÈNE [145] and Augustus [156]) allow the knowledge of expert users to influence the prediction. Chen [46] incorporates information from mass spectrometry of proteins to a gene finding program.

Many gene finders preprocess the sequence, by using programs that detect repetitive DNA elements, such as RepeatMasker [152]. These elements account for a significant portion of eukaryotic genomes, for example they account for more than a half of the human genomic sequence [51]. Sequence repeats often contain irrelevant protein coding sequences or pseudogenes that may confuse a gene finder. The location of repeats in a genome thus an important information in gene finding.

Another possible source of information is the location of CpG islands. The pair of adjacent nucleotides CG is underrepresented in human DNA because the nucleotide C in such a pair is often chemically altered by methylation and then mutates more easily than unmethylated nucleotides. CpG islands are short genomic regions relatively rich in CG pairs. They are typically left unmethylated and occur in the promoter regions close to the start of a gene [51]. It is possible to find CpG islands by sequence analysis [88], and verify them by experimentally testing for their lack of methylation. CpG islands may suggest the possible location of gene boundaries. They were used in systems for predicting promoters and transcription start sites, for example by Bajic and Seah [12].

1.5 Methods for combining evidence in gene finding

Now we turn our attention to methods for incorporating additional evidence, in particular sequence similarity, into gene finders. Some systems first gather the gene predictions of *ab initio* gene finders and information from various external sources, and then predict genes by combining this information without considering properties of the query DNA sequence. Other approaches incorporate external evidence directly to *ab initio* gene finders to increase their accuracy. In our gene finder, we use the latter approach, since we believe that the query DNA sequence contains valuable information that should be considered together with any evidence. In the following survey, we concentrate in particular on approaches for incorporating evidence into the probabilistic framework of HMM-based gene finders, although we mention other approaches as well.

1.5.1 Hidden Markov models with multiple outputs

As we have seen, HMMs provide a convenient probabilistic framework for *ab initio* gene finding. In this section, we present several extensions of gene finding HMMs that allow them to represent additional evidence. In all of these extensions, the HMM is modified so that it generates k sequences in parallel, one character from each sequence in each step. Given k output sequences, we may find the most probable path generating all of them, as before. One of these sequences might be the DNA sequence, while the others represent additional information in the form of labels from some fixed alphabet.

These extensions are most easily described as Bayesian networks. A Bayesian network is a generative probabilistic model with N variables arranged as the vertices of a directed acyclic graph. We generate values for the variables in topological order, so that the values of parents are generated before the value of their children. Consider now variable X, with parents $X_1, \dots, X_k$. The parameters of the Bayesian network specify the conditional probability $\Pr(X = x \mid X_1 = x_, \dots X_k = x_k)$, for all combinations of the values $x, x_1, \dots, x_k$. Thus, once the values of the parent variables are fixed, we can generate the value of X from this conditional distribution. An HMM generating a sequence of a fixed length n can be represented as a Bayesian network with $2n$ variables: for each emitted character, we have one variable representing the character itself and one variable representing the hidden state emitting the character (see Figure 1.8). To generalize an HMM to more

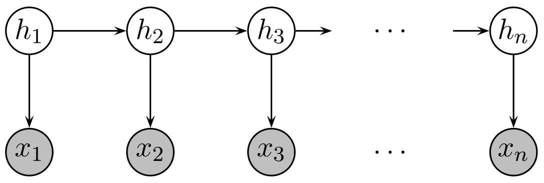


Figure 1.8: A hidden Markov model represented as a Bayesian network. The top row of variables represents the state path, $h_1, \ldots, h_n$. The bottom row represents the emitted DNA sequence, $x_1, \ldots, x_n$. Conditional probabilities in the Bayesian network are defined by the initial, transition, and emission probabilities of the HMM: $\Pr(h_1) = s_{h_1}$, $\Pr(h_i|h_{i-1}) = a_{h_i,h_{i-1}}$, and $\Pr(x_i|h_i) = e_{h_i,x_i}$. The observed variables, which indicate the DNA sequence, are shaded in the figure.

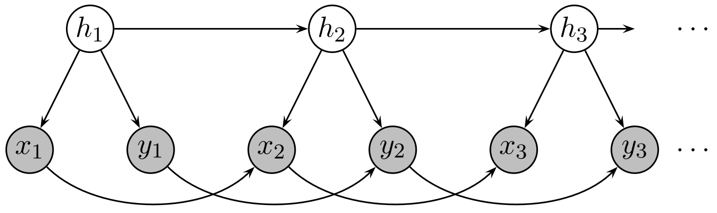


Figure 1.9: Probabilistic model of TwinScan gene finder [95] as a Bayesian network. Each variable h_i represents one state of the HMM, variable x_i represents the i-th nucleotide of the query DNA sequence, and y_i represents the conservation between this nucleotide and some other genome. The possible values of y_i are labels standing for match, mismatch and unaligned position in the alignments between two genomes. Notice that each nucleotide x_i in the figure has two parents: state h_i and the previous nucleotide x_{i-1}. This corresponds to using emission tables of the first order in the HMM. (TwinScan, in fact, uses emission tables of order five.)

outputs, we replace each of the n slices by a more complicated Bayesian network, with more than two variables.

TwinScan, by Korf *et al.* [95], is a gene finder based on Genscan [37] that generates two sequences. One is the query DNA sequence, and the other one represents alignments to a different genome. This sequence is over a three letter alphabet: one character of the alphabet represent a match in the alignment, one represents mismatch, and one represents gaps and nucleotides in unaligned genomic regions. In each step, the characters of the two sequences are generated independently, which reduces the number of model parameters that need to be estimated. Figure 1.9 shows the TwinScan model as a Bayesian network.

A similar approach was used by Pavlović *et al.* [128] to combine predictions of several gene finding programs. Each prediction was turned into a separate output sequence. The query DNA sequence was not used in the model. Output sequences were independently generated in each state, although authors also study a variant with more complex dependencies. Their model does not enforce consistent reading frame across the gene, such consistency is enforced in a post-processing stage.

Recently, Frey *et al.* [68] have used a complicated Bayesian network to find genes in expression array data. They have chosen a set of candidate exons and measured the expression level of each exon in several different tissues. The goal is to decide which candidate exons are true exons, and

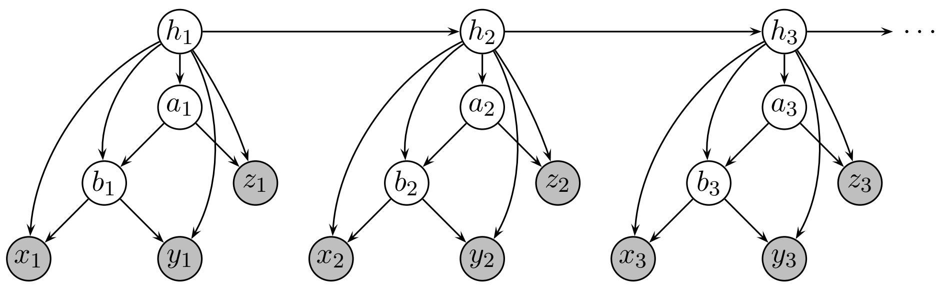


Figure 1.10: A simple phylogenetic hidden Markov model depicted as a Bayesian network. Each variable h_i represents one state of the HMM, variables x_i, y_i, z_i represent nucleotides of three species from one column of a multiple genome alignment, and the variables a_i and b_i represent the ancestral sequences. Observed variables are shaded. For example, the value of x_i depends on its ancestor b_i and on the state h_i. The state determines mutation rate, since mutations occur more frequently in non-coding regions.

to decide which exons belong to the same gene. In general, true exons have higher expression levels than decoys and expression levels within one gene are correlated across different tissues. This signal is, however, diluted by very strong experimental noise. They use a Bayesian network for this problem. Each slice of the network corresponds to one putative exon, rather than a single nucleotide.

TwinScan [95] uses alignment with a single genome to predict genes. As multiple genomes have become available, researchers have attempted to use multiple sequence alignments to further improve prediction accuracy. Pedersen and Hein [130] introduced phylogenetic HMMs to gene finding. These models find shared genes in a multiple alignment of several genomes. Sequences in the multiple alignment cannot be assumed independent, since they are closely related by evolution. Therefore, the authors arrange the sequences in the leaves of a Bayesian network with their topology identical to the phylogenetic tree representing the evolutionary history of the sequences (see Figure 1.10). Different variants of phylogenetic HMMs for gene finding were also investigated by McAuliffe *et al.* [118], Siepel and Haussler [151], and Gross and Brent [76].

The program SAGA by Chatterji and Pachter [44] is a novel technique for finding genes in multiple homologous genomic regions. Unlike phylogenetic HMMs, it does not require prior alignment of the sequences, and the alignment is actually never explicitly constructed. It is assumed that the input sequences represent independent samples, generated by the same HMM. This HMM has many features typical for *ab initio* HMM gene finders, but its exon number distribution and emission tables are biased specifically to match the genes on the input. The parameters of the model and gene predictions are iteratively improved by Gibbs sampling. Thus, after each iteration, gene predictions in all input sequences will tend to be more similar to each other, and the parameters of the model will tend to be fitted more closely to the sequences on input.

1.5.2 Positional score modification

In this section, we explore approaches that deviate from the Bayesian network framework introduced in the previous section, but still incorporate external evidence to a hidden Markov model. In an HMM, the joint probability $\Pr(H, X)$ of sequence X and state path H is computed as a product of emission and transition probabilities (see Equation (1.1)). Therefore, it is natural to incorporate evidence in the form of additional multiplicative terms in this product.

This is the approach taken in the HMMGene gene finder by Krogh [97]. In the first step, a probability distribution over all possible labels is computed at each position of the sequence based on local alignments. Let $p_{i,\ell}$ be the predicted probability of label ℓ at position i (for each position i, we have that $\sum_i p_{i,\ell} = 1$). For example, an EST alignment increases the probability of that position being inside a coding region or UTR. If no alignment is detected at a position, the probability distribution over all labels is uniform.

In the second step, the most probable state sequence H^* is found, but the probability $\Pr(H, X)$ is modified by multiplying it, at each position i, by the term p_{i,ℓ_i} where ℓ_i is the label corresponding to state h_i:

$$\Pr(H, X) = s_{h_1} \left(\prod_{i=1}^{n-1} e_{h_i,x_i} \cdot p_{i,\ell_i} \cdot a_{h_i h_{i+1}} \right) e_{h_n,x_n} \cdot p_{n,\ell_n}.$$

The Viterbi algorithm can easily be modified to handle the modified version of $\Pr(H, X)$. However, the modified $\Pr(H, X)$ values no longer actually constitute a valid probability distribution over all pairs H, P of fixed length.

Many parts of the HMMGene method are somewhat arbitrary. No systematic way for handling several alignments on one position is given. The amount by which an alignment increases the probabilities of a particular label is an arbitrary constant chosen by the author, scaled by the significance of the alignment.

GenomeScan by Yeh *et al.* [173] uses a similar method to incorporate protein homology. Its authors select the most significant alignment from each cluster of overlapping alignments. Then, they consider a single position in the center of the alignment. The probabilities of all state sequences H that label that position as being from a coding region are raised and the probabilities of other state sequences are lowered so that $\Pr(H, X)$ is still a valid probability distribution. Their formula for modifying probabilities has a probabilistic interpretation, at least in the case of a single alignment. The amounts by which the alignments increase and decrease the path probabilities depend on the tenth root of the alignment's BLAST P-score, where the P-score is an estimate of the probability of an alignment of that strength occurring at random in noise sequences. The tenth root was picked arbitrarily, because the original P-scores seemed too low to estimate the probability of a spurious alignment for the purposes of gene prediction.

An important difference between GenomeScan and HMMGene is that given a particular alignment, GenomeScan alters the probability at one position only, whereas HMMGene boosts the probability independently at each position covered by the alignment.

A slightly different method is used in EUGÉNE [145], which is not based on an HMM but on a directed graph with a node for every combination of position and label. Weights of edges in this graph roughly correspond to transition and emission probabilities in an HMM, but the overall structure is not a generative model. The resulting gene prediction is the path in graph with highest product of edge weights. For each edge of the graph, we may have several different edge weights estimated from different sources, such as sequence composition, protein alignments, cDNA alignments, splice site signal detectors. These estimates are combined by convex combination to make a single edge weight for each edge. The weights of the individual sources in this convex combination are estimated from the training data.

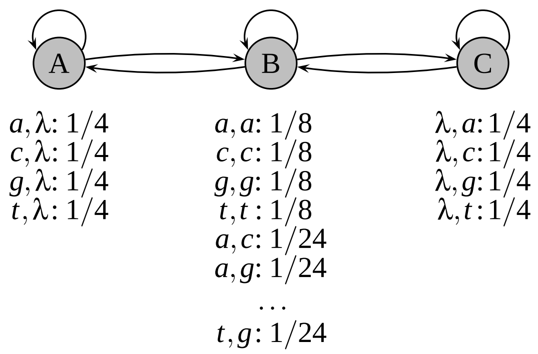


Figure 1.11: A simple pair HMM. Symbol λ in the emission probability tables represents empty string. State B generates ungapped portion of the alignment. State A generates characters only in the first sequence, and state C generates characters only in the second sequence. Alignment gaps induced by states A and C have geometrically distributed lengths.

1.5.3 Pair hidden Markov models

In the previous sections, we have reviewed several methods that break the problem of gene finding into two steps. First, a general search tool is used to find local alignments between the query DNA and a sequence database. Next, this information is incorporated to some gene finding method. The main disadvantage of the two-step method is that the initial general-purpose alignment algorithm does not take into account gene structure. Thus, alignments of a protein or EST with the query DNA may extend beyond exon boundaries to surrounding introns, and alignments of two homologous genes may have misaligned splice sites. Such mistakes are then propagated to the second stage, and may affect the accuracy of gene finding.

This problem is avoided by performing gene finding and alignment together in a single step. Such process can be modeled by a pair HMM. Pair HMMs are HMMs that generate two sequences at the same time, but where a state of a model can generate a character in one sequence or both sequences. Pairs of characters generated in the same step correspond to homologous positions. If only one character is generated in a given step, it corresponds to a sequence position in that sequence with no homolog in the other sequence, due to insertion or deletion. Simple pair HMMs, such as the one in Figure 1.11, can be used to represent a traditional global alignment of two sequences [60] (a global sequence alignment is required to cover the whole extent of the two input sequences, whereas local alignments may cover only short conserved regions). Note that by contrast, the multiple output HMMs introduced in Section 1.5.1 have an alignment of the output sequences fixed and in each step generate a character in each output sequence. If the alignment contains a gap, they generate a special character, for example a dash. On the other hand, the output sequences of pair HMMs do not identify which pairs of characters were emitted in the same step.

The program SLAM, by Alexandersson *et al.* [5], predicts genes simultaneously in two homologous genomic sequences, under the assumption that they have the same exon structure. Their paired HMM has separate states for exons, introns, signals and intergenic region, as in HMMs for gene finding. Each state emits pairs of sequences with conservation patterns typical for sequence feature represented by the state. DoubleScan, by Meyer and Durbin [120], is similar, but can also predict genes with different exon-intron structure. GeneWise, by Birney *et al.* [20], uses pair HMMs to align a protein sequence to a genomic sequence. The non-coding states emit characters only in the genomic sequence, while coding states emit a triple of nucleotide in the genomic sequence and a single amino acid in the protein sequence.

The main disadvantage of pair HMMs is their high running time. Given two sequences generated by a pair HMM, we do not know, which pairs of characters from these two sequences were generated at the same time. The running time of the modified Viterbi algorithm that finds the most probable alignment of two sequences and their annotation is proportional to the product of the sequence lengths. Although such a running time is infeasible in many situations, different heuristics can be used to make the pair HMM approach more practical [5, 120]. This approach is also hard to extend to multiple sources of information because its running time grows exponentially with the number of sequences.

Several early algorithms for spliced alignment were not using the formal probabilistic framework of pair HMMs. For example, PROCRUSTES, by Gelfand *et al.* [72], first identifies all potential splice sites in the query DNA sequence, and then selects the gene structure that uses a subset of these splice sites and aligns best to a given protein. Similar ideas are explored in the gene finders AAT by Huang *et al.* [87], and Aln, by Gotoh [75].

1.5.4 Rule-based systems

Some gene finders are not based on probabilistic models, but instead use a series of hand-crafted rules to predict genes based on evidence. For example, the Ensembl annotation pipeline [56] first aligns known proteins, ESTs, and cDNAs to the query DNA sequence. The resulting alignments are then filtered, adjusted, and assembled to complete gene structures by a series of rules that try to mimic decisions made by human annotators. A similar strategy is used in the EAnnot pipeline of Ding *et al.* [59] and in AIR, by Florea *et al.* [65]. Unlike most gene finders, these pipelines can annotate several splicing variants of a single gene.

Rule-based systems were also used in *ab initio* gene finding. Murakami and Takagi [121] and Rogic *et al.* [140] observe that we can improve the accuracy of *ab initio* gene finding by combining results of several gene finders by simple rules.

In systems based on probabilistic models or other machine learning approaches, most parameters can be usually estimated from the training data. Therefore, they are easy to adapt to new data sets or even new sources of evidence. On the other hand, rule based methods are typically based on arbitrary decisions and *ad hoc* parameters. Allen *et al.* [6, 7] have created two systems, Combiner and Jigsaw, that work in a similar way to rule-based systems such as Ensembl, yet their rules are extracted from training data by automated methods. They collect evidence from numerous sources, including EST and protein alignments, and *ab initio* gene predictions. Then, for each position of the sequence they construct a feature vector whose elements indicate the presence or absence of individual sources of evidence, and possibly their strength. Then, they partition the space of all possible evidence feature vectors using an automatically constructed decision tree. Each leaf of the tree corresponds to one element of the feature space partition. For each partition element, they estimate probability of the individual sequence elements, based on the training data. These decision trees are used instead of *ad hoc* rules. The algorithms that build the trees recognize which sources of evidence are most reliable and hence should be followed with highest priority.

1.6 Evaluation of gene finding accuracy

In the previous sections, we have seen many different approaches to gene finding, ranging from *ab initio* gene finders using only the query DNA sequence, to pipelines integrating diverse sources of evidence. The main criterion for comparing different gene finders is their prediction accuracy.

The measures currently used to evaluate gene prediction accuracy were introduced by Burset and Guigó in their evaluation study published in 1996 [40]. Since then, several other authors have made independent comparative studies of multiple gene finders on various testing sets [79, 129, 139, 136]. The prediction accuracy is typically measured on three levels. At the nucleotide level, we compare the set of nucleotides predicted as coding with the set of nucleotides that actually are coding. At the exon level, we compare the set of predicted exons with the set of actual exons. And, at the gene level, we compare the set of predicted genes with the set of actual genes. An exon is correctly predicted if both of its boundaries are correct, and a gene is correctly predicted if its entire exon-intron structure is correct.

At each level, we evaluate two measures: sensitivity and specificity. Sensitivity measures the ratio of correct predictions to all correct elements at that level. For example, nucleotide sensitivity is the fraction of real coding nucleotides that are predicted as coding. Specificity measures the ratio of correct predictions to all predictions. For example, exon specificity is the fraction of all predicted exons that occur with the same boundaries in the reference gene annotation. We need to consider both measures to reasonably compare gene finders. Some gene finders may have high sensitivity, but the correct predictions are lost in a sea of false positives. Other gene finders may have lower sensitivity, but their few predictions are very likely to be correct.

In general, modern gene finders achieve relatively high accuracy at the nucleotide level, but the accuracy at the exon level, and especially at the gene level, is much lower. It is much easier to determine the approximate locations of most exons than to determine their correct boundaries and connect them to genes. For example, the *ab initio* gene finder Genscan [37] achieves 98% nucleotide sensitivity, 82% exon sensitivity, and only 44% gene sensitivity on the Rosetta testing set, which consists of 117 human genes developed by Batzoglou *et al.* [15] (see Table 4.3 for more results from this testing set).

The accuracy measures are even more complex in the presence of alternative splicing in the testing set. In our experiments, we compare predicted genes with a reference annotation using the evaluation program Eval by Keibler and Brent [90]. At the nucleotide level, Eval considers as coding all nucleotides in the union of coding regions of all alternative transcripts of a gene. At the exon level, Eval considers the set of all unique exons, pooled from all transcripts. Note that exons in this set may overlap. Thus, a program predicting only a single splice variant per gene will never achieve 100% sensitivity if the testing set contains genes with alternative splicing, but it may achieve 100% specificity. At the gene level, Eval considers each gene as a set of transcripts, and a gene is correctly predicted if at least one of its transcripts is correctly predicted.

An important issue in gene finding accuracy assessment is the reliability of the reference testing set annotation. Ideally, the reference annotation is a manually curated set of genes with good supporting evidence, for example in the form of full-length cDNAs. However, while we can create reliable sets of gene annotations, it is much harder to verify that there are no other real genes or splicing variants in the testing sequences. While sensitivity requires only a representative subset of real genes, and can be thus assessed relatively reliably, specificity requires us to reject some predicted genes as definitely incorrect. This rejection is much harder to achieve.

1.7 Summary

In this chapter, we have introduced the problem of gene finding and described existing work in the field, with the emphasis on methods for combining external evidence with *ab initio* gene finding methods. We have also described hidden Markov models, an important tool for gene finding and other bioinformatics tasks.

Our gene finder combines a hidden Markov model, which captures many local sequence properties, with multiple sources of external evidence. In 2001, when we started this work, most HMM-based gene finders were using at most one source of external evidence, such as proteins [173] or other genomes [95]. Several early approaches for using multiple sources of evidence in an HMM for gene finding, such as HMMGene [97] and GENIE [106] choose at most one source of evidence to influence the score at each particular sequence position. In recent years, the emphasis was on methods for combining information from multiple genome alignments [44, 76, 130, 151]. Recently, several methods for combining high-quality evidence were published [6, 7, 56, 59, 65]. They differ significantly from our work, since they do not contain an explicit model of the query DNA sequence. Several new, as yet unpublished methods, for evidence combination have appeared in the ENCODE gene prediction workshop in May 2005, and we will mention them in our experiments in Section 4.4.2.

A recent study by Eyras *et al.* [62] on the chicken genome confirms that gene prediction accuracy still needs to be improved. They have verified a sample of splice site pairs from gene predictions of TwinScan [95], SGP-2 [127] and Ensembl [56] by RT-PCR. Almost all Ensembl predictions could be verified, but many real genes are missing from this set. On the other hand, TwinScan and SGP-2 predict many more genes, but less than a half of the predicted splice site pairs not identified by Ensembl could be verified experimentally. This study shows that gene finders with an *ab initio* component, such as SGP-2 and TwinScan, can supplement conservative annotations built by methods relying on high quality information. However, their utility would increase if their predictions were more reliable. Also, the study does not even attempt to verify predictions on the gene level. Yet, to infer correct protein sequence encoded by a gene, we need to know the exact boundaries of all of its coding exons.

Both TwinScan and SGP-2 use only genome alignments to predict genes. In our work, we use evidence from multiple sources to improve the prediction accuracy. On the other hand, our gene finder does not require high quality information, such as full-length cDNAs, used in Ensembl. It can use less reliable sources of evidence, such as alignments of ESTs and other genomes. When no evidence is available at all, our gene finder will produce *ab initio* predictions.

Chapter 2

Evidence Combination in Gene Finding

In this chapter, we introduce a general probabilistic framework facilitating the combination of different types of additional evidence in gene finding. In our framework, we model individual sources of evidence as experts providing probability distributions over all possible labelings of the query sequence. The resulting prediction is then a combination of such expert predictions.

Our framework is based on a hidden Markov model. While some approaches use HMMs to score potential coding regions or find one most probable labeling and combine such output with other sources of evidence, we modify the weights of individual state paths of the HMM according to the available evidence. In this way, we also consider the probabilities assigned by the HMM to sub-optimal paths that are supported by the evidence, which is not possible if only a single path or an incomplete set of coding regions is picked before considering the evidence.

Evidence combination in gene finding is challenging. The information provided by different sources is incomplete. That is, one source of evidence typically cannot help to distinguish among all possible labels at a given site. Also, some sources of evidence may offer advice only for some parts of the sequence. Our framework allows the expression and combination of such incomplete information in a very natural way.

In this chapter, we first introduce the general architecture of our framework and its components. The individual components are combined together in two steps that we in turn describe in greater detail. In the first step, we combine together the incomplete information from different sources. We design a new combination method and show that it extends traditional approaches for expert combination. We also study several natural variants of our method. Since some sources of evidence are more reliable than others, we assign each source of evidence a weight, and discuss the problem of weight optimization on the training data. Information merged from all sources is then combined with a hidden Markov model for gene prediction to obtain the final result. At the end of the chapter, we discuss related approaches for combining incomplete information from the research literature.

2.1 Overview of advisor architecture

Our model combines two kinds of probabilistic models: a single hidden Markov model for gene finding and multiple *advisors*, each representing one type of additional evidence. Each component of the system returns a probability distribution over all possible labelings. These probability dis-

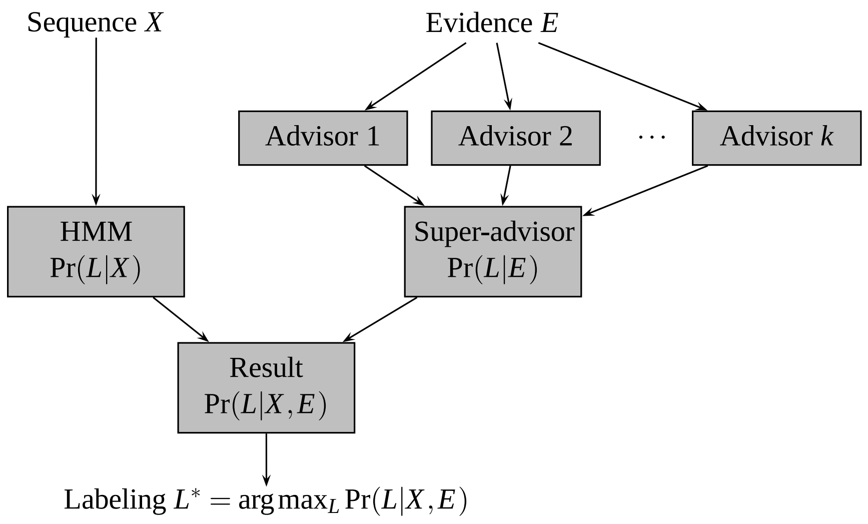


Figure 2.1: Overview of model architecture.

tributions are combined to produce one probability distribution over all possible labelings. The most probable labeling (or the most probable state path) in this labeling is then chosen as a final prediction.

We use two steps to combine probability distributions (see Figure 2.1). First, the distributions produced by the advisors are joined into one super-advisor distribution. This super-advisor is then combined with the distribution given by the HMM. In particular, the HMM defines a probability $\Pr(L|X)$ for every query DNA sequence X and labeling L. Similarly, for each labeling L, the super-advisor defines a probability $\Pr(L|E)$, where E is the evidence available to the advisors. These are combined to yield a single probability distribution $\Pr(L|X,E)$, and the optimal labeling, $\arg\max_L \Pr(L|X,E)$, is chosen as the final prediction of the true label at each site.

In the rest of this section we will discuss the individual components in more detail.

2.1.1 The base hidden Markov model for gene finding

The hidden Markov model is used in the advisor architecture to model basic gene structure, the composition properties of different sequence elements, splicing and transcription signals, and the length distributions of these elements. Generalized HMMs that allow non-geometric length distribution and higher-order Markov chains can also be used in the advisor architecture. In our implementation, we use an HMM similar to the models used in Genscan [37] or Augustus [156]; for more details, see Section 4.1.

In the HMM, each state is assigned a corresponding label, but several states can have the same label. Each state sequence $H = h_1, h_2, \ldots, h_n$ corresponds to a labeling $L = \ell_1, \ell_2, \ldots, \ell_n$, where ℓ_i is the label of state h_i. The probability of a labeling L is the sum of the probabilities of all state sequences whose labeling is L. As we have discussed in Section 1.2.1, the problem of finding the most probable labeling is greatly simplified when each state sequence corresponds to a different label sequence. This condition is satisfied in our HMM for gene finding. Therefore, we can focus on finding the most probable state path through the HMM.

The HMM forms the basis of our gene finding system and has several roles. It models the length and composition of individual sequence elements and the signals at their boundaries, which is a task for which HMMs have been shown to be well suited [37, 96]. It also enforces that labelings with improper gene structure have zero probability, such as when an intergenic region is followed by an intron, or when the reading frames of two consecutive coding regions do not match. The prediction of the HMM is then enhanced by external information in the form of advisors.

2.1.2 Advisors and the super-advisor

In our model, an advisor is an electronic or human expert concentrating on a small portion of the gene finding task. For example, an advisor might focus on splice site prediction or sequence similarity search. At each position in the sequence, every advisor estimates the probability that a given label is the true label at that position or that a set of labels contains the true label.

An advisor typically does not have enough information to estimate a probability distribution of all labels at all positions. For example, an advisor predicting donor signals does not know how to estimate the probability that a position is inside an intron. Therefore, we allow an advisor to provide only partial information.

Definition 1 (Advice of an advisor). *Let Σ be a finite set of labels. The* advice *of advisor a at position i of the sequence consists of a partition $\pi_{i,a}$ of the set Σ and a probability distribution $p_{i,a}(S)$ over all sets S in the partition $\pi_{i,a}$. The value $p_{i,a}(S)$ is an estimate of the probability that the correct label at position i is in set S, given the evidence available to advisor a.*

The advisor thus need not provide a complete probability distribution over all labels but possibly only a coarser probability distribution over the sets of labels that make up the partition $\pi_{i,a}$. Also, note that the partition $\pi_{i,a}$ can be different at different positions of the sequence. We will drop the sequence index i from this notation when the position is clear from the context.

As an example, let Σ consist of the following five labels: I for intron, $0, 1, 2$ for coding region in the three possible reading frames, and X for intergenic region. The following examples constitute valid advice of an advisor a at a single position i in the sequence:

- $\pi_a = \{\{0\}, \{1, 2, I, X\}\}$, $p_a(\{0\}) = 0.6$, $p_a(\Sigma \setminus \{0\}) = 0.4$. This corresponds to the statement, "Position i is in a coding region at reading frame 0, with 60% probability."
- $\pi_a = \{\{X, I\}, \{0, 1, 2\}\}$, $p_a(\{X, I\}) = 0.34$, $p_a(\{0, 1, 2\}) = 0.66$. This corresponds to the statement, "Position i belongs to a coding region with probability 66%."
- $\pi_a = \{\Sigma\}$, $p_a(\Sigma) = 1$. This corresponds to the situation in which advisor a cannot produce any advice at position i. We call such advice *vacuous*.

In Section 2.4, we will show how to combine a set of advisors into a single super-advisor a^*. The super-advisor has the same form as an advisor, except that the super-advisor estimates the probability of each individual label, at each position in the sequence, so that the partition π_{i,a^*} is always complete.

The super-advisor defines a probability distribution $\Pr(L|E)$, over all labelings of the sequence. We will assume for simplicity that the labels of different positions in the sequence are independent. Thus, the probability of a particular labeling is the product of the superadvisor's probabilities of the labels at individual positions. This assumption is of course false; we will discuss possible solutions in Section 2.7.

Since the super-advisor considers each position independently, many labelings with improper gene structure have non-zero probability. The super-advisor alone does not provide meaningful predictions of gene structure, but it is flexible and can be easily extended to accommodate more information in the form of new advisors.

2.1.3 Related work

Our method for combining evidence in gene finding consists of two steps. In the first step, we combine several advisors to one super-advisor. In the second step, we combine the super-advisor and the HMM together to form a single probability distribution. Both steps can be seen as a special case of expert opinion combination, also called combination of classifiers [107]. In this framework, we are given a set of classifiers, or experts. Each expert assigns probability to each possible event from some fixed finite set of events. We want to combine their probability distributions to a single probability distribution over the same set. The goal is to assign high probability to the true event, which is not known in advance.

Methods for expert combination were introduced in many different contexts. For example, ensemble methods, such as bagging [24] and AdaBoost [67], train several classifiers on different samples drawn from a training set, and combine them to increase the prediction accuracy compared to a single classifier. In the on-line learning model [86], the data set is revealed one point at a time, and after each point we adjust the weights of individual experts so that the overall prediction accuracy is guaranteed to be close to the prediction accuracy of the best expert.

Neither of these approaches is directly applicable to our problem. In both combination steps in our framework, the experts are fixed, and thus ensemble methods cannot be used. The training data is entirely available in advance, so on-line algorithms are not appropriate. In Section 2.2.1, we discuss two popular methods for combining experts, the linear and logarithmic opinion pools. We show that a simple modification of the logarithmic opinion pool can be used to combine the super-advisor and the HMM, using Bayesian principles. However, we cannot combine the advisors to a super-advisor by traditional expert combination methods, as the information provided by the advisors is incomplete. Therefore, we propose a new combination method, described in Section 2.4. In Section 2.8, we compare our advisor framework with other systems for representing incomplete information.

Our advisor combination method allows us to assign weights to advisors. The problem of optimizing weights for linear combination of experts is well studied [13, 82, 108, 131]. One can optimize weights with respect to various criteria, such as minimizing mean squared error [82, 131], the number of misclassified training samples [108], or just by ranking expert performance [13]. These methods do not apply directly to our problem, since our combination method is more complex than linear combination. We discuss the problem of optimizing weights with respect to the maximum likelihood criterion in Section 2.6. We show that weight optimization for a simplified version of our combination method is equivalent to parameter estimation for Bayesian networks.

2.2 Combination of hidden Markov model and super-advisor

Both the hidden Markov model and the super-advisor produce a probability distribution over all possible sequence labelings. We would like to combine these two distributions to one final distribution, and then choose the most probable labeling as our final prediction. In particular, for a given DNA sequence X, additional evidence E, and labeling L, the HMM defines a probability $\Pr(L|X)$ and the super-advisor defines a probability $\Pr(L|E)$. We wish to combine these two probabilities to obtain $\Pr(L|E,X)$. We can do so under the assumption that the DNA sequence X and the evidence E are conditionally independent given the labeling L, that is,

$$\Pr(X,E|L) = \Pr(X|L)\Pr(E|L). \tag{2.1}$$

Following the approach of Tax *et al.* [158], we can use this assumption and Bayes Theorem to derive $\Pr(L|E,X)$:

$$\begin{aligned}
\Pr(L|X,E) &= \Pr(X,E|L)\cdot\frac{\Pr(L)}{\Pr(X,E)} \\
&= \Pr(X|L)\cdot\Pr(E|L)\cdot\frac{\Pr(L)}{\Pr(X,E)} \\
&= \Pr(L|X)\cdot\frac{\Pr(X)}{\Pr(L)}\cdot\Pr(L|E)\cdot\frac{\Pr(E)}{\Pr(L)}\cdot\frac{\Pr(L)}{\Pr(X,E)} \\
&= \frac{\Pr(L|X)\cdot\Pr(L|E)}{\Pr(L)}\cdot\frac{\Pr(X)\cdot\Pr(E)}{\Pr(X,E)}.
\end{aligned}$$

Since the term $\Pr(X)\cdot\Pr(E)/\Pr(X,E)$ does not change with L, the overall probability of a labeling is proportional to the product of the probability assigned by the HMM and the probability assigned by the super-advisor, divided by the prior probability of the labeling:

$$\Pr(L|X,E) \propto \Pr(L|X)\frac{\Pr(L|E)}{\Pr(L)}. \tag{2.2}$$

For the purposes of combination, we thus need to define a prior distribution over all labelings L. For this prior, as for the advisors, we assume independence of positions. The prior distribution is then conveniently in the same form as the super-advisor prediction, and can be efficiently handled by the algorithm for computing the most probable labeling in Section 2.2.2. In particular, let prior(ℓ) be the prior probability of a label ℓ (specifically, the relative frequency with which this label occurs in our training sequences). Then the prior probability of a labeling $L = \ell_1\ell_2\ldots\ell_n$ is:

$$\Pr(L) = \prod_{i=1}^{n}\text{prior}(\ell_i). \tag{2.3}$$

The conditional independence assumption (2.1) required to obtain formula (2.2) is not true in practice, as the information used by the advisors and by the HMM is not completely independent. Still, this formula is not unreasonable to use if one takes care in the design of a system using this framework. For example, in our gene finder, the HMM and advisor predictions are based on different features. The HMM's parameters capture sequence properties such as composition bias, signals, and length distributions, while the advisors are based on sequence conservation with other species and experimental evidence, such EST database similarities. Still, there is likely some dependency.

The formula for combination of the super-advisor and the HMM can be further generalized by incorporating a parameter α that emphasizes or reduces the effect of the super-advisor:

$$\Pr(L|X,E) \propto \Pr(L|X)\frac{\Pr(L|E)^\alpha}{\Pr(L)^\alpha}. \tag{2.4}$$

As we will see in Section 2.7.2, using parameter $\alpha < 1$ will help us deal with problems caused by the assumption that positions are independent in the super-advisor distribution $\Pr(L|E)$.

In order to compute the exact probability $\Pr(L|X,E)$ based on either formula (2.2) or (2.4), we would need the normalization constant $\sum_{L'} \Pr(L'|X) \cdot \Pr(L'|E)^\alpha / \Pr(L')^\alpha$. Although it can be computed, it is not necessary in our application: we only want to find the most probable labeling, not its probability. For that, we do not need to know the constant multiplicative factor.

Finally, we note that if the super-advisor distribution $\Pr(L|E)$ equals the prior distribution $\Pr(L)$, the combined distribution $\Pr(L|X,E)$ equals the HMM distribution $\Pr(L|X)$. Therefore, if there is no additional information available for some position, the super-advisor should return the prior probability distribution, so as to not change the distribution after combination.

2.2.1 Linear and logarithmic opinion pool

The combination of the super-advisor and the HMM in our framework can be viewed as an instance of expert opinion combination. The simplest and frequently used formulas for combining expert opinions that are expressed as probabilities are the linear opinion pool and the logarithmic opinion pool [73]. In a linear opinion pool, the output distribution is a convex combination of the input distributions, while in a logarithmic opinion pool, it is a normalized product of the input distributions. Formally, let $\Pr_i(\theta)$ be the probability assigned to an elementary event θ by expert i, and $\Pr_0(\theta)$ be the combined probability. The logarithmic opinion pool is defined as

$$\Pr_0(\theta) = \frac{\prod_{i=1}^k \Pr_i(\theta)^{w_i}}{\sum_{\theta'} \prod_{i=1}^k \Pr_i(\theta')^{w_i}}, \tag{2.5}$$

and the linear opinion pool is

$$\Pr_0(\theta) = \sum_{i=1}^k w_i \cdot \Pr_i(\theta), \tag{2.6}$$

where the w_i are non-negative weights that sum to one.

Formula (2.4) can be viewed as a generalized logarithmic opinion pool of three experts, in which we allow negative weights. The elementary event θ now corresponds to a labeling L, $\Pr_1(L)$ is the probability $\Pr(L|X)$ defined by the HMM, $\Pr_2(L)$ is the probability $\Pr(L|E)$ defined by the super-advisor, and $\Pr_3(L)$ is the prior probability $\Pr(L)$. The expert weights are $w_1 = 1$, $w_2 = \alpha$ and $w_3 = -\alpha$.

Tax *et al.* [158] conclude that a formula analogous to our combination formula (2.2) has a Bayesian foundation if we combine multiple classifiers, each using independent feature vectors describing the instance. On the other hand, the linear opinion pool can be justified when the experts provide the same sort of probability estimate, with a zero-mean additive error; the impact of error is decreased by averaging the expert predictions. They also study the performance of linear and logarithmic opinion pools on real and generated data. The results show that the logarithmic opinion pool performs better with higher number of classes, especially when the probability estimates provided by experts do not differ significantly from the correct probabilities. Although in our case, the information used by the advisors and the HMM is not completely independent, a generalized logarithmic opinion pool seems reasonable to use.

2.2.2 Algorithm to incorporate super-advisor into HMM

The main advantage of the formula based on the logarithmic opinion pool is that the most probable labeling, $L^* = \arg\max_L \Pr(L|X, E)$, can be efficiently computed by a straightforward modification to the Viterbi algorithm. In contrast, we do not know of any efficient algorithm for combining the super-advisor and an HMM by the linear opinion pool.

The Viterbi algorithm for HMMs finds the most probable state path, $H^* = \arg\max_H \Pr(H|X)$. If every labeling corresponds to a unique state path, it will also find the most probable labeling. Our modified algorithm computes the most probable state path, $H^* = \arg\max_H \Pr(H|X, E)$, in the probability distribution of state paths defined as follows:

$$\Pr(H|X,E) \propto \frac{\Pr(H|X) \cdot \Pr(L_H|E)^\alpha}{\Pr(L_H)^\alpha}, \tag{2.7}$$

where L_H is the labeling corresponding to a state path H. As for HMMs, the probability of a labeling, $\Pr(L|X, E)$, equals the sum of probabilities of all corresponding state paths. Therefore if every labeling corresponds to a unique state path, our algorithm will compute the most probable labeling, $L^* = \arg\max_L \Pr(L|X, E)$.

The algorithm is very similar to the Viterbi algorithm, but in each step, we multiply the emission probability by the probability of the corresponding label provided by the super-advisor for the current position in the sequence and divide it by the prior of the label. Thus, we obtain the following recurrence, which is a simple modification of recurrence (1.2) from the Viterbi algorithm:

$$V[i,k] = \begin{cases} s_k \cdot e_{k,x_1} \cdot r_{1,k} & \text{if } i = 1, \\ \max_l V[i-1,l] \cdot a_{l,k} \cdot e_{k,x_i} \cdot r_{i,k} & \text{otherwise.} \end{cases} \tag{2.8}$$

In this formula, $r_{i,k} = p_{i,s}(\ell)^\alpha/\text{prior}(\ell)^\alpha$, where ℓ is the label of state k and $p_{i,s}$ is the advice of the super-advisor at position i. In the same way, we can also extend the Viterbi algorithm for generalized HMMs. Note that now, $V[i, k]$ is no longer a probability, as we are not computing the normalization factor from Formula (2.7).

2.3 Expressing evidence as advisors

In the previous sections we have defined the concept of an advisor and described a method for combining the super-advisor with an HMM-based gene finder. Before we describe methods for combining multiple advisors to a single super-advisor, we want to develop a better sense of advisor properties by considering how to represent various types of evidence used in gene finding as advisor advice. The goal of this section is to illustrate the usefulness as well as some limitations of our advisor framework. For more details about the advisors used in our actual implementation, see Section 4.3.

Table 2.1: Label set Σ used in our gene finder, ExonHunter.

Label(s)	Meaning
x	intergenic region
b, B	translation start site, two strands
e, E	translation stop site, two strands
d, D	donor site, two strands
a, A	acceptor site, two strands
i, I	intron, two strands
$0, 1, 2$	coding region on forward strand, 3 positions within a codon
$3, 4, 5$	coding region on reverse strand, 3 positions within a codon

The choice of label set Σ has a strong influence on the ease of advisor representation. The label set should satisfy the condition that each labeling corresponds to at most one gene structure annotation (some labelings are not biologically meaningful and thus do not correspond to any annotation), and that each gene structure annotation corresponds to exactly one labeling. We could use, for example, a small set Σ, consisting of four labels: intergenic, intron, coding region on forward strand, and coding region on reverse strand. Provided that a labeling contains at least one intergenic position, we could uniquely determine the reading frame of all coding regions and thus a complete gene structure. However, if some information source could indicate not only the presence of a coding region but also the reading frame, we would not be able to express this detail in an advisor.

To increase the expressiveness of advisors, we use a much richer set of labels in our gene finder, listed in Table 2.1. These labels allow advisors to provide information for all signals marking the boundaries of coding regions, as well as determine the reading frame of coding regions.

Advisors can easily represent information about location of signals. We can incorporate simple probabilistic models of signals directly into the HMM, but not all methods for signal detection can be expressed in an HMM framework. On the other hand, advisors can use arbitrary machine learning methods for signal recognition. At every position, the advice of such an advisor will be a binary partition, such as $\{\{b\}, \Sigma \setminus \{b\}\}$, and an estimate of the probability that that position is the location of the signal. Optionally, the advisor may produce vacuous advice at some positions if it cannot reliably estimate the probability that the position is a signal. However, there is a problem with advisors for signals, since they typically rely on sequence information. They thus violate the assumption of independence between the sequence and the advisor information that we needed to obtain formula (2.2) for combining the super-advisor and the HMM. This problem may be less pronounced if the advisor uses features not captured in the HMM, such as long-range sequence dependencies or homology information.

Sequence similarity information, in the form of protein, EST, and genome alignments is perhaps the most readily available source of external evidence for gene finding. Different sources of alignments have different properties. These differences can be reflected in the choice of the label partition that a particular advisor will use. For example, a protein alignment indicates a presence of a coding region and also can be used to determine the strand and the reading frame. Therefore, it is appropriate to use a binary partition of the form $\{\{0\}, \Sigma \setminus \{0\}\}$, where the label 0 can be replaced by the label corresponding to any coding frame. On the other hand, an EST alignment may not directly indicate strand or frame. This uncertainty can be expressed with the partition $\{\{0,1,2,3,4,5\}, \Sigma \setminus \{0,1,2,3,4,5\}\}$. In addition, EST alignments can extend to UTRs (which we label with the intergenic label, x). Therefore, we should use the set of labels $\{0,1,2,3,4,5,x\}$ in the partition to represent EST information.

For positions not covered by any alignment, an advisor can either use the same partition and predict a lower probability of a coding region, or simply return vacuous advice with the trivial partition $\{\Sigma\}$. We use the second option, as the probability of unmatched true exons is hard to estimate and depends on the coverage of the database used, rather than an intrinsic properties of the sequences involved.

User knowledge can also be expressed as an advisor. This advisor returns vacuous advice at all positions except for those positions where the user wants to prevent or encourage a region to be predicted as coding, with whatever level of detail is appropriate.

Finally, any gene prediction program can be used as an advisor as well. However, as we have discussed, it is preferable that advisors use information independent from the information used in the HMM. The predictions of another *ab initio* gene finder will be likely based on the same information as the HMM, and will be biased towards similar predictions. Therefore, they may enforce errors made by the HMM. The goal of the advisors is to bring new information into the gene finding process.

As we have seen, the advisor framework conveniently captures available information at a single position from a wide range of sources. However, the assumption that positions are independent causes difficulties in expressing information pertaining to whole regions or dependencies between positions. For example an EST may cover parts of two exons and thus it will align to two regions of the genome separated by an intron. To indicate that the region between the two alignments is likely an intron, we can increase the probability of each individual position being from intron, but we cannot indicate that the whole region should be one uninterrupted intron. Another example is evidence suggesting that a certain region of the sequence is very likely to contain a translation start site. In our framework, we can only express probabilities of individual positions. If the region is long, no single position is very likely to contain the translation start site, so the effect is muted.

To construct an advisor from external evidence, we need to specify a label set partition $\pi_{i,a}$ and a probability distribution $p_{i,a}(S)$ over all sets S in $\pi_{i,a}$. As we have seen, the partition is determined manually, based on known properties of a particular evidence source. Some evidence sources may explicitly provide an estimate of the probability $p_{i,a}(S)$. For example, probabilistic models for signal detection may produce a probability that a sequence position is instance of the signal. In other cases, we may need to estimate the probabilities from a training set by counting for every set S in the partition π_a, how often is the true label in this set. We study this problem in more detail in Section 4.3.1.

2.4 Combination of advisors into super-advisor

In this section, we study how to combine multiple advisors to one super-advisor. The advisors, as well as the super-advisor, assume independence of individual positions in the sequence. Therefore, it is reasonable to combine advisor predictions at each sequence position separately.

Let us consider a fixed position in the sequence. Each advisor a chooses a partition π_a of the label set Σ and specifies a probability distribution $p_a(S)$ over the label sets S in π_a. Based on the information provided by the advisors, we want to estimate a probability distribution $\overline{x} = (x_1, \ldots, x_{|\Sigma|})$ over the labels in set Σ.

We want to find a vector $\overline{x}$ that best summarizes the advice provided by the individual advisors and use it as the probability distribution of labels given by the super-advisor at the sequence position in question.

The problem of combining advisor predictions is again related to the expert combination problem. However, standard expert combination rules, such as the linear or logarithmic opinion pool, cannot be used here, because the individual advisors do not completely specify their probabilistic distributions. Therefore, we propose a new combination rule in Section 2.4.1 that finds a probability distribution that minimizes the distance to a set of incompletely specified distributions in the form of advisors. We show in Section 2.4.2 that this rule exhibits intuitive properties in important special cases. In particular, it generalizes the linear opinion pool, and is equivalent to it when all advisors specify complete probability distributions. In Section 2.5, we consider additional variations of our combination rule and present experiments comparing their performance. The combination rule and its variants use weights that represent reliability of each advisor's predictions. In Section 2.6, we study methods for training these weights.

2.4.1 Combining advisors to minimize distance to the super-advisor

As we have noted above, combination of multiple advisors to one super-advisor is complicated by the different partitions produced by different advisors. One natural property of a combination formula is that if all advisors give the same advice with a complete probability distribution, the super-advisor prediction should be the same as well. This property holds for both the linear and logarithmic opinion pools, assuming the weights add to one. We will extend this property to advice with partial probability distributions: our method will construct a super-advisor prediction consistent with all advice, if such a prediction exists. In practice however, the advice of different advisors is often inconsistent. Then, we will choose a super-advisor prediction as close as possible to all advisors.

Formally, we will consider the probability $p_a(S)$ for each advisor a and each set S of the partition π_a as a constraint on the super-advisor's prediction. If possible, the super-advisor prediction $\overline{x}$ will satisfy all these constraints: for every set S in π_a, the vector $\overline{x}$ will satisfy $\sum_{j \in S} x_j = p_a(S)$. If no vector $\overline{x}$ can satisfy all the constraints, we will choose a vector $\overline{x}$ that violates them as little as possible.

For this purpose, we define a distance between a probability vector $\overline{x}$ and the advice of a single advisor a as follows: Assume $\pi_a = \{S_1, \ldots, S_k\}$, and that the prediction of advisor a is the vector $\overline{p} = (p_a(S_1), \ldots, p_a(S_k))$. Given a vector of label probabilities $\overline{x}$, we will denote $\overline{x}(S)$ the sum of probabilities of labels in set S, that is, $\overline{x}(S) = \sum_{j \in S} x_j$. For a vector $\overline{x}$ and the partition π_a we produce a vector of sums $\overline{x}' = (\overline{x}(S_1), \ldots, \overline{x}(S_k))$ and then measure the distance between the vectors $\overline{p}$ and $\overline{x}'$ using the square of the L_2 norm: $||\overline{p} - \overline{x}'||_2^2 = \sum_{i=1}^{k} (p_a(S_i) - \overline{x}(S))^2$. However, we modify this formula to give higher weight to sets with smaller prior probability, because the same absolute change in a probability of such a label will have greater impact on the overall prediction, once we combine it with the HMM. The prior of a label set S is defined as the sum of the prior probabilities of its constituent labels $\text{prior}(S) = \sum_{j \in S} \text{prior}(j)$.

With this in mind, we define the distance between a vector $\overline{x}$ and the advice of an advisor a by the following formula:

$$\mathrm{dist}_a(\overline{x}) = \sum_{S\in\pi_a} \frac{1}{\mathrm{prior}(S)} \cdot \left(p_a(S) - \overline{x}(S)\right)^2, \tag{2.9}$$

The vector $\overline{x}$ of super-advisor predictions will be chosen so that it represents a valid probability distribution and minimizes the weighted sum of distances from the predictions of individual advisors. This leads to the following constrained optimization problem, where w_a is the positive weight of advisor a:

$$\begin{aligned} \text{minimize} \quad & \sum_a w_a \sum_{S\in\pi_a} \mathrm{dist}_a(\overline{x}) && (2.10)\\ \text{subject to} \quad & \sum_{j\in\Sigma} x_j = 1 && (2.11)\\ & x_j \geq 0 \quad \text{for all labels } j \in \Sigma && (2.12) \end{aligned}$$

The solution of this program is a probability vector $\overline{x}$, where x_j is the probability of label j in the super-advisor prediction. If there exists a solution consistent with the advice of all advisors, then this solution has zero objective value and is chosen as the optimum.

The solution of the optimization problem (2.10) may have some probabilities x_j equal to 0. In such a case, label j is not allowed at all at the current position in the final gene structure. Since the advisors and their combination may make errors, it is useful to prevent zero or extremely small label probabilities in the super-advisor vector. In our gene finder, we constrain the values x_j to be at least $\mathrm{prior}(j)/100$ and at most $100 \cdot \mathrm{prior}(j)$.

2.4.1.1 Quadratic programming in advisor combination

The optimization problem (2.10) is a convex quadratic program. In general, a convex quadratic program is an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\overline{x}^T H \overline{x} + \overline{c}^T \overline{x} \\ \text{subject to} \quad & A\overline{x} \geq \overline{b}, \end{aligned}$$

where $\overline{x}$ is the unknown (column) vector of length n, H is a positive semidefinite $n \times n$ matrix, A is an $n \times m$ matrix, $\overline{c}$ is a vector of length n and $\overline{b}$ is a vector of length m. In our case, the length of the unknown vector x is $n = |\Sigma|$. We have $m = n + 2$ linear constraints in matrix A: a lower bound for each variable, and a matching upper and lower bound for the sum of the x_i.

The objective value in problem (2.10) is the sum of terms of the form

$$\frac{w_a}{\mathrm{prior}(S)} \cdot \left(p_a(S) - \overline{x}(S)\right)^2 = \frac{w_a \cdot p_a(S)^2}{\mathrm{prior}(S)} - \frac{2w_a \cdot p_a(S)}{\mathrm{prior}(S)} \left(\sum_{j\in S} x_j\right) + \frac{w_a}{\mathrm{prior}(S)} \left(\sum_{j\in S} x_j\right)^2. \tag{2.13}$$

The first term on the right-hand side is a constant and can be ignored, the second term will contribute the amount $-2w_a \cdot p_a(S)/\mathrm{prior}(S)$ to c_j for every $j \in S$, and the third term will contribute the amount $2w_a/\mathrm{prior}(S)$ to $h_{j,k}$ for every $j, k \in S$. Clearly H is positive semidefinite, since $(1/2)\overline{x}^T H \overline{x}$ is a sum of terms of the form $(w_a/\mathrm{prior}(S))\overline{x}(S)^2$.

Our convex quadratic program, although typically quite small, needs to be solved at every position of the sequence. Therefore, the running time is quite important. Fortunately, convex quadratic programs with integer or rational coefficients can be solved by interior point methods in polynomial time in the length of the input, measured in bits (a detailed treatment can be found in [162]). In addition to algorithms with good theoretical performance, there exist numerous practical implementations. In our gene finder, we use the function `nag_opt_lin_lsq` from the NAG C Library [126]. The running time for the convex quadratic programs at each sequence position is reasonable compared to the other components of our gene finding system (see Section 2.5.4 for experimental data).

2.4.2 Properties of advisor combination

In this section, we study properties of the combination formula defined as the solution to the quadratic program (2.10). First, we show that if all advisors produce a complete probability distribution, the super-advisor will be a linear combination of them. Thus, the linear opinion pool is a special case of our method. We will also illustrate the influence of weights and priors in the case of binary partitions and we show that by adding a single advisor with advice equal to prior we can avoid under-constrained quadratic programs with multiple optima.

2.4.2.1 Linear combination as a special case of advisor combination

The following lemma shows that when several advisors use the same partition, we can obtain an equivalent quadratic program by combining these advisors to a single advisor using linear combination.

Lemma 2. *All advisors that produce advice with the same partition π at a particular sequence position can be replaced by a single advisor whose weight is the sum of the weights of the original advisors, and whose advice is a linear combination of their advice.*

Proof. Let A be the set of all advisors that use the partition π and let S be a partition set in π. The part of the objective function concerning the advice of advisors in A for set S has this form:

$$\frac{1}{\mathrm{prior}(S)} \cdot \sum_{a \in A} w_a \left(p_a(S) - \overline{x}(S)\right)^2 . \tag{2.14}$$

We denote $W = \sum_{a \in A} w_a$, and rearrange the sum as follows:

$$\begin{aligned}
&\sum_{a \in A} w_a \left(p_a(S) - \overline{x}(S)\right)^2 \\
&\quad = \sum_{a \in A} w_a p_a(S)^2 - 2 \left(\sum_{a \in A} w_a p_a(S)\right) \overline{x}(S) + \underbrace{\left(\sum_{a \in A} w_a\right)}_{W} \overline{x}(S)^2 \\
&\quad = \sum_{a \in A} w_a p_a(S)^2 + W \left(-2\frac{\sum_{a \in A} w_a p_a(S)}{W} \overline{x}(S) + \overline{x}(S)^2\right) \\
&\quad = \underbrace{\sum_{a \in A} w_a p_a(S)^2 - W \left(\frac{\sum_{a \in A} w_a p_a(S)}{W}\right)^2}_{C} + W \left(\frac{\sum_{a \in A} w_a p_a(S)}{W} - \overline{x}(S)\right)^2
\end{aligned} \tag{2.15}$$

Therefore, the part of the objective function concerning the advice of advisors in A for set S is equal to

$$\frac{1}{\text{prior}(S)}\left(\sum_{a\in A} w_a\right)\left(\frac{\sum_{a\in A} w_a p_a(S)}{\sum_{a\in A} w_a} - \overline{x}(S)\right)^2 + \frac{1}{\text{prior}(S)}C, \tag{2.16}$$

where C is a constant not depending on vector $\overline{x}$. Omitting the constant term, we have obtained an expression corresponding to one advisor with weight $\sum_{a\in A} w_a$ and whose prediction for set S is equal to $(\sum_{a\in A} w_a p_a(S))/(\sum_{a\in A} w_a)$. □

Corollary 3. *If the advice of all advisors uses a complete partition,* i.e., $\pi_a = \{\{j\} : j \in \Sigma\}$, *then the super-advisor prediction is a convex combination of the advisor advice, weighted by the advisor weights. In particular, the super-advisor prediction for label j is*

$$x_j = \frac{\sum_a w_a p_a(\{j\})}{\sum_a w_a}. \tag{2.17}$$

This corollary shows that in the special case when all advisors provide a complete probability distribution, our combination rule is equivalent to the linear opinion pool. Even the advisor weights map directly to the weights in the linear opinion pool. This behavior of the combination method is encouraging, because the linear opinion pool is a very natural combination method. We have discussed that it is desirable to make the information used by advisors independent of the information used by the HMM. However, the information sources of individual advisors might be dependent on each other (for example several advisors using different methods to predict the same signal). In such situations, linear combination is an appropriate combination rule and indeed it is used to combine their predictions in a special case of the quadratic program (2.10).

2.4.2.2 Advisors with binary partitions and the influence of priors

In our system, most of the advisors provide predictions in the form of bipartitions, in which one partition set consists of one label or a small set of labels of interest, and the other partition set contains all other labels. Consider the special case of this scenario in which each advisor partitions the labels into two sets: a singleton set and the rest, and each label is in the singleton set of exactly one advisor. Note, that multiple advisors with the same singleton set can be combined into one advisor using Lemma 2.

Lemma 4. *Assume that the set of advisors consists of one advisor for each label j; this advisor has weight w_j, partition $\{\{j\}, \Sigma\setminus\{j\}\}$, and assigns probability p_j to label j. Moreover, assume that $\sum_{j\in\Sigma} p_j \le 1$. Then the super-advisor $\overline{x}$ satisfies the following*

$$x_j = p_j + C\cdot\frac{\text{prior}(j)(1-\text{prior}(j))}{w_j}, \tag{2.18}$$

where C is a normalizing factor independent of the label j.

Proof. The objective function (2.10) can be written using this notation as follows:

$$\sum_{j\in\Sigma} w_j\cdot\left[\frac{(p_j - x_j)^2}{\text{prior}(j)} + \frac{((1-p_j)-(1-x_j))^2}{1-\text{prior}(j)}\right] = \sum_{j\in\Sigma}\frac{w_j}{\text{prior}(j)(1-\text{prior}(j))}\cdot(p_j - x_j)^2 \tag{2.19}$$

Using constraint (2.11), we replace one variable x_k with $1 - \sum_{j \neq k} x_j$. At the optimum, $\overline{x}$, all partial derivatives equal zero, which gives the system of linear equations:

$$-2\frac{w_j(p_j - x_j)}{\text{prior}(j)(1 - \text{prior}(j))} + 2\frac{w_k(p_k - 1 - \sum_{j' \neq k} x_{j'})}{\text{prior}(k)(1 - \text{prior}(k))} = 0 \quad \text{for each } j \in \Sigma \setminus \{k\}. \tag{2.20}$$

The unique solution of this system has the form

$$x_j = p_j + \frac{B}{A} \cdot \frac{\text{prior}(j)(1 - \text{prior}(j))}{w_j} \quad (1 \leq j \leq n) \tag{2.21}$$

$$\text{where} \quad A = \sum_{j \in \Sigma} \frac{\text{prior}(j)(1 - \text{prior}(j))}{w_j} \quad \text{and} \quad B = 1 - \sum_{j \in \Sigma} p_j.$$

Since B is non-negative, this solution clearly satisfies constraints (2.12) and (2.11) and thus it is a solution of the quadratic program. We will set $C = B/A$ to obtain the claim of the lemma. $\square$

This lemma has the following intuitive meaning. If the sum of the advisor predictions, $\sum_{j \in \Sigma} p_j$, equals one, every x_j will equal p_j. Otherwise, the remainder by which the sum of advisor predictions differs from 1 is divided among labels, so that the surplus amount assigned to label j is proportional to $\text{prior}(j)(1 - \text{prior}(j))/w_j$. The difference between the super-advisor value x_j and the advisor value p_j is smaller for advisors with higher weight w_j, which is consistent with the intuitive meaning of advisor weight: their predictions should be respected more closely.

The lemma also illustrates the influence of the prior term $1/\text{prior}(S)$ in the definition of $\text{dist}_a(\overline{x})$. The quantity $\text{prior}(j)(1 - \text{prior}(j))$ is maximized when $\text{prior}(j) = 1/2$ and decreases to zero as $\text{prior}(j)$ goes to zero or one. Thus, the absolute difference between x_j and p_j will be smaller for labels with very high or very low priors.

In gene finding, individual labels have very different priors, and it turns out to be important to consider this issue. For example, more than half of the human genome is covered by intergenic regions, while coding regions are estimated to cover only 1.2% of the genome [52]. In addition, in our gene finder we also have separate labels for the signals at the boundaries of coding regions (translation start, translation stop, donor and acceptor splice site). These have even lower frequency than coding regions; for example, donor sites occur approximately once in seventeen thousand bases in the human genome annotation [52].

Let us consider an example of two labels j and k with very different priors; for example, suppose $\text{prior}(j) = 0.5$ and $\text{prior}(k) = 0.0001$. Let us assume that due to the influence of advisors, the super-advisor prediction for one of the labels j and k increases by 0.01, compared to its prior. In the case of label j, the probability changes from 0.50 to 0.51, whereas in the case of label k, it changes from 0.0001 to about 0.01. When we use formula (2.2) to combine the super-advisor and the HMM, the super-advisor will be divided by the prior. Thus the probability of labelings that have label j at this particular position will increase $0.51/0.5 = 1.02$ times compared to the probability defined by the HMM, whereas the probability of labelings that have label k will increase 100 times. Therefore, any artifacts of advisor combination have much greater impact in final prediction for labels with very small prior. To prevent such artifacts, changes in the probabilities of labels with small prior have higher penalty in the definition of the distance $\text{dist}_a(\overline{x})$. The previous lemma confirms that this measure has the desired impact in at least some circumstances. Changes in a label with prior very close to one also have higher penalty in the previous lemma, because then the rest of the labels must have prior very close to zero.

Although the prior term in the definition of $\mathrm{dist}_a(\overline{x})$ is theoretically justified, our experiments in Section 2.5.4 demonstrate that even if we omit this term, the gene finding accuracy of our gene finder does not change very much. However, this behavior might be specific to our set of advisors.

2.4.2.3 Under-constrained advisor combination

It may be the case that values of some labels are not specified by any advisor or combination of advisors. In particular, we can have a set of labels S such that the partition of every advisor contains some superset of S. In such a case, the objective value of our quadratic program (2.10) is the same for all vectors $\overline{x}$ that differ only in the distribution of probabilities within the set S. Thus, the quadratic program is under-constrained and has multiple optimal solutions, one of which will be chosen essentially arbitrarily by the quadratic program solver.

The problem of under-constrained systems can be solved by adding the prior distribution of labels as an additional advisor at each site, with low weight. In this way, each label is in a singleton set for at least one advisor. If the weight of the prior advisor is sufficiently small, the influence on labels specified by other advisors will be negligible. On the other hand, the probability of labels that are not specified by any other advisor will be divided proportionally to their priors, as is demonstrated in the following lemma.

Lemma 5. *Suppose that all advisors assign labels j and k to the same partition set. If we add a single advisor whose partition is complete and whose advice is the prior probability for each label, then in the super-advisor prediction $\overline{x}$, the ratio x_j/x_k will be equal to their prior ratio,* $\mathrm{prior}(j)/\mathrm{prior}(k)$.

Proof. Fix the sum of $x_j + x_k$ to be equal to some constant s, and consider how the objective function of the quadratic program (2.10) changes as we vary the ratio of x_j to x_k. The only terms in the objective function that change are the ones corresponding to the added prior advisor with some weight w:

$$\frac{w}{\mathrm{prior}(j)}(\mathrm{prior}(j) - x_j)^2 + \frac{w}{\mathrm{prior}(k)}(\mathrm{prior}(k) - x_k)^2. \tag{2.22}$$

We set $x_k = s - x_j$, take the derivative, and obtain that this expression is minimized when $x_j = s \cdot \mathrm{prior}(j)/(\mathrm{prior}(j)+\mathrm{prior}(k))$ and $x_k = s \cdot \mathrm{prior}(k)/(\mathrm{prior}(j)+\mathrm{prior}(k))$. Notice that both x_j and x_k are between 0 and s and thus constitute a valid assignment of probabilities. The optimal ratio between x_j and x_k is $\mathrm{prior}(j)/\mathrm{prior}(k)$ for any fixed sum s, including the case when s equals the optimal value of $x_j + x_k$ in the quadratic program (2.10). □

Corollary 6. *Assume that all advisors produce vacuous advice with the trivial partition $\pi_a = \{\Sigma\}$. If we add the prior advisor, the super-advisor prediction will be equal to the prior distribution.*

This corollary shows that if all advisors produce vacuous advice, the super-advisor prediction will be equal to the prior. In this case, any influence of the super-advisor is canceled out in formula (2.2) for combining the HMM and the super-advisor, as the probability after combination equals the HMM's probability times the ratio between the super-advisor prediction and the prior. As we noted before, if all advice is vacuous for each position of the sequence, the overall prediction will be simply the most probable labeling as defined by the HMM.

Note that Lemma 5 does not hold if the factor $1/\mathrm{prior}(S)$ is omitted from the definition of $\mathrm{dist}_a(\overline{x})$, which is another point in favor of including this factor in the distance formula.

2.5 Variants of advisor combination

We can create variations of the advisor combination rule by changing the definition of the distance $\mathrm{dist}_a(\overline{x})$ between an advisor a and the super-advisor probability vector $\overline{x}$. In this section we briefly discuss variants based on the L_1, and L_∞ measures, and on the relative entropy. Then we discuss a naive combination method that simplifies the problem of advisor combination by converting all advice to complete probability distributions and combines them by linear combination. Such a simple combination rule loses some information originally supplied by the advisors. We conclude with a small experimental study illustrating the impact of the combination method used.

2.5.1 Distance measured by L_1 and L_∞

Instead of using the squared Euclidean distance L_2, we could use L_1 distance

$$\mathrm{dist}_a^{(L_1)}(\overline{x}) = \sum_{S \in \pi_a} |p_a(S) - \overline{x}(S)| \,. \tag{2.23}$$

Note that for simplicity, we ignore the priors for now. Compare this formula to a modified version of dist_a, without the $1/\mathrm{prior}(S)$ term:

$$\mathrm{dist}_a^{(L_2^2)}(\overline{x}) = \sum_{S \in \pi_a} (p_a(S) - \overline{x}(S))^2 \,. \tag{2.24}$$

Minimizing the weighted sum of L_1 distances can be expressed as a linear program and thus solved in polynomial time. In the program, we introduce a new variable $y_{a,S}$ for every constraint to remove the absolute values:

$$\begin{array}{rl} \text{minimize} & \displaystyle\sum_a w_a \sum_{S \in \pi_a} y_{a,S} \\ \text{subject to} & y_{a,S} \geq p_a(S) - \overline{x}(S) \\ & y_{a,S} \geq -(p_a(S) - \overline{x}(S)) \\ & \displaystyle\sum_{j \in \Sigma} x_j = 1 \\ & x_j \geq 0 \quad \text{for all labels } j \in \Sigma \end{array}$$

We can also construct a combination rule based on the L_∞ distance. We obtain one constraint for each $p_a(S)$, and minimize the maximum difference from a single constraint:

$$\mathrm{dist}_a^{(L_\infty)}(\overline{x}) = \max_{a, S \in \pi_a} w_a \, |p_a(S) - \overline{x}(S)| \,. \tag{2.25}$$

This formulation can also be expressed as a linear program by introducing a single variable y equal to the maximum difference in the optimal solution:

$$\begin{array}{rl} \text{minimize} & y \\ \text{subject to} & y \geq w_a (p_a(S) - \overline{x}(S)) \\ & y \geq -w_a (p_a(S) - \overline{x}(S)) \\ & \displaystyle\sum_{j \in \Sigma} x_j = 1 \\ & x_j \geq 0 \quad \forall j \in \Sigma \end{array}$$

We can better understand the differences between the L_1, L_∞ and squared L_2-based distance measures by considering the simple case of two labels $\Sigma = \{0, 1\}$ and several advisors of equal weight, each giving non-vacuous advice that can be represented by a single number $p_a(\{1\})$. Combination based on the $\mathrm{dist}_a^{(L_2^2)}$ measure will select the mean of the $p_a(\{1\})$ values as the final super-advisor prediction x_1 for label 1, and $\mathrm{dist}_a^{(L_\infty)}$ will select the mean of the two extreme values (the minimum and maximum $p_a(\{1\})$. This suggests that the measure $\mathrm{dist}_a^{(L_\infty)}$ seems undesirable, because it depends only on outliers among advisors; other advisors have no influence.

For the $\mathrm{dist}_a^{(L_1)}$ measure, the result is the median of the $p_a(\{1\})$ values. To see that this is the case, consider the change of the objective value when we add some positive constant ϵ to x_1. By this change, $\mathrm{dist}_a^{(L_1)}(\overline{x})$ increases by ϵ for all advisors a with prediction $p_a(\{1\})$ smaller than the median. On the other hand, the distance will decrease by at most ϵ for all advisors with prediction greater than median. Since the number of advisors is the same in both groups, the objective value will not decrease for any positive value of ϵ. The argument is analogous for negative values of ϵ.

One disadvantage of the L_1 formulation is that in some situations, it has many optimal solutions, some of which may be counter-intuitive. If there is an even number of advisors (say $2k$) in the previous example, and we order them by their value of $p_a(\{1\})$, then any value between the predictions $p_a(\{1\})$ of the kth advisor and the one after that is optimal. This does not seem to be a problem, but similar behavior in other situations may cause problems. For example, consider the set of labels $\Sigma = \{A, B, C\}$ and these three (bipartite) advisors, all of the same weight 1:

$$\begin{array}{ll} p_{a_1}(\{A\}) = 0.4 & p_{a_1}(\{B, C\}) = 0.6 \\ p_{a_2}(\{B\}) = 0.4 & p_{a_2}(\{A, C\}) = 0.6 \\ p_{a_3}(\{C\}) = 0.4 & p_{a_3}(\{A, B\}) = 0.6 \end{array} \tag{2.26}$$

These three advisors are incompatible. The optimum solution when using $\mathrm{dist}_a^{(L_2^2)}$ is $(x_A = 1/3, x_B = 1/3, x_C = 1/3)$, but $\mathrm{dist}_a^{(L_1)}$ leads to the same objective value for many other solutions, for example, the vector $(x_A = 0.4, x_B = 0.4, x_C = 0.2)$. This is not very good because the advisors give no indication that C should have lower probability than the rest.

In a situation with non-equal weights, $\mathrm{dist}_a^{(L_1)}$ performs even worse. Consider the same advisors, but with an increased weight on the first advisor. No matter how little this weight is increased, the optimal solution will always have $x_A = 0.4$. The actual value of the weight does not have any influence, as long as it is greater than the other two. Such behavior of advisor weights is counter-intuitive and would create difficulties for training of advisor weights.

In contrast, for $\mathrm{dist}_a^{(L_2^2)}$, the value of x_A is a smoothly increasing function $0.4 - 1/(10w + 5)$ of the weight w of the first advisor. For example, if the first advisor has weight $w = 1.5$, the solution is $(x_A = 0.35, x_B = 0.325, x_C = 0.325)$, while if the first advisor has weight $w = 2$, the solution is $(x_A = 0.36, x_B = 0.32, x_C = 0.32)$.

2.5.2 Distance measured by relative entropy

We can also use relative entropy to define the distance between an advisor a and the probability distribution represented by the vector $\overline{x}$. The relative entropy (also known as cross entropy or Kullback-Leibler (KL) divergence) of two discrete probability distributions P and Q is defined as follows:

$$H(Q||P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)}. \tag{2.27}$$

It was introduced by Kullback and Leibler [104], and it is often used to measure the divergence of two probability distributions. Relative entropy is always non-negative and it is equal to zero only if the two distributions are equal. However it is not symmetric, that is, $H(Q||P)$ is not always equal $H(P||Q)$.

The asymmetry of relative entropy leads to two natural variants of the combination rule:

$$\begin{aligned}
\mathrm{dist}_a^{(KL1)}(\overline{x}) &= \sum_{S\in\pi_a} \overline{x}(S)\log\frac{\overline{x}(S)}{p_a(S)},\\
\mathrm{dist}_a^{(KL2)}(\overline{x}) &= \sum_{S\in\pi_a} p_a(S)\log\frac{p_a(S)}{\overline{x}(S)}.
\end{aligned}$$

Both distance measures lead to convex objective functions with respect to vector $\overline{x}$ because both $-\log(x)$ and $x\log(x/c)$, for a positive constant c, are convex functions. Therefore, they can be optimized efficiently by interior point methods [23].

The following two lemmas examine the behavior of the two measures in the special case when all advisors produce a complete probability distribution, analogously to Lemma 2.

Lemma 7. *All advisors that produce advice with the same partition π at a particular sequence position can be replaced under the $\mathrm{dist}_a^{(KL1)}$ combination rule by a single advisor, whose weight is the sum of weights of original advisors, and whose advice is proportional to the weighted geometric mean of their advice.*

Proof. Let A be the set of all advisors that predict partition π and let S be a partition set in π. The part of the $\mathrm{dist}_a^{(KL1)}$ objective function concerning the advice of advisors in A for set S has the form

$$\begin{aligned}
\sum_{a\in A} w_a\cdot\overline{x}(S)\cdot\log\frac{\overline{x}(S)}{p_a(S)} &= \sum_{a\in A}\overline{x}(S)\cdot\log\frac{\overline{x}(S)^{w_a}}{p_a(S)^{w_a}}\\
&= \overline{x}(S)\cdot\log\frac{\overline{x}(S)^{\left(\sum_{a\in A}w_a\right)}}{\prod_{a\in A}p_a(S)^{w_a}}\\
&= \left(\sum_{a\in A}w_a\right)\cdot\overline{x}(S)\cdot\log\frac{\overline{x}(S)}{\left(\prod_{a\in A}p_a(S)^{w_a}\right)^{1/\left(\sum_{a\in A}w_a\right)}}.
\end{aligned}$$

Notice that $\left(\prod_{a\in A}p_a(S)^{w_a}\right)^{1/\left(\sum_{a\in A}w_a\right)}$ is the weighted geometric mean of the advisor predictions for the set S. Let us denote this quantity $g(S)$, and let $G=\sum_{S\in\pi}g(S)$. The expression we have obtained has almost the desired form, except that the sum G of the weighted geometric means $g(S)$ is not necessarily equal 1. To normalize them we consider the overall contribution of all advisors in the set A:

$$\begin{aligned}
&\left(\sum_{a\in A} w_a\right)\sum_{S\in\pi}\overline{x}(S)\log\frac{\overline{x}(S)}{g(S)} \\
&\quad= \left(\sum_{a\in A} w_a\right)\sum_{S\in\pi}\left(\overline{x}(S)\cdot\log\frac{\overline{x}(S)}{g(S)/G}+\overline{x}(S)\log G\right) \\
&\quad= \left(\sum_{a\in A} w_a\right)\sum_{S\in\pi}\overline{x}(S)\log\frac{\overline{x}(S)}{g(S)/G}+\left(\sum_{a\in A} w_a\right)\left(\sum_{S\in\pi}\overline{x}(S)\right)\log G \\
&\quad= \left(\sum_{a\in A} w_a\right)\sum_{S\in\pi}\overline{x}(S)\log\frac{\overline{x}(S)}{g(S)/G}+\left(\sum_{a\in A} w_a\right)\log G
\end{aligned} \tag{2.28}$$

In the last step we have used the fact that $\sum_{S\in\pi}\overline{x}(S)=1$ for all feasible vectors $\overline{x}$, hence the last term does not depend on $\overline{x}$ and can be omitted from the objective function. Therefore, the advisors in the set A have the same contribution to the objective function as a single advisor with weight $\sum_{a\in A} w_a$ and the prediction for set S equal to $g(S)/G$, which is the normalized weighted geometric mean. □

Lemma 8. *All advisors that produce advice with the same partition π at a particular sequence position can be replaced under the $\mathrm{dist}_a^{(KL2)}$ combination rule by a single advisor, whose weight is the sum of the weights of the original advisors, and whose advice is a linear combination of their advice.*

Proof. The objective function is a sum of terms of the form $w_a p_a(S)\log(p_a(S)/\overline{x}(S))$. This can be rewritten as a sum of two terms $w_a p_a(S)\log p_a(S) - w_a p_a \log\overline{x}(S)$, where the first term does not depend on $\overline{x}$ and can thus be ignored.

As in the previous lemma, we denote by A the set of advisors with partition π and study the part of the objective function concerning the advice of advisors in A for some set $S\in\pi$:

$$\sum_{a\in A} -w_a p_a \log\overline{x}(S) = -\left(\sum_{a\in A} w_a\right)\left(\frac{\sum_{a\in A} w_a p_a}{\sum_{a\in A} w_a}\right)\log\overline{x}(S). \tag{2.29}$$

By adding the constant term

$$\left(\sum_{a\in A} w_a\right)\left(\frac{\sum_{a\in A} w_a p_a}{\sum_{a\in A} w_a}\right)\log\left(\frac{\sum_{a\in A} w_a p_a}{\sum_{a\in A} w_a}\right), \tag{2.30}$$

we can obtain that the part of the objective function concerning the advice of advisors in A for a set S is, up to a constant term, equal to

$$\left(\sum_{a\in A} w_a\right)\left(\frac{\sum_{a\in A} w_a p_a}{\sum_{a\in A} w_a}\right)\log\frac{\left(\frac{\sum_{a\in A} w_a p_a}{\sum_{a\in A} w_a}\right)}{\overline{x}(S)}. \tag{2.31}$$

Thus, the advisors in set A can be replaced by a single advisor with weight $\sum_{a\in A} w_a$ and prediction for set S equal to the linear combination $\sum_{a\in A} w_a p_a / \sum_{a\in A} w_a$. □

These lemmas show that for both distances based on relative entropy, we can combine several advisors with the same partition to a single advisor. In particular, if all advisors provide complete probability distributions over all labels, the advisor obtained by their combination will be equal the super-advisor obtained by minimizing the relative entropy. In case of $\mathrm{dist}_a^{(KL2)}$, the super-advisor prediction will be simply a linear combination of the advice, as we would also obtain by using our original squared L_2 based distance (see Lemma 2). However for $\mathrm{dist}_a^{(KL1)}$, the super-advisor prediction will be a weighted geometric mean of the advice, which corresponds to the logarithmic opinion pool.

A formula analogous to $\mathrm{dist}_a^{(KL2)}$ is used by Vomlel [166]. The author studies the problem of constructing a multivariate distribution based on a set of inconsistent constraints in the form of marginals, when each constraint specifies a marginal probability distribution over a subset of all variables. He proposes to minimize a weighted sum of relative entropies between the target distribution and the constraints. Our problem is not completely identical since the advisors are not the marginals of a multivariate distribution.

2.5.3 Naive advisor combination

Advisor combination is difficult because advisors do not provide complete probability distributions. A simple option is to express their advice as a distribution over all labels and then use a standard combination rule, such as the linear opinion pool. If an advisor a has a set S with more than one label in its partition π_a, we can divide the probability $p_a(S)$ among the individual members of set S proportionally to their prior probabilities. The super-advisor vector $\overline{x}$ obtained by this naive combination method has the form:

$$x_j = \sum_{\substack{a \\ S\in\pi_a : j\in S}} \frac{w_a \cdot p_a(S) \cdot \mathrm{prior}(j)}{\mathrm{prior}(S) \cdot \sum_{a'} w_{a'}}. \tag{2.32}$$

When we divide the probability $p_a(S)$ among the labels in the set S, we obtain values that are not directly based on evidence and thus may decrease the reliability of the system. For example, assume that one advisor predicts that the probability of coding region in frame 0 is 0.5 and other advisor predicts that the probability of coding region in one of the frames 0, 1, or 2 is 0.6 (*i.e.,* the second advisor is not able to distinguish which frame is correct). If we divide the value of the second advisor between the 3 coding frames, the advice given by the two advisors conflicts, because one assigns probability 0.5 to reading frame 0 and the other 0.2. If the two advisors have equal weight, the super-advisor advice for reading frame 0 produced by the naive combination will be 0.35. This value is not directly supported by any evidence.

Although the naive method leads in this case to a conflict between two advisor predictions, it is possible to construct a vector $\overline{x}$ that agrees with the original advice of both advisors. For example, we may set probability of frames 1 and 2 to be 0.05 and probability of frame 0 to be 0.5. All of the distance based methods we have discussed will produce such a consistent vector.

We will also consider a variant of the naive method that ignores all vacuous advice. Although in principle vacuous advice can be converted to a complete probability distribution just like any other advice, such a conversion would produce the prior distribution of labels. Several advisors with vacuous advice could force the advice of the super-advisor to be close to the prior, and lower its influence in combination with the HMM. Since vacuous advice is used at positions where a particular advisor cannot reliably provide any information, it seems more advisable to ignore this advice entirely than to replace it with the prior distribution, and thus weaken the super-advisor. We will call this variation of the naive method the improved naive method.

The naive combination method artificially creates advice with a complete probability distribution over all labels for each advisor. We could require that advisors supply advice in this form directly. However, many sources of evidence can be expressed more naturally as a probability distribution over the partition sets of some coarser partition, and the necessity to provide a probability value for each label separately would likely require extensive training data. Training becomes especially difficult if some labels occur rarely, as is the case for many of the labels used in our gene finder. In this context, we can view the advisor framework with the naive combination method as a convenient method for decreasing the number of parameters to train.

2.5.4 Experimental comparison of advisor combination methods

In the previous sections, we have introduced several methods for combining advice of multiple advisors and briefly discussed their theoretical properties. In this section, we will compare their influence on the accuracy of gene finding on testing data. Our experiment tests eight advisor combination methods. We include the three measures $\text{dist}_a^{(L_1)}$, $\text{dist}_a^{(L_2^2)}$, $\text{dist}_a^{(L_\infty)}$, based on L_n norms, introduced in Section 2.5.1. We also include the relative entropy measures $\text{dist}_a^{(KL1)}$ and $\text{dist}_a^{(KL2)}$ introduced in Section 2.5.2. We evaluate our original squared L_2 based distance that uses the prior, dist_a, from Section 2.4.1. And finally, we consider the two versions of the naive advisor combination discussed in Section 2.5.3. The naive method converts all advice to a complete probability distribution over the set of labels, and the improved naive method applies this process only to non-vacuous advice.

We use each of these measures to combine the advice of several advisors to a super-advisor prediction. The super-advisor prediction is then combined with the HMM. We report the running time of the advisor combination phase, as well as the exon level sensitivity and specificity of the overall gene prediction. Recall from Section 1.6 that exon sensitivity is the proportion of annotated exons that are predicted completely correctly, and exon specificity is the proportion of predicted exons that are identical to exons found in the annotation.

The test was done on three human genomic sequences, ENr131, ENr233, and ENr332, from the ENCODE project [61]. Each is about 500,000 bases long, and in total they contain 533 annotated coding exons. In this experiment, we have used a set of advisors based on several protein databases (human, mouse, chicken, and fruit fly), human and mouse ESTs, the mouse genome, and sequence repeats to predict human genes. The weights of all advisors were set equal to one, and the prior distribution was added as an additional advisor with weight 0.01 (see Lemma 5). Super-advisor values were constrained to differ at most 100-fold from the prior probability of each label for all methods except the two naive ones. We have raised both the super-advisor and the prior to the power $\alpha = 0.02$ in Formula (2.4) for combining the HMM and the super-advisor, as described in Section 2.7.2. Further details of the advisor and HMM construction, and of the testing and training sets can be found in Chapter 4.

Table 2.2 shows the prediction accuracy of ExonHunter with different advisor combination methods. The hidden Markov model without advisors has 63% sensitivity and 58% specificity. Our original squared L_2 based distance dist_a yields 76% exon sensitivity and 67% specificity. Several other combination methods have very similar performance: $\text{dist}_a^{(KL2)}$, $\text{dist}_a^{(KL1)}$, $\text{dist}_a^{(L_2^2)}$, and even the improved naive method. On this data set, $\text{dist}_a^{(KL2)}$ and dist_a actually lead to identical gene

Table 2.2: Comparison of ExonHunter accuracy with different advisor combination methods on a testing set with 533 coding exons. Our original combination measure dist_a has similar accuracy as $\mathrm{dist}_a^{(KL2)}$, $\mathrm{dist}_a^{(KL1)}$, $\mathrm{dist}_a^{(L_2^2)}$, and the much simpler improved naive method. The table also shows the running time of the advisor combination. As expected, the two naive methods are fastest.

Advisor combination	Exon sensitivity	Exon specificity	Running time
no advisors	63%	58%	–
dist_a (original)	76%	67%	15 min
improved naive	76%	66%	19 s
naive	71%	60%	3 min
$\mathrm{dist}_a^{(L_2^2)}$	77%	66%	16 min
$\mathrm{dist}_a^{(L_1)}$	68%	62%	20 min
$\mathrm{dist}_a^{(L_\infty)}$	54%	54%	12 min
$\mathrm{dist}_a^{(KL2)}$	76%	67%	151 min
$\mathrm{dist}_a^{(KL1)}$	76%	67%	107 min

predictions, while the improved naive method differs from them in 3% of exons. The naive method, $\mathrm{dist}_a^{(L_1)}$, and especially $\mathrm{dist}_a^{(L_\infty)}$ perform significantly worse; the prediction using advisors that are combined with $\mathrm{dist}_a^{(L_\infty)}$ is actually worse than the prediction without any advisors.

To study the performance of advisor combination in greater detail, we have compared the super-advisor prediction and the prior for the correct label at each position. If the super-advisor prediction is higher than the prior, the super-advisor will boost the probability of the correct label in the HMM. Since the testing set annotation contains alternatively spliced genes, we have computed a reference labeling by choosing a subset of non-overlapping gene variants maximizing the total length of coding regions, and consider the labels in this labeling as "correct". At each position we compute the log-odds ratio $\log_2 \Pr(\ell)/\mathrm{prior}(\ell)$, where $\Pr(\ell)$ is the super-advisor prediction for the reference label ℓ. The distribution of these ratios for various combination methods is shown in Figure 2.2. For all methods, the interval $[0, 1)$ contains the highest number of examples, because for many positions all advisors give vacuous advice and thus the super-advisor prediction equals the prior.

The reason for the poor behavior of $\mathrm{dist}_a^{(L_\infty)}$ is the relatively high number of correct labels in the lowest bin $[-7, 6)$ (lower values do not occur, as we have constrained $\Pr(\ell)$ to be at least $\mathrm{prior}(\ell)/100$). The distributions of log-odds ratios for the methods based on relative entropy are very similar to the distribution for our original measure. The values of the two naive methods are not constrained to be within interval $[-\log_2 100, \log_2 100]$, and indeed some sites have values above the upper boundary of the interval (not shown). But most of the distribution is concentrated close to 0, that is, the influence of the advisors is weak. This effect is even stronger when we include vacuous advice in the naive combination approach. The improved naive combination has stronger signal and better performance in gene finding.

Table 2.2 also shows the running time of different advisor combination methods in a simple prototype implementation. Measurement was done on 1.7GHz Pentium 4 processor. As expected, the two naive methods are fastest. Methods using linear and quadratic programming work 50

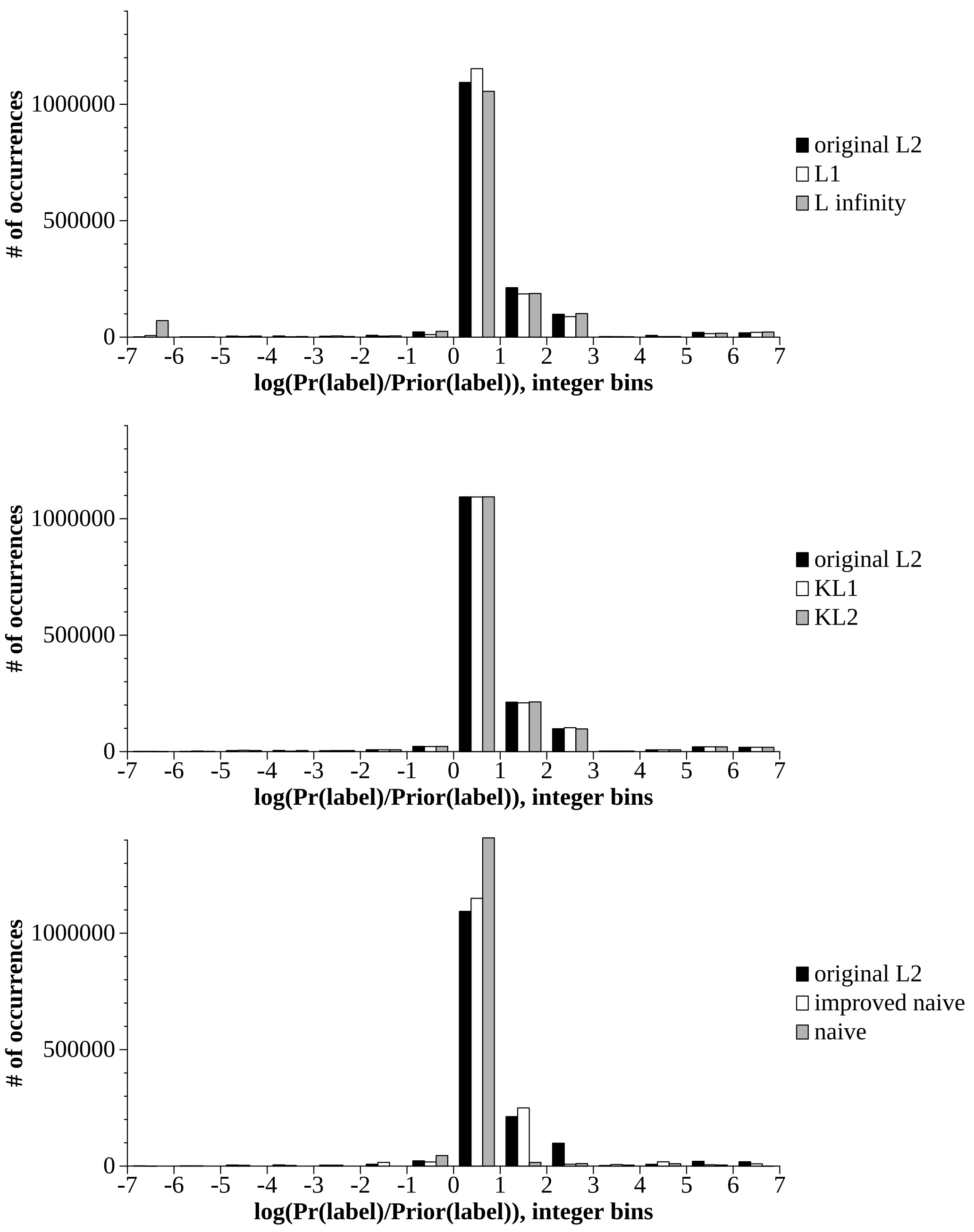


Figure 2.2: Performance of different advisor combination methods, measured as log of the ratio between the super-advisor prediction and prior probability of the correct label at each position. If this value is positive, the super-advisor boosts the probability of the correct label. If it is negative, the super-advisor decreases the probability of the correct label is decreased. Each figure compares our original square L_2 based measure (shown in black bars) with two alternatives.

times slower; relative entropy methods are about 7-10 times slower than quadratic programming. In practice however, the running time of advisor combination using the quadratic programming is dominated by the other steps in gene finding, such as sequence database search and the Viterbi algorithm for decoding; recall that these optimization problems are quite small.

In this experiment, we have observed that several advisor combination methods, including our original squared L_2 measure, yield very similar accuracy on our set of advisors. If the speed of advisor combination is important, the improved naive combination method is a good candidate, as it provides good performance and high speed. The squared L_2-based distance measure provides a stronger signal for the gene finder and is reasonable to use when the speed of advisor combination is not the most important factor. Finally, the comparison of the two naive methods suggests that it is very beneficial that advisors have an option to produce vacuous advice that is ignored by the advisor combination.

2.6 Training of advisor weights

The application of our combination rule (2.10) or any of its variants requires that each advisor a has an associated weight w_a that reflects our confidence in its predictions. We need a method to estimate these weights from training data. Assume we are given a training data set $\mathcal{T}$, which is a list of pairs $(\ell, \overline{p})$, where ℓ is the true label at some position in the sequence and $\overline{p}$ is a vector of the predictions of all advisors for that position. For each advisor a, then, the vector $\overline{p}$ contains the partition π_a and the values $p_a(S)$ for all $S \in \pi_a$.

The probability distribution returned by the super-advisor at a position is a function of the advisor predictions $\overline{p}$ for that position and the vector of advisor weights $\overline{w}$. In particular, let $g(\overline{p}, \overline{w}, \ell)$ be the probability assigned by the super-advisor to label ℓ, provided that $\overline{p}$ is the vector of advisor predictions and $\overline{w}$ is the vector of advisor weights. In our combination rule, the value $g(\overline{p}, \overline{w}, \ell)$ corresponds to the variable x_ℓ in the solution of the quadratic program (2.10). Intuitively, we want to choose weights $\overline{w}$ so that the value $g(\overline{p}, \overline{w}, \ell)$ is high for all pairs $(\ell, \overline{p})$ in the training set. We need to define some optimization criterion formalizing this intuition, and then find a method for optimizing $\overline{w}$ under this criterion.

One possibility is to use maximum likelihood estimation, choosing weights that maximize the probability of the true labels in our training set. Assume that the training set contains independently chosen samples, and therefore, the joint probability of all true labels is the product of probabilities of true labels for individual samples. Therefore, we want to find the weight vector $\overline{w}$ that maximizes the following product over all $\overline{w}$:

$$\prod_{(\ell,\overline{p})\in\mathcal{T}} g(\overline{p}, \overline{w}, \ell), \tag{2.33}$$

or equivalently, its logarithm, which we will denote $L(\overline{w})$

$$L(\overline{w}) = \sum_{(\ell,\overline{p})\in\mathcal{T}} \log g(\overline{p}, \overline{w}, \ell). \tag{2.34}$$

Again, the positional independence assumption is unlikely to hold in practice, but is made to simplify the computation.

The problem of optimizing the weights for the quadratic program is difficult because of the complex relationship between the weights, advisor probabilities, and the result of advisor combination.

Therefore, in this section, we concentrate on two simple special cases when advisor combination can be characterized by a simple formula.

In particular, if all advisors produce complete partitions, the super-advisor prediction is a linear combination of the advisor probabilities (Corollary 3). We will show that in this case the problem is identical to optimizing the weights of a certain Bayesian network, and we may use the classical expectation-maximization (EM) algorithm [58].

Next we study a more general problem where some advisors produce a complete probability distribution and others vacuous information. Then the super-advisor will be a linear combination of the non-vacuous advice, but the set of advisors with non-vacuous advice can be different at each data point. Weight optimization for this case is equivalent to optimizing weights for the improved naive combination method. As we have seen in Section 2.5.4, this combination method works well in practice, which motivates our interest in this problem.

Unfortunately this problem turns out to be more difficult. Although we are still able to represent it as a Bayesian network, due to atypical parameter tying we are unable to use the EM algorithm, and instead we optimize the weights by use of a general package for constrained optimization.

2.6.1 Weights for linear combination

In this section we assume that each advisor provides a complete distribution over the set of labels Σ. According to Corollary 3, advisor combination using our squared L_2-based method is equivalent to linear combination of advisors, so

$$g(\overline{p}, \overline{w}, \ell) = \sum_a w_a p_a(\ell), \tag{2.35}$$

under the assumption that the sum of all weights w_a is one.

In a probabilistic setting, the super-advisor distribution of labels can be viewed as a mixture of the distributions provided by the individual advisors. Typically, the individual components of a mixture model are fixed distributions, but in our case they are different at each position, determined by the advisor advice.

The situation in our case can be represented by the Bayesian network depicted in Figure 2.3. This Bayesian network has three variables: the true label ℓ, vector $\overline{p}$ containing predictions of all advisors, and a randomly chosen advisor a. When we use this network as a generative model, we first sample an advisor a from the probability distribution given by the advisor weights. Independently of the chosen advisor, we generate a vector $\overline{p}$ containing the advice of all advisors from some arbitrary fixed distribution that is not important for our application. Finally, given advisor a and all advice $\overline{p}$, we generate a label from the probability distribution given by the advice of advisor a in vector $\overline{p}$, that is $\Pr(\ell|\overline{p}, a) = p_a(\ell)$. All parameters of the network are fixed except the advisor weights, which will be estimated by the training procedure. In the training set, we observe the values of ℓ and $\overline{p}$ but not a.

This Bayesian network does not have an intuitive meaning, but it has the convenient property that the likelihood of the observed data in the network is closely related to the quantity $g(\overline{p}, \overline{w}, \ell)$ which we want to maximize:

$$\Pr(\ell, \overline{p}) = \Pr(\overline{p}) \cdot \sum_a P(a) \Pr(\ell|\overline{p}, a) = \Pr(\overline{p}) \cdot \sum_a w_a \cdot p_a(\ell) = \Pr(\overline{p}) \cdot g(\overline{p}, \overline{w}, \ell). \tag{2.36}$$

The probability $\Pr(\overline{p})$ does not depend on the weights $\overline{w}$, and therefore optimizing weights to maximize likelihood of observed data in this network is equivalent to optimizing the log likelihood

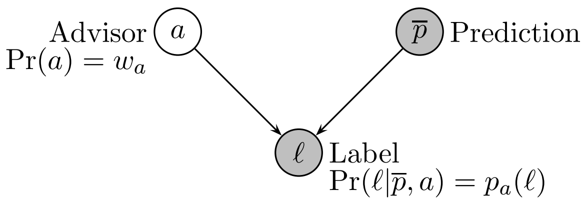


Figure 2.3: Bayesian network for which $\Pr(\ell|\overline{p}) = g(\overline{p}, \overline{w}, \ell)$. The observed variables are shown in grey.

$L(\overline{w})$ of the true labels in linear combination of advice. Therefore, we can use standard methods for learning Bayesian network parameters to learn the weights for advisor combination.

One popular method for optimizing parameters of Bayesian networks is the expectation-maximization (EM) algorithm [58]. This algorithm starts with some initial vector of parameters $\overline{w}^0$, and iteratively improves them. In each iteration, we compute a vector of parameters $\overline{w}^{t+1}$ that yields a higher or equal likelihood of the observed data than the previous vector $\overline{w}^t$. The following lemma shows how to apply the EM algorithm to our case.

Lemma 9. *Let $\overline{w}^t$ be a weight vector and let weight vector $\overline{w}^{t+1}$ be defined as follows:*

$$w_a^{t+1} = C \cdot \sum_{(\ell,\overline{p}) \in \mathcal{T}} \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)}, \tag{2.37}$$

where C is a normalizing constant not depending on a such that $\sum_a w_a^{t+1} = 1$. Then, $L(\overline{w}^{t+1}) \geq L(\overline{w}^t)$.

Proof. Instead of optimizing the likelihood of the observed data directly, the EM algorithm in each step optimizes the expected log likelihood of the complete data. Since only part of the data is observed, we define a distribution over the complete data using the old weights $\overline{w}^t$ and then compute the expected log likelihood of the complete data $Q(\overline{w}|\overline{w}^t)$ as a function of the new weights $\overline{w}$.

In our case, the likelihood of the complete data is $\Pr(a, \overline{p}, \ell|\overline{w}) = \Pr(\overline{p}) \cdot w_a \cdot p_a(\ell)$, where a is unobserved. We estimate the probability of a, $\Pr(a|\overline{p}, \ell, \overline{w}^t)$, using the old weights $\overline{w}^t$ and then compute the expectation of the log likelihood $Q(\overline{w}|\overline{w}^t)$:

$$\begin{aligned}
Q(\overline{w}|\overline{w}^t) &= E\left[\log \prod_{(\ell,\overline{p}) \in \mathcal{T}} \Pr(a, \overline{p}, \ell|\overline{w}) \middle| \mathcal{T}, \overline{w}^t\right] \\
&= \sum_{(\ell,\overline{p}) \in \mathcal{T}} E[\log \Pr(a, \overline{p}, \ell|\overline{w})|\mathcal{T}, \overline{w}^t] \\
&= \sum_{(\ell,\overline{p}) \in \mathcal{T}} E[\log \Pr(a, \overline{p}, \ell|\overline{w})|\ell, \overline{p}, \overline{w}^t] \\
&= \sum_{(\ell,\overline{p}) \in \mathcal{T}} \sum_a \Pr(a|\overline{p}, \ell, \overline{w}^t) \cdot \log \Pr(a, \overline{p}, \ell|\overline{w}) \\
&= \sum_{(\ell,\overline{p}) \in \mathcal{T}} \sum_a \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)} \log\left(\Pr(\overline{p}) \cdot w_a \cdot p_a(\ell)\right)
\end{aligned}$$

Theorem 1 in Dempster *et al.* [58] states that if $Q(\overline{w}|\overline{w}^t) \geq Q(\overline{w}^t|\overline{w}^t)$, then the log likelihood of the observed data does not decrease. Therefore, in each iteration, the EM algorithm first computes the function $Q(\overline{w}|\overline{w}^t)$, and in the second step, it finds a vector $\overline{w}^{t+1} = \arg\max_{\overline{w}} Q(\overline{w}|\overline{w}^t)$. If this is not easy to achieve, the generalized EM algorithm allows to choose $\overline{w}^{t+1}$ such that at least the inequality $Q(\overline{w}^{t+1}|\overline{w}^t) \geq Q(\overline{w}^t|\overline{w}^t)$ is satisfied. In our case, it is easy to maximize $Q(\overline{w}|\overline{w}^t)$:

$$\begin{aligned}
Q(\overline{w}|\overline{w}^t) &= \sum_{(\ell,\overline{p})\in\mathcal{T}} \sum_a \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)} \log\left(\Pr(\overline{p}) \cdot w_a \cdot p_a(\ell)\right) \\
&= \sum_a \underbrace{\sum_{(\ell,\overline{p})\in\mathcal{T}} \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)}}_{q_a} \log w_a + \underbrace{\sum_{(\ell,\overline{p})\in\mathcal{T}} \sum_a \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)} \log\left(\Pr(\overline{p}) \cdot p_a(\ell)\right)}_{\alpha} \\
&= \sum_a q_a \log w_a + \alpha
\end{aligned}$$

The term denoted α does not depend on $\overline{w}$, so we only need to maximize $\sum_a q_a \log w_a$ subject to $\sum_a w_a = 1$, which is achieved by setting $w_a = q_a / \sum_{a'} q_{a'}$. Indeed, as the relative entropy $H(P_1||P_2)$ is non-negative for any probability distributions P_1 and P_2 (see Section 2.5.2), we have $\sum_x P_1(x) \log P_1(x) \geq \sum_x P_1(x) \log P_2(x)$. This implies that the weights w_a should be proportional to values q_a to maximize $Q(\overline{w}|\overline{w}^t)$.

Therefore if we set

$$w_a^{t+1} = \frac{q_a}{\sum_{a'} q_{a'}} = C \cdot \sum_{(\ell,\overline{p})\in\mathcal{T}} \frac{w_a^t \cdot p_a(\ell)}{\sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)}, \qquad (2.38)$$

$Q(\overline{w}^{t+1}|\overline{w})$ is maximized, and the likelihood of the training data is increased or remains the same. □

The lemma allows us to iteratively improve the weights by the use of the EM algorithm. Informally, the computation of new weights works as follows. For each pair $(\ell, \overline{p})$ in the training set we compute the combined prediction $g(\overline{p}, \overline{w}^t, \ell) = \sum_{a'} w_{a'}^t \cdot p_{a'}(\ell)$ using the old weights $\overline{w}^t$. Notice that only predictions for the correct label ℓ are used in this computation. Then for each advisor a, we compute what portion of the combined value was contributed by this advisor, and this portion is added to w_a. After summing these quantities for all pairs in the training data, we normalize the weights by dividing them by their sum.

In this way, if all advisors have high predictions for a given correct label, their weights increase by a relatively small contributions. However, if one advisor predicts much higher probability for the correct label than the other advisors, its weight increases much more.

The lemma states that the likelihood improves or remains the same in each step but it does not guarantee convergence to a global maximum. Under some conditions the EM algorithm is guaranteed to converge to a stationary point of the likelihood function [168]; unfortunately, our case does not satisfy conditions of these theorems. In particular, Wu [168] assumes that the sequence of weight vectors converges to some interior point in the space of all feasible weight vectors, which often not the case in our problem.

2.6.2 Weights for linear combination including some vacuous advice

Now we turn our attention to a more complex case, where each advisor returns either a complete probability distribution over all labels, or simply provides vacuous advice. According to Lemma 2, the super-advisor prediction for label ℓ will be a linear combination of the non-vacuous advice:

$$g(\overline{p}, \overline{w}, \ell) = \frac{\sum_{a \in A(\overline{p})} w_a \cdot p_a(\ell)}{\sum_{a \in A(\overline{p})} w_a}, \tag{2.39}$$

where $A(\overline{p})$ is the set of advisors with non-vacuous advice in the vector $\overline{p}$ containing collected advice of all advisors. Unfortunately, since the set of advisors $A(\overline{p})$ is different at each sequence position, we cannot assume that the normalization factor $\sum_{a \in A(\overline{p})} w_a$ is always equal to one.

This causes the problem to be much more difficult than weight training for linear combination, as in the previous section. We will show that it is still possible to convert the problem to an instance of parameter training in a Bayesian network, but the parameters of the network will be tied together by the normalization factor $\sum_{a \in A(\overline{p})} w_a$. Due to this, we are not able to easily adapt the EM algorithm.

Instead, in our experiments, we use a generic package for constrained optimization to directly optimize the log likelihood $L(\overline{w})$ with respect to the weight vector $\overline{w}$, as in this optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{(\ell, \overline{p}) \in \mathcal{T}} \log g(\overline{p}, \overline{w}, \ell) \\ \text{subject to} \quad & \sum_a w_a = 1 \\ & x_a \geq \epsilon. \end{aligned}$$

Note that we require the weights to be at least ϵ to avoid undefined values. To solve this optimization problem, we use the function `nag_opt_nlp` from the NAG C Library [126], which is an implementation of a sequential quadratic method [64]. The function requires the gradient of the objective function which is easily computed:

$$\frac{\partial \log g(\overline{p}, \overline{w}, \ell)}{\partial w_a} = \sum_{(\ell, \overline{p}) \in \mathcal{T} : a \in A(\overline{p})} \left(\frac{p_a(\ell)}{\sum_{a' \in A(\overline{p})} w_{a'} p_{a'}(\ell)} - \frac{1}{\sum_{a' \in A(\overline{p})} w_{a'}} \right). \tag{2.40}$$

As an alternative to this generic method we have attempted to extend the EM algorithm from the previous section to this more general scenario. We use a new Bayesian network, shown in Figure 2.4, by adding variable A, which represents the set of advisors with non-vacuous advice at the current position. The set A is generated first, from a fixed distribution. Given the set A, one advisor a is chosen from set A, so that $\Pr(a|A) = w_a / \sum_{a' \in A} w_{a'}$. Given the set A we also generate vector $\overline{p}$ containing advice of all advisors from some fixed distribution. Exact form of the distribution is not important, as long as it always generates vectors such that the advice of advisors in A is a complete distribution over all labels and all advisors not in A give vacuous advice. Finally, given the advisor a and the advisor predictions $\overline{p}$, the true label ℓ is generated so that the probability of each ℓ is $p_a(\ell)$.

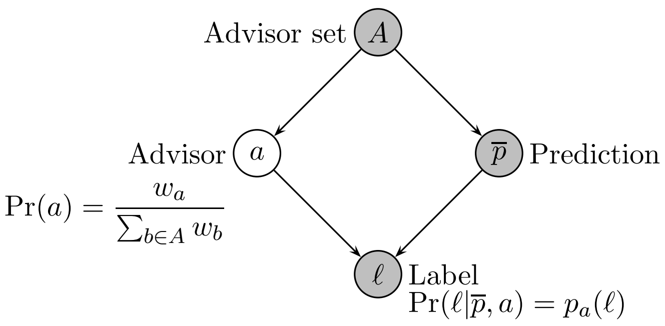

Advisor set
A
Advisor
a
$\overline{p}$
Prediction
$\Pr(a) = \frac{w_a}{\sum_{b \in A} w_b}$
ℓ
Label
$\Pr(\ell|\overline{p}, a) = p_a(\ell)$

Figure 2.4: Bayesian network for which $\Pr(\ell|\overline{p}, A) = g(\overline{p}, \overline{w}, \ell)$. Observed variables are shown in grey.

The observed data in this network consists of A, $\overline{p}$ and ℓ. As in the previous section, the likelihood of the observed data is closely related to the advisor combination result, $g(\overline{p}, \overline{w}, \ell)$:

$$\begin{aligned}
\Pr(A, \overline{p}, \ell) &= \Pr(A) \cdot \Pr(\overline{p}|A) \cdot \sum_a \Pr(a|A) \Pr(\ell|\overline{p}, a) \\
&= \Pr(A) \cdot \Pr(\overline{p}|A) \cdot \sum_a \frac{w_a}{\sum_{a' \in A} w_{a'}} \cdot p_a(\ell) \\
&= \Pr(A) \cdot \Pr(\overline{p}|A) \cdot g(\overline{p}, \overline{w}, \ell).
\end{aligned}$$

Again, the term $\Pr(A) \cdot \Pr(\overline{p}|A)$ does not depend on the weight vector $\overline{w}$. Therefore maximizing the likelihood of the training set in this Bayesian network with respect to the weight vector $\overline{w}$ is equivalent to maximizing the log likelihood $L(\overline{w})$ of the correct labels in the super-advisor prediction. As in Lemma 9, we could attempt to use the EM algorithm to optimize the weights; unfortunately we were not able to optimize $Q(\overline{w}|\overline{w}^t)$ with respect to weights. Nonetheless, the following lemma at least gives the expression for $Q(\overline{w}|\overline{w}^t)$, to illustrate the difficulties involved.

Lemma 10. *Let $\overline{w}^t$ be a weight vector, and let $\overline{w}^{t+1}$ be a weight vector such that $Q'(\overline{w}^{t+1}|\overline{w}^t) \geq Q'(\overline{w}^t|\overline{w}^t)$ where*

$$Q'(\overline{w}|\overline{w}^t) = \sum_a \sum_{A: a \in A} q_{a,A} \cdot \log \frac{w_a}{\sum_{a' \in A} w_{a'}} \tag{2.41}$$

and

$$q_{a,A} = \sum_{(\ell, \overline{p}) \in \mathcal{T}: A(\overline{p}) = A} \frac{w_a^t \cdot p_a(\ell)}{\sum_{a' \in A} w_{a'}^t \cdot p_{a'}(\ell)}. \tag{2.42}$$

Then, $L(\overline{w}^{t+1}) \geq L(\overline{w}^t)$.

Proof. Analogously to Lemma 9 the result follows from the basic properties of the generalized EM algorithm. The expected log likelihood of the complete data, $Q(\overline{w}|\overline{w}^t)$, in this case can be computed as follows:

$$\begin{aligned}
Q(\overline{w}|\overline{w}^t) &= E\left[\log \prod_{(\ell,\overline{p})\in\mathcal{T}} \Pr(A,a,\overline{p},\ell|\overline{w})\middle|\mathcal{T},\overline{w}^t\right] \\
&= \sum_{(\ell,\overline{p})\in\mathcal{T}} \sum_{a\in A(\overline{p})} \Pr(a|A,\overline{p},\ell,\overline{w}^t)\cdot\log\Pr(A,a,\overline{p},\ell|\overline{w}) \\
&= \sum_{(\ell,\overline{p})\in\mathcal{T}} \sum_{a\in A(\overline{p})} \frac{w_a^t\cdot p_a(\ell)}{\sum_{a'} w_{a'}^t\cdot p_{a'}(\ell)} \log\left(\Pr(A)\cdot\Pr(\overline{p}|A)\cdot\frac{w_a}{\sum_{a'\in A(\overline{p})} w_{a'}}\cdot p_a(\ell)\right)
\end{aligned}$$

By reorganizing the last expression we obtain that $Q(\overline{w}|\overline{w}^t) = Q'(\overline{w}|\overline{w}^t) + \alpha$ where Q' is defined in the lemma statement and α is a constant with respect to the new vector of weights $\overline{w}$:

$$\alpha = \sum_{(\ell,\overline{p})\in\mathcal{T}} \sum_{a\in A(\overline{p})} \frac{w_a^t\cdot p_a(\ell)}{\sum_{a'} w_{a'}^t\cdot p_{a'}(\ell)} \log\left(\Pr(A)\cdot\Pr(\overline{p}|A)\cdot p_a(\ell)\right) \tag{2.43}$$

Thus, if $Q'(\overline{w}^{t+1}|\overline{w}^t) \geq Q'(\overline{w}^t|\overline{w}^t)$, then also $Q(\overline{w}^{t+1}|\overline{w}^t) \geq Q(\overline{w}^t|\overline{w}^t)$, and by the properties of the EM algorithm the likelihood of the observed data will increase or stay the same. □

To use the result of this Lemma for computation, we would need a procedure for maximizing, or at least increasing, the value of the function $Q'(\overline{w}^{t+1}|\overline{w}^t)$, which is much more complicated than the function we have obtained in Lemma 9. We are not aware of a simple approach for this, and instead of optimizing Q' by numerical methods, we have opted to use the previously mentioned methods to directly optimize the overall likelihood of the labels in the training set.

Instead of trying to optimize weights for the improved naive method, which is a difficult problem, we might try to use weights optimized for the naive method, which does not allow vacuous advice. These weights may be optimized easily by the EM algorithm given in the previous section. However, such simplification can lead to undesirable results. Consider an an advisor that returns non-vacuous advice only rarely, but whenever it does, its advice is reliable. Such an advisor should clearly have a high weight in the improved naive method. However, the naive method replaces all vacuous advice expressed by this advisor with the prior, lowering the advisor's usefulness. Consequently, the weight finding algorithm assign it a low weight.

Consider an example with three advisors: one which reports the prior, and two other advisors a and b. Let the advice for the correct label ℓ_1 on the first position be: prior$(\ell_1) = 0.1$, $p_a(\ell_1) = 0.5$, $p_b(\ell_1) = 0.6$. Further, suppose that at the second position, advisor b gives vacuous advice, and the rest of the advice is as before, with prior$(\ell_2) = 0.1$ and $p_a(\ell_2) = 0.5$. Assume we fix the weight of prior at 0.01. The optimal weights, optimizing the likelihood of the correct labels, for the improved naive combination are approximately $w_a = 0.21$, $w_b = 0.78$. Using these weights, we obtain super-advisor predictions for the correct labels at the two positions 0.57 and 0.48 respectively, and the overall likelihood is their product, 0.27. However, optimal weights for the naive method give $w_a = 0.99$, and the advisor b is given zero weight. When these are used by the improved naive method, we obtain super-advisor predictions 0.50 and 0.50, with the lower likelihood of 0.25. We see that a slight increase in the second position was traded for a larger decrease in the first position, and the overall training data likelihood dropped. Since the advisors used in our gene finder have a high proportion of vacuous advice, it is important to take vacuous advice into account explicitly.

2.6.3 Experiments

In this section, we explore the influence of advisor weights and their training on the prediction accuracy of our gene finder. To train the weights, we use a training set of 13 human genomic sequences from the ENCODE project [61]. We use the same experimental settings as in Section 2.5.4, including the testing set consisting of three ENCODE sequences. We have also used the same set of advisors: those based on human, mouse, chicken, and fruit fly proteins, human and mouse ESTs, the mouse genome, sequence repeats, and one advisor that gives the prior label distribution. In total, we have 23 distinct advisors because some sources of information yield multiple advisors. In particular, from EST alignments we create separate advisors for predicting exons and introns, and from protein alignments we have separate advisors for exons, introns, start sites, stop sites, and for the short pieces of intergenic sequence adjacent to potential start and stop sites (see details in Section 4.3).

To train the weights, we collect all sequence positions from the training set where at least two advisors give non-vacuous advice (one of which is always the prior). There are over five million such positions, out of nine million nucleotides in the training data set. The remaining positions are useless for training because all advisors except the prior provide the vacuous advice and thus the result of the combination will be equal to the prior for any advisor weights. Thus, our training set is representative of the positions where weights are important, though not necessarily of all genomic positions.

Then we convert every advisor with non-vacuous advice to a full probability distribution by dividing the probability of each partition element among the individual labels proportionally to their prior probability, as in the improved naive method. We extract the value of the correct label from this distribution and include it in the training set. Thus, at every sequence position we construct a list containing probabilities of the correct label from all advisors with non-vacuous advice. Note that in the presence of alternative splicing and overlapping genes, some sequence positions may have more than one correct label. For simplicity we use a single reference labeling that is obtained by selecting a set of non-overlapping gene structures from the training set annotation, as described in Section 2.5.4.

Using this data, we optimize the log likelihood of the correct labels in the combined prediction, as described in Section 2.6.2. We require that all weights are higher than $\epsilon = 10^{-8}$. The weights obtained in this manner will be optimized for the improved naive combination method, but they can also be viewed as an approximation of optimal weights for the squared L_2-based method.

First, to assess the impact of the weights on gene finding accuracy, we have generated 10 random weight vectors. Each advisor, including the prior, was given a random weight from the set $\{10^0, 10^{-1}, 10^{-2}, \ldots, 10^{-8}\}$, and the weight vector was normalized to sum to one. These weights were used to combine advisors by the improved naive combination method, and then combined into the HMM as in Section 2.5.4. The results of our gene finder on the testing set, shown in Table 2.3, illustrate that the accuracy tends to be higher when the prior advisor is given low weight. Weight vectors that give high weight to the prior will yield super-advisor predictions close to the prior probability, and consequently will weaken the influence of advisors on the HMM. Weight vectors with low prior probability also tend to give higher log likelihood to the training set, although the correlation between the log likelihood and the prediction accuracy is not as strong as one might like. This suggests that the likelihood of the correct labels is perhaps not the ideal objective function for training the weights.

To eliminate the influence of the prior weight, we have generated a second set of 20 random

Table 2.3: Exon sensitivity and specificity with random advisor weights, ordered by the weight of the prior advisor. We also list the geometric mean of the likelihood of a label over all labels in the training set $\mathcal{T}$. This value is obtained by a monotonic transformation from $e^{L(\overline{w})/|\mathcal{T}|}$ from the objective value $L(\overline{w})$, defined in Equation (2.34).
Weight vectors with high prior advisor weight weaken the super-advisor influence, which leads to lower objective value and worse prediction accuracy.

Exon sn.	Exon sp.	w_{prior}	Probability of a label in $\mathcal{T}$
76%	65%	$3 \cdot 10^{-9}$	0.42
75%	66%	$4 \cdot 10^{-7}$	0.35
74%	67%	$3 \cdot 10^{-6}$	0.36
70%	64%	$3 \cdot 10^{-6}$	0.39
75%	61%	$5 \cdot 10^{-6}$	0.36
75%	66%	$2 \cdot 10^{-3}$	0.34
71%	63%	$3 \cdot 10^{-3}$	0.38
73%	60%	$8 \cdot 10^{-3}$	0.34
70%	59%	$3 \cdot 10^{-2}$	0.30
64%	59%	$3 \cdot 10^{-1}$	0.29

Table 2.4: Exon sensitivity and specificity with random advisor weights when the prior weight is small. Statistics were computed on 20 random weight vectors. All experiments yield very similar prediction accuracy.

	min	max	mean	std. dev.
Exon sensitivity	73.9%	76.7%	75.3%	0.8%
Exon specificity	65.7%	67.3%	66.6%	0.4%

weight vectors. This time we have fixed the weight of the prior at 10^{-8} and assigned all the other weights randomly from the set $\{10^0, 10^{-1}, 10^{-2}, \dots, 10^{-6}\}$, ensuring that the weight of every advisor is at least 100 times bigger than the weight of the prior. The results are shown in Table 2.4. We see much less variance in the prediction accuracy between individual weight vectors than in Table 2.3.

Finally, we have used all the random vectors as starting points for the weight optimization function. Each run has produced a different set of weights, which may be different local minima of the objective function. The running time of the optimization ranged between several minutes and several hours on 1.7GHz Pentium 4 processor.

We have taken the two optimized weight vectors that give the highest likelihood and used them for gene finding in both improved naive and dist_a method for advisor combination. As we can see in Table 2.5, these weights do not achieve any significant improvement of prediction accuracy compared to uniform weight 1 for all advisors and weight 0.01 for prior (the weight vector used in Section 2.5.4). Moreover, we see that the accuracy obtained with uniform or optimized weights is not significantly greater than the accuracy using random weights with low prior weight, as shown in Table 2.4.

Table 2.5: Exon sensitivity and specificity with optimized advisor weights for two different combination methods on a testing set of three ENCODE regions with 533 coding exons. The use of optimized weights leads to almost identical accuracy as the use of uniform weights. However, the uniform weight vector has lower objective value than the two optimized weight vectors, as shown by the last column. This column lists the geometric mean of the likelihood of a label over all labels in the training set $\mathcal{T}$, as in Table 2.3.

	Improved naive		dist_a		
Weights	Exon sn.	Exon sp.	Exon sn.	Exon sp.	Prob. of a label in $\mathcal{T}$
Best found	76.6%	66.8%	76.4%	66.8%	0.43
Second best	76.7%	66.8%	76.4%	66.6%	0.43
Uniform	76.0%	66.4%	76.4%	66.6%	0.38

This discouraging performance of optimized weights can be due to various reasons. Perhaps the global optimum of the likelihood function $L(\overline{w})$ has better performance and could be found by improved optimization methods. Although this is a possibility, the best weights we have obtained in our experiments do improve the likelihood considerably compared to the uniform weights, but gene finding accuracy is not improved. Thus it seems that a better explanation would be that the maximum likelihood $L(\overline{w})$ is not a good objective function. It only requires that likelihood of the correct label is high but perhaps we should also require that the likelihood of other labels is low, although it is not clear how to formulate this into a tractable objective function. We could also try to improve the results by giving different weights to different training samples. For example, we may try to increase the weight of rare labels that may be underrepresented in the training set, or increase the weight of positions where the HMM without advisors makes mistakes and thus the correct advice is more important. Finally, it is possible that for our current set of advisors the choice of weights (as far as prior weight is low) does not significantly influence the results. However, weights may be more important for other sets of advisors and thus, their optimization may remain an important problem. For the set of advisors considered in this experiment, the uniform weights seem to be a good choice. Their performance is quite good, and they decrease the number of parameters that require training.

2.7 Addressing the independence assumption

In order to achieve flexibility and computational feasibility of our probabilistic model, we use extensive independence assumptions. In particular, our advisors assume complete independence of all individual positions. Hidden Markov models are able to model only very limited set of dependencies between different regions of the sequence, because they have only a constant-size memory. Clearly, these assumptions are not realistic.

As we have already seen, some types of evidence refer to whole regions, and are thus hard to express as advisors. However, the positional independence assumption in the advisor model leads to problems even with some advisors that can be expressed for each position separately. Consider for example a protein alignment indicating the probable location of a coding region. Based on characteristics of the alignment, such as its length and score, we may estimate that each individual position within the alignment is coding with some probability p. Thus the probability of each labeling that has a coding region in this entire stretch of DNA will be multiplied by $(p/r)^k$ where r is the prior probability of the coding region label (assuming for simplicity that there is only one such label) and k is the length of the alignment. For typical values of p, r, and k, this number will be very large and will dominate the information obtained from the HMM. Thus we will force a coding region everywhere a protein alignment occurs, even though some of those occurrences are false positives. The problem is that the value p is based on the information from the whole alignment, but then it is applied many times at each position of the alignment independently.

This is in contrast to the method of TwinScan [95], which uses only local information. TwinScan's HMM emits the query DNA sequence in parallel with a sequence representing the pattern of matches and mismatches of the sequence alignments (see Section 1.5.1). Since the alignment sequence is emitted by HMM states of order 5, TwinScan examines the pattern of matches and mismatches in a window of length six at each position.

In this section, we describe two simple and practical heuristics that lessen the impact of the position independence assumption. We evaluate their performance experimentally and discuss a possible generalization of the advisor framework that avoids the position independence assumption.

2.7.1 Selection of super-advisor positions

The advisors used in our system tend to provide information based on the same source (for example, the same alignment) at several consecutive positions. Since their advice is treated by the advisor framework as independent at each position, the advisor's influence on the HMM prediction is too strong, as we have discussed above.

One possibility how to avoid this problem is to use only one position from each alignment. For example, GenomeScan [173] selects one position from each protein alignment and increases the probability of all gene structures that label this position as coding region (see Section 1.5.2). Although this is easy to do for one advisor, it is not immediately clear how to consistently choose positions from multiple advisors based on heterogeneous sources of information.

We use the following strategy. First, the super-advisor prediction is computed at all positions. Then a subset Q of positions is chosen so that every two chosen positions are at least some threshold T apart. All other positions are replaced with the prior distribution, which means they will have no effect on the HMM prediction. The set Q is chosen so that it contains the positions that are most informative, that is, those that will have greatest impact on the HMM prediction. For the purpose of this selection we measure the "informativeness" of a position with super-advisor advice $\overline{x}$ as follows:

$$I(\overline{x}) = \max_{j\in\Sigma} \left|\log \frac{x_j}{\mathrm{prior}(j)}\right| = \log \max_{j\in\Sigma} \left\{ \frac{x_j}{\mathrm{prior}(j)}, \frac{\mathrm{prior}(j)}{x_j} \right\}. \tag{2.44}$$

In the HMM, the probability of a labeling with label j is multiplied by the term $x_j/\mathrm{prior}(j)$, and thus $\left|\log \frac{x_j}{\mathrm{prior}(j)}\right|$ measures the super-advisor's impact on the HMM prediction. We only consider the label with the greatest impact.

We choose Q so that the sum of $I(\overline{x})$ for positions included in Q is maximized, subject to the constraint that every two positions in Q are at least T positions apart. The optimal set Q can be computed by a dynamic programming in $O(n)$ time. For every position $i = 1, 2, \ldots, n$, we compute the cost $Q[i]$ of the best selection from the first i positions. For each i we need to consider two cases: either the optimal solution contains i or not. In the former case it cannot contain any of the positions $i-1, i-2, \ldots, i-T+1$. Thus we obtain the following recurrence:

$$Q[i] = \max\{Q[i-1], Q[i-T] + I(\overline{x}_i)\}. \tag{2.45}$$

After experimenting with different values of T, we chose to set $T = 50$. This threshold seems to be sufficient to prevent strong dependencies between positions in Q to negatively influence prediction accuracy. A typical coding region will contain two or three points with advisor information, as the mean length of human internal exons is 145 [51]. Note that the choice of the threshold T presumably should depend on the properties of advisors used. If advisors do not have dependencies spanning long regions, a lower threshold can be used.

We include the most informative positions in the subset Q because those are usually most reliable. For example if advisors produce conflicting advice for a given position, their advice will to some degree cancel out and thus such a position is less likely to be selected. A similar phenomenon occurs also near the boundaries of coding regions. In our experiments, the super-advisor often increases the coding region probability even several positions beyond its boundary, due to evidence from alignments that extend to adjacent non-coding regions. Such mistakes may cause incorrect coding region boundaries in the final prediction. But luckily, the super-advisor probability given to coding region labels is often lower near the coding region boundary than in its middle (see Section 4.3) and thus, the subset Q often contains the more informative, and correct, positions from inside the coding region rather than less informative positions from its boundary. The same phenomenon may, however, cause problems when the HMM incorrectly predicts a shorter coding region and the advice showing its full extent is skipped.

One disadvantage of this strategy is that a small change in the super-advisor prediction can cause a different subset of positions Q to be chosen, and this can cause unpredictable changes in the final prediction. Such discontinuous behavior leads to difficulties in optimizing advisor parameters for the best overall performance.

2.7.2 Choice of exponent α

To avoid the discontinuous behavior caused by using the set of most informative positions, we can use all positions but lower the degree of the super-advisor influence in the overall prediction. In particular, Equation (2.4), which we use for combining the HMM and the super-advisor, permits us to raise the ratio of the super-advisor to the prior to some power α. For $\alpha = 1$, we can derive the combination rule from the conditional independence assumption between the DNA sequence and evidence used by advisors, but we could still use different exponents heuristically.

Choosing one out of every T position for subset Q is roughly equivalent to using $\alpha = 1/T$ at every super-advisor position. Indeed, if the super-advisor advice for a certain label j is constant over some region of length T, then the probability of any labeling that has label j through this region will be multiplied by the same amount regardless of whether we choose one position of the region with $\alpha = 1$ or all T positions with $\alpha = 1/T$.

However, the two methods can lead to different results when the super-advisor signal changes rapidly within a short region. Consider for example a region much shorter than T in which the super-advisor probability of a coding region is much higher than in the surrounding area (this can be the result, for example, of a short alignment). Depending on the informativeness of other positions nearby, the subset Q may or may not contain one of the positions with elevated coding region probability. On the other hand, including all positions, but with α set to $1/T$, will guarantee

Table 2.6: Comparison of exon specificity and sensitivity for several values of T in the subset method and corresponding values $\alpha = 1/T$ in the exponent method of addressing position independence assumption. For each experiment, we also list how many exons predicted for threshold $T - 1$ are predicted without any change for threshold T.

	Subset			Exponent $\alpha = 1/T$		
T	Exon sn.	Exon sp.	Shared with $T-1$	Exon sn.	Exon sp.	Shared with $T-1$
48	75.8%	64.5%	—	76.6%	66.7%	—
49	75.2%	64.9%	93.7%	76.4%	66.5%	99.8%
50	76.6%	65.6%	96.1%	76.4%	66.6%	99.5%
51	76.4%	66.2%	95.1%	76.4%	66.6%	100.0%
52	76.0%	65.4%	95.7%	76.4%	66.7%	99.7%

that we include the signal, but in a weaker form because its effect will be raised to some power t/T close to zero, where t is the length of the region. Thus, the elevated coding region probability has the strongest influence on the HMM when we use a subset Q that contains a position from the region, weaker influence when we set α to $1/T$, and no influence when we use a subset Q that does not contain a position from the region. The choice of exponent α leads to more predictable results but the influence of the super-advisor on the HMM may not be sufficient in some cases.

Although the two methods we have described are only simple heuristics, we will see in the next section that they both lead to good performance in practice.

2.7.3 Experimental comparison

In this section, we present an experimental comparison of the two proposed heuristics for dealing with our position independence assumption. We have used the same experimental details as in Section 2.5.4, and we have combined advisors by optimizing the original dist_a measure. When we use the super-advisor prediction, at full strength, and at every position, the gene finder has 61% exon sensitivity and 41% exon specificity. This performance is far worse than the 63% exon sensitivity and 58% specificity achieved by the HMM without any advisors. The poor performance of using full-strength advice at each position illustrates the problems caused by uncritical use of the position independence assumption and the need to address them in practice.

When we apply either the subset method introduced in Section 2.7.1 or exponent $\alpha = 1/T$ discussed in Section 2.7.2, our accuracy significantly improves. Table 2.6 shows the performance of both methods for several values of T close to our default value $T = 50$. The overall performance of both methods is comparable. However the predicted gene structures between successive values of T change more when we use the subset method than when we use the exponent method. This is the result of the discontinuous behavior of the subset method with respect to small changes in various parameters. A small change in T or in the super-advisor prediction causes a different subset Q of positions to be chosen and this can cause changes in the most probable gene structure. As a consequence, the exponent method seems preferable in practice.

2.7.4 Relaxing the position independence assumption

As we have seen, the position independence assumption leads to serious problems, and although some of them can be addressed by simple heuristics, it would be desirable to remove this assumption altogether.

One generalization of the advisor framework is to make the probability of each label in the super-advisor prediction dependent on the previous label. This is a change similar to changing ordinary HMM states to first order states. The super-advisor s would, for every position i and every two labels $j, k \in \Sigma$, specify a conditional probability $p_{i,s}(j|k) = \Pr(\ell_i = j|\ell_{i-1} = k, E)$, where E is the evidence available to the advisors and $L = \ell_1\ell_2 \dots \ell_n$ is a labeling of the input sequence X. Then the overall probability assigned by the super-advisor to a given labeling is the product:

$$\Pr(L|E) = \prod_{i=1}^{n} p_{i,s}(\ell_i|\ell_{i-1}). \tag{2.46}$$

This requires only a small change in the Viterbi algorithm for computing the most probable labeling described in Section 2.2.2. The advice of a first-order advisor would consist of a separate partition $\pi_{a,k}$ for every possible label k at the previous position, and the probability $p_{a,k}(S|k)$ for every set of labels $S \in \pi_{a,k}$. The prior probability could also use conditional probabilities and thus capture not only relative abundance of individual labels, but also in effect the geometric length distribution of individual sequence features, such as exons.

When using first order advisors, we could omit the labels for signals from the label set Σ used in ExonHunter (see Table 2.1). Signals would be represented as transitions between two different labels. For example, the start site would be a transition from intergenic to coding region in frame 0. An advisor predicting start site signal would increase probability of such a transition, and it would produce vacuous advice for all labels k other than intergenic at previous position. An advisor based on alignment evidence would increase the probability of staying in a coding region at a position inside an alignment, but would not change the probability for example of moving from intron to coding region, as such increase is not implied by the evidence.

As we have not implemented this approach, we cannot compare its performance with the advisor framework proposed in the first part of this chapter. One possible source of problem is that conflicting evidence at one position will not cancel out. In the current framework, if one advisor assigns high probability to intergenic region and one assigns high probability to coding region, the super-advisor prediction will be somewhere between the two extremes suggested by the two advisors. In the first-order advisor framework, the super-advisor can easily simultaneously increase the probability of staying in intergenic region and staying in exon, sending the HMM two strong contradictory signals. Nonetheless, the first-order advisor framework is an interesting extension of our current work and is worth further study.

2.8 Other approaches to incomplete information

Our advisor evidence combination framework first represents evidence regarding a single sequence position as partial probabilistic statements about the correct label at that position, and then combines several such partial statements into one probability distribution. Partial probabilistic statements are the characteristic feature of our framework, and make it very flexible, but are also a source of difficulties in advisor combination. In this section, we will discuss some approaches from the literature for representing and combining partial information and show how they relate to our work.

2.8.1 Dempster-Shafer theory of evidence

The Dempster-Shafer theory of evidence [148] replaces the probability distributions with a more general notion of a belief function. Consider a set of possible worlds Θ (in the context of advisor advice for one position, Θ will be equal to the set of labels Σ). Let 2^Θ be the set of all subsets of Θ. In the Dempster-Shafer theory, a mass function (called a basic probability assignment in [148]) is a function $m : 2^\Theta \to [0, 1]$ such that $m(\emptyset) = 0$ and $\sum_{A\subseteq\Theta} m(A) = 1$. A mass function m is then used to define a belief function $\text{bel} : 2^\Theta \to [0, 1]$:

$$\text{bel}(A) = \sum_{B\subseteq A} m(B). \tag{2.47}$$

Intuitively, a mass function quantifies the weight of evidence specifically for the set A and a belief function quantifies our total belief that some element of A is the correct answer. This belief comes from both evidence for the set A, and evidence for its subsets. A regular probabilistic distribution is a special case of a mass function that assigns non-zero values only to singleton sets. In particular, if $m(\{j\}) = \Pr(j)$ for all $j \in \Theta$, then for every set $a \subseteq \Theta$ we have $\text{bel}(A) = \Pr(A)$. In general though, the mass function can be non-zero for sets with more than one element, and as a result, we can have $\text{bel}(A \cup B) > \text{bel}(A) + \text{bel}(B)$ for two disjoint sets A and B.

The advice of an advisor may seem similar to the Dempster-Shafer theory, because it also assigns probabilities to sets of labels. However, we require that those sets are disjoint, or in other words, that they form a partition π_a of the label set Σ. Thus, the advice of an advisor a can also be understood as a special case of a mass function, for which $m(S) = p_a(S)$ for $S \in \pi_a$ and $m(S) = 0$ otherwise.

Two mass functions m_1 and m_2 can be combined by Dempster's rule of combination:

$$m(A) \propto \sum_{A_1\cap A_2=A} m_1(A_1)m_2(A_2). \tag{2.48}$$

In particular, the combined mass function of two advisors with the same partition will correspond to an advisor with the same partition and probabilities proportional to the product of the two original probabilities. When we combine two advisors with different partitions, we obtain an advisor whose partition contains all pairwise intersections of the sets from the two original partitions.

We could mechanically apply Dempster's rule for the advisor combination, but such application would not be consistent with the philosophy of the Dempster-Shafer theory. An advisor expresses evidence in favor of some label set S as a distribution corresponding to the mass function $m(S) = p$, $m(\Sigma\backslash S) = 1-p$ for some probability p. However, Shafer instead suggests to represent such evidence for a set S by the mass function $m(A) = p$, $m(\Sigma) = 1 - p$. After combining several mass functions of this form, we would obtain a complicated mass function, not corresponding to any probability distribution, and it would be difficult to combine it with the probabilistic framework of the HMM gene finder.

Halpern and Fagin [80] argue that mass functions of a certain form are a good way of representing evidence, and Dempster's rule is appropriate to combine independent evidence represented in this form. The mass function obtained by combining multiple pieces of evidence can be converted to a probability distribution by combining it with the mass function of some prior distribution. However, as individual sources of evidence in gene finding are often not independent, this approach is not directly applicable.

2.8.2 Maximum entropy principle

Entropy and relative entropy are used in statistical inference to choose a single distribution among those that satisfy a given set of consistent constraints [150]. The maximum entropy principle prescribes the choice of the distribution of maximum entropy among all distributions that satisfy the constraints. By maximizing the entropy, we preserve as much uncertainty as possible about the events for which no information is available. The maximum entropy principle can be generalized to minimum cross entropy principle which prescribes to choose the distribution minimizing relative entropy to a known prior distribution. When the prior is uniform, the minimum cross entropy and maximum entropy principles are identical. Shore and Johnson [150] prove that minimum cross entropy is the only criterion that satisfies a certain set of natural axioms.

We can view individual advisors as constraints, but since they are not necessarily consistent, the maximum entropy principle cannot be applied directly. Interestingly, a certain parameter smoothing method for maximum entropy principle leads to a formula similar to our squared L_2 based advisor combination. The fuzzy maximum likelihood principle [45] allows the resulting distribution to violate some of the constraints, but a penalty for violating them is then added to the objective function. Therefore, we are seeking the probability distribution q that minimizes $H(q||\text{unif})+U(q)$, where $H(q||\text{unif})$ is the relative entropy of q given a uniform prior distribution, and $U(q)$ is the penalty function. A frequently used penalty function has the form of a weighted sum of squares of differences between q and constraints. Such a penalty function corresponds to the logarithm of a Gaussian distribution centered around the constraints [45]. Our advisor combination method also minimizes a weighted sum of squares of differences between the combined probability and constraints supplied by the advisors. However, we do not use the relative entropy term.

2.8.3 Bayesian approach

Methods for combining probability distributions are typically somewhat arbitrary and do not have any justification in probability theory. This is also the case of our advisor combination measure and its variants discussed in Section 2.4. Although we have demonstrated that our method performs well in practice and proved that it has some intuitively desirable properties, we were not able to derive it in a probabilistic framework. The naive method and improved naive method for advisor combination can be represented as a Bayesian networks for the purpose of weight training (see Section 2.6), but these Bayesian networks do not correspond to our intuitive view of evidence in gene finding. Note however that combination of the super-advisor and the HMM has a valid probabilistic basis under the assumption that the evidence and DNA sequence are independent given a labeling.

Grove and Halpern [77] discuss an example of updating a prior probability distribution based on new information in the form of a partial probabilistic statement. They show that if we combine the prior and the new information by the cross-entropy minimization discussed in the previous section, we will get unintuitive results. They argue that this is likely to be the case of every mechanical rule for combination of probability distributions. Instead, one should extend the probabilistic model to include the source of the new information. The new information becomes a random event in the expanded space, and we incorporate the new information simply by switching to a conditional probability distribution.

In the case of advisor combination, we would need a joint probability distribution of the correct label and all sources of evidence $\Pr(\ell, e_1, \dots, e_k)$. Then observing particular instances of evidence at a current position, we would use the conditional probability $\Pr(\ell | e_1, \dots, e_k)$ as the super-advisor prediction. To make such a model tractable, we would need a simple characterization of the joint probability distribution with a small number of parameters. This can be achieved, for example by making independence assumptions expressed in the form of a Bayesian network. In this approach, it would be more difficult to add and remove sources of evidence, as that would require changes in the topology of the Bayesian network as well as re-estimation of its parameters. Thus, although the Bayesian approach has better theoretical foundations, it is perhaps less suitable for advisor combination.

2.9 Summary

In this chapter, we have introduced a new and flexible framework for expressing evidence in gene finding and combining it with a hidden Markov model. The evidence is expressed in the form of advisors that specify a partial probability distribution over the set of possible labels at every position of the DNA sequence. Since we allow advisors to supply only partial information, we are able to express a wide range of evidence sources in a natural way and transparently handle missing information.

The presence of partial information prevents us from combining advisors using a simple combination rule. Therefore we design a method based on minimizing the squared L_2 distance between partial probability distributions supplied by the advisors and the resulting super-advisor prediction. We show that our method is a generalization of linear combination to the case of partial probability distributions and compare it to several other methods, both on simple artificial examples and on real data. We can assign higher importance to some advisors by increasing their weight in the advisor combination formula. We have studied methods for automatically optimizing the weights on a training data set.

The super-advisor prediction defines a probability distribution over all possible labelings of the query sequence. We combine this distribution with the distribution defined by the HMM and find the most probable state path in the combined distribution by a modified Viterbi algorithm.

We have made several independence assumptions to achieve computational efficiency and modeling simplicity. In particular, information in advisors is represented independently for each position. This assumption caused decreased prediction accuracy, but the problem can be solved by simple heuristics that perform well in practice. It remains an open problem how to remove the assumption that positions are independent and allow advisors to express information about whole regions of sequence, while maintaining the flexibility to express incomplete information.

While our method allows incorporation of multiple sources of evidence, we do not require all sources to be available. When no additional information is available, the system performs as a typical *ab initio* gene finder. Adding and removing sources of information is easy: it only requires estimation of new advisor weights, but our experiments show that at least in some situations, we can use uniform weights for all advisors without negative impact on the prediction accuracy.

Chapter 3

Spaced Seeds for Protein Coding Regions

The main topic of this thesis is the use of evidence in gene finding. This evidence often comes from sequence databases containing sequences of proteins, ESTs, or genomes. In order to use these databases in gene finding, we need to find local alignments, or regions of statistically significant sequence similarity, between the target sequence and the sequences in the database.

Local alignments are also used in many other areas of bioinformatics. Pairs of sequences in high scoring alignments are usually conjectured to be *homologous*, that is, to share a common evolutionary origin. Comparative studies of homologous sequences help to elucidate the evolutionary history, function and structure of biological sequences [161].

The standard dynamic programming algorithm by Smith and Waterman [153] can identify all local alignments between two sequences with score above a given threshold, but unfortunately its running time is proportional to the product of their lengths. The large size of many sequence databases (even a single genome may contain billions of nucleotides) forces us to use heuristic algorithms, such as BLAST [8], which run much faster but are not guaranteed to find all alignments.

In a 2002 paper, Ma *et al.* [116] introduced a technique for increasing the sensitivity of BLAST-like algorithms. The standard nucleotide BLAST program starts by finding short exact matches between the two input sequences, and then extending them to longer alignments. Ma *et al.* argue that instead, one should start by locating groups of non-consecutive matches in a prescribed formation called a *spaced seed*. They demonstrate that a carefully chosen spaced seed increases the sensitivity of homology search heuristics under a simple probabilistic model of alignments as well as on real genomic sequences with practically no effect on the running time.

In our gene finding application, we are especially interested in alignments between homologous coding regions from two organisms. These have special properties. For example, there is a three-periodic structure, originating in the genetic code that translates a triplet of nucleotides to one amino acid. In this chapter, we show that spaced seeds tailored specifically to properties of homologous protein coding regions have higher sensitivity to these regions than the original seed used by Ma *et al.* in their program PatternHunter.

To find such seeds, we first create models of sequence conservation in homologous coding regions and represent them as hidden Markov models. We also define *vector seeds*, a new framework that generalizes spaced seeds and the seeding strategies used by other programs, such as protein BLAST [8] and BLAT [92]. The space of practical vector seeds is very big, providing the flexibility to tailor

```
GGTGTGCTCTCTGACAAAGCC
GTTGTTCTTAATGAGAAAGCA        101110110001110111110
  11101    11101               11101    11101
               11101                         11101
```

Figure 3.1: The left side shows a short alignment with three hits of the spaced seed `11101`, two of them overlapping. The right side shows that the hit positions can be determined from a simple representation of the alignment in which a match is denoted by a `1` and mismatch by a `0`.

seeds that are appropriate for particular needs.

Selection of an optimal seed for a particular application requires an algorithm for computing seed sensitivity under a given probabilistic model representing a typical alignment. Keich *et al.* [91] have developed such an algorithm for spaced seeds and a very simple alignment model. In our work, we have extended this algorithm to handle vector seeds in addition to spaced seeds and more complex alignment models in the form of HMMs.

Finally, we demonstrate that seeds chosen based on our improved models lead to significant improvements in sensitivity over BLAST or PatternHunter on testing data consisting of homologous coding regions.

Spaced seeds, as introduced by Ma *et al.*, inspired a flurry of subsequent work. In the two years following their work, at least 15 papers on the topic of spaced seeds were published. Our contributions were originally published in 2003 [26, 27], and then appeared in journal form in 2004 [29] and 2005 [30]. Throughout the text, we mention related parallel and subsequent work. Further developments in the area of spaced seeds are discussed in Section 3.6.

3.1 Introduction to spaced seeds

This section describes spaced seeds, as introduced by Ma *et al.* [116], and their terminology. Spaced seeds generalize the algorithm used in nucleotide BLAST (BLASTN) to identify homologous regions in DNA sequences. BLAST first finds short exact matches, called *hits*. A BLAST hit consists of several consecutive positions (the default is 11). For each, an alignment is built that extends the hit on both sides. If the alignment score exceeds a threshold, the alignment is reported. Some significant alignments do not contain 11 consecutive matches; thus, they are not discovered by BLAST. To find hits, we create a hash table of all the words of length 11 in one of the sequences and then search for each word of length 11 from the other sequence in the table.

In the generalization introduced by Ma *et al.* [116], a hit consists of several non-consecutive matches in a prescribed configuration, called a spaced seed. A seed can be represented as a binary string, in which a `1` denotes a position that is required to match and a `0` denotes a position that is not required to match. For example, the seed `11101` requires three consecutive matches followed by one position which may or may not match, and another match, as shown in Figure 3.1. The seed corresponding to a BLAST hit is simply `11111111111`. To find hits, we now hash only the positions where the seed has a '1'. For example for the seed `11101`, we will hash 4-tuples $x_i x_{i+1} x_{i+2} x_{i+4}$ from the input sequence $x_1 x_2 \dots x_n$, for all possible values of i.

The performance of a particular spaced seed can be characterized by its false negative and false positive rate. The *false negative rate* measures the fraction of real alignments that do not contain any hit of the seed. A complementary measure, called *seed sensitivity*, is the fraction of alignments that contain at least one hit. These are the alignments that can be detected by the alignment algorithm. Conversely, the *false positive rate* measures the fraction of all pairs of unrelated positions from the two input sequences that contain a hit of the seed. Each such hit increases the running time, since the algorithm will attempt to extend it to a full alignment and fail. A seed with false positive rate q is expected to produce roughly qnm false hits, where n and m are the lengths of the two input sequences. The actual number is slightly lower, since some position pairs are either true hits or too close to the sequence boundary to produce a hit, but this effect is negligible. Also, many false hits will overlap each other and a clever algorithm handle them at the same time.

Ideally, we want seeds with low false positive and negative rates. Unfortunately, there is a trade-off between these two measures. Clearly, longer BLAST seeds have fewer false positives and more false negatives than shorter ones, but the situation is more complicated for spaced seeds. Ma *et al.* propose a simple probabilistic model of alignments to characterize the sensitivity of a spaced seed. In this model, a local alignment is represented as a binary sequence, where `1` represents a match and `0` a mismatch (see Figure 3.1). We model only ungapped alignments, as seed hits are found only inside the ungapped portions of all alignments. The probabilistic model has two parameters, N and p, and represents an aligned region of length N, where each position is a match independently with probability p. Formally, it is a sequence of N independent Bernoulli random variables $X_0, X_1, \ldots, X_{N-1}$, with $\Pr(X_i = 1) = p$ for each i. The sensitivity of a seed is then the probability that an alignment sampled from this model has a hit. Computation of seed sensitivity is a non-trivial task, and we will discuss it in more detail in Section 3.4.

To compute the false positive rate, we need a model of alignment of two unrelated sequences. We will assume that two unrelated nucleotides match with probability 1/4, and therefore the false positive rate can be computed as the probability of a hit in an alignment sampled from the Bernoulli model for $p = 1/4$ and N equal to the length of the seed. To have a hit in this model, we require character `1` at every position where the seed has a `1`. Therefore, the probability is 4^{-W}, where W is the number of ones in the seed. We will call the number of ones in a seed its *weight*.

Since all spaced seeds of the same weight have the same predicted false positive rate, we should use the seed that has the highest sensitivity out of all seeds of a fixed weight. Indeed, Ma *et al.* show that PatternHunter's seed, `111010010100110111`, has the highest sensitivity of all seeds of weight 11 and length at most 20 in the Bernoulli alignment model with parameters $N = 64$ and $p = 0.7$ [116]. Its sensitivity is 47%, compared to the BLAST consecutive seed of the same weight, which has sensitivity only 30%. Even the BLAST seed of weight 10 has lower sensitivity, 41%, and four times higher false positive rate. Thus, using the PatternHunter seed of weight 11, we may expect to find more alignments, in shorter time, than using the BLAST seed of weight 10.

Why is there such a big difference in sensitivity between the BLAST and PatternHunter seeds? The answer is that while the expected number of hits in an alignment is roughly the same for both seeds, the BLAST seed has more alignments that do not have any hit, and a much higher expected number of hits among those alignments that have at least one hit (see Table 3.1). This is because BLAST hits often cluster together. If the BLAST seed has a hit at some position i of the alignment, it has probability 0.7 to have another hit at position $i + 1$, since ten out of the eleven matches required for a hit are guaranteed by the presence of the hit at position i. For the PatternHunter seed, this probability is only $0.7^6 \approx 0.12$, since six more matches are required (see Figure 3.2).

In this thesis, we extend the framework of spaced seeds in two ways. First, in the next section, we define a richer set of seeds, and in Section 3.3, we design more complex probabilistic models of

Table 3.1: Let X be the random variable representing the number of seed hits in an alignment sampled from the Bernoulli model with parameters $N = 64$ and $p = 0.7$. This table compares the sensitivity ($\Pr(X \geq 1)$), the expected number of hits in one alignment ($E[X]$), and the expected number of hits in an alignment that is guaranteed to have at least one hit ($E[X|X \geq 0]$). The expected number of hits is slightly lower for the PatternHunter seed, since it is longer and thus it has fewer positions where a hit may occur. However, the two seeds differ much more in the number of hits in alignments with a guaranteed hit. This is because BLAST hits are more clustered together.

Seed	$\Pr(X \geq 1)$	$E[X]$	$E[X\|X \geq 0]$
BLAST: `11111111111`	0.30	1.1	3.6
PatternHunter: `111010010100110111`	0.47	0.9	2.0

BLAST seed:
1111111111
 111111111**1**

PatternHunter seed:
111**0**01**0**01**0**01**0**1**0**111
 11**1**00**1**00**1**00**1**0**1**011**1**

Figure 3.2: Consider an alignment sampled from the Bernoulli model with match probability p. If the BLAST seed has a hit at some position i of the alignment, it has probability p to have another hit at position $i+1$, since only one additional match is required (shown in bold). On the other hand, the PatternHunter seed requires 6 additional matches, and thus the probability of a match at position $i+1$ is only p^6.

alignments that better represent homologous coding sequences.

3.2 Vector seeds

PatternHunter, which uses the spaced seeds discussed in the previous section, has greatly greatly improved the sensitivity compared to BLAST, while still having the same or even better running time. Similar strategies were developed by other researchers. In particular, Kent [92], in his program BLAT allows a fixed number of mismatches in the region that makes up a hit. For example, we may require at least 11 matches in a region of length 12. The mismatch may occur at any of the twelve positions.

Vector seeds, which we introduced in [30], unify and further generalize the hit definitions used by PatternHunter and BLAT. They can also be applied to protein homology search, where programs traditionally use more complicated hit definitions reflecting the properties of amino acid substitution matrices used to score alignments.

To define a hit in the vector seed model, we represent an ungapped pairwise local alignment as a sequence of real numbers, each corresponding to a position in the alignment. We call such a sequence of positional scores an *alignment sequence*. The binary sequence representation of alignments used in the previous section is an example of such an alignment sequence.

Definition 11. A vector seed *is an ordered pair $Q = (w, T)$, where w is a weight vector $(w_1, \ldots, w_M)$ of non-negative real numbers and T is a threshold value.*

An alignment sequence $X = (x_1, x_2, \ldots, x_n)$ contains a hit to the seed *Q at position k if the dot product of the weight vector and the alignment sequence of length M beginning at position k is* *at least the threshold T:*

$$\sum_{i=1}^{M}(w_i \cdot x_{k+i-1}) \geq T.$$

The number of nonzero positions in the weight vector w is the support *of the seed.*

We first demonstrate the versatility of the vector seed framework by showing how to express several examples of hit definition as vector seeds. These examples include the hit definitions used in PatternHunter, BLAT and protein BLAST. Then, we discuss the algorithm for finding hits under the vector seeds definition. Seeds with high support are impractical, but seeds with low support can be easily implemented by only modest changes in the existing algorithms. Finally, we will show that under the simple Bernoulli model used to evaluate spaced seeds by Ma *et al.* [116], vector seeds achieve very good performance.

3.2.1 Expressiveness of vector seeds

Spaced seeds, described in Section 3.1, are a special case of vector seeds. To cast them in the vector seed framework we will use binary alignment sequences, with a `1` representing a match and a `0` representing a mismatch. To construct a vector seed (w, T) equivalent to a spaced seed Q, we set the weight vector w equal to the spaced seed string, and the threshold will be equal to the weight of the seed Q. For example, the nucleotide BLAST seed `1111` of weight 4 is equivalent to the vector seed $((1, 1, 1, 1), 4)$ and the spaced seed `11101` is equivalent to the vector seed $((1, 1, 1, 0, 1), 4)$.

Similarly, the seeding strategy used in BLAT can be formulated as a vector seed over the binary alignment sequence. For example, a BLAT hit definition that requires at least 4 matches in a region of length 5 corresponds to the vector seed $((1, 1, 1, 1, 1), 4)$. The vector seed framework immediately allows us to combine the BLAT and PatternHunter seeding strategy by introducing positions of weight zero to the BLAT vector seed, as in the vector seed $((1, 1, 1, 0, 1, 1), 4)$. As we will see in Section 3.2.3, such seeds actually perform very well.

However, vector seeds can also encode more complicated concepts. To illustrate this, we consider half-gapped seeds introduced by Chen and Sung [47]. A half-gapped seed can be represented as a string over alphabet $\{\texttt{0}, \texttt{1/2}, \texttt{1}\}$. As in a spaced seed, a match is required at positions with a `1` in the seed, whereas positions with a `0` are ignored. In addition, each of the four nucleotides has a set of neighbors, and we require that any position with a `1/2` in the seed is either a match or a pair of neighboring nucleotides. Similar seeds are also considered by Schwartz *et al.* [147] and Noé and Kucherov [123] (see also Section 3.6.2).

We can represent a half-gapped seed with h half-match positions and m match positions as a vector seed by using an alignment sequence in which the value $1 + (h + 2)/(h + 1)$ stands for a match, value 1 for neighbors, and value 0 for non-neighbors. We can construct a weight vector w by replacing each `1` in the half-gapped seed by weight $h + 1$ and each `1/2` by weight 1. We set threshold T to $(h + 2)m + h$. For example, the half-gapped seed $(1, 1/2, 0, 1/2)$ corresponds to the vector seed $((3, 1, 0, 1), 6)$ in an alignment sequence where a match is represented by $4/3$, neighbor by 1, and non-neighbor by 0. The dot product of the weight vector and an alignment sequence achieves the highest value $T + h/(h + 1)$ when all positions match. If one of the match positions of the seed is not a match in the alignment, the score is at most $T + h/(h + 1) - (h + 2) + h + 1 < T$. If one of the half-match positions of the seed is a non-neighbor in the alignment, the score is at most $T + h/(h + 1) - (h + 2)/(h + 1) < T$. Therefore, only sequences that constitute a hit of the half-gapped seed constitute a hit in the corresponding vector seed and vice versa.

Although in this thesis we concentrate on homology search in nucleotide sequences, we note here that we can also apply vector seeds to protein homology search. Nucleotide alignments are usually scored by simple scoring matrices that have only two scores: one for any pair of matching nucleotides and one for any pair of mismatching nucleotides. On the other hand, protein alignments are scored by substitution matrices such as BLOSUM62 [84], which define different scores for different matching and mismatching amino acid pairs. The score of a mismatch in such a matrix reflects the similarity of chemical properties of the two amino acids, or the process of evolution and its effect on the sequence. BLAST for protein sequences [8] defines a hit as several consecutive positions whose total score exceeds a given threshold.

The protein BLAST hit definition can be expressed as a vector seed. We can represent the alignment between protein sequences $Y = y_1 y_2 \dots y_n$ and $Z = z_1 z_2 \dots z_n$ by the alignment sequence of positional scores, $(s_{y_1,z_1}, s_{y_2,z_2}, \dots, s_{y_n,z_n})$, where $S = (s_{i,j})$ is the scoring matrix, for example the BLOSUM62 matrix [84]. The protein BLAST settings requiring three consecutive positions with total score at least 13 in a hit correspond to the vector seed $((1, 1, 1), 13)$.

Spaced seeds do not work well for protein homology search for two reasons. First, the protein BLAST seed is very short, and thus it is hard to improve it by spacing. Also, spaced seeds consider only matches and mismatches, and not the richer similarity measure introduced by amino acid substitution matrices. On the other hand, vector seeds and their generalizations allowed researchers to achieve improvements compared to protein BLAST seed. For example, Brown [32] reports a collection of eight vector seeds that achieve almost the same sensitivity as the protein BLAST seed while reducing the number of false positives four to five times. In his approach, a hit is a position where at least one vector seed from the collection has a hit.

Kisman *et al.* [94] report that tPatternHunter, a version of PatternHunter for searching protein sequences, uses a collection of seeds of a special kind. Each seed is in a effect a vector seed over a BLOSUM62 alignment sequence with a binary weight vector. They define a hit as a hit of the vector seed satisfying an additional constraint that all positions with weight one have non-negative BLOSUM62 score. To represent this seed as a vector seed without additional constraints, we change the alignment sequence so that all negative scores are set to $-\infty$. Then the same weight vector and the same threshold define exactly the hits that score above threshold and do not have negative scores on non-zero positions. Similar seeds were used by Brown and Hudek [33] for searching in nucleotide sequences with ambiguous characters that represent sets of nucleotides.

The examples in this section demonstrate that vector seeds generalize numerous approaches to hit definition in homology search, and allow to study them in a unified way. Using vector seeds, we can for example combine advantages of spaced seeds and BLAT seeds, or transfer techniques developed for nucleotide sequences to protein homology search.

3.2.2 Identifying hits in a sequence database

We have shown that vector seeds are very flexible and generalize several popular definitions of a hit for heuristic homology search. However, as the next extreme example shows, not every vector seed is practical. Let us assume that we want to find ungapped alignments of length 100 with at least 70% matches. Then the vector seed with the weight vector consisting of 100 consecutive positions of weight one and threshold 70 has perfect performance: 100% sensitivity and no false positives. However, finding hits of this seed is equivalent to finding the alignments themselves, and so the seed does not help us to achieve that task at all.

Practical seeds are those that have relatively small support, that is, a small number of positions with non-zero weight. For such seeds, we can organize the computation similarly as for spaced seeds. For a seed with support k, we could first create a hash table of all k-tuples from one sequence, storing only positions that have non-zero weight in the weight vector of the seed. Then, for each k-tuple x from the other sequence, we generate the set Y of all k-tuples that would produce a hit, and search for each $y \in Y$ in the hash table.

Consider for example the nucleotide seed $((1,1,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1), 13)$ with support 15. For every position in the second sequence, we have to examine all matching 15-letter hash table keys in the set Y. As the set Y contains all the keys that have at most two mismatches out of 15 positions of weight one, its size is

$$\sum_{i=0}^{2} \binom{15}{i} 3^i = 991.$$

If many of these 991 hash table entries are empty, the running time of the look-up phase may dominate the extension phase of the homology search, rendering the seed impractical.

On the other hand, if the first input sequence is large, the expected number of entries in each hash table entry will be greater. For example, if the first sequence contains one billion nucleotides, the hash table for a seed with support 15 will have the expected number of nucleotides per entry almost one. In such a case, we will perform an extension for almost all lookups, and the extension time will again dominate the lookup phase.

Therefore if the support of a seed is sufficiently small with respect to the size of the first sequence, the running time of homology search is dominated by extension attempts for false hits. In such a case, the false positive rate is a good measure of running time induced by the vector seed. Hence, we seek vector seeds of small support that achieve high sensitivity for a particular false positive rate.

3.2.3 Predicted performance of vector seeds

We have performed a simple experiment to verify the usefulness of vector seeds. We have studied vector seeds that combine the features of both spaced seeds and BLAT seeds allowing a fixed number of mismatches at arbitrary positions. We have considered all seeds (w, T) with binary weight vectors of length at most 18 containing at most 8 zeroes and between 9 and 15 ones. For a seed with support s we have allowed threshold T between $s-2$ and s, so allowing at most two mismatches at positions with weight one in the weight vector.

We have evaluated the sensitivity and false positive rate of these seeds using the simple probabilistic model discussed in Section 3.1. In particular, we compute the sensitivity as the probability of at least one hit in an alignment sequence sampled from the simple Bernoulli model, in which all positions are independent, alignments are of a constant length 64, and the probability of a match at each position is 0.7. We compute the false positive rate as the probability of a hit in a random alignment sequence sampled from the Bernoulli model, with match probability 0.25 and length equal to the length of the seed. This alignment sequence corresponds to two short unrelated DNA sequence windows aligned to each other. We compute both probabilities by the algorithm given in Section 3.4.

Our results are summarized in Figure 3.3 and Table 3.3. Seeds that permit both mismatches and the structure of spaced seeds have a large advantage over either alone. For example, the optimal spaced seed of weight 10 has sensitivity 60%. The BLAT seed BLAT-13-15 of length 15 allowing

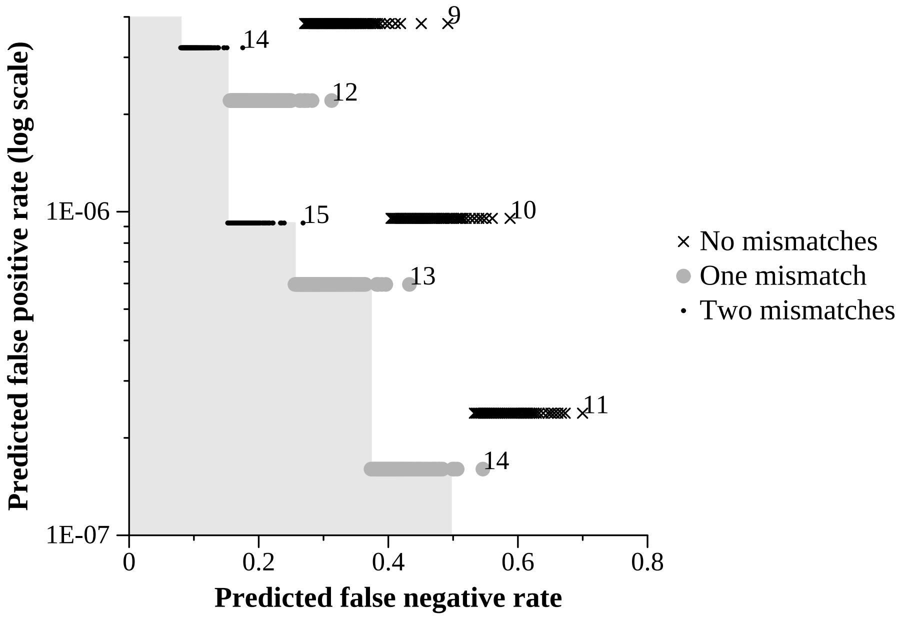


Figure 3.3: Predicted false negative and false positive rate of vector seeds under the simple Bernoulli model, with match probability 0.7 and length 64. Seeds in a horizontal row in the figure have the same support and threshold, and consequently the same false positive rate. The support is shown next to the least sensitive seed in the row. Seeds in the white area are outperformed in both false positives and false negatives by some seed on the boundary of the grey area. We show only seeds with false positive rate between 10^{-7} and $4 \cdot 10^{-6}$. Best vector seeds of support 13 and threshold 12, and support 15 and threshold 13 have much higher sensitivity and somewhat lower false positive rate than the best spaced seed of weight 10.

two mismatches has sensitivity 73% and a slightly lower false positive rate. By introducing spaces to this seed we improve the sensitivity to 85%.

Seeds of support 15 may not be practical, as there are 4^{15} possible hash table entries, and for each position in the second sequence we need to check more than 900 of those (see the discussion in Section 3.2.2). The more practical vector seed VS-12-13, allowing one mismatch, has sensitivity 74%, which is comparable to the BLAT-13-15 seed, and its false positive rate is only two-thirds that of BLAT-13-15. This seed therefore outperforms both the optimal spaced seed and the BLAT seed of support 15. Another interesting seed is VS-11-12. It is almost as sensitive as VS-13-15, but has support only 12, which is preferable for shorter sequences. Its false positive rate is about 2.3 times higher than the false positive rate of spaced seeds of weight 10.

3.3 Probabilistic models of conserved coding regions

As we have seen, spaced seeds optimized with respect to a simple Bernoulli alignment model improve sensitivity of homology search. However, such a model does not capture the properties of real alignments. Perhaps, even better seeds can be obtained by optimizing them with respect to more realistic models. We concentrate on modeling alignments of protein coding regions. Such alignments

Table 3.2: Predicted sensitivity and false positive rate of selected vector seeds under the simple Bernoulli model with match probability 0.7 and length 64. In the table, BLAST-s is the unspaced seed of length s, BLAT-t-s is unspaced seed of length s with $s - t$ allowed mismatches, PH-s is the optimal spaced seed of weight s, and VS-t-s is the optimal vector seed of support s and threshold t. The optimal vector seed VS-13-15 has higher sensitivity than both the corresponding BLAT seed and a spaced seed with similar false positive rate. Vector seeds VS-12-13 and VS-11-12 provide practical alternatives, with smaller hash tables.

Weight vector	T	Support	Name	Sensitivity	False positive rate
1111111111	10	10	BLAST-10	41%	9.5×10^{-7}
1110101100011011	10	10	PH-10	60%	9.5×10^{-7}
111111111111111	13	15	BLAT-13-15	73%	9.2×10^{-7}
111111011011101111	13	15	VS-13-15	85%	9.2×10^{-7}
111101100110101111	12	13	VS-12-13	74%	6.0×10^{-7}
111011001011010111	11	12	VS-11-12	84%	2.2×10^{-6}

provide valuable information for gene finding, as we demonstrate in Chapter 4. Therefore, we need tools that detect alignments of homologous coding regions with high sensitivity.

Evolutionary forces conserve protein coding regions more strongly than they conserve non-coding regions. Although higher conservation makes sequence homologies in coding regions easier to detect, they pose unique challenges for homology search programs. Recall that the amino acid sequence of a protein is encoded in DNA as a sequence of triplet codons, with each triplet representing one amino acid according to the genetic code shown in Figure 1.2. However, the code is redundant: some amino acids have multiple encodings. A nucleotide mutation causing a change in the amino acid sequence of the encoded protein may disrupt a function of the protein. Therefore, such mutations are relatively rare. On the other hand, *silent* mutations that leave the protein sequence the same occur much more often, so similar proteins can differ greatly in corresponding coding regions in DNA.

Another difficulty is caused by introns, since a protein alignment may correspond to several short alignments of exonic DNA, separated by long intron gaps. The task is easier if the location of exons is known prior to local alignment, as we may then concatenate exons of each gene together and ignore intronic sequences. However, we assume that we do not know exon location, as is the case in gene finding.

Alignments in coding regions have specific properties not modeled well by the simple Bernoulli model. First, the Bernoulli model assumes equal mutation rate at all alignment positions. However, in protein coding regions, the third positions in codons can often mutate silently. For example, a mutation from C to T or vice versa is always silent in the third position of a codon. Therefore, the third positions in codons of even closely related proteins can undergo substantial mutation. There also can be substantial within-codon dependencies in alignment positions. Finally, coding sequence alignments (and, actually, non-coding alignments) often are quite inhomogeneous: some regions are highly conserved between the two species while their neighbors are not.

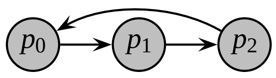


Figure 3.4: HMM representation of the model $M^{(3)}(p_0, p_1, p_2)$. Each state is labeled with emission probability of '1'. Each transition has probability 1.

The simple nucleotide BLAST seed [8], as well as the original spaced seed used in PatternHunter [116], do not reflect the properties of protein coding regions. By contrast, Kent and Zahler [93] implicitly used the spaced seed `110110110` in their alignment program, WABA. This seed represents their intuition that coding DNA has 3-periodic structure, and that the third position of a codon is less conserved than the first two. Kent and Zahler report a substantial improvement over BLAST in sensitivity in detecting homologous coding sequences.

We obtain still better seeds for protein coding regions by modeling properties of the alignments in these regions with probabilistic models. We describe three models, which increase in complexity and fidelity. All of our models can be represented as HMMs, and therefore we can compute seed sensitivity for all three models using a general algorithm that works for an arbitrary hidden Markov model. We will describe such an algorithm in Section 3.4. All models presented in this section generate binary alignment sequences, where a `1` represents a match and a `0` represents a mismatch. Such models can be used for designing both spaced seeds and vector seeds. We also extend the probabilistic model so that the length of the alignment is a random variable, chosen from a fixed distribution of alignment lengths.

3.3.1 Three-periodicity

The most obvious property of alignments in coding regions is their three-periodicity and that some of the positions of a codon are less conserved than others. A simple extension of the Bernoulli model, which we call $M^{(3)}$, encapsulates this idea. Model $M^{(3)}(p_0, p_1, p_2)$ represents random alignments where the match probability depends on its relative codon position, but the positions within the alignment are still independent. Formally, it is a sequence of independent Bernoulli random variables $X_0, X_1, \ldots$ where $\Pr(X_i = 1) = p_{i \bmod 3}$. This model can be expressed as the simple three-state HMM depicted in Figure 3.4.

Table 3.3 shows the parameters of the model $M^{(3)}$, estimated from our training set of alignments between human and fruit fly protein coding regions and between human and mouse protein coding regions. More details about the data sets can be found in Section 3.5. In both cases, the third position is much less conserved than the others, and the first position is somewhat less conserved than the second.

3.3.2 Dependencies within codon

Model $M^{(3)}$ models the different conservation levels within codons arising from the redundancy of the genetic code. However, there are also dependencies among codon positions, as demonstrated by Table 3.4. The first row of the table shows the probabilities of all triplets as estimated by model $M^{(3)}$, under the assumption that codon positions are independent. The second row shows the probabilities of all triplets in our training set. In particular, the triplets `000` and `111` occur in the data more often than expected by the $M^{(3)}$ model.

Table 3.3: Parameters of model $M^{(3)}$ estimated from our training sets consisting of alignments between human and mouse and human and fruit fly coding regions (more detailed description can be found in in Section 3.5.1). The third position of the codon is the least conserved, with probability of a match only 40% in alignments between human and fruit fly.

Data set	p_0	p_1	p_2
Human/mouse	0.82	0.87	0.61
Human/fruit fly	0.67	0.77	0.40

Table 3.4: Comparison of the probabilities of the eight possible conservation patterns within a codon in the models $M^{(3)}$ and in the training data set consisting of alignments of human and fruit fly coding regions (see Section 3.5.1). Triplets consisting of three mismatches or three matches (`000` and `111`) occur more often in the data than is predicted by the $M^{(3)}$ model.

	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
Model $M^{(3)}$	0.05	0.03	0.15	0.10	0.09	0.06	0.31	0.20
Training data	0.11	0.04	0.12	0.06	0.06	0.03	0.32	0.27

To model dependencies within codons, we use another model, $M^{(8)}$. The model has eight parameters $p_{\mathtt{000}}, p_{\mathtt{001}}, \ldots, p_{\mathtt{111}}$ and represents a random alignment as a sequence of codons. Each codon has conservation pattern $x \in \{\mathtt{0},\mathtt{1}\}^3$ with probability p_x; the sum of $p_{\mathtt{000}}, p_{\mathtt{001}}, \ldots, p_{\mathtt{111}}$ is 1. In this model, the positions within one codon have arbitrary dependencies specified by the parameters $p_{\mathtt{000}}, \ldots, p_{\mathtt{111}}$, yet the individual codons are independent of each other. Formally, this model is a sequence of independent triples of Bernoulli random variables $(X_0, X_1, X_2), \ldots, (X_{3i-3}, X_{3i-2}, X_{3i-1}), \ldots,$ such that $\Pr(X_{3i} = a, X_{3i+1} = b, X_{3i+2} = c) = p_{abc}$. Model $M^{(8)}$ can be represented by the HMM shown in Figure 3.5. This HMM has three states for each possible triplet, deterministically generating the appropriate binary pattern. The random process is encoded by transition probabilities that choose one of the codon patterns at each codon boundary. It is possible to use this HMM to generate alignments sequences whose length not divisible by three. Figure 3.6 shows an alternative HMM for the same purpose, which is less intuitive, but has only 8 states, instead of the 24 needed in the first model. Since the running time of the algorithm for computing seed sensitivity is cubic in the number of states, this model may speed the sensitivity computation up to 27 times.

3.3.3 Inhomogeneity of alignments

The previous models assume that alignments have roughly the same conservation rate across their length. In fact, the conservation pattern of a typical alignment is highly non-uniform. Many alignments include short, highly conserved regions, surrounded by less well-conserved regions. This is not surprising, as highly conserved regions are more likely to be functional parts of the proteins [172].

We address this problem with a new model, depicted in Figure 3.7. This model consists of four copies of the HMM for $M^{(8)}$ shown in Figure 3.6, where each copy represents regions with a different level of conservation. We model changes in the conservation rate by allowing transitions

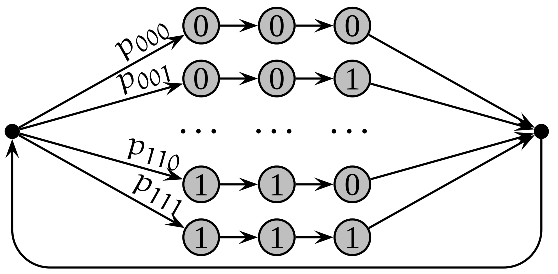
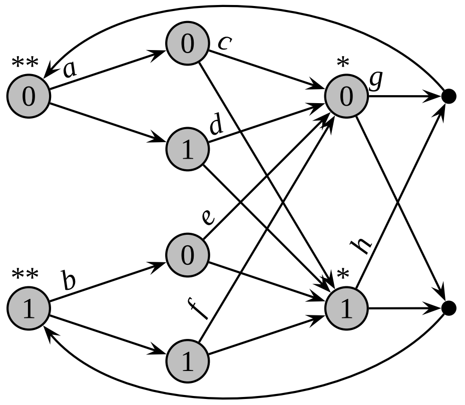


Figure 3.5: A simple HMM representation of the model $M^{(8)}$ with parameters $p_{000}, \ldots, p_{111}$. Each state is labeled with emission probability of '1'. The two black dots represent *silent* states, that is, states that do not emit any character. They can be eliminated by connecting each of the non-silent states in the third column directly to each non-silent states in the first column with appropriate p_{ijk} transition probabilities.



$$a = (p_{000} + p_{001})/(p_{000} + p_{001} + p_{010} + p_{011})$$
$$b = (p_{100} + p_{101})/(p_{100} + p_{101} + p_{110} + p_{111})$$
$$c = p_{000}/(p_{000} + p_{001})$$
$$d = p_{010}/(p_{010} + p_{011})$$
$$e = p_{100}/(p_{100} + p_{101})$$
$$f = p_{110}/(p_{110} + p_{111})$$
$$g = h = p_{000} + p_{001} + p_{010} + p_{011}$$

Figure 3.6: HMM representation of the model $M^{(8)}$ with parameters $p_{000}, \ldots, p_{111}$. This model is less intuitive but smaller than the model in Figure 3.5. Each state of the HMM is labeled with emission probability of '1'. Each state non-silent has two outgoing transitions. One of them has the transition probability shown in the picture; the other has probability such that they add to one. The states labeled with stars will be used to connect this model to a larger model shown in Figure 3.7.

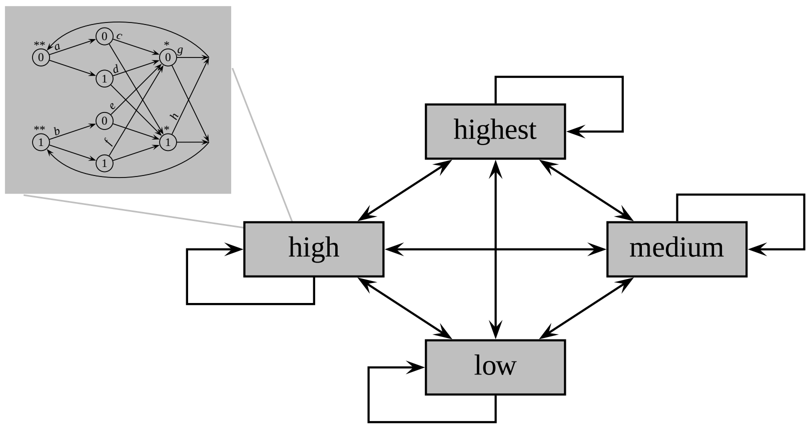


Figure 3.7: To allow transitions between different conservation levels, we add transitions from all copies of both states labeled by a star in Figure 3.6 to all copies of both states labeled by two stars.

Table 3.5: Overview of the parameters of the $M^{(4\times 8)}$ models, obtained by the Baum-Welch algorithm from the training sets described in Section 3.5.1. The columns labeled p_0, p_1, and p_2 show the probability of a match in the first, second, and third position of a triplet generated in each of the four submodels. We see that individual submodels correspond to regions with different conservation levels. In the fruit fly data set, the two highest conservation levels differ mainly in the probability of a match in the third codon position.

Conservation	Human/mouse			Human/fruit fly		
level	p_0	p_1	p_2	p_0	p_1	p_2
Highest	0.97	0.97	0.80	0.84	0.90	0.68
High	0.91	0.95	0.64	0.83	0.93	0.46
Medium	0.83	0.90	0.56	0.78	0.88	0.35
Low	0.58	0.66	0.43	0.49	0.57	0.33

between the four modules after each triplet. We will call this model $M^{(4\times 8)}$.

Given a training data set, the parameters of the $M^{(3)}$ and $M^{(8)}$ models can be estimated simply by counting the patterns of matches and mismatches at individual codon positions. We cannot use this method to train the $M^{(4\times 8)}$ model, since we do not know which of the four copies should generate each codon. We use the standard Baum-Welch algorithm [16, 135] to train the model. Table 3.5 shows the conservation rates for the four modules obtained by training. Although the Baum-Welch algorithm is not guaranteed to produce the optimal model parameters, the models we have obtained characterize the data better than the simpler $M^{(8)}$ model, as we will see in Section 3.5.2.

Somewhat similar hidden Markov models were previously used by Li and Miller [109] to characterize background mutation rates of long stretches of alignment. Their models have four states representing difference conservation levels. Since they model general nucleotide alignments, their models do not exhibit the three-periodic structure characteristic of protein coding regions.

3.3.4 Length of alignments

The Bernoulli model, as well as our models of homologous coding regions can be used to generate sequences of arbitrary length. Many authors, including Ma *et al.* [116], Choi and Zhang [49], and Buhler *et al.* [36], consider random alignments of length 64, a somewhat arbitrary constant chosen by Ma *et al.* [116] and propagated thereafter.

Since different seeds are optimal for different alignments lengths [49], we have decided to consider the length of the alignment as a random variable, samples from some probability distribution. We require that there is an upper bound N on the longest alignment length. The sensitivity of a seed in a model with variable alignment lengths is the weighted average of the probabilities for models with fixed lengths up to N.

In our experiments in Section 3.5, we use an empirical distribution, observed on the training data, where we discard the longest 5% of alignments to keep the threshold N reasonable for fast computation of sensitivity. Instead of the empirical distribution, we could also use a parametric distribution. We have considered this possibility for modeling lengths of protein alignments [25]. However, we observe that the particular parameters of the length distribution do not have high impact, since longer alignment lengths typically yield the same optimal seed. For example, consider the $M^{(8)}$ model, trained on alignments between human and fruit fly. In this model, the same seed is the most sensitive for all lengths between 39 and 196, out of all seeds of weight 10 and length at most 18. Still, in the $M^{(3)}$ model, two seeds oscillate between lengths 82 and 121—one is optimal when the length is equal one modulo three, and the other is optimal for other lengths. In this case, a small change in length distribution might change the overall optimum seed, but these two seeds appear to have very similar sensitivity.

3.4 Algorithm for computing sensitivity of vector seeds under an HMM

Here, we show how to compute the sensitivity of a vector seed to detect alignments generated by a hidden Markov model. Our method is based on the algorithm by Keich *et al.* [91] for computing the sensitivity of a spaced seed in the simple Bernoulli alignment model. We extend this algorithm to work for vector seeds instead of only spaced seeds, and to work for hidden Markov models, not just Bernoulli models.

We are particularly interested in computing the seed sensitivity under a hidden Markov model since all our models of homologous coding regions have this form. We will use this algorithm in our experiments in Section 3.5 to compare the sensitivity of many seeds and choose the best.

The running time of the algorithm is exponential in the length of the seed for some seeds. In addition, it only computes the sensitivity of a single seed. The set of all spaced seeds of a given weight is infinite, as they can contain arbitrary number of zeroes. For vector seeds, the weights and the threshold may vary as well. To choose the most sensitive seed for a particular application, we typically evaluate the sensitivity of a large family of potential seeds and choose the best one. For example, for spaced seeds, were were able to consider all seeds of length at most 18 and weight 10. Although seed optimization is computationally extensive, once we optimize a seed for a particular alignment model, we can use it for homology searches in all situations in which the model characterizes alignment properties reasonably well.

Assume that we are given a vector seed $Q = (w, T)$ and an HMM $\mathcal{H}$ that generates alignment sequences. We want to compute the probability of seed Q having at least one hit in an alignment sequence of length N generated by the HMM. This probability is the *sensitivity* of the seed Q to the model $\mathcal{H}$.

Each step of the generative process consists of the emission of one character in the current state, followed by the transition to the next state. The characters generated by the HMM are drawn from a small finite set D of real numbers, and correspond to individual positions of the alignment sequence. For example, we typically use binary alignment sequence to characterize nucleotide alignments and sequence of BLOSUM62 scores to characterize protein alignments.

We first introduce notation that will allow us to describe events occurring in the generative process of the HMM. Let V be the set of states of the HMM. Let $\mathit{Start}(u)$ be the event that the HMM starts generating in state u. Using the notation introduced in Section 1.2, $\Pr(\mathit{Start}(u)) = s_u$, where s_u is the initial probability of state u. Let $\mathit{Emit}(x)$ be the event that the HMM generates sequence x in the first $|x|$ steps. For example, if x has length 1, the probability $\Pr(\mathit{Emit}(d) \mid \mathit{Start}(u))$ is equal to the emission probability $e_{u,d}$, for any state u. Let $\mathit{Hit}(i, Q)$ be the event that the sequence generated in the first i steps contains a hit of the seed Q. Finally, let $\mathit{State}(i, u)$ be the event that after i transitions the HMM is in state u. Notice that for any states u and v, the probability $\Pr(\mathit{State}(1, v) \mid \mathit{Start}(u))$ equals the transition probability $a_{u,v}$.

We will proceed by dynamic programming. The subproblem is the following: for any $i \le N$, sequence $x \in D^*$ and $u \in V$, let $P_Q(i, x, u)$ be the probability that the sequence generated by the HMM in i steps contains a hit of the seed Q, provided that the HMM starts in state u and x is a prefix of the generated sequence, or using our notation:

$$P_Q(i, x, u) = \Pr(\mathit{Hit}(i, Q) | \mathit{Start}(u), \mathit{Emit}(x)). \tag{3.1}$$

According to this definition, the value $P_Q(N, \lambda, u)$ equals the probability that the seed has a hit in an alignment of length N generated by the HMM starting in state u. By weighting these probabilities by the initial probability of each individual state, we obtain the sensitivity of the seed Q:

$$\Pr(\mathit{Hit}(N, Q)) = \sum_{u \in V} s_u \cdot P_Q(N, \lambda, u). \tag{3.2}$$

Technically, $P_Q(i, x, u)$ is undefined if $\Pr(\mathit{Start}(u), \mathit{Emit}(x)) = 0$. We will recognize two cases when this may happen. First, if $\Pr(\mathit{Start}(u)) = s_u = 0$, we will define $P_Q(i, x, u)$ for notational convenience as the probability $\Pr(\mathit{Hit}(i, Q) | \mathit{Start}(u), \mathit{Emit}(x))$ in an HMM with initial probabilities modified so that s_u is non-zero (the value of $P_Q(i, x, u)$ will be the same for any non-zero s_u). Similarly, we will extend the definition of any other probability conditional on $\mathit{Start}(u)$. Second, if $\Pr(\mathit{Emit}(x) | \mathit{Start}(u)) = 0$, the value $P_Q(i, x, u)$ can be arbitrary, since it will always be multiplied by 0 in the dynamic programming algorithm.

We will compute the value of $P_Q(i, x, u)$ for only a limited set of sequences $x \in D^*$. Before we define this set, we need to introduce several new terms. Let M be the length of the seed weight vector w, and let $H \subseteq D^M$ be the set of all hits of seed Q, that is, the set of all sequences x of length M such that $x \cdot w \ge T$. Let H_P be the set of all prefixes of sequences in H. We will call these sequences *possible hits*, because each $x \in H_P$ can be extended by adding additional characters in D to obtain some $y \in H$. Let H_G be the set of sequences x such that any sequences of length M that has x as a prefix belongs to H. We will call sequences in H_G *guaranteed hits*, because their extension to length M is always a hit. Note that by definition, $H \subseteq H_G \subseteq H_P$. The value

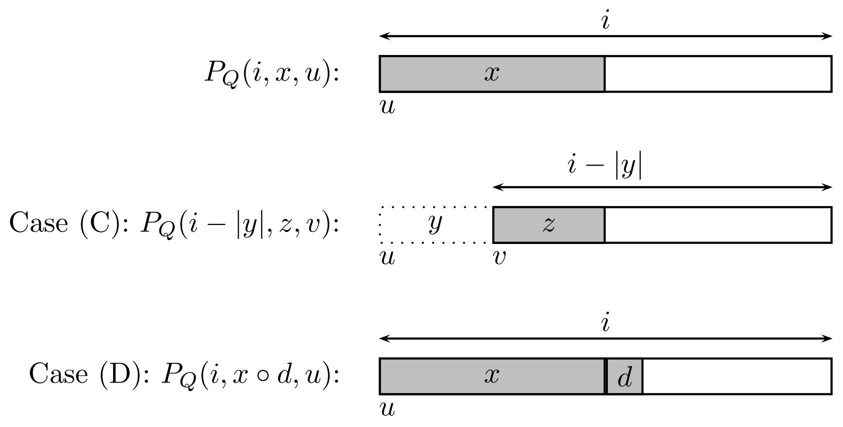


Figure 3.8: Illustration of cases (C) and (D) of the dynamic programming algorithm for computing sensitivity of a vector seed Q. The bar represents the random alignment generated by an HMM; shaded parts represent the fixed prefix of the alignment sequence. A state generating the first character of a block is shown under the left boundary of the block.

$P_Q(i,x,u)$ will be computed for sequences x in the set H_* of possible hits and single-character extensions to possible hits that are not guaranteed hits:

$$H_* = H_P \cup \{x \circ d \mid x \in H_P \setminus H_G, d \in D\}, \tag{3.3}$$

where $\circ$ denotes concatenation of two sequences, to distinguish it from the dot product. Let *suffix*(x,Q) be the longest proper suffix z of x such that z is a possible hit.

Using this notation, we can express the algorithm for computing the probability $P_Q(i,x,u)$ by the recurrent formula given in the following theorem (see also illustration in Figure 3.8).

Theorem 12. *Let Q be a vector seed and $\mathcal{H}$ a hidden Markov model. Then for any state u, sequence $x \in H_*$ and integer $i \geq 0$, the probability $P_Q(i,x,u)$ can be computed by the following recurrent formula:*

$$P_Q(i,x,u) = \begin{cases} 0 & \textit{if } i < M \quad \textit{(A)} \\ 1 & \textit{if } i \geq M \textit{ and } x \in H_G \quad \textit{(B)} \\ \sum_{v\in V} p_v \cdot P_Q(i-|y|,z,v) & \textit{if } i \geq M \textit{ and } x \notin H_P\textit{, where} \\ & \quad x = y\circ z,\ z = \textit{suffix}(x,Q), \\ & \quad p_v = \Pr(\textit{State}(|y|,v) \mid \textit{Start}(u), \textit{Emit}(x)) \quad \textit{(C)} \\ \sum_{d\in D} q_d \cdot P_Q(i, x\circ d, u) & \textit{if } i \geq M \textit{ and } x \in H_P \setminus H_G\textit{, where} \\ & \quad q_d = \Pr(\textit{Emit}(x\circ d) \mid \textit{Start}(u), \textit{Emit}(x)) \quad \textit{(D)} \end{cases}$$

Proof of the theorem and further discussion of the algorithm is organized as follows. First, we will show in Lemma 13 that to obtain the desired values $P_Q(N,\lambda,u)$ by this recurrent formula, we need to compute $P_Q(i,x,u)$ only for $i \leq N$ and $x \in H_*$. Then, in Lemma 14 we show the correctness of the recurrent formula. Finally, in Theorem 15 we analyze the running time of the algorithm and further details, including the efficient computation of H_*, *suffix*(x,Q), and the probabilities p_v and q_d needed in the formula. Figure 3.9 shows an example of the computation of $P_Q(i,x,u)$ for a short seed.

$Q = ((2,1),3)$
$D = \{0,1,2\}$

$H = \{(1,1),(1,2),(2,0),(2,1),(2,2)\}$
$H_P = H \cup \{\lambda,(1),(2)\}$
$H_G = H \cup \{(2)\}$
$H_* = H_P \cup \{(0),(1,0)\}$

$$\begin{tikzcd} A \arrow[loop above, "a_{A,A}=1/2"] \arrow[r, bend left, "a_{A,B}=1/2"] & B \arrow[l, bend left, "a_{B,A}=1"] \end{tikzcd}$$

$e_{A,0}$=1/2	$e_{B,0}$=1/4
$e_{A,1}$=1/4	$e_{B,1}$=1/4
$e_{A,2}$=1/4	$e_{B,2}$=1/2
s_A=1/2	s_B=1/2

Computation of $P_Q(i,x,u)$ for $i \in \{2,3\}$ (values for $i < 2$ are zero):

	$i=2$		$i=3$	
$x \in H_*$	$u=A$	$u=B$	$u=A$	$u=B$
λ	$\frac{1}{2}P_Q(2,(0),A)$ $+\frac{1}{4}P_Q(2,(1),A)$ $+\frac{1}{4}P_Q(2,(2),A)$ $=13/32$	$\frac{1}{4}P_Q(2,(0),B)$ $+\frac{1}{4}P_Q(2,(1),B)$ $+\frac{1}{2}P_Q(2,(2),B)$ $=5/8$	$\frac{1}{2}P_Q(3,(0),A)$ $+\frac{1}{4}P_Q(3,(1),A)$ $+\frac{1}{4}P_Q(3,(2),A)$ $=85/128$	$\frac{1}{4}P_Q(3,(0),B)$ $+\frac{1}{4}P_Q(3,(1),B)$ $+\frac{1}{2}P_Q(3,(2),B)$ $=93/128$
(0)	$\frac{1}{2}P_Q(1,\lambda,A)$ $+\frac{1}{2}P_Q(1,\lambda,B)$ $=0$	$1\cdot P_Q(1,\lambda,A)$ $+0\cdot P_Q(1,\lambda,B)$ $=0$	$\frac{1}{2}P_Q(2,\lambda,A)$ $+\frac{1}{2}P_Q(2,\lambda,B)$ $=33/64$	$1\cdot P_Q(2,\lambda,A)$ $+0\cdot P_Q(2,\lambda,B)$ $=13/32$
(1)	$\frac{3}{8}P_Q(2,(1,0),A)$ $+\frac{1}{4}P_Q(2,(1,1),A)$ $+\frac{3}{8}P_Q(2,(1,2),A)$ $=5/8$	$\frac{1}{2}P_Q(2,(1,0),B)$ $+\frac{1}{4}P_Q(2,(1,1),B)$ $+\frac{1}{4}P_Q(2,(1,2),B)$ $=1/2$	$\frac{3}{8}P_Q(3,(1,0),A)$ $+\frac{1}{4}P_Q(3,(1,1),A)$ $+\frac{3}{8}P_Q(3,(1,2),A)$ $=5/8$	$\frac{1}{2}P_Q(3,(1,0),B)$ $+\frac{1}{4}P_Q(3,(1,1),B)$ $+\frac{1}{4}P_Q(3,(1,2),B)$ $=1/2$
(2)	1	1	1	1
(1,0)	$\frac{5}{6}P_Q(0,\lambda,A)$ $+\frac{1}{6}P_Q(0,\lambda,B)$ $=0$	$\frac{1}{2}P_Q(0,\lambda,A)$ $+\frac{1}{2}P_Q(0,\lambda,B)$ $=0$	$\frac{5}{6}P_Q(1,\lambda,A)$ $+\frac{1}{6}P_Q(1,\lambda,B)$ $=0$	$\frac{1}{2}P_Q(1,\lambda,A)$ $+\frac{1}{2}P_Q(1,\lambda,B)$ $=0$
$(1,1)$	1	1	1	1
$(1,2)$	1	1	1	1
$(2,0)$	1	1	1	1
$(2,1)$	1	1	1	1
$(2,2)$	1	1	1	1

Result: $\Pr(Hit(3,Q)) = \frac{1}{2}P_Q(3,\lambda,A) + \frac{1}{2}P_Q(3,\lambda,B) = 89/128 \approx 0.69$

Figure 3.9: Example of execution of the dynamic algorithm for computing vector seed sensitivity.

Lemma 13. *To compute the value of $P_Q(i,x,u)$ by the recurrent formula (A)-(D) for some $i \in \{0,1,\ldots,N\}$, $x \in H_*$, and $u \in V$, we need only the values $P_Q(i',x',u')$ such that both $x' \in H_*$ and either i' is less than i or $i' = i$ and x' is strictly longer than x.*

Proof. Clearly, the lemma holds for the base cases (A) and (B) of the recurrent formula. Case (C) applies if $x \in H_* \setminus H_P$. We need values $P_Q(i-|y|,z,v)$ where $z = \mathit{suffix}(x,Q)$ and $x = y \circ z$. By the definition of $\mathit{suffix}(x,Q)$, z must by in H_P and hence in H_*. Since z is a proper suffix of x, y is of length at least 1, and therefore $i-|y|$ is less than i. Finally, case (D) applies when $x \in H_P \setminus H_G$. We need the values of $P_Q(i,x\circ d,u)$, for $d \in D$. Clearly, $x \circ d$ is in H_* and is longer than x. □

This lemma shows that we can compute the probabilities $P_Q(i,x,u)$ in order of increasing i. For each i, we compute from the longest strings in H_* to the shortest. In the example in Figure 3.9, this means proceeding from left to right and, for each column, from its bottom to its top.

Lemma 14. *Let $i \in \{0,1,\ldots,N\}$, $x \in H_*$, and $u \in V$ such that $\Pr(\mathit{Emit}(x) \mid \mathit{Start}(u)) > 0$. Then, the recurrent formula given by cases (A)-(D) computes the correct value of $P_Q(i,x,u)$ as defined in Equation (3.1).*

Proof. Case (A) recognizes that a seed of length M cannot have a hit in a region shorter than M. In case (B), since sequence x is in H_G, we are guaranteed that first M characters of the sequences constitute a hit.

In case (C), since x is not in the set of possible hits H_P, a hit will not start at the first position of the alignment, so only following positions need to be considered. In particular, the longest suffix z of string x in the set H_P corresponds to the first possible position where a hit can begin. We need to consider all possible states v in which the HMM can start to generate the suffix z; the characters of y only matter in that we must emit them all, and this influences how we get to the beginning of z.

$$\begin{aligned}
P_Q(i,x,u) &= \Pr(\mathit{Hit}(i,Q) \mid \mathit{Start}(u), \mathit{Emit}(x)) \\
&= \sum_{v \in V} \Pr(\mathit{State}(|y|,v) \mid \mathit{Start}(u), \mathit{Emit}(x)) \\
&\qquad \cdot \Pr(\mathit{Hit}(i,Q) \mid \mathit{Start}(u), \mathit{State}(|y|,v), \mathit{Emit}(x))
\end{aligned}$$

Properties of hidden Markov models imply that if we know that the HMM is in state v after the first $|y|$ characters, the characters emitted after the first $|y|$ steps do not depend on the events $\mathit{Start}(u)$ and $\mathit{Emit}(y)$. Since a hit cannot start on any of the first $|y|$ positions, we can consider only the remaining alignment sequence of length $i-|y|$ that is generated starting in state v:

$$\begin{aligned}
&\Pr(\mathit{Hit}(i,Q) \mid \mathit{Start}(u), \mathit{State}(|y|,v), \mathit{Emit}(x)) \\
&= \Pr(\mathit{Hit}(i-|y|,Q) \mid \mathit{Start}(v), \mathit{Emit}(z)) \\
&= P_Q(i-|y|,z,v).
\end{aligned}$$

Note that $\Pr(\mathit{Start}(v), \mathit{Emit}(z))$ is zero only if $\Pr(\mathit{State}(|y|,v) \mid \mathit{Start}(u), \mathit{Emit}(x))$ is zero. Therefore values of P_Q that are not properly defined are multiplied by zero.

Finally, case (D) provides a formula for combining the subproblems where the first $|x|+1$ characters are fixed into a subproblem where only $|x|$ characters are fixed. Similarly as for case C, $P_Q(i,x\circ d,u)$ is not defined if $\Pr(\mathit{Emit}(x\circ d)|\mathit{Start}(u)) = 0$. In that case, q_d must be zero, and therefore the formula works correctly. □

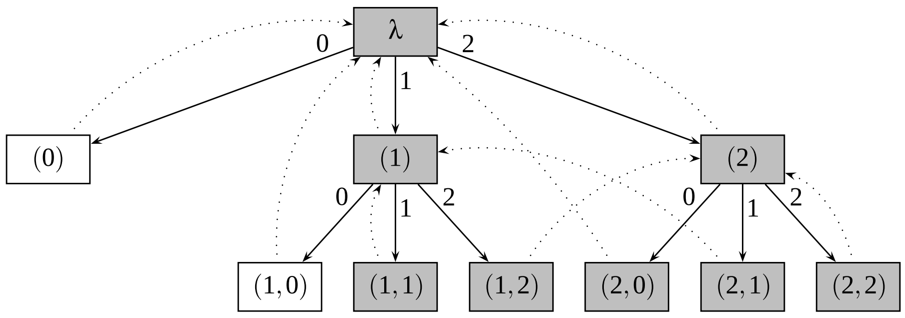


Figure 3.10: The trie representing set H_* for the seed $Q = ((2,1),3)$ and set $D = \{0,1,2\}$, used in Figure 3.9. Shaded boxes represent possible hits, that is, sequences from H_P. Dotted edges connect each sequence x with its value *suffix*(x,Q). Note that in practice we do not store the whole sequence in each node, as that would increase the running time and memory requirements.

Li *et al.* [110] show how to efficiently organize the computation of a similar recurrent formula so that the running time to compute sensitivity of one spaced seed in a Bernoulli model is $O(|H_*| \cdot N)$. When we extend their methods to a more complicated case of vector seeds and hidden Markov models, we obtain the following result.

Theorem 15. *The probability that a given vector seed Q has a hit in an alignment sequence of length N generated by a given HMM with set of states V emitting numbers from a set D can be computed in $O(|H_*| \cdot |V|^2 K)$ time for preprocessing and $O(K)$ time for each of $O(|H_*| \cdot |V| \cdot N)$ subproblems, where $K = \max(|V|, |D|)$. Hence, if the HMM and its emission set D are both of constant size, the running time is $O(|H_*| \cdot N)$.*

Proof. In the first step, the algorithm will compute the set H_* and mapping *suffix*(x,Q) from $H_* \setminus \{\lambda\}$ to H_P. Since the set H_* is closed under the prefix operation, we can represent it conveniently as a trie. A trie is a tree with edges labeled by elements of D. Each node of the trie corresponds to one element of H_* obtained by concatenating the labels of edges on the path from root to this node (see an example of such a trie in Figure 3.10). The definition of H_* implies that all internal nodes of the trie correspond to sequences from H_P and all leaves correspond to sequences that are either in H_G or in $H_* \setminus H_P$. To build the trie, we keep for each node the length of its corresponding sequence x and the dot product between x and a prefix of the seed's weight vector w. In addition, before we start building the trie, we compute the best and worst possible score for each suffix of the weight vector of the seed Q. We can compute the best possible score for a suffix of the weight vector as the dot product of the suffix and an alignment sequence consisting of the maximum element in D. Similarly, we can obtain the worst possible score of a suffix by using an alignment sequence consisting of the minimum element in D.

If we have just created a new node of the trie, we compute the smallest and highest score its descendants may achieve. Then we can determine whether it is a possible or guaranteed hit by comparing these values with the seed's threshold. If the current node is not a possible hit, it will be a leaf. Otherwise, if we have not yet achieved length M, we create its $|D|$ children and continue recursively. The whole process takes $O(|H_*|)$ time.

Then, we compute *suffix*(x,Q), for each $x \in H_*$, by creating a link from the node corresponding to x to a node corresponding to *suffix*(x,Q). We will proceed from the root, level by level. The first two levels are special cases. The value of *suffix*(λ, Q) is undefined, and *suffix*$(d, Q) = \lambda$ for each one-letter sequence d. Now, consider some node corresponding to a sequence x of length greater than one. Let y be the sequence of its parent, that is, $x = y \circ d$, and let *suffix*$(y, Q) = y'$. Then, *suffix*(x, Q) is either $y' \circ d$ or its suffix. The node corresponding to $y' \circ d$ can be found by following the suffix link of the parent to y' and then moving to its child along the edge labeled d. Notice that since $y' \in H_P$, and $|y'| < |x| - 1 < M$, y' is not a leaf. Therefore, a node for sequence $y' \circ d$ exists. If $y' \circ d$ is in H_P, then clearly *suffix*$(x, Q) = y' \circ d$. Otherwise, *suffix*$(x, Q) =$ *suffix*$(y' \circ d, Q)$. Since $|y' \circ d| < |x|$, the suffix link for $y' \circ d$ is already computed, so we copy it to make a suffix link for x. Since we can find the correct link for each node in constant time, this step takes $O(|H_*|)$ time as well.

We will also need for each sequence x in H_* a link from its node to the node corresponding to the prefix y such that $y \circ$ *suffix*$(x, Q) = x$. Note that y is a prefix of x and its length can be computed by comparing the lengths of x and *suffix*(x, Q). Therefore, all of these links can be computed by a depth-first search of the tree, in which we keep the nodes on the current path from root in an array indexed by their length and find the node corresponding to $|y|$ in constant time. This, too, can be therefore done in $O(|H_*|)$ time.

To complete the preprocessing, we need to compute the probabilities p_v and q_d needed in cases (C) and (D) of the recurrence formula in Theorem 12. We will use a modification of the standard forward algorithm for HMMs [17, 135] to compute values $B_{u,v,x}$ and $C_{u,x}$ defined as follows:

$$B_{u,v,x} = \Pr(\mathit{Emit}(x), \mathit{State}(|x|, v) \mid \mathit{Start}(u))$$

$$C_{u,x} = \Pr(\mathit{Emit}(x) \mid \mathit{Start}(u)) = \sum_{v \in V} B_{u,v,x}$$

The conditional probabilities required then now be computed using these two formulas:

$$p_v = \Pr(\mathit{State}(|y|, v) \mid \mathit{Start}(u), \mathit{Emit}(\overbrace{yz}^{x})) = \frac{B_{u,v,y} \cdot C_{v,z}}{C_{u,x}} \tag{3.4}$$

$$q_d = \Pr(\mathit{Emit}(x \circ d) \mid \mathit{Start}(u), \mathit{Emit}(x)) = \frac{\sum_{v \in V} B_{u,v,x} \cdot e_{v,d}}{C_{u,x}} \tag{3.5}$$

To compute the values of $B_{u,v,x}$, we proceed along the trie representing the set H_* and compute the values for x from the values for its parent. At the root of the tree, $B_{u,u,\lambda} = 1$, while $B_{u,v,\lambda} = 0$ for all $u \neq v$. Now, suppose we are to consider the node for the string $x = y \circ d$, where $d \in D$, and we have already computed the value of $B_{u,v,y}$. We can obtain $B_{u,v,x}$ by the following simple sum:

$$\begin{aligned} B_{u,v,x} &= \Pr(\mathit{Emit}(x), \mathit{State}(|x|, v) \mid \mathit{Start}(u)) \\ &= \sum_{w \in V} \Pr(\mathit{Emit}(y), \mathit{State}(|y|, w) \mid \mathit{Start}(u)) \cdot e_{w,d} \cdot a_{w,v} \\ &= \sum_{w \in V} B_{u,w,y} \cdot e_{w,d} \cdot a_{w,v} \end{aligned}$$

Therefore, at each node, we spend $O(|V|)$ time to compute the value $B_{u,v,x}$, and we do this for all combinations of $u, v \in V$, so the overall running time of this process is $O(|H_*| \cdot |V|^3)$. Clearly, this time is also sufficient to compute all necessary values of $C_{u,x}$ and p_v. Finally, to compute values q_d using Equation (3.5) for all u, x, and d, we need $O(|H_*| \cdot |V|^2 \cdot |D|)$ time.

Once all these values are precomputed, every subproblem of the algorithm can be solved in $O(|V| + |D|)$ time, since we need $O(|V|)$ time in case (C) and $O(|D|)$ time in case (D). This completes the proof of the theorem. $\square$

The memory required for the algorithm does not depend on the alignment length, N, because for each subproblem $P_Q(i, x, u)$, we will only need to access subproblems $P_Q(i', x', u')$ for which i' is at least $i - M$. Therefore, we only need to keep the values for the last $M + 1$ values of i. In addition to the $O(|H_*| \cdot M \cdot |V|)$ memory needed to store the intermediate values of P_Q, we also need $O(|H_*| \cdot |V| \cdot K)$ memory to store the probabilities computed in the preprocessing.

Note that the running time and the memory size depends on the size of the set H_*, which can be exponential in the length of the seed. The size of H_* is bounded from above by the number of all sequences of length at most M, which is $O(D^M)$. For many practical seeds, the size if H_* is actually smaller. For example, for a spaced seed with k zeroes, the size of H_* is at most $M2^{k+1} + 2$, because the set of hits H contains exactly the sequences obtained by replacing an arbitrary subset of the seed's zeroes by ones. Thus, the size of H is 2^k, and the size of its prefix set, H_P, is at most $M2^k + 1$, while every element x of $H_* \setminus H_P$ can be written as $y \circ 0$ for some $y \in H_P$. If we used BLAST consecutive seed of length M, then H_* would have exactly $2M + 1$ elements, while the BLAT seed of length M and threshold $M - q$ has $O(M^{q+1})$ elements in H_*.

In practice, the algorithm proved quite efficient in our experiments. We were able to evaluate the theoretical sensitivity of all spaced seeds with weight 10 and length at most 18 in alignments of length 195 generated by an HMM with 32 states in approximately four hours, on a 2.4 GHz Pentium IV workstation.

3.4.1 Algorithm extensions

The algorithm for computing seed sensitivity can be extended in several ways. First, we can consider multiple hits of a seed. Here, to start the extension phase of the alignment, we require that the alignment contains at least p hits at least ℓ positions apart, where ℓ is at least M to guarantee that the hits are non-overlapping. Multiple hits of a consecutive seed are used in gapped BLAST, by Altschul *et al.* [9]. PatternHunter [116] also supports multiple hits to a spaced seed. In both contexts, two hits of a seed lead to fewer false positives than its one-hit counterpart with comparable sensitivity [9, 116].

We can compute the probability of multiple non-overlapping hits of a seed at least ℓ positions apart in a random alignment sequence by a simple extension of our algorithm. We will keep matrices $P_{Q,a}$, where $P_{Q,a}(i, x, u)$ is the probability of at least a hits in a sequence of length i starting with x and with first state u. The recurrence is the same as before, except for the following modifications. First, $P_{Q,1}(i, x, u) = P_Q(i, x, u)$ as before. Second, if x is a guaranteed hit and $a > 1$, then we need $a - 1$ hits in the rest of the sequence, which leads to the following formula:

$$P_{Q,a}(i, x, u) = \sum_{v \in V} r_v \cdot P_{Q,a-1}(i - \ell, \lambda, v).$$

In this formula, r_v is the probability that in step ℓ the model will be in state v, that is,

$$\begin{aligned} r_v &= \Pr(\mathit{State}(\ell, v) \mid \mathit{Start}(u), \mathit{Emit}(x)) \\ &= \sum_{w \in V} \Pr(\mathit{State}(|x|, w) \mid \mathit{Start}(u), \mathit{Emit}(x)) \cdot \Pr(\mathit{State}(\ell - |x|, v) \mid \mathit{Start}(w)) \\ &= \sum_{w \in V} \frac{B_{u,w,x}}{C_{u,x}} \cdot \Pr(\mathit{State}(\ell - |x|, v) \mid \mathit{Start}(w)). \end{aligned}$$

Values of r_v can be easily computed in the pre-processing stage. First, we compute the probability $\Pr(\mathit{State}(k, v) \mid \mathit{Start}(u))$ for all states u and v and for all distances $k \leq \ell$. This can be done in $O(|V|^3\ell)$ time. Computation of r_v is then done in $O(|V|)$ time for every $u, v \in V$ and $x \in H_G$. The number of subproblems of the dynamic programming algorithm grows by a factor of p, where p is the number of required hits. Each subproblem still takes the same time to compute.

We can also generalize the algorithm to compute the probability that at least one seed in a collection of seeds $\mathcal{Q}$ has a hit in an alignment sampled from the model. This is achieved by changing the definition of the set of hits H to include hits to any seed in $\mathcal{Q}$. Sets H_P, H_G and H_* are changed accordingly. During the trie construction, shown in the proof of Theorem 15, we need to compute dot products for prefixes of all seeds from $\mathcal{Q}$ in each node, and then determine if any of them guarantees, or at least makes possible, the hit of the corresponding seed. This slows down preprocessing by a factor of $|\mathcal{Q}|$. Of course, the set H_* may also grow by adding multiple seeds. The rest of the algorithm remains unchanged.

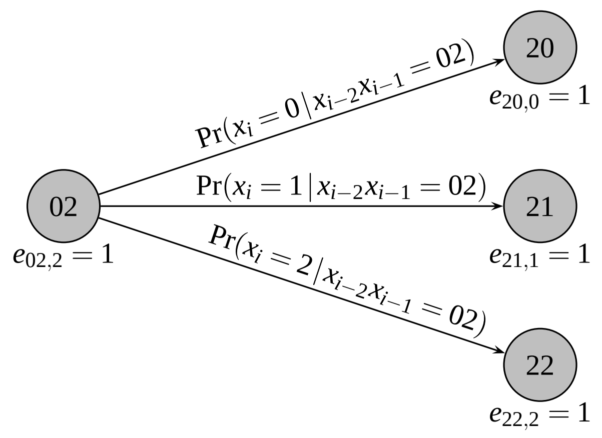
Finally, in our models of homologous coding regions, the length of the region can be a random variable sampled from a fixed probability distribution. To accommodate such models, we need to compute the probability of a hit $\Pr(\mathit{Hit}(N, Q))$ for all possible alignment lengths N from 1 to N^*, where N^* is the maximum length with non-zero probability. Luckily, when we run our algorithm for alignments of length N^*, we obtain also values $P_Q(N, \lambda, u)$ for all $N \leq N^*$. From these values we can compute the hit probabilities for all values of N in $O(N^* \cdot |V|)$ time, using Equation (3.2). Hence, the running time of computing the sensitivity of a seed under a variable length model is asymptotically the same as the running time of computing the sensitivity under the model, for fixed length N^*.

3.4.2 Related algorithms

Our dynamic programming algorithm is an extension of the algorithm by Keich *et al.* [91] for computing the sensitivity of a single spaced seed under the Bernoulli model. The trie construction from Theorem 15 was inspired by the algorithm by Li *et al.* [110] for computing the sensitivity of a collection of seeds under the Bernoulli model. It was not presented in our original papers [29, 30]. Instead, we originally represented elements of H_* as integers, and assumed they can manipulated in constant time, which was true in the cases we have considered. Even then, the preprocessing was slower by a factor of M.

The algorithm by Li *et al.* [110] works in $O((|\mathcal{Q}| + L) \cdot |H_P \setminus H_G|)$ time. Our algorithm, if run for a set of spaced seeds under the Bernoulli model, has the same running time, as in that case $D = \{0, 1\}$ and $|V| = 1$, and $|H_*| - O(|H_P \setminus H_G|)$.

Our algorithm is more complex than the algorithm for the Bernoulli model since the hidden Markov model introduces dependencies between states and sequences that are not present in the



20
$e_{20,0} = 1$
$\Pr(x_i = 0 \mid x_{i-2}x_{i-1} = 02)$
02
$e_{02,2} = 1$
$\Pr(x_i = 1 \mid x_{i-2}x_{i-1} = 02)$
21
$e_{21,1} = 1$
$\Pr(x_i = 2 \mid x_{i-2}x_{i-1} = 02)$
22
$e_{22,2} = 1$

Figure 3.11: A hidden Markov model equivalent to a Markov chain of order k over alphabet D has D^k states, one for each combination of k characters. State $d_1 d_2 \dots d_k$ always emits d_k and represents the situation when $d_1 d_2 \dots d_k$ are the k most recently emitted characters. There is a transition from state $d_1 d_2 \dots d_k$ to state $d_2 \dots d_k d$ with probability $\Pr(x_i = d \mid x_{i-1..i-k} = d_1 d_2 \dots d_k)$ defined by the Markov chain. All other transitions have zero probability. The figure shows outgoing transitions from a state 02 in the hidden Markov model corresponding to a Markov chain of order 2 over alphabet $\{0, 1, 2\}$.

Bernoulli model. For example, case (C) of the algorithm for the Bernoulli model is simply

$$P_Q(i, x) = P_Q(i - |y|, z, v),$$

whereas in our algorithm it is

$$P_Q(i, x, u) = \sum_{v \in V} \Pr(\mathit{State}(|y|, v) \mid \mathit{Start}(u), \mathit{Emit}(x)) \cdot P_Q(i - |y|, z, v).$$

Therefore, our algorithm requires computation of various probabilities in the pre-processing stage. Construction of the trie, and thus of the sets H_P, H_Q, and H_*, is also modified in our algorithm, to handle the definition of hit of a vector seed using the dot product.

Buhler *et al.* [36] consider alignments generated by a Markov chain instead of a hidden Markov model. In a Markov chain of order k, the probability of each character depends on the k previously generated characters, as for a state of order k of a generalized HMM discussed in Section 1.2.3. A Markov chain of order k over alphabet D can be represented as a hidden Markov model of with D^k states, as shown in Figure 3.11. Therefore, we can directly apply our algorithm, resulting in an algorithm with $O(|H_*| \cdot |D|^{3k} + |H_*| \cdot |D|^{2k+1} \cdot N)$ running time for a Markov chain of order k. The running time can be further greatly reduced by observing that many probabilities considered in our algorithm are always zero in the case of Markov chains. For example, $P_Q(i, x, u)$ is zero, unless sequence x is a prefix of the sequence corresponding to the state u, or vice versa.

The algorithm by Buhler *et al.* [36] computes the sensitivity of a spaced seed in an alignment of length N sampled from a Markov chain of order k in $O(|H_P \setminus H_G| \cdot N \cdot 2^k)$ time. They represent the set H_P as a deterministic finite automaton (similar to the trie used in Theorem 15), and can further reduce its size by constructing a smaller equivalent automaton. In their experiments, size of the automaton, and as well as the overall running time, is typically reduced 3 to 30 times by this procedure.

Choi *et al.* [48] also give an algorithm for computing the sensitivity of a spaced seed under the Bernoulli model. Their algorithm runs in $O(N \cdot M \cdot 2^{2(M-W)})$ time, where M is the length of the seed and W its weight. Since $|H_P| = O(M \cdot 2^{M-W})$, this algorithm is asymptotically worse than the algorithms by Keich *et al.* and Buhler *et al.* [36, 91].

Subsequently to our work, Kucherov *et al.* [103] extended the algorithm by Buhler *et al.* [36] to a very general problem characterized by three finite automata. The first automaton characterizes the set of all alignments that have a hit of a seed. Therefore, the technique can be applied to spaced seeds, vector seeds, collections of seeds, and other natural hit definitions. The second automaton characterizes the set of allowed alignments. In the simplest case, it can simply enforce a condition that the length of an alignment is a given constant. We might also consider more restrictive classes of alignments, for example those that have a certain overall score. Finally, the last automaton defines a probability distribution over alignments. This automaton is equivalent to a hidden Markov model of the type we consider. While the running time of our algorithm is cubic in the number of states of the HMM, the running time of their algorithm is only quadratic.

3.5 Experiments

In this section, we demonstrate that our models of homologous coding regions introduced in Section 3.3 are good predictors of seed sensitivity on real data. Then, we select the spaced seeds with the highest predicted sensitivity and compare their performance with several other programs for finding homologous coding regions. We will mostly concentrate on spaced seeds, since they are supported by the PatternHunter software [19], and can be thus easily applied in practice, but at the end of the section we will also demonstrate the improvements we can obtain by using vector seeds instead.

We have carefully constructed the data sets used in these experiments. Our goal was to create a set of biologically meaningful alignments of protein coding regions. Therefore we started with a set of very strong protein alignments and mapped them back to alignments of their coding regions. Some of the resulting nucleotide alignments have relatively low conservation rate and present a challenge for homology search. Yet, since they come from a very significant protein alignment, they are likely to correspond to a true homology.

3.5.1 Data sets and models

We have conducted our experiments on two separate data sets of protein coding region alignments: human *vs.* fruit fly and human *vs.* mouse. To construct both data sets, we first found statistically significant inter-species protein alignments. Then, we filtered these so that each protein occurs in at most one alignment. Next, we split the resulting sets into testing and training halves.

We then mapped the protein sequences to their coding sequences in the genomes. In this way, we transformed the alignments of pairs of proteins into alignments of pairs of DNA sequences. Note that one protein alignment can yield several DNA alignments because the coding regions for each protein can be interrupted by non-coding introns (see Figure 3.12). We call the DNA alignments in this set *fragments*. We discarded weakly conserved and very short fragments because they cannot be detected by local alignment programs using the spaced seed method.

Finally, note that some alignment fragments contain gaps. However, to find a fragment by an

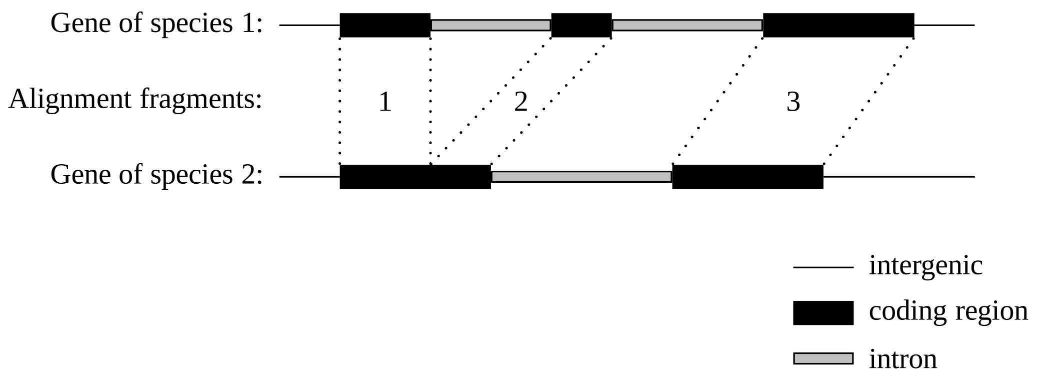


Figure 3.12: One protein alignment may correspond to several shorter alignments of protein coding regions, called fragments. The figure shows hypothetical homologous genes from two organisms. A single alignment of the two proteins encoded by these genes can be mapped in this case to three alignment fragments of the corresponding coding regions.

alignment program using the spaced seed method, the seed hit must be within an ungapped region. Thus, we broke gapped fragments in the training set into *ungapped* fragments, and again discarded weak and short fragments. Since a single hit to an ungapped fragment is sufficient to discover the entire gapped fragment by the spaced seed method, in our testing set we consider the number of gapped fragments that have a seed hit in at least one of their ungapped regions.

By this process, we have ensured that our data sets contain biologically meaningful nucleotide alignments of protein coding regions. Only the coding regions of related proteins are aligned, codon boundaries are always correctly aligned, and alignments do not extend to non-coding parts of the genome.

The initial data set consisted of all human, fruit fly (*Drosophila melanogaster*) and mouse proteins from the SwissProt database [21], release 40.38 from December 2002, for which a correctly annotated coding regions could be found in the GenBank database as of January 2003. The initial alignments were created by protein BLAST 2.0.8 [8] keeping only alignments expected to occur less than 10^{-30} times in a random database, as reported by BLAST's E-value statistics. The resulting set contained 339 alignments between human and fruit fly proteins and 675 alignments between human and mouse proteins. In the final data sets, we filtered out gapped fragments with alignment score less than 16, using the scoring scheme that scores each match by $+1$, mismatch by -1, gap opening by -5, and gap extension by -1. We have also filtered out ungapped fragments with less than ten matches. We show the sizes and mean alignment lengths in both data sets in Table 3.6.

We used the training set of ungapped fragments to estimate the parameters of our homologous coding regions models $M^{(3)}$, $M^{(8)}$, $M^{(4\times 8)}$. We have also estimated the single parameter of the simple Bernoulli model with uniform mutation rate at each position, which we will call $M^{(1)}$ in this section. We estimated the parameters of $M^{(1)}$, $M^{(3)}$, and $M^{(8)}$ by counting frequencies of corresponding conservation patterns, while we estimated the parameters of the $M^{(4\times 8)}$ model by the Baum-Welch algorithm [16, 135]. We estimated the length distribution of alignments from the set of ungapped fragments as well.

We use these models to estimate the probability of a hit of individual spaced and vector seeds in a random alignment. The false positive rate of a seed is estimated using the background model with match probability 1/4 and length equal to the length of the seed. We use the algorithm

Table 3.6: Parameters of the data sets. For each pair of species we have a training set containing ungapped alignment fragments and a testing set containing alignment fragments that may contain gaps.

	Training set (ungapped)		Testing set (gapped)	
Data set	$n =$	mean length	$n =$	mean length
Human/fruit fly	972	104	810	138
Human/mouse	2171	120	1660	152

from Section 3.4 to compute both sensitivity and false positive rate of a seed. To evaluate seed performance, we have computed how many gapped alignments from the testing set contain a hit of each seed and can be thus potentially found by an alignment program using this seed.

3.5.2 Our models as predictors of seed performance

In Section 3.3, we have described three models capturing properties of sequence conservation patterns in protein coding regions. In this section, we will study how well the sensitivity predicted under each of the models corresponds to the sensitivity measured on the testing data set. Good models should assign higher probability to seeds which perform better in practice, and the probability predicted by the model should correspond to sensitivity on real data. We use data obtained from alignments between human and fruit fly.

Figure 3.13 compares predicted and real sensitivity of all 24,310 seeds of weight 10 and length at most 18. The predicted probability increases with the sensitivity of the seed on testing data in models $M^{(3)}$, $M^{(8)}$, and $M^{(4\times 8)}$. This is in contrast to the simple Bernoulli model $M^{(1)}$ where there is no clear correspondence between predicted and real sensitivity. Models $M^{(4\times 8)}$ and $M^{(8)}$ exhibit better quality of ordering among the top seeds as well as among the worst seeds. In addition to that, $M^{(4\times 8)}$ is clearly the best predictor of the real sensitivity; in fact, the correlation between the estimated sensitivity by $M^{(4\times 8)}$ and the actual sensitivity is the highest, at $r^2 = 0.963$. Sensitivity is consistently underpredicted in all models, since the training set consists of ungapped fragments, which are slightly shorter than the gapped fragments in the testing set.

Table 3.7 further demonstrates the ordering capabilities of each of the models. The two seeds performing best on the testing data are correctly identified by the $M^{(4\times 8)}$ model. The $M^{(8)}$ models also ranks them in top two, but in the reversed order. Note that these seeds happen to be mirror images of each other, so in real applications where the genes can occur on both strands, they are expected to perform approximately the same.

The two seeds classified as the best in $M^{(3)}$ model have ranks 3 and 4 on the testing data. On the other hand, there is no clear correspondence between ranks in the $M^{(1)}$ model and ranks on the testing data. The consecutive BLAST seed is among the worst seeds in any of the considered models, and actually performs near the worst on the data.

3.5.3 Optimal spaced seeds for homologous coding regions

We have shown in the previous section that our probabilistic models are good predictors of seed performance. Here, we concentrate on the best seeds of weight 10 under individual models and

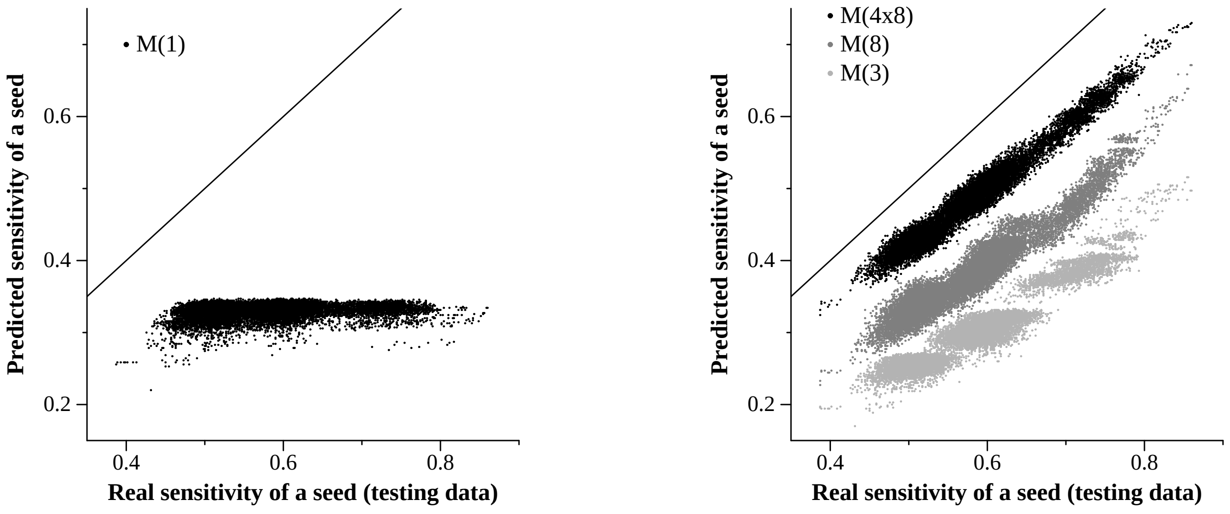


Figure 3.13: Sensitivity of all spaced seeds of weight 10 and length at most 18 on the testing set versus their predicted sensitivity under different probabilistic models trained on the set of human/fruit fly alignments. Each dot represents one seed under one model. The diagonal line is the graph of the function $y = x$, representing the ideal predictor. Our models, shown in the right plot, exhibit a much stronger correlation between predicted and actual sensitivity than the simple Bernoulli model shown in the left plot.

Table 3.7: The table shows the ranks of several selected seeds in the testing data set as well as under each of the considered probabilistic models trained on the set of human/fruit fly alignments. Best and worst ranks are highlighted.

	Rank				
Seed	Testing data	$M^{(4\times8)}$	$M^{(8)}$	$M^{(3)}$	$M^{(1)}$
11011000011011011	**1**	**1**	2	17	9746
11011011000011011	2	2	**1**	16	9746
11011001011001011	3	5	5	**1**	17990
11001011001011011	4	4	6	2	17945
11011011011000011	4	6	4	43	19124
111001001001010111	8120	10426	10350	3253	**1**
111010100100100111	13164	8638	8175	4426	**1**
1111111111	24295	24285	24233	**24310**	**24310**
101010101010101011	**24309**	**24310**	**24310**	24298	24306
110101010101010101	**24309**	24309	24309	24296	24306

Table 3.8: Performance of selected seeds on both testing sets. Columns labeled Hit indicate how many fragments in the testing set have a hit of the seed. Columns labeled PH indicate how many of these fragments overlap an alignment discovered by the PatternHunter program using the seed.

		Fruit fly		Mouse	
Seed	Name	Hit	PH	Hit	PH
110 110 000 110 110 11	DATA-OPT	86%	85%	92%	92%
110 110 110 000 110 11	M8-OPT	86%	85%	92%	92%
110 110 110 110 11	WABA	80%	79%	90%	90%
111001001001010111	M1-OPT	60%	57%	86%	86%
1111111111	BLAST	43%	43%	81%	81%
101010101010101011	WORST	39%	39%	79%	79%

compare their performance in practice with existing approaches.

Table 3.8 lists the seeds we selected for further examination. For each seed we show its sensitivity on both testing sets (human *vs.* fruit fly and human *vs.* mouse) computed as the fraction of alignments that have a hit of the seed. We have also run the PatternHunter local alignment program [116] on the exonic sequences in the testing set, obtained by masking non-coding regions to letter N. In the table, we list the fraction of testing fragments overlapping an alignment reported by PatternHunter.

The seed DATA-OPT has the highest sensitivity on both testing sets. It is also the most sensitive seed under the $M^{(4\times 8)}$ models trained on both training sets. The next seed, M8-OPT, has rank 2 on both testing sets. It is the optimal seed under the $M^{(8)}$ model trained on both training set and under $M^{(3)}$ trained for alignments between human and mouse.

As we can see, these two seeds consist of triples `110`, and `000`, reflecting the fact that the last position of a codon is least conserved. With this in mind, Kent and Zahler [93] developed the seed `110110110` for their alignment program WABA, before spaced seeds were introduced in PatternHunter. We list the performance of WABA seed of weight 10 in the table.

The seed M1-OPT is optimal under the $M^{(1)}$ model with parameters estimated on our training set. Note that it is different from the PatternHunter seed `111010110001101l` of weight 10 optimized for the Bernoulli model with match probability 0.7 and length of alignment 64, since the match probabilities differ, and the alignment length is random. Seed M1-OPT and the consecutive seed used in nucleotide BLAST do not reflect codon structure, and as a result have much lower sensitivity. We also include the seed globally worst on the testing set.

In both data sets, the seeds that take into account the three-periodic structure of the coding regions (DATA-OPT, M8-OPT, and WABA), have higher sensitivity than other seeds. Further, the optimal seeds under models $M^{(4\times 8)}$ and $M^{(8)}$ have significantly better performance than the WABA seed. The DATA-OPT seed reduces the number of alignments without a hit by 20% compared to the WABA seed in the fruit fly set and by 15% in the mouse set. This increase in sensitivity does not come at a cost of greatly increased running time. The running time of PatternHunter increased by at most 3% compared to the M1-OPT seed.

Note that there is much less difference between the best and the worst seeds on the mouse set than on the fruit fly set, since alignments between human and mouse are much more conserved,

Table 3.9: Sensitivity and running time of various methods for aligning homologous coding sequences. The DATA-OPT and M8-18-8 rows correspond to using spaced seeds with PatternHunter; the other rows correspond to other programs. The M8-18-8 seed is a weight 8 seed that offers sensitivity comparable to or superior to TBLASTX with vastly reduced running times.

	Fruit fly		Mouse	
Seed	Sensitivity	time (s)	Sensitivity	time (s)
DATA-OPT	84.7%	14.0	91.6%	21.9
TBLASTX	94.7%	123.7	93.6%	891.1
BLAT	58.1%	9.0	80.7%	12.6
M8-18-8	94.6%	19.9	97.2%	37.3

and are thus easier to detect by any seed. Still, there is a clear separation between performance of the seeds tailored to detecting alignments in coding regions and other seeds.

As we have discussed in Section 3.3, similar proteins can be encoded by very different nucleotide sequences due to redundancy of the genetic code. Our spaced seeds increase the sensitivity by concentrating on those codon positions that are most likely to be conserved. Still, we use these seeds in a general nucleotide aligner PatternHunter which takes into account only nucleotide matches and mismatches, and may discard some alignments that are weak on nucleotide level and strong on the protein level. Some programs avoid this problem by translating the nucleotide sequence to protein sequence in all six reading frame and applying protein homology search. We have compared the performance of PatternHunter using our optimal seed DATA-OPT with two such programs: TBLASTX from the BLAST suite [8] and translated BLAT [92].

Table 3.9 summarizes the running time and sensitivity of individual programs on the sequences with non-coding regions masked in our data sets from human and mouse and from human and fruit fly. The running time was measured on 1.4GHz Intel Xeon processor.

Here, we see that although TBLASTX is more sensitive than our seed, it is very slow: TBLASTX required fifteen minutes to align mouse and human exonic sequences, while our program required 22 seconds. To see if a more sensitive seed might still beat TBLASTX, we computed the optimal seed for model $M^{(8)}$ with eight ones in a seed length of at most eighteen. In Table 3.9, we call this seed M8-18-8. Using this seed, `11000011011011`, we found 97.2% of the mouse fragments, with a running time of 37 seconds. This is still 24 times faster than TBLASTX, yet has fewer than half as many false negatives. While this seed does generate more false positives, as detected in the slower running time, the running time is only two times higher than the running time for the weight 10 seeds.

In experiments on the full genomic regions containing the exons, results were also not comparable: TBLASTX required 7.26 hours (26141 s) to align these regions, while PatternHunter using the DATA-OPT seed required 23 minutes (1394 s). Using the more sensitive weight 8 seed, the alignment still required only two hours (7219 s).

BLAT also fared substantially less well than our alignment method. At its default seed length setting of five amino acids, BLAT found 80.7% of alignments, about 1.73 times faster than our method, which found 92.5% of them. BLAT did not have a false positive rate comparable to our method until we reduced the seed length to three amino acids; then, it took almost three hundred

Weight vector	T	Support	Name	Actual sensitivity	Predicted sensitivity	False positives
11011000011011011	10	10	DATA-OPT	86.0%	67.2%	$9.5 \cdot 10^{-7}$
111101100110101111	12	13	VS-12-13	74.7%	54.0%	$6.0 \cdot 10^{-7}$
11011001011000011011011	12	13	CVS-12-13	91.1%	74.4%	$6.0 \cdot 10^{-7}$
1111110110111101111	13	15	VS-13-15	83.8%	63.1%	$9.2 \cdot 10^{-7}$
11011011000011011001011011	13	15	CVS-13-15	96.0%	80.9%	$9.2 \cdot 10^{-7}$
12011011000011011001011011	14	15	CVS2-14-15	95.6%	80.1%	$8.0 \cdot 10^{-7}$
12011011000011011001011012	15	15	CVS2-15-15	94.8%	79.2%	$7.0 \cdot 10^{-7}$
12011012000011011001011012	16	15	CVS2-16-15	94.1%	78.1%	$6.0 \cdot 10^{-7}$
12011012000011011001012012	17	15	CVS2-17-15	93.2%	76.9%	$5.0 \cdot 10^{-7}$
12011012000012011001012012	18	15	CVS2-18-15	92.7%	75.4%	$4.2 \cdot 10^{-7}$
12012012000012011001012012	19	15	CVS2-19-15	91.6%	73.7%	$3.4 \cdot 10^{-7}$
12012012000012012001012012	20	15	CVS2-20-15	90.7%	71.7%	$2.8 \cdot 10^{-7}$
12012012000012012002012012	21	15	CVS2-21-15	88.8%	68.4%	$2.1 \cdot 10^{-7}$

Table 3.10: Performance of vector seeds on homologous coding regions. VS-T-S are spaced seeds of support S and threshold T optimized for the simple Bernoulli model, as shown in Table 3.10. CVS-T-S are vector seeds optimized for the $M^{(8)}$ model trained on alignments between human and fruit fly. The seed CVS-13-15 has very high sensitivity, both predicted and actual. CVS2-T-S are variants of this seed have weight 2 on some positions. They allows us to set the false positive rate at a finer scale.

times as long to do its alignments, while still only finding 90.7% of alignments. However, we note that BLAT also attempts to stitch alignments together across introns, which goes beyond the problem of identifying homologous coding alignments we consider.

Our experiments show that PatternHunter, using our optimal seeds, is substantially more effective than existing sequence alignment packages tailored to align homologous coding regions, offering greater sensitivity and much lower running time.

3.5.4 Vector seeds for homologous coding regions

In Section 3.2, we have introduced vector seeds and shown that they improve predicted sensitivity under the simple Bernoulli model compared to both spaced seed and BLAT seeds allowing a fixed number of mismatches. In this section, we will apply vector seeds to the search of homologous coding regions.

We have considered vector seeds over binary alignment sequence, with one representing a match and zero a mismatch. First we have investigated 1372 binary vector seeds (v, T) with support s between 10 and 15 and threshold T between s and $s-2$. The seeds we have investigated have codon structure: they can be divided into triplets, where each triplet is either $(0, 1, 0)$, $(1, 1, 0)$, or $(0, 0, 0)$, since these triples were the building blocks of the best coding spaced seeds in Table 3.7.

We have used the $M^{(8)}$ model trained on the set of alignments between human and fruit fly to select best vector seeds. We have also computed their actual sensitivity, defined as the fraction of gapped alignments in the testing set containing a hit. Results for some interesting seeds are shown in Table 3.10.

We have seen in Table 3.2, the vector seed VS-13-15, with support 15 and allowing 2 mismatches, has significantly higher sensitivity than any spaced seed or BLAT seed of comparable false positive rate. Its coding counterpart, CVS-13-15, improves actual sensitivity on our test set to 96%, compared to 86% achieved by the best coding spaced seed. Meanwhile, the sensitivity of VS-13-15, which was not optimized for coding regions is only 83%. One disadvantage of this seed is its high support, which requires hash tables with 4^{15} entries. Seed CVS-12-13 with support 13 has sensitivity 91.1%, also higher than the spaced seed.

We have also explored the effect of using non-binary weights in vector seed. In particular, we have considered all seeds obtained from the seed CVS-13-15 by increasing the weight of the middle positions of some of codons to two and raising the threshold by one. A hit of such a seed can have either one mismatch from the positions with weight two or at most two mismatches from the positions with weight one. Thus the false positive rate as well as sensitivity decreases with increasing number of twos. For each level of false positive rate we report seed with the highest predicted sensitivity in Table 3.10. The seed CVS2-21-15, which has weight 0 or 2 at all middle codon positions, has sensitivity slightly above the sensitivity of the spaced seed DATA-OPT. Its false positive rate is less than one quarter of the false positive rate of seeds CVS-13-15 and DATA-OPT. This experiment shows that simple changes in weights can achieve finer control over false positive rates.

3.6 Related work

The optimized spaced seeds for homology search were introduced by Ma *et al.* in 2002 [116]. Previously, collections of random spaced seeds were used in homology search by Califano and Rigoutsos [43] and, for very long seeds, by Buhler [35]. Spaced seeds can also be used for lossless filtration, where the goal is to find all alignments of a certain length and similarity level with 100% sensitivity. Such schemes were studied for example by Burkhardt and Kärkkäinen [39] and Kucherov *et al.* [102]. Beyond sequence similarity search, Preparata *et al.* [132] apply a specially constructed family of spaced probes to the problem of sequencing by hybridization and Kucherov *et al.* [102] design spaced probes for expression arrays.

In this section, we briefly survey recent developments in the area of spaced seeds for heuristic homology search. Much of this work appeared within a very short time, and different authors often worked in parallel on similar approaches. An alternative treatment of the topic can be found in the survey by Brown *et al.* [34].

Spaced seeds were quickly adopted in several homology search programs, including PatternHunter [116, 110], megaBLAST, from the BLAST suite [119], BLASTZ [147], and YASS [124]. They were also used by Brown and Hudek [34] to construct multiple sequence alignments and by Flannick and Batzoglou [63] to align a sequence to an existing multiple sequence alignment.

3.6.1 Theoretical properties of seeds

Although we can evaluate the sensitivity of individual seeds under a particular probabilistic model of alignments using algorithms such as the one presented in Section 3.4, we would also like to characterize properties of a seed more generally, across ranges of possible model parameters, such as alignment length and conservation level.

Keich *et al.* [91] prove that in some sense the consecutive seed $\tilde{Q}$ is worse than any other seed Q of the same weight in an infinitely long alignment, sampled from the simple Bernoulli model, at any conservation level. In particular, the leftmost end of the first hit of Q is more likely to occur within the first n positions of the alignment than the first hit of $\tilde{Q}$. Also, the expected value of the leftmost end of the first hit of Q is smaller than the expected value of the leftmost end of the first hit of $\tilde{Q}$. However, since $\tilde{Q}$ is shorter than Q, its rightmost end may occur sooner, even if its leftmost end appears later.

In alignments of finite length sampled from the Bernoulli model, some seeds perform even worse than the consecutive seed $\tilde{Q}$. Choi *et al.* [48] and Buhler *et al.* [36] notice that periodic seeds are less sensitive than the BLAST seed of the same weight in alignments of any length and any conservation level. A periodic seed is one that has ones at equally spaced positions $0, k, 2k, \ldots,$ for example `10101` or `1001001`.

Buhler *et al.* [36] study how the sensitivity of a seed grows as the length of an alignment sampled from a Bernoulli model goes to infinity. If s_N is the sensitivity of a particular seed in an alignment of length N, the following holds

$$\lim_{N\to\infty} \frac{1 - s_N}{\beta \cdot \lambda^N} = 1,$$

where β and λ are positive constants not depending on N. Therefore, we can characterize the asymptotic performance of a seed Q by its λ_Q and β_Q constants. This characterization defines a linear order on the set of all seeds. Buhler *et al.* use this general theory to prove that the value of $\lambda_{\tilde{Q}}$ of the BLAST consecutive seed $\tilde{Q}$ is at least as big as λ_Q is for any other seed Q. In addition, they construct a seed with λ_Q strictly lower than $\lambda_{\tilde{Q}}$. This proves that the BLAST seed is not asymptotically optimal.

Although this asymptotic characterization of seed performance is quite elegant, some questions remain open. It is not clear, for example, how good the asymptotic approximation is for realistic values of N. Also, the relationship between constants λ_Q and β_Q and the the conservation level of an alignment is not understood.

3.6.2 Generalized spaced seeds

Sensitivity of spaced seeds can be further improved by considering various variations of the hit definition. One example of this approach are the vector seeds introduced in this thesis. In this section, we will discuss several other generalizations of spaced seeds proposed in the literature.

Many authors [30, 32, 110, 157, 170] consider collections of seeds. A collection of seeds has a hit in an alignment if at least one seed in the collection has a hit. Li *et al.* [110] demonstrate that in the simple Bernoulli model of alignments, doubling the number of seeds in the collection is roughly equivalent to decreasing the weight of a single seed by one. However, doubling the collection size increases the number of false positives by a factor of two, while lowering the weight by one increases it by a factor of four. Thus, in the ideal case, the collection of seeds would lead to a 50% reduction in running time compared to a seed of lower weight. These time savings are somewhat diminished in practice by the overhead associated with creating and searching two separate hashing tables.

Csűrös and Ma [55] note that a vector seed over a binary alphabet is a special case of a seed collection. For example, the vector seed $((1, 1, 0, 1), 2)$ is equivalent to the collection of spaced seeds `11`, `101`, and `1001`. Technically, the equivalence may break near the alignment boundaries, for example if the alignment ends with two ones. We may avoid this problem by padding both ends of the alignment with a short stretch of mismatches. Sometimes the collection of spaced seeds $\mathcal{Q}$ corresponding to a vector seed (w, T) has a subset $\mathcal{Q}' \subset \mathcal{Q}$ such that $\mathcal{Q}'$ has almost as high sensitivity as $\mathcal{Q}$ but lower false positive rate. Consequently, Csűrös and Ma recommend using a well chosen subset $\mathcal{Q}' \subset \mathcal{Q}$ instead of the original vector seed (w, T). Seeds in the collection $\mathcal{Q}'$ are called daughter seeds of the seed (w, T). An advantage of daughter seeds over arbitrary collections of seeds is that all daughter seeds can share one hash table, with keys containing all positions of the parent seed with non-zero weight.

Csűrös and Ma [55] propose another improvement of vector seeds. As we have seen in Sections 3.2 and 3.5.4, some vector seeds with very high sensitivity and low positive rate are unfortunately impractical due to high memory requirements. To address this problem, Csűrös and Ma [55] propose so called relaxed seeds. A relaxed seed consists of a regular spaced seed used for hashing and a secondary vector seed. If the spaced seed has a hit, their algorithm first checks whether the same position is also a hit of the vector seed, and only triggers alignment extension if this is the case. In effect, the secondary seed specifies which positions are considered first in the extension phase, and if these positions do not have sufficiently high score, the extension is not continued. Secondary seeds significantly reduce the expected time spent in finding and extending hits compared to spaced seeds. They also decrease memory requirements compared to vector seeds, since only part of the seed is hashed.

In a completely different vein, Csűrös [54] proposes variable weight spaced seeds. In this framework, the weight of the seed depends on the composition of the string to be hashed. Rare strings, expected to produce fewer false positives, use seeds with smaller weight, which increases the sensitivity of the search.

Chen and Sung [47] introduce so called half-gapped seeds, which we have mentioned already in Section 3.2.1. In their framework, some pairs of nucleotides are designated as “neighbors”. A mismatch consisting of two aligned neighbors is called a half-match. A half-gapped seed requires matches at some positions, and either matches or half-matches at others. Chen and Sung motivate the use of their seeds simply by the desire to provide a more fine-grained control over the false negative rate, and do not prescribe how to choose neighbors for each nucleotide. However, Noé and Kucherov [123] observe that seeds of this type can be used to distinguish between transitions and transversions. Transitions are mutations between the pyrimidines C and T or between the purines A and G. They are more frequent than other mutations, called transversions [141]. Therefore we can define neighbor pairs (A,G), and (C,T), and use half-gapped seeds that allow either a match or a transition at certain positions. As we have seen in Section 3.2.1, half-gapped seeds in general, and thus also the transition-constrained seeds of Noé and Kucherov, are a special case of vector seeds. Seeds allowing transitions at some positions were also suggested earlier by Schwartz *et al.* [147].

3.6.3 Probabilistic models of alignments

One of our main contributions in this chapter is the introduction of realistic models of homologous coding regions. In parallel work, Buhler *et al.* [36] represent a coding region as a three-periodic non-homogeneous Markov chain of order 5. That is, their model is a hidden Markov model with the same topology as our $M^{(3)}$ model, but with each state of order 5. This model is more general than our $M^{(8)}$ model, because it not only captures dependencies within a codon but also between adjacent codons. It is comparable to our $M^{(4\times 8)}$ model, although not completely equivalent, as it does not characterize the regions of the alignment as having high or low conservation. One advantage of Markov chain models is that their parameters can estimated simply by counting frequencies of 6-tuples in the sequences, whereas our $M^{(4\times 8)}$ model requires the use of the iterative Baum-Welch algorithm [16].

Buhler *et al.* [36] also model alignments of non-coding regions as Markov chains of order up to 5. While more realistic models of coding regions allow better estimation of seed sensitivity and lead to optimal seeds with improved performance, Buhler *et al.* do not observe much improvement in non-coding seeds when they increase the order of the Markov chain above one.

Li *et al.* [110] use our $M^{(3)}$ model with fixed parameters $(0.8, 0.8, 0.5)$, so that the overall identity level is 0.7, as in their simple Bernoulli model. In their experiments on a set of homologous mouse and human EST sequences, seeds optimized under this model perform better than seeds optimized under our $M^{(8)}$ model. This may be caused by the presence of untranslated regions in ESTs, which are not considered in our model. Kucherov *et al.* [103] extend our $M^{(3)}$, $M^{(8)}$ and $M^{(4\times 8)}$ models to emit sequence of matches, transitions (mutations between A or G, or between C and T) and transversions (other mutations).

Hidden Markov models and Markov chains characterize local properties of alignments. Kucherov *et al.* [101] concentrate on modeling global alignment properties. In particular, alignments reported by local alignment programs are always homogeneous, which means that no partial segment of the alignment scores higher than the whole alignment. Kucherov *et al.* consider the set of all homogeneous alignments of a certain length and score, and assign equal probability to each. This model yields slightly different optimal seeds which are more sensitive on testing data than the seeds optimized for the Bernoulli model. Csűrös and Ma [55] drop the requirement of homogeneity and use the model, in which each alignment of a given length and number of matches has equal probability.

Finally, all models mentioned so far represent ungapped alignments. The use of ungapped alignment models can be justified by the fact that a single seed is always located within a single ungapped region of an alignment. In our work [30], we characterize gapped protein alignments as a sequence of ungapped fragments. The number of ungapped fragments in an alignment as well as their lengths are random variables. We use this model to estimate seed sensitivity as a probability that at least one of the ungapped fragments in a random alignment contains a hit. Noé and Kucherov [123] propose a model of gapped alignments in which each position is a gap with some small probability. Note that their model is not consistent with traditional alignment models where length of a gap is typically modeled by a geometric distribution. They use this model to estimate the probability that an alignment will have two hits of a seed in a certain distance and with a limited number of gaps separating them.

3.6.4 Complexity issues

In Section 3.4 we have described an algorithm for computing sensitivity of a vector seed under a hidden Markov model of alignments. We have also discussed several other similar algorithms from literature. The running time of all these algorithms is in the worst case exponential in the length of the seed. This raises the question whether more efficient algorithms exist, or whether the problem can be proved NP-hard. Another important algorithmic problem in this area is to find the seed with highest sensitivity out of all seeds of a given weight. Both problems can be extended to collections of seeds or other generalized seeding strategies.

Most hardness results do not consider the predicted sensitivity of a seed under a probabilistic model, but rather the explicit sensitivity, defined as the fraction of a given set of alignments that contain a hit. For example, Li *et al.* [110] prove that it is NP-hard to find the seed or collection of k seeds of a given weight and length that have maximum sensitivity on a given set of alignments. Nicolas and Rivals [122] and Brown [32] study further variants of this problem.

Predicted sensitivity was considered by Li *et al.* [110] who prove that it is NP-hard to compute the probability of a hit of a spaced seed collection in a uniformly generated random binary string, even if the string has the same length as all the seeds in the collection. Recently, Li *et al.* [111] prove that it is NP-hard even to compute a sensitivity of a single seed in a uniformly generated random binary string. Nicolas and Rivals [122] prove that it is NP-hard to determine if a single seed has a hit in all possible alignments of a fixed length and number of matches, even if the length of the alignment is encoded in unary.

In the face of these results, researchers have developed various heuristics and approximation algorithms to select good seeds. Xu *et al.* [170] study the problem of selecting a collection of k seeds from a set of seeds listed as input that will optimize the sensitivity on a given set of alignments. Their alignments are sampled from a probabilistic model, as a way to estimate the seed sensitivity under the model. They formulate the problem as an integer linear program and solve it by linear relaxation and randomized rounding techniques. Their algorithm is guaranteed to find a collection of seeds with sensitivity within a factor of $1 - 1/e \approx 0.63$ from the optimal solution. In their experiments, this algorithm finds seed collections that achieve over 90% of the sensitivity achievable by the optimal solution.

Li *et al.* [110] and Sun and Buhler [157] design collections of seeds by the greedy algorithm that in each step adds one seed to a collection so that the increase in sensitivity is as high as possible. Sun and Buhler [157] speed up this process by creating a random sample of alignments that do not have a hit to already chosen seeds. Although the sampling algorithm is slow, it allows them to evaluate the sensitivity of potential candidates much faster. Further heuristic approaches for selecting one or multiple seeds are presented, for example, by Buhler *et al.* [36], Choi *et al.* [48], and Brown [32].

3.7 Summary

In this chapter we have presented several improvements in the area of spaced seeds for homologous coding regions. We have designed realistic probabilistic models capturing properties of homologous protein coding regions to better predict seed sensitivity. We have also defined vector seeds, a new framework that unifies and further extends several previously used seed definitions. Finally, we have extended an existing algorithm for computing seed sensitivity to handle our more general probabilistic models as well as the more general vector seed definition.

Using our algorithm and models, we have obtained new seeds that significantly increase sensitivity of homology search in coding regions. We have confirmed this by experiments in two data sets: one containing alignments between human and mouse, and one containing alignments between human and fruit fly. Increased sensitivity of homology search programs with respect to coding regions is particularly useful in gene finding, where alignments of coding regions serve as a valuable source of evidence. With this goal in mind, we have tried to emulate the properties of the coding region alignments encountered in gene finding when constructing our data sets.

Although the process of selecting optimal seeds can be computationally intensive for complex probabilistic models or longer seeds, the same seeds were optimal or close to optimal in both our data sets. This suggests that a seed optimized for homology search between a particular pair of organisms would perform well for other pairs of organisms as well. Therefore once a spaced seed is optimized, it can be reused in many related homology search tasks.

We also survey many generalizations of the spaced seed framework, and associated probabilistic models and algorithms that appear in the recent literature. They provide a bewildering selection of methods for seeding homology search. Some methods, such as vector seeds or seed collections, improve sensitivity and false positive rate, but introduce additional overhead needed for more extensive hashing. Perhaps the most pressing question in this area is to provide an independent assessment and comparison of these methods.

Chapter 4

ExonHunter: a Comprehensive Eukaryotic Gene Finder

In this chapter, we present ExonHunter, a practical gene finder that combines the advisor framework introduced in Chapter 2 and a generalized HMM for gene finding described in the thesis of Vinař [164]. We will also build upon the results from Chapter 3, since some of the advisors used in ExonHunter use spaced seeds to find homologous coding regions.

First, we turn our attention to the hidden Markov model that forms the basis of ExonHunter. Compared to other such models previously used in gene finding, it introduces several novel features. These improvements are given in full detail in Vinař's thesis [164]. We describe the model briefly here for completeness.

Next, we study the problem of how to represent evidence from different sources in the form of advisors. Since most of our advisors come from comparison of the query sequence with sequence databases containing known proteins, ESTs, and other genomes, we will discuss in particular how to represent the information from a sequence alignment in the form of an advisor. We present a simple general scheme for estimating advisor parameters from training data that can be used for a variety of evidence sources. Still, for each data source, we need to select appropriate tools for finding alignments, post-process their output, and choose the label set partition best suited for advice based on this source of information.

Finally, we evaluate the overall prediction accuracy of our gene finder on several testing sets. We use testing sets for which the predictions of several gene finders are available. This allows us to compare our results to those of well established as well very recent gene finders. On our test sets, ExonHunter outperforms *ab initio* gene finders and gene finders that use only alignments with other genomes as evidence. Several recent approaches for integrating additional evidence in gene finding achieve higher accuracy, and the comparison suggests directions for improving our program. We concentrate on the design and evaluation of our gene finder for human sequences, but to demonstrate that our approach can be adapted to other species, we show results for finding genes in the genome of the fruit fly (*Drosophila melanogaster*). We also study the contribution of individual advisors to the overall accuracy.

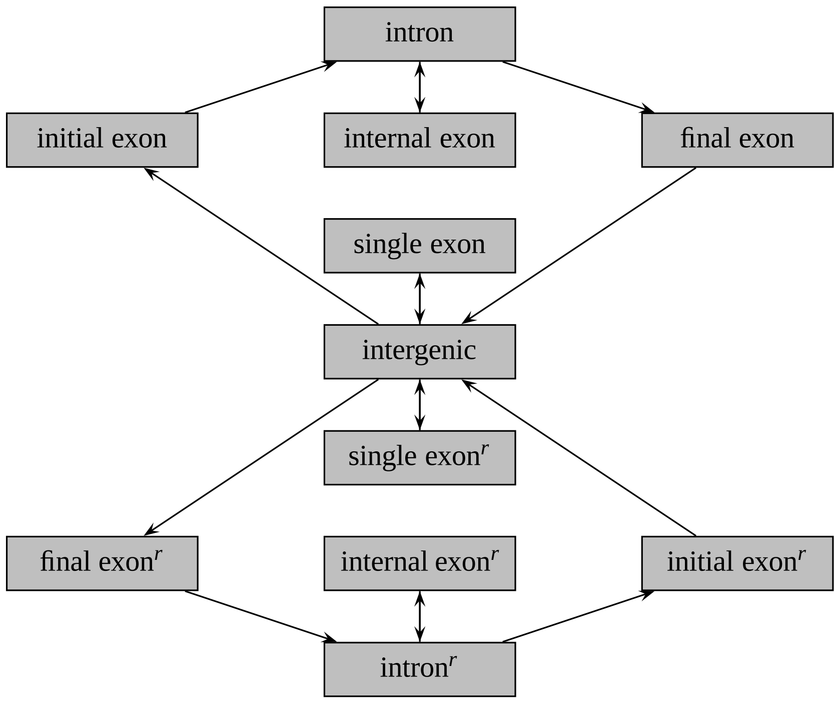


Figure 4.1: Overall scheme of the HMM used in ExonHunter. Each box represents a group of states. The upper half of the figure models a gene on the forward strand, while the bottom half models a gene on the reverse strand.

4.1 Extended HMM for gene finding

In this section, we briefly outline the hidden Markov model used in ExonHunter and several novel features that distinguish it from other similar models for gene finding. A detailed description and justification of these features can be found in Vinař's thesis [164]. We start with a brief description of the overall topology of the model and then describe its individual features in more detail.

4.1.1 Model topology

The schematic overview of the model topology is depicted in Figure 4.1, and follows what is now a usual pattern, popularized by Genie [105] and Genscan [37]. The model consists of a submodel for the intergenic region and two submodels for genes on the two DNA strands. For each strand we treat one-exon genes separately. The submodels for multi-exon genes consist of an intron model and models of initial, internal, and final exons. (Here, by exons, we understand only the protein coding exons.) We have experimented with adding simple models of UTRs between the intergenic and coding region models. However, these did not seem to increase prediction accuracy, and thus we omitted them in the final version.

The intron model on each strand consists of three identical copies which are used to keep the reading frame consistent between exons, as depicted in Figure 4.2. For example, the intron submodel 0 ensures that if an exon ends with a nucleotide in frame 0, the next exon will start with

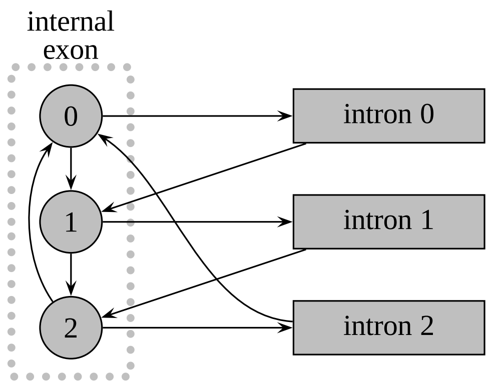


Figure 4.2: Transitions between internal exon and intron states in the HMM. Three copies of the intron submodel help to keep the reading frame consistent between adjacent exons. Shaded boxes denote submodels consisting of multiple states, dotted boxes contain states or groups of states for which the time spent in the box has an explicit length distribution, and circles represent single states and their label.

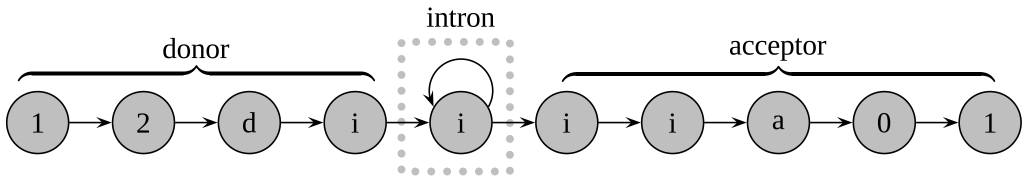


Figure 4.3: Topology of intron 0 submodel from Figure 4.2. In this picture, the models of both donor and acceptor splice site signal extend two nucleotides to coding region, therefore Figure 4.2 would have to be modified, so that the transition from the intron 0 submodel goes to the exon state in frame 2. Note that signal windows used in practice are typically longer than shown in this figure.

a nucleotide in frame 1. The figure shows only transitions between the intron model and internal exon model; transitions from the initial exon model to the intron model and from the intron model to the final exon model preserve the reading frame in the same fashion.

Each of the copies of the intron submodel consists of a submodel emitting donor splice site signal, followed by a state emitting the interior part of an intron, followed by a submodel emitting acceptor splice site signal (see Figure 4.3). Note that acceptor and donor splice site signals may overlap with the coding region. In such a case the transitions in Figure 4.2 need to be adjusted accordingly. For example, if both acceptor and donor splice site signals overlap two bases with coding region, they in total add one full codon and one nucleotide to the coding region in addition to coding nucleotides generated by the internal exon submodel. Therefore the transition from the intron 0 module is to exon frame 2 instead of 1, to account for the four coding nucleotides generated in the intron submodel. The HMM also contains a translation start signal model at the start of the first (or only) coding exon in each gene, and a stop signal model at the end of the last (or only) coding exon.

Each state of the HMM has one label from the set of labels introduced in Table 2.1. For simplicity, we require that the HMM starts and ends in the intergenic state. If this is true, then the model topology ensures that each labeling corresponds to at most one state path with non-zero probability, and thus the most probable state path found by the Viterbi algorithm corresponds to the most probable labeling. However, if we include the UTR submodels, this is no longer true, because UTRs have the same label as intergenic regions. Therefore several state paths with the same labeling but with different transcription start site or introns within UTRs may exist. The most probable labeling corresponds to several such paths, but it is possible than none of them has high enough probability to be the most probable state path detected by the Viterbi algorithm [28] (see also Section 1.2.1). This may have been a part of the reason why our simple models of UTRs do not increase prediction accuracy.

4.1.2 Length distributions

In this section, we will describe modeling of lengths of exons, introns, and intergenic regions. As we have discussed in Section 1.2.3, the length of a region generated by a single state with a self-loop transition is geometrically distributed. As the distributions of exon, intron, and intergenic region lengths observed in genomes are not geometric, generalized states with explicit duration are often used in gene finding [37, 156].

In our HMM, we use so called geometric-tail distributions. These distributions consist of a head with an arbitrary distribution, and a geometrically decaying tail, as shown in Figure 4.4. For the head of the distribution, we use a smoothed version of the distribution observed in the training data. The single parameter of the geometric distribution used in the tail is fitted to training data by maximum likelihood estimation. We have developed a modified Viterbi algorithm that finds the most probable state path in an HMM with geometric-tail state durations that runs in time $O(nt)$, where n is the length of the sequence and t is the start of the tail [31], assuming the HMM size is a constant. The parameter t allows us to balance running time and modeling accuracy: for $t = 1$ we get a geometric length distribution and linear running time, for $t = n$ we get arbitrary length distributions and quadratic running time. It is possible to achieve a reasonable approximation of exon and intron lengths for practical values of t.

Intergenic regions are typically much longer than introns and exons, and we would require very high values of t to capture the distribution adequately. Therefore we use a distribution in the form of a step-function, which consists of t blocks of length t. Within each block the probability of all lengths is a constant, and when we collapse the blocks to points, we get a geometric-tail distribution with tail starting at t (see Figure 4.5). Therefore the distribution of the first t^2 lengths is approximated by an arbitrary step function, which yields a better approximation than the geometric distribution. We have also modified the Viterbi algorithm for this scenario so that it runs in $O(nt)$ time [25].

Intron and intergenic length distributions simply give the durations of a single state, which emits the internal part of the intron or intergenic region, between the signals at their respective boundaries. However, coding regions require three-periodic structure, with separate states emitting the nucleotides in the first, second, and third position of the codon, as depicted in Figure 4.3. Luckily, it is possible to extend the model and the corresponding algorithm so that the explicit duration applies to a whole submodel [31]. The situation is a little bit more complicated in the final exon (and the initial exon on the reverse strand), which has to end exactly at the end of the codon. Therefore we modify the length distribution so that only lengths divisible by three are allowed, and add two states to properly finish exons that start in the middle of a codon (see Figure 4.6).

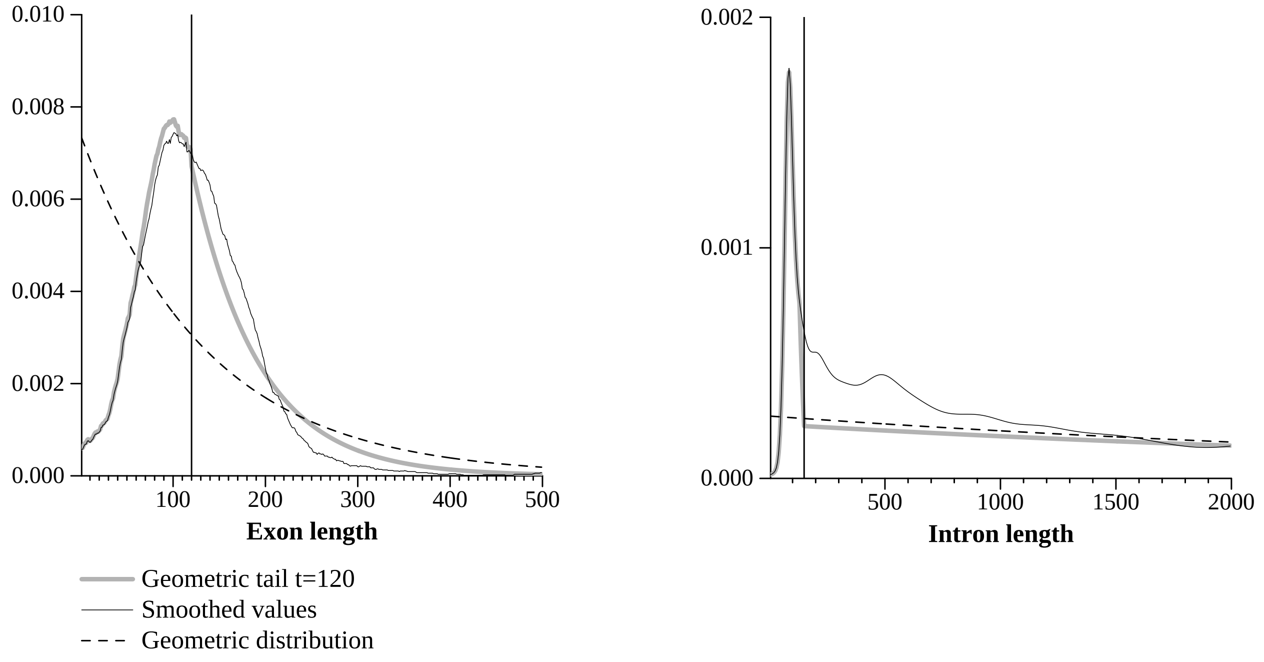


Figure 4.4: Geometric length distribution of human exons and introns from the training portion of chromosome 22 (see Section 4.2 for description of the data sets). The geometric tail starts after the vertical line. For comparison, geometric distribution and smoothed training data distribution is also shown.

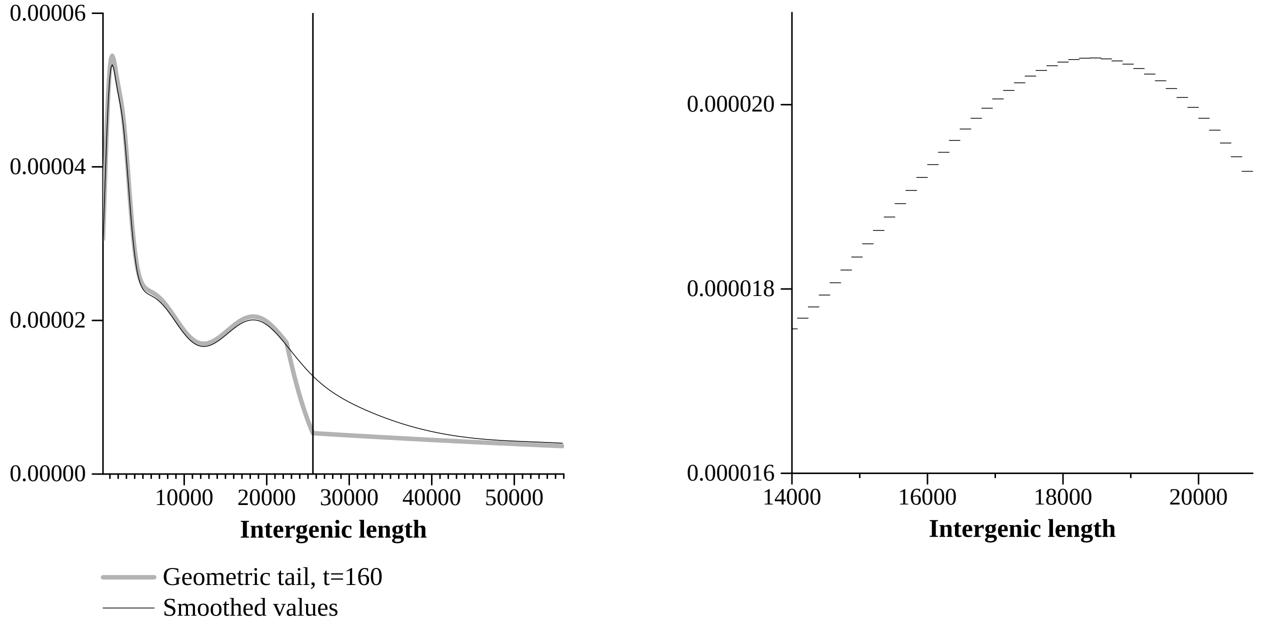


Figure 4.5: Geometric length distribution of human intergenic regions from training portion of chromosome 22. The right plot shows detail of the step-function character of the geometric tail distribution, with step 160.

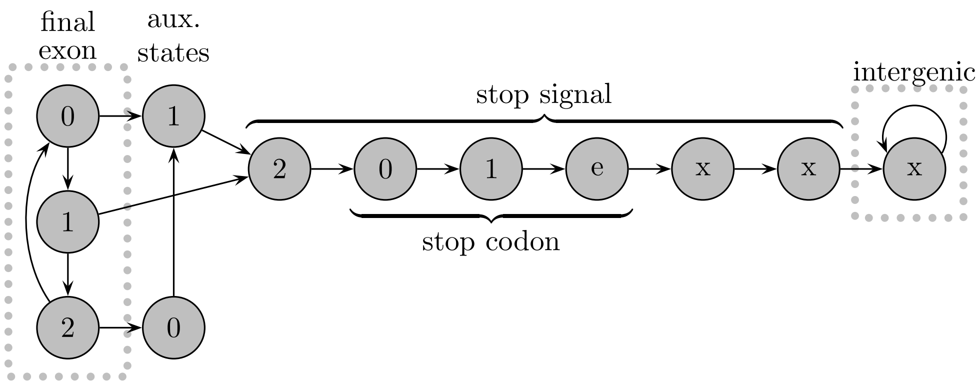


Figure 4.6: The translation stop signal consists of the stop codon and a short window surrounding it. The final exon submodel always produces a multiple of three bases, but may start, and therefore also end, in any of the three frames. The auxiliary states help to ensure that the frame is always correct when entering the signal.

4.1.3 Signal and content models

The HMM consists of states representing signals and states representing sequence regions such as exons, introns, and intergenic regions. States for sequence regions are of order 4, which means that each character depends on the four previously emitted characters. Emission probabilities are estimated from the training data. For every combination of four consecutive nucleotides we need the distribution over all possible nucleotides that can follow them. If some combination of four nucleotides is rare, we may not have enough data to estimate this distribution. If so, we use interpolation [144]. That is, we combine the observed fourth order distribution with the third or even lower order distribution that may have been estimated on more numerous data. The emission probabilities are interpolated only for combinations of four nucleotides with fewer than 100 training data samples.

We train the emission probabilities on sequences in which sequence repeats were masked, that is rewritten with letter N denoting an unknown nucleotide. Sequence repeats account for a significant portion of eukaryotic genomes, and when we include them in training, they often overwhelm other data. For example, non-repeat parts of intergenic regions may have low emission probability in an intergenic state trained on a mixture of repeat and non-repeat sequences, and thus may be erroneously predicted as coding. Consequently, prediction accuracy decreases significantly when repeats are included in the training data.

A single state representing one of the sequence regions generates a long stretch of sequence with the same set of emission parameters. In contrast, signals are represented as chains of states, each state generating one position of a short signal window, using a separate set of emission probabilities. The splice site signals depicted in Figure 4.3 are examples of such signal models. A probabilistic model consisting of a chain of states of order k is called a position weight matrix of order k [174].

We have developed higher order trees [28], a generalization of position weight matrices, which allows nucleotides of the signal window to be generated in some arbitrary fixed order. In a higher order tree of order k, each nucleotide depends on an arbitrary set of at most k previously generated nucleotides. Trees of order one are known as Chow-Liu trees [50], and were previously used for biological signal modeling by Agarwal and Bafna [4] and Cai *et al.* [42]. The main reason for using higher-order trees or Chow-Liu trees is the observation that some signals have dependencies between nonadjacent positions [37]. To use a higher order tree, we need to find a topology, that is, the order in which nucleotides will be generated and a set of predecessors for each state. We compute the topology by a heuristic algorithm that attempts to optimize the likelihood of training data for a fixed order k [28]. Once the topology is fixed, we estimate the emission probabilities similarly as for states generating sequence regions, including the use of interpolation for combinations of predecessor nucleotides with insufficient data.

For some signals, the emission probabilities do not change sharply at each position, but rather they are similar at adjacent positions but change slowly over an interval. In those cases we use windowed position weight matrices [37], which differ from regular position weight matrices only in their training: the emission table for each position is trained on examples pooled from a short window around the position. This technique increases the amount of training data and thus allows the use of higher order states.

Our HMM contains signal models of donor and acceptor splice sites, transcription start site and transcription stop site. In addition, we use a windowed model of the region at the end of an intron, just before the acceptor splice site. This region is typically rich in the nucleotides C and T (pyrimidines). We also use a windowed model of the signal peptide in the initial exon after start site signal. The signal peptide (sometimes also called signal sequence) is a signal that directs the transport of proteins to endoplasmic reticulum. It occurs only in transmembrane and secretory proteins. Therefore, we have a special submodel that emits a sequence using either the signal peptide submodel or regular coding region states. This signal peptide model is trained on those genes in the training set for which a signal peptide prediction program by Zhang and Henzel [175] identifies a high scoring signal peptide. Note that our special submodel potentially introduces two state paths with the same labeling, one that emits signal peptide and one that emits a regular sequence of codons. We avoid this problem in the Viterbi algorithm by treating the whole submodel as a special generalized state generating the whole signal window.

4.1.4 Dependence on GC content

The GC content of a given sequence is the proportion of G and C nucleotides in the sequence. GC content usually varies within a genome. In mammalian genomes, high GC regions have higher gene density and shorter introns [51]. Figure 4.7 illustrates a similar effect in our training data.

Since the statistical characteristics of genes change with GC content, some gene finders, such as Genscan [37], have different parameters for different GC content ranges. In Genscan it is assumed that the input DNA sequence is relatively short and homogeneous. Therefore the GC content is computed for the sequence as a whole and a corresponding parameter set is used. In ExonHunter in contrast, we compute the GC content in a sliding window of length 1000 and change the parameters with changing GC content. We use four different GC content levels for human genome and three for fruit fly. Besides changing the HMM parameters with GC content, we also change the prior probability of individual labels. This prior probability distribution is needed in Formula (2.4) for combining the super-advisor and the HMM, and it is also used as an additional advisor with a low weight (see Lemma 5).

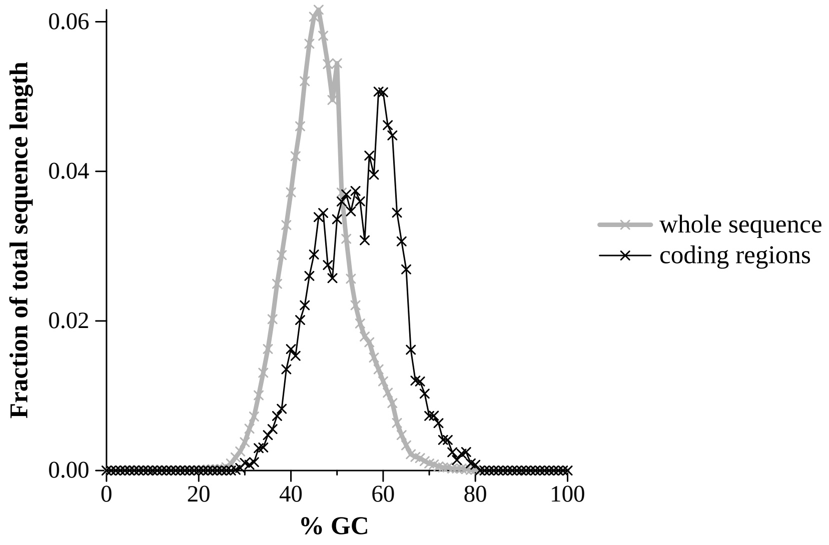


Figure 4.7: Relative frequency of different GC content levels in training portion of the human chromosome 22. Data from the whole sequence compared with data from coding regions only. Each point corresponds to a bin spanning one per cent of the GC content range.

4.2 Training and testing data sets

Most of the experiments presented in this thesis were done on human genomic sequences. We also show some results for fruit fly *Drosophila melanogaster* to demonstrate that our approach can be extended to other species. An overview of the sets used in our experiments is shown in Table 4.1.

For testing, we use the Rosetta set of 117 human single-gene sequences developed by Batzoglou *et al.* [15]. The advantage of this set is that the results of several gene finders for this set are available in the literature. On the other hand, the sequences in the set are quite short, and the gene annotation has not been updated with more recent experimental evidence from the past five years. We also test our gene finder on a testing set from the ENCODE project [61]. The goal of the ENCODE project is to identify all functional elements in the human genome, including protein coding genes. In the pilot phase of the project, approximately 1 per cent (30 Mb) of the genome from 44 different regions was chosen for detailed analysis. A gene finding workshop was organized in May 2005, with the goal of comparing the performance of different gene finders [1]. A hand curated gene annotation of 13 ENCODE regions was released before the workshop as a training set, while the rest were a testing set. In this thesis, we split the ENCODE set into training and testing sets in the same way. We perform some partial experiments only on three regions from the ENCODE testing sets, namely ENr131, ENr233, and ENr332. We use the hand curated HAVANA annotation [11] as a reference annotation of all ENCODE regions in our training and testing sets.

We have trained all advisor parameters for the human genome on the ENCODE training set. The many parameters of the HMM require more training data. Therefore, we have also used the set of 1284 human single-gene sequences collected by Stanke [156] for training the Augustus gene finder. We have removed 81 sequences from this set due to significant similarities to the Rosetta set used for testing. For training models of splice site signals, we include a set of 15,263 human splice sites from SpliceDB (downloaded May 26, 2005) [41]. Finally, intergenic region lengths were trained on a part of human chromosome 22 annotated with the RefSeq annotation [133], downloaded

Table 4.1: Overview of the data sets used in our experiments and their sizes.

Set	Length	Genes	Coding exons
Human training sets:			
ENCODE training	9M	137	1,597
Augustus	11M	1203	6,151
SpliceDB	2.5M	(15,263 splice site pairs)	
Chromosome 22	18M	196	1,786
Human testing sets:			
Rosetta	0.6M	117	464
ENCODE testing	21M	296	2,782
ENCODE three sequences	1.5M	41	533
Fruit fly sets:			
Training (chr. 3L)	52M	5,835	22,578
Testing (chr. 2L)	22M	3,380	10,021

from the UCSC Genome browser [89] on July 2, 2005. RefSeq annotation is based on the newest experimental data and a part of it is hand curated.

For fruit fly experiments, we have downloaded genomic data from the UCSC Genome browser (assembly from April 2004) annotated with its RefSeq annotation. We have used chromosome arm 3L for training and chromosome arm 2L for testing.

As we have discussed in Section 1.1, our gene finder assumes that genes do not overlap and that each gene produces only one transcript. In practice these assumptions are not true, and the annotation of a training set does not correspond to a single labeling. To simplify training, we create a reference labeling by choosing the subset of non-overlapping transcripts from the annotation that maximizes the total length of coding regions in the labeling. For testing, we compare our prediction to all annotated transcripts.

4.3 Advisor construction

In Chapter 2, we have described how to combine the advice of multiple advisors with an HMM-based gene finder. In order to use the advisor framework, we need to express the evidence from a variety of sources in the form of advisors. We need to determine how to partition the labels for each advisor and how to assign probabilities to sets in the partition based on the evidence.

Some gene finders using evidence in a similar manner as our advisor system (for example HMMGene [97] and GenomeScan [173]) obtain probabilities by hand-crafted rules from the statistical significance of the local alignments used as evidence. Such rules need to be manually adjusted for new sources of information. Instead, we estimate the probabilities by a simple training procedure from training data. Our training procedure operates on a simplified representation of each source evidence, consisting of a set of intervals with associated scores. Therefore, it can be applied to different alignment-based sources of evidence.

We will then discuss the sources of evidence and corresponding advisors that we use for gene finding in the human and fruit fly genomes. Issues specific to each particular source of information will influence how we convert sequence alignments to intervals, how we assign them scores and what label partitions will the advisors use. Biological properties of individual sources of evidence and their use in existing gene finders are discussed in Section 1.4.

4.3.1 Interval representation of alignment-based advisors

In this section, we will describe a simple unified procedure for estimating advisor parameters from training data. To unify the procedure, we will present available evidence as a set of intervals. Each interval is assigned a score and a label set S.

Local sequence alignments between a database of known proteins, ESTs, or genomic sequences can be represented in this framework by identifying each alignment with an interval that contains query sequence positions covered by it. The score of an interval may be the score of the corresponding alignment. Intervals are not, however, required to directly correspond to alignments. For example, two alignments that are adjacent in a known protein but some distance apart in the query sequence indicate an intron in the query sequence between the two alignments. We may create an interval corresponding to this likely intron.

An advisor a using the interval representation of evidence will produce vacuous advice at each position not covered by any interval of its type. At each position covered by an interval, it will use a binary partition $\pi_a = (S, \Sigma - S)$. The probability $p_a(S)$ assigned to the chosen set S depends on the score of the interval and on the distance of the current position to the nearest interval boundary. If a sequence position is inside multiple intervals, we choose the interval maximizing $p_a(S)$. The advisor either uses the same partition $\pi_a = (S, \Sigma - S)$ at every point of the interval or repeats a pattern of three partitions periodically to indicate the reading frame in coding regions. Such a three-periodic pattern can be completely specified by the set S at the first position of the interval.

We will estimate the values of $p_a(S)$ from the training data set. First, we partition the range of possible scores and the range of possible distances into buckets, creating a two-dimensional bucket grid. For each two-dimensional bucket, the probability $p_a(S)$ is estimated as the true positive rate, or the fraction of sites in the bucket labeled by a label from S. The score buckets separate intervals into several confidence levels, and the probability $p_a(S)$ also depends on the distance from the interval boundary, which will result in lower true positive rates near the interval boundaries when input intervals often extend beyond the boundary of coding regions or of the other features they are supposed to indicate.

The bucketing helps us to reduce the number of parameters so that we can estimate $p_a(S)$ more reliably. We keep the number of score buckets quite small, typically 5 or less, depending on the amount of training data. On the other hand, it does not help much to put two adjacent points from the same interval to the same distance bucket, because the label at two adjacent positions is highly correlated. However, greater distances have fewer training data points, as only intervals longer than $2d$ have points with distance at least d from both end points. Therefore we create a separate bucket for every distance d from 0 to some threshold D, and one for all distances greater than D. The threshold D is chosen so that 40% of intervals have length at least $2D$. In the last bucket, we weight samples so that the total weight of all samples from each interval is one. Otherwise extremely long intervals tend to outweigh all others.

Once the distance buckets are fixed, we then determine the score buckets so that scores with similar false positive rates are grouped together. The goal is to group more reliable intervals together and assign them high probability $p_a(S)$. Formally, we measure the homogeneity of a bucket by its entropy. Consider a two-dimensional bucket with distance d and score range $[s_1, s_2]$, and let $n(s_1, s_2, d)$ be the number of training samples in this bucket, and $p(s_1, s_2, d)$ be the true positive rate among these samples. Then the entropy of this bucket is

$$h(s_1, s_2, d) = p(s_1, s_2, d) \cdot \log p(s_1, s_2, d) + (1 - p(s_1, s_2, d)) \cdot \log(1 - p(s_1, s_2, d)). \tag{4.1}$$

We weight the entropy of each bucket by $n(s_1, s_2, d)$ and then choose a fixed number of score bucket boundaries so that the weighted sum of entropies of all two-dimensional buckets is minimized. Note that minimizing entropy is equivalent to maximizing the likelihood of the training data, as the likelihood of training samples in a particular bucket with $n(s_1, s_2, d)p(s_1, s_2, d)$ positive samples and $n(s_1, s_2, d)(1 - p(s_1, s_2, d))$ negative samples is

$$p(s_1, s_2, d)^{n(s_1,s_2,d)p(s_1,s_2,d)}(1 - p(s_1, s_2, d))^{n(s_1,s_2,d)(1-p(s_1,s_2,d))}. \tag{4.2}$$

By taking the logarithm of this expression, we obtain $h(s_1, s_2, d)$.

This optimization problem can be solved by a simple dynamic programming algorithm [70] in $O(\sigma^2 BD)$ time where σ is the number of different scores, B is the desired number of buckets, and D is the distance threshold. For every possible score s that can be the largest score in the bth score bucket, we try all possible largest scores for the previous score bucket, and choose the one yielding the highest weighted sum of bucket entropies. We need $O(D)$ time in each step to compute the entropies for the current score range and all possible distances. To avoid unbalanced partitions with very small buckets that may suffer from insufficient training data, we restrict each score bucket so that it contains at least a $1/(3B)$ fraction of the training data. This constraint can be easily integrated in the dynamic programming by considering only score buckets that meet this constraint. An example of score buckets and their associated true positive rates obtained by this process is shown in Figure 4.8

In the training procedure we consider positions of each intervals as separate training samples. However, if we have several overlapping intervals, only the one with the highest true positive rate will be used at each position. Others contribute to training but not to actual use, and may create biases in the true positive rate. Therefore we remove excessive overlaps before using intervals for both training and testing. We do this by a simple greedy algorithm that considers intervals in the order of descending score and discard each interval that overlaps by more than 60% with some non-discarded interval of higher score.

Intuitively, the true positive rate $p_a(S)$ should be higher than the prior probability of the set S, if the interval represents evidence supporting the labels of S. However, occasionally some buckets have the true positive rate lower than the prior. This happens because the prior varies with the GC content (see Section 4.1.4), and if the true positive rate $p_a(S)$ is only slightly higher than the overall prior of the set S, it may be higher than the prior in some GC content levels and lower in others. When $p_a(S)$ is lower than the prior, it decreases the probability of all gene structures that have label in set S at that particular position, which is not the intended behavior when we are using alignments as positive evidence. Therefore we filter out all advice lower than the prior from all advisors based on interval representation, and replace it with vacuous advice.

The bucketing scheme presented in this section can be used to estimate parameters of any source of evidence represented as a set of intervals with scores. In the following sections we will describe several advisors using the interval scheme to represent evidence based on sequence alignments.

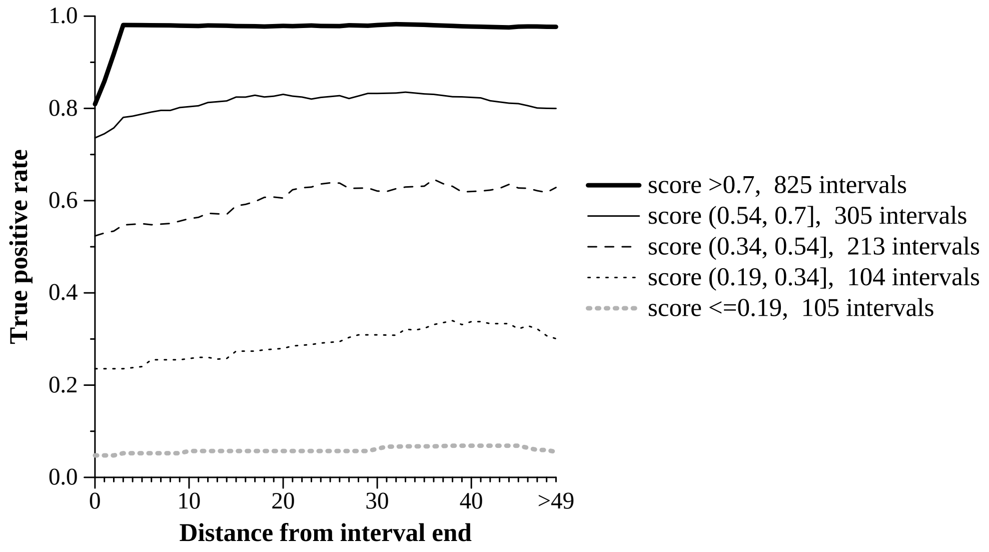


Figure 4.8: This graphs shows an example of the results obtained by the bucketing procedure. The x-axis corresponds to the distance of a position from the nearest interval boundary. The right-most point represents distances greater than 48. Every line of the graph corresponds to one score bucket. The y-axis represents the true positive rate for each bucket, that is, the fraction of positions labeled by a label from the set S associated with each interval. The data shown in this graph come from alignments between human proteins and the ENCODE training set, explained in more detail in Section 4.3.2. We see that the five score buckets have quite different true positive rates.

4.3.2 Advisors based on protein alignments

Some proteins encoded in the query sequence, or their homologs in other species, will already be known. Therefore, local alignments between a database of known proteins and the query sequence may help us to locate genes. In our experiments, we use the SwissProt database [21] of hand-curated proteins, release 47.3 (from June 2005). We extract proteins of several selected species from the database and use proteins from each species separately. We use proteins from human, mouse, chicken, and the fruit fly to predict genes in human genomic sequences, and proteins from the fruit fly and the roundworm *Caenorhabditis elegans* to predict genes in fruit fly sequences. The database contains 12,330 human, 9,449 mouse, 1,170 chicken, 2,231 fruit fly, and 2,657 roundworm proteins.

We align proteins with the query sequence using BLASTX [74]. When running BLASTX, we use the default BLOSUM62 matrix [84] to score the alignment and increase the gap penalties so that the penalty for starting a gap is 11 and for extending a gap by one more position is 2. We discard alignments with low scores that are expected to occur at least 0.05 times just by chance in our database, using the E-value statistics computed by BLAST. We also discard alignments that contain long gaps potentially spanning introns.

Each species yields several advisors. The first advisor predicts coding regions and the reading frame. Therefore the set S in its partition will contain the reading frame label indicated by the alignment, changing in a three-periodic pattern. Each interval corresponds to the extent of one alignment with 30 nucleotides removed at each end. For alignments shorter than 60 nucleotides we keep only the one or two middle codons. The purpose of this trimming is to avoid excessive

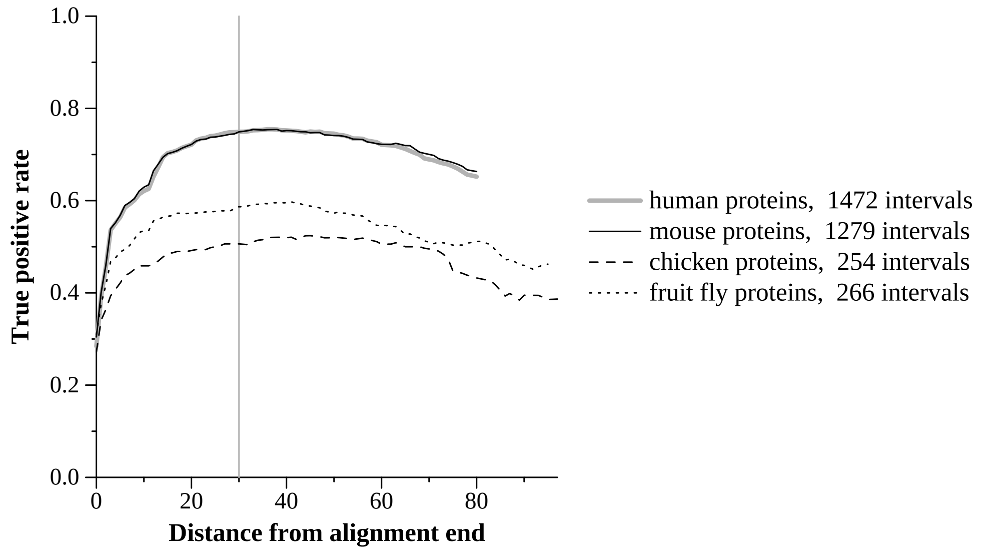


Figure 4.9: True positive rate of protein advisors as a function of the distance from the alignment boundary on the ENCODE training set. Note that mouse and human proteins yield very similar true positive rates, although their sets of alignments differ. The true positive rate drops fast close to the alignment boundary, especially for human and mouse proteins. We cut 30 positions from both ends of each alignment to avoid many false positives. Alignments inside true coding regions tend to be shorter, and therefore the true positive rate decreases somewhat at higher lengths where only longer alignments are counted.

overlaps with introns for alignments that extend beyond exon boundaries. Figure 4.9 illustrates that if alignments are not trimmed, the true positive rate drops fast close to the alignment boundary and this may cause prediction errors.

To score the reliability of intervals, we use the BLOSUM62 score computed by BLASTX, divided by the length of the alignment. We have observed that this interval scoring scheme is more strongly correlated with the true positive rate than the total BLOSUM62 score of the alignment (see Figure 4.10). This is possibly caused by long relatively weak alignments between proteins and pseudogenes (inactivated copies of genes). These alignments can be more easily distinguished by their low score per position than by the total score, which may be relatively high for long, weak alignments.

If two alignments are adjacent in the protein and non-adjacent in the query sequence, we construct a second advisor that predicts an intron in the gap between the two alignments. Its score equals the sum of scores of the two adjacent exon intervals, that is, it is the sum of two BLOSUM62 per position values. We consider two alignments adjacent in the protein if their distance or overlap is at most 10 amino acids. We only report introns of length between 100 and 10,000. Although both shorter and longer introns exist, shorter gaps between alignments may be also a result of an insertion within an exon or a weakly conserved region, while longer gaps may correspond to parts of two genes that coincidentally match the same protein. As in the case of exons, each intron interval is trimmed by 15 bases on both sides.

If the alignment includes a protein's start codon, we create one advisor that predicts the start codon label and one that predicts the intergenic label in the window of length 100 adjacent to the hypothetical start codon suggested by the alignment. Analogously we have two more advisors for

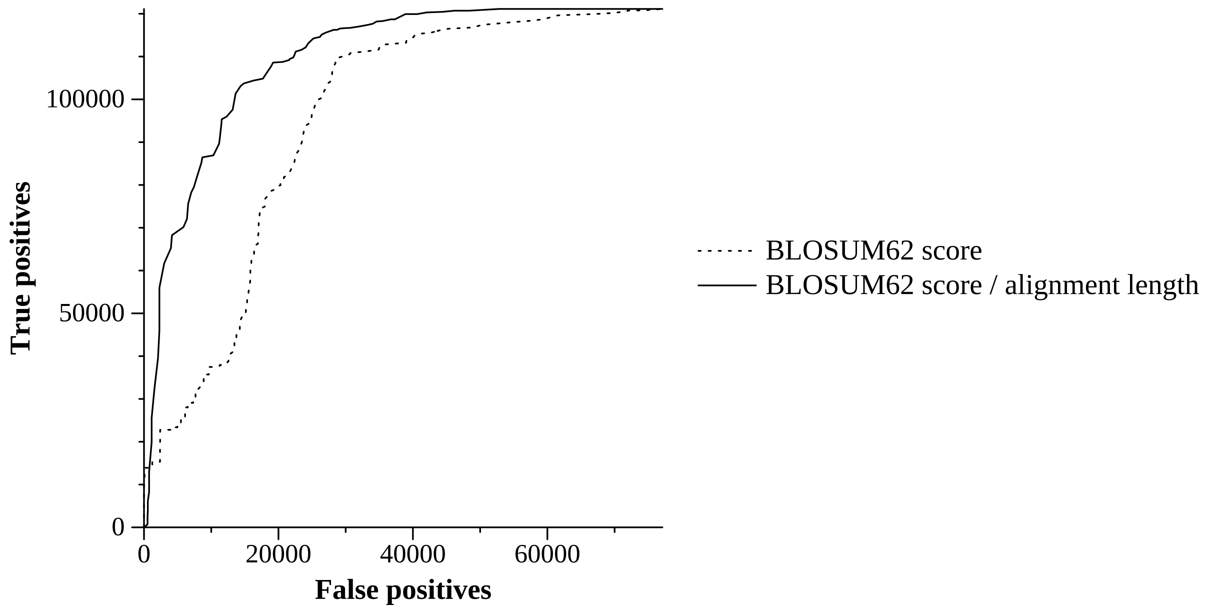


Figure 4.10: Receiver operating characteristic (ROC) curve of the total alignment score and alignment score divided by alignment length. For each of the two scoring methods we sort all positions of all intervals by decreasing interval score and then for each prefix of the sorted sequence we count the number of true and false positives and plot it as a one point of the curve. An ideal scoring method would have all true positives preceding all false positives; that is, the curve would go through the upper left corner of the graph. A random permutation of samples would give a curve near the diagonal. The graph is based on alignments between mouse proteins and ENCODE training set. The curve for BLOSUM62 score per position is further from the diagonal, showing that true positives and false positives are better separated by this scoring method. This happens because some high-scoring false positives are very long, with low score per position.

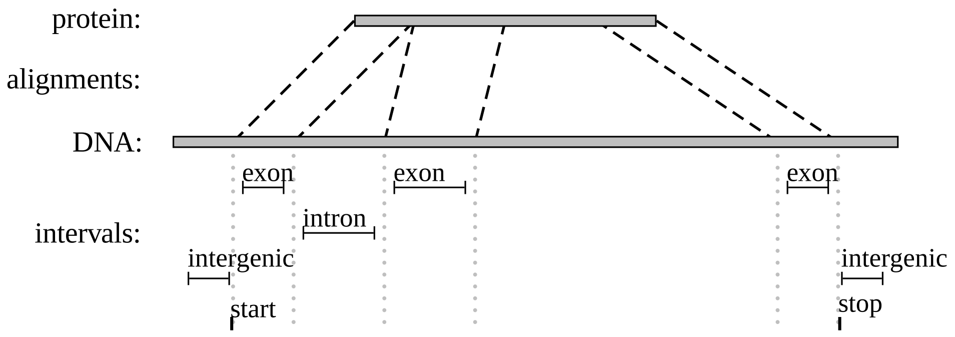


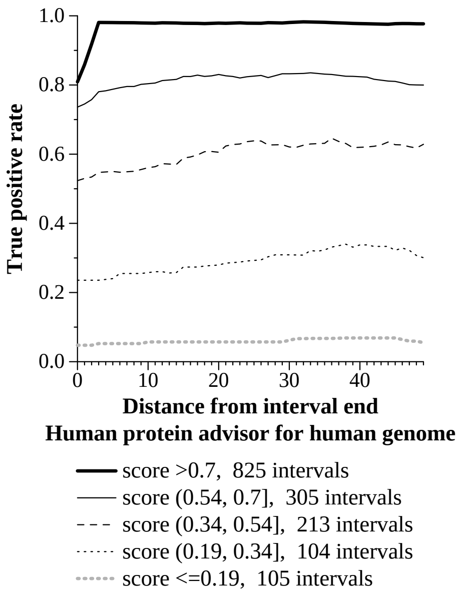
Figure 4.11: Overview of intervals produced from a set of hypothetical alignments with one protein. One advisor uses the three intervals inside alignments to predict exons, one advisor uses the gap between two alignments adjacent in protein to predict intron, two advisors predict short intergenic region before the first and after the last alignment respectively, and two advisors predict start codon and stop codon.

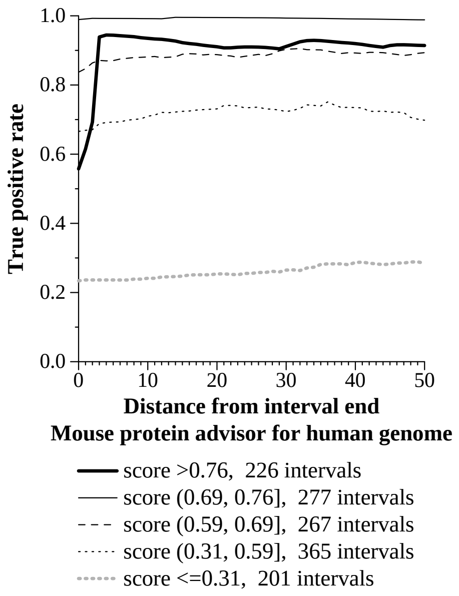
stop codons. The overview of all intervals created from a set of hypothetical alignments with one protein is shown in Figure 4.11. For human gene finding, we create start and stop codon advisors only from human and mouse proteins, as alignments of chicken and fruit fly proteins with the human genome contain start and stop codons only rarely.

A protein database may contain proteins corresponding to several splicing variants of the same gene. However, our gene finder is capable of predicting only one variant. When we encounter alignments that suggest overlapping exon and intron intervals, indicating the possible presence of alternative splicing, we remove the overlapping portions from both intervals and leave the choice of the splicing variant to the HMM. This post-processing is also done for the EST advisors presented in the next section.

Figures 4.12 and 4.13 show the parameters of several protein advisors used in ExonHunter. The human protein advisor for predicting exons in human genome has five clearly separated score buckets with true positive rate growing with interval score. For mouse protein the first four buckets behave in a similar way, but the highest scoring intervals have very low true positive rate close to the interval boundary. This is caused by a large number of false positives among the alignments shorter than 60 nucleotides, which are reduced to only one or two codons by our trimming scheme. These short alignments have high score per position but relatively low overall score and may represent spurious matches. Perhaps they can be filtered out by setting a higher threshold on the alignment score when running BLASTX, but such a step may omit many good alignments as well. The fruit fly genome has relatively few pseudogenes [81]. Therefore the true positive rates are very high in all score buckets for the exon advisor based on fruit fly proteins. In introns, the situation is reversed. The advisor for the human genome has true positive rates close to one for all buckets. In the fruit fly, genome many introns are shorter than our intron length lower bound of 100 nucleotides. Therefore, we filter out many genuine matches, and the rest have relatively low true positive rate. Improvements could be probably achieved by using intron length cutoffs adjusted for the fruit fly genome. Even with current settings, though, high scoring intron intervals do provide reliable information for the gene finder.

All protein advisors use the true positive rates produced by the bucketing algorithm introduced in Section 4.3.1. It is possible that in some instances we may not have sufficient training data to





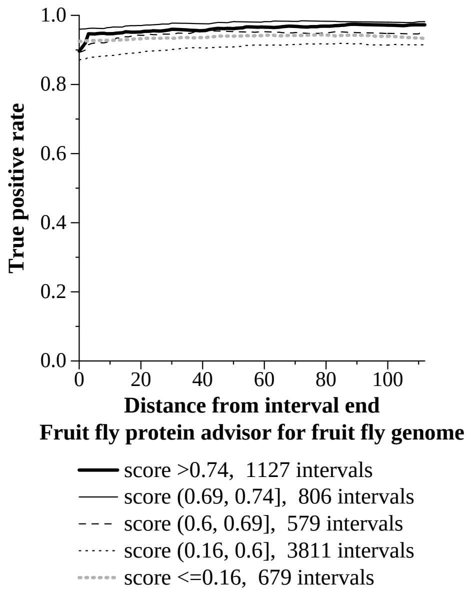


Figure 4.12: True positive rates obtained by the bucketing procedure for exon protein advisors used in ExonHunter. While human genome advisors based on human and mouse protein data have buckets with a wide range of true positive rates, the fruit fly genome advisor based on fruit fly protein data has very high positive rates for all score ranges.

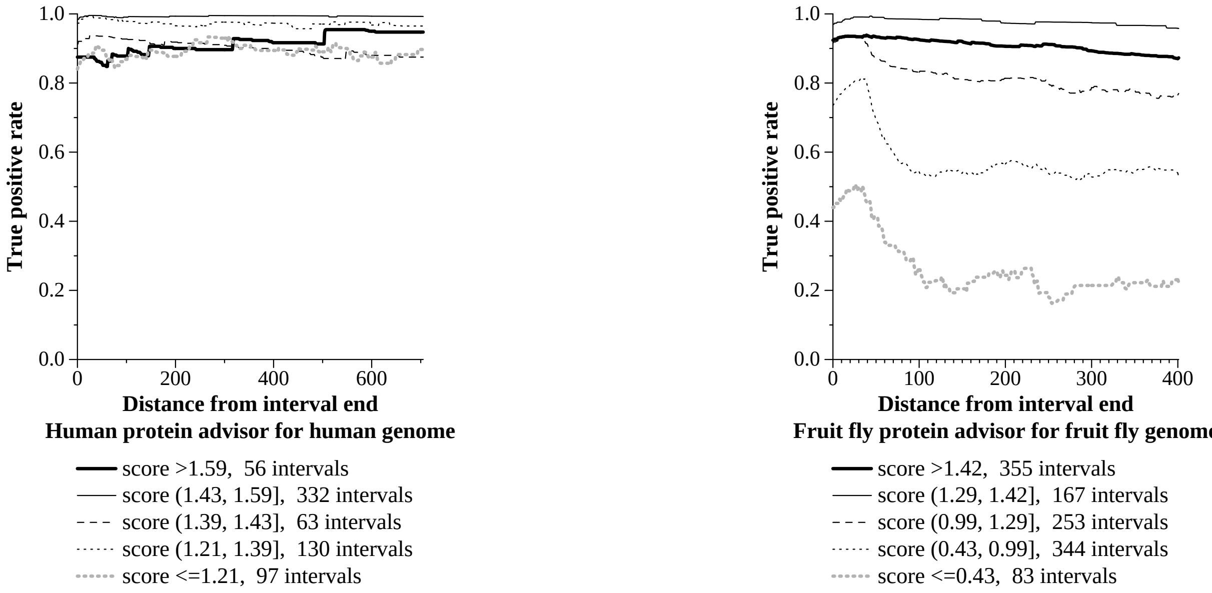


Figure 4.13: True positive rates obtained by the bucketing procedure for the intron protein advisors used in ExonHunter. The human genome advisors based on human protein data have high true positive rates. The fruit fly genome advisor based on fruit fly protein data achieves high true positive rates for high scoring buckets, but low scoring buckets contain many false positives.

estimate these parameters reliably or that we would use a new version of a database with parameters estimated on an older version. Therefore we have tested how sensitive the gene finding results are with respect to changes in true positive rates. We have first run ExonHunter with advisors derived from mouse protein alignments on the testing set consisting of three ENCODE sequences with 533 annotated coding exons. We have used the improved naive method described in Section 2.5.3 to combine the mouse protein advisors and the prior advisor into the super-advisor. Then, we varied the parameters used by the mouse protein advisors, and observed the changes in ExonHunter prediction accuracy. The results are shown in Table 4.2. We have tried parameters obtained by the bucketing algorithm run on human and chicken protein alignments, as well as the original mouse true positive rates, artificially raised or lowered by multiplying or dividing each probability by 3/2. We have set all values that exceeded one after multiplication to one. Finally, we have also trained the true positive rates with all scores in one bucket.

All parameter modifications yield very similar gene annotations. When we use the original distance based method for combining advisors instead of the improved naive combination method, we see even smaller changes, because this method constraints the super-advisor prediction to be at most 100 times more than the prior probability. The exon advisor increases the probability of one the six coding frame labels at each position. These labels have prior probability less than 0.005 in most GC content levels. Therefore if the exon advisor is used alone, any probability above 0.5 will make the super-advisor's value exactly 100 times the prior.

This experiment demonstrates that the mouse protein advisors improve the gene prediction results compared to using the HMM only, and that protein advisors are robust with respect to changes in true positive rate estimates. It also shows that although our bucketing scheme separates alignments into clusters with different true positive rates, it does not actually lead to improvements

Table 4.2: Influence of the true positive rate estimates on the prediction accuracy. We use mouse protein alignments with true positive rates estimated by the bucketing algorithm described in Section 4.3.1 on human and chicken protein alignments, as well as with artificially raised and lowered parameters obtained from mouse alignments, and parameters obtained from all alignments grouped in one score bucket. Exon sensitivity and specificity of ExonHunter on the three ENCODE testing sequences is almost identical for all parameters we have tried. Also, each run produced 97–99% of the exons predicted with original mouse parameters, only 87% of which are matched when we use no advisors.

	Exon Sn.	Exon Sp.	Shared with orig. parameters
Original mouse tables	68%	59%	–
Human protein tables	68%	59%	98.4%
Chicken protein tables	68%	59%	98.2%
Original tables times 3/2	69%	60%	97.5%
Original tables times 2/3	68%	59%	97.5%
One score bucket	68%	59%	98.9%
No advisors	63%	58%	87.3%

in prediction accuracy, compared to using only one bucket.

4.3.3 Advisors based on EST alignments

Molecules of mRNA harvested from cells and then sequenced provide direct evidence that a particular part of genomic sequence is transcribed. Sequenced fragments of mRNA molecules, called expressed sequence tags (ESTs), are available in large numbers for many species of interest. We can use ESTs from multiple species to predict genes in one genome.

In our experiments we use the TIGR gene index database [134], which contains ESTs merged into longer transcripts. Such merging reduces redundancy and speeds up search. For human gene finding we use the human gene index (release 16, February 2005) and the mouse gene index (release 15, February 2005). For fruit fly gene finding we use the fruit fly gene index (release 10, September 2004), and the mosquito (*Anopheles gambiae*) gene index (release 8, February 1, 2005).

As with proteins, we use each species as a separate source of evidence. ESTs are aligned to genome using the SIM4 program, by Florea *et al.* [66]. This program aligns the ESTs and attempts to identify correct splice sites. Since SIM4 is relatively slow, we first find similarities between the query sequence and the EST database using the PatternHunter local alignment program [116], with our coding seed `110 110 010 110 110 11` (this is the optimal seed of weight 11 and length at most 20 under the hidden Markov model for human-fruit fly alignments, see Section 3.5). Then we run SIM4 only for ESTs with at least one match.

For each species we create two advisors: one for exons and one for introns. They are created similarly as for the protein advisors for exons and introns. The score of an exon interval is the identity level of the alignment between the query DNA and the EST, as reported by SIM4. The score of an intron interval is the mean of the two adjacent exon intervals. Since SIM4 attempts to find appropriate splice sites, we do not experience problems with alignments extending beyond exon boundaries. Therefore, we trim both intron and exon intervals by only three nucleotides on

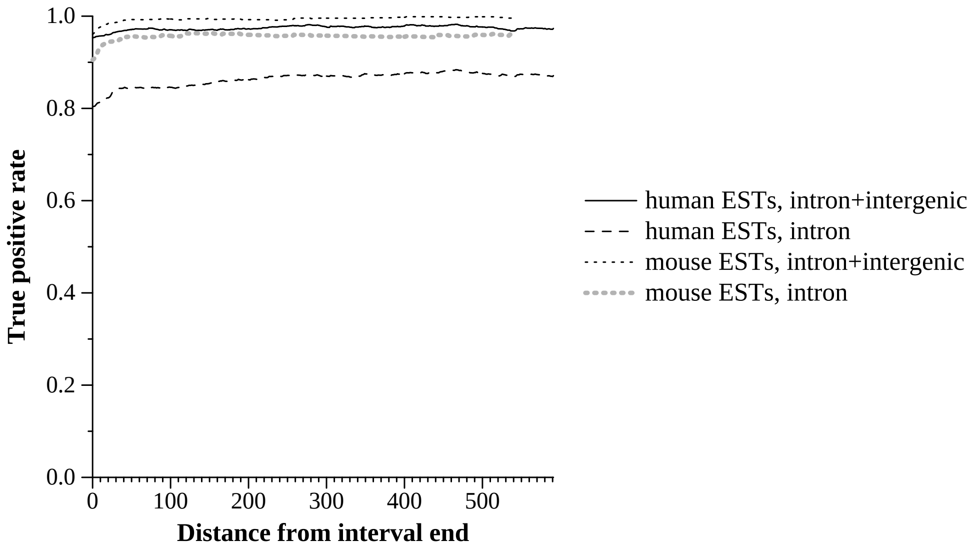


Figure 4.14: True positive rates of EST intron advisors for different label set partitions. Intron advisors use label set partition $\{S, \Sigma \setminus S\}$, where S contains either the two intron labels, or labels for both intron and intergenic regions. We include the intergenic label in the set S to account for introns in UTRs. A bigger set S inevitably leads to fewer false positives, but the difference between the two sets is very small in intervals from mouse ESTs, since very few intervals are in the intergenic regions.

each side to avoid overlaps with the splice site signals.

Another more important distinction is that while protein alignments indicate presence of a coding region, as well as its reading frame, EST matches may occur either in a coding region or in an untranslated region of the mRNA (UTR). Therefore, exon advisors originating from ESTs predict the probabilities for the partition $\pi_a = \{S, \Sigma \setminus S\}$ where the set S contains labels for all six reading frames, start and stop codon signals, and intergenic region. (Recall that in our set of labels, the intergenic label is used for UTRs as well.) Similarly, introns may occur between two coding regions or between two parts of an UTR. Therefore, the label set S of the intron advisor contains the intron labels for both strands and the intergenic region label representing introns in UTRs. If we use ESTs from different species than the query sequence, we typically see fewer matches in untranslated regions (see Figure 4.14). This is because untranslated regions are typically not as well conserved in evolution as coding regions. Therefore, the advisor using mouse ESTs to help with human genome gene finding only has the intron labels in its set S. This does not increase the number of false positives very much, and the influence of an advisor with this partition is much stronger. The prior probability of intron labels alone is much lower than the prior of intron and intergenic combined, which is over 97% in our training set of human genomic sequences.

4.3.4 Advisors based on genome alignments

Local alignments between two genomes highlight areas that are well conserved by evolution, and thus likely to have an important biological function. Protein coding regions are one example of such areas. Genome alignments can help us find genes that are transcribed only in small quantities or under special conditions, and thus are less likely to be detected by EST sequencing. For finding genes in the human genome, we use the genomes of mouse, chicken, and fruit fly. For finding genes in the fruit fly genome, we use the mosquito genome and the genome of another fruit fly species, *Drosophila pseudoobscura*.

To effectively use genomic alignments in gene finding, we need to consider two problems: how to find local alignments between two genomes, and how to determine which parts of the found alignments correspond to protein coding genes. We have discussed the first problem in Chapter 3, where we described spaced seeds that increase the sensitivity of similarity search between homologous coding regions, and thus help us find more alignments. In our experiments, we use the PatternHunter local alignment program [116] with our high-sensitivity coding seed `110 110 010 110 110 11`.

Not all alignments found by a local alignment program correspond to a coding region. Even those that do may extend beyond the exon boundaries to the surrounding introns or UTRs. PatternHunter uses a simple scoring scheme in which the score for a nucleotide match is +1 and mismatch -1. This scoring scheme is adequate for finding generic regions of high sequence similarity, but does not distinguish an alignment of two homologous coding regions from an alignment of two homologous introns or an alignment of a coding region with a pseudogene. Such alignments can be distinguished to some degree by their typical mutation patterns. As we have discussed in Section 3.3, coding regions have a high proportion of silent mutations that do not change the amino acid sequence of the encoded protein. It is possible to distinguish mutation patterns typical for protein coding regions by comparing the ratio of synonymous and non-synonymous substitution rates in the alignment (this is called the K_A/K_S ratio) [160]. These substitution rates are obtained by a phylogenetic analysis using a statistical model of codon evolution. Functional protein coding genes have higher synonymous mutation rates than pseudogenes, because non-synonymous mutations often have negative impact on the protein function, and thereby reduce the fitness of organisms that contain them.

Instead of using phylogenetic analysis, we simply translate the nucleotide alignment found by PatternHunter to six protein alignments, one for each reading frame, and score these alignments with the BLOSUM62 protein scoring matrix [84]. In each of the six reading frames, we find the segment of the alignment with maximum BLOSUM62 score that does not contain any frameshifts, gaps whose length is not a multiple of three. Thus we obtain six alignment segments, one for each frame. From them we choose the one with the highest score and turn it into an interval, by trimming seven codons from each side. The score of each trimmed interval is the BLOSUM62 score per position.

The BLOSUM62 matrix helps to identify alignments between pairs of homologous coding regions by assigning higher scores to mutations that occur often in known protein alignments, giving negative score to rare mutations, and heavily penalizing stop codons disrupting a coding region in one of the sequences. Its disadvantage is that it assigns the same score to two identical aligned codons as to two different codons coding for the same amino acid. We could avoid this problem by using a codon substitution matrix containing scores for all 64^2 pairs of codons. Recently, Schneider *et al.* [146] have developed such a matrix by observing mutation rates in a large set of coding region alignments.

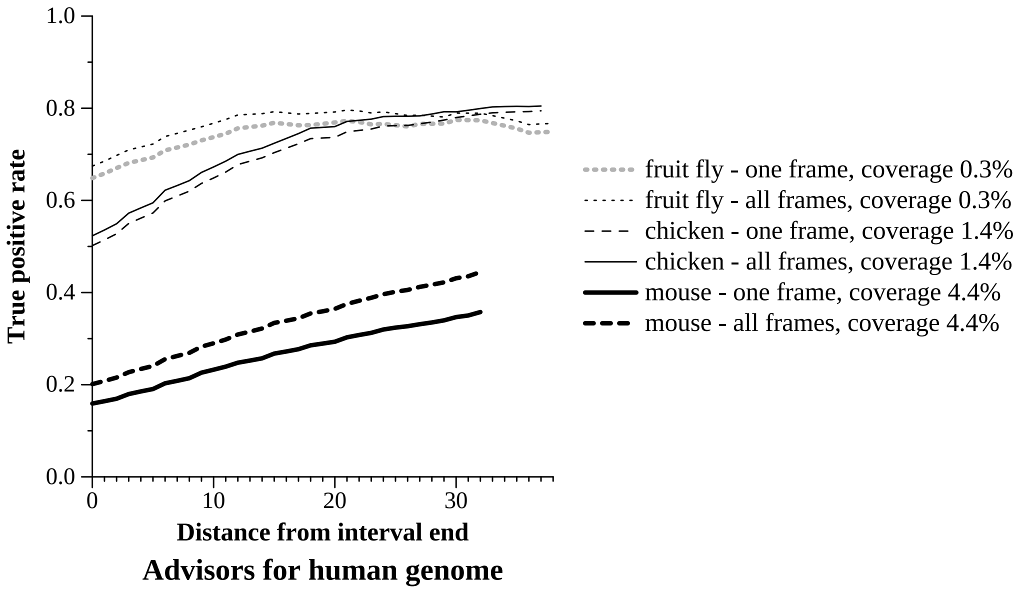
Our alignment rescoring procedure implies a reading frame for each position of the alignment. Most of the advisors based on genomes use the label of this single reading frame as the label set S in their label partition. The reading frame is sometimes predicted incorrectly by this method. Therefore, we also consider an alternative advisor that includes all six reading frames in its partition set S. Figure 4.15 shows that for most genomes the two variants have very similar true positive rates, and therefore it seems reasonable for the advisor to provide more specific information by using the smaller set S containing only the inferred reading frame. The mouse genome advisor predicts a wrong reading frame at about 20% of interval positions inside coding regions; therefore, we use the advisor that does not specify the reading frame when using genomic alignments between human and mouse. It might be possible to determine the reading frame reliably for some mouse genome alignments. For example, perhaps the reading frame tends to be correct in the alignments where the BLOSUM62 score of the highest scoring frame is much higher than the scores of all the other frames. We could use such information to divide mouse genome alignments between two advisors, one that would indicate frame and one that would not. At present, however, we use only a single advisor.

The figure also shows how the true positive rate changes with evolutionary distance. Many homologies between the mouse and human genomes are outside coding regions and our method does not filter all of them out. The chicken and fruit fly genomes have fewer homologies in non-coding regions, so their true positive rate is higher. The fruit fly genome is not very useful for finding human genes because it has very few hits. The chicken genome has comparable true positive rate, but its alignments cover a much greater portion of the genome. For finding genes in the fruit fly genome, genomic alignments with *Drosophila pseudoobscura* lead to similar true positive rate as mouse genome alignments for human. The mosquito genome, however, yields a very high true positive rate for finding coding regions.

Figure 4.15 shows all intervals grouped to one score bucket. In advisor training, we use several buckets. Buckets with low scores have quite low true positive rates. To eliminate many false positives in these buckets, we eliminate all advice that raises the exon probability less than 10 times above the prior.

The two-stage process that we use to find intervals for genome advisors could be replaced by translating both genomic sequences to proteins in all six frames and then comparing these, as is done for example in TBLASTX from the BLAST family of programs [8]. This process may find more alignments as some regions are similar on protein level but not on nucleotide level. But it is also quite slow, and thus not suitable for comparing long genomic sequences. We have seen in Section 3.5 that well-chosen spaced seeds achieve good sensitivity in finding homologous coding regions. Our rescoring method also allows the use of different scoring schemes, including models of codon evolution, whereas TBLASTX considers only protein sequences.

The rescoring phase of our procedure can probably be further improved to recognize homologies from coding region with greater accuracy especially for close species, such as human and mouse. For example, we assume that there is at most one coding region in each nucleotide alignment, but nucleotide alignments can sometimes span several exons if the introns between them are sufficiently conserved. In that case, our method will either choose the highest scoring exon, or (incorrectly) a region containing several exons and the introns between them. For example, in our set of alignments between the ENCODE human training set and the mouse genome, 0.6% of intervals span at least two exons. It might be better to apply the rescoring procedure on smaller windows and then choose all windows with sufficiently high BLOSUM62 score in one of the three reading frames, or even build a simple HMM discriminating between coding and non-coding alignments. Another possible line of improvement is to use a scoring matrix that considers not only amino acids but also the codons encoding them.



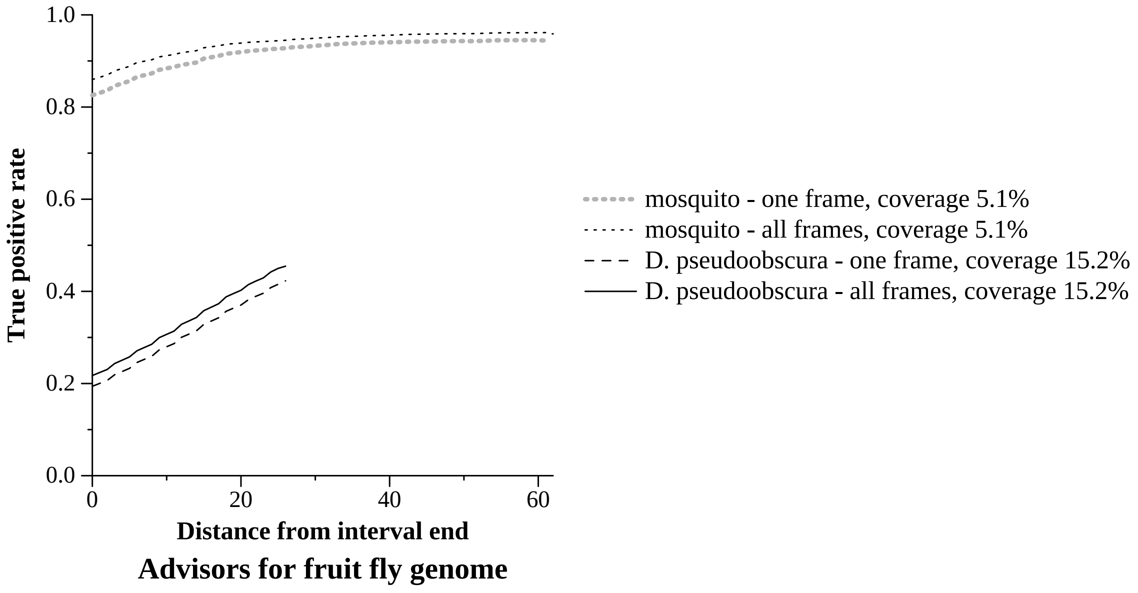


Figure 4.15: True positive rates of genome alignment advisors for different label set partitions. Genome advisors use label set partition $\{S, \Sigma \setminus S\}$, where S contains either the single reading frame label implied by the rescoring procedure or all six reading frame labels together. The top figure shows both variants of advisors for gene finding in the human genome, the bottom figure shows advisors for the fruit fly genome. The difference between these two variants is greatest for the mouse genome, and so we include all six reading frames in the set S for this genome. In all other cases the inferred reading frame seems quite reliable. The graph legends also show the portion of the training set covered by the intervals, for each advisor.

4.3.5 Advisors based on sequence repeats

A significant portion of eukaryotic genomes consists of repetitive elements. These elements typically do not occur inside coding regions, and thus an advisor can boost probability of introns and intergenic regions at positions covered by repeats.

We use RepeatMasker [152] to find a set of likely repeats, and these are converted to a slightly modified interval representation. All positions covered by repeat intervals are grouped into one distance bucket, and instead of score buckets, we divide sequence repeats into four categories, and estimate the true positive rate of each category separately.

The first category contains kinds of repeats that may occur in coding regions, and therefore the advisor ignores them. This category includes low complexity repeats, which are sequences very rich in one or two nucleotides, and simple repeats whose periodicity is a multiple of three. A simple repeat is a region containing many consecutive copies of a short pattern. If the length of this pattern is divisible by three, it can be part of a coding region in which one or a few amino acids repeat periodically.

The second category contains satellites. These are repeats that again occur in many tandem copies, although they are longer than the simple repeats from the previous category. They occur mostly in the telomere and centromere regions of a chromosome, which are regions important for stability and replication of the chromosome. As these regions do not contain genes, the repeat advisor predicts only the intergenic label for satellite repeats.

The third category contain simple repeats whose periodicity is not a multiple of three. These are rarely coding, so at such positions, we increase the probability of the label set S consisting of the intron and intergenic labels. Finally, the fourth category consists of all other repeats. At such positions we again predict "intron or intergenic." This category includes interspersed elements that occur in many distant parts of the genome. Some of these contain active or inactive copies of protein coding genes. In gene finding, we are not interested in identifying these genes in interspersed elements, as they are handled by the repeat masking process. Yet, such genes may confuse the gene finder and disrupt the prediction of adjacent genes.

The repeat advisor is very important. All other advisors presented so far tend to increase the probability of exons and introns. This advisor balances them by increasing the probability of intergenic regions. Other gene finding programs usually either ignore repeats altogether, or mask repeats in the query sequence by a special character N, representing an unknown nucleotide. In this approach, all sequence information contained in the masked sequences is lost, and cannot be used by an HMM. Repeat advisors allow us to keep the original sequence information, while increasing the probability of intron and intergenic region labels.

4.4 Experiments

In all experiments described in this section, we combine advisors by the distance-based combination method described in Section 2.4.1. As we have not observed in improvement of prediction accuracy by training advisor weights (see Section 2.6.3), we assign weight one to all advisors. We add the prior distribution of labels as an advisor with weight 1/100. To combine the super-advisor and the HMM, we use Formula (2.4), with exponent $\alpha = 0.02$. The low exponent helps to handle problems with position independence, as discussed in Section 2.7.2. Unless explicitly stated otherwise, we use all of the advisors based on proteins, ESTs, genomes, and repeats that we described in Section 4.3.

Table 4.3: Comparison of ExonHunter, Genscan, and several programs using mouse genome alignments on the Rosetta set containing 117 human genes. The results for Rosetta, SLAM, TwinScan, and SGP-1 were reported by Alexandersson *et al.* [5], who did not report gene statistics. The row labeled *EH* gives the results achieved by ExonHunter with all advisors. The row labeled *EH (nonhuman)* corresponds to ExonHunter results without advisors originating in human datasets.

	Gene Sn.	Gene Sp.	Exon Sn.	Exon Sp.	Nucl. Sn.	Nucl. Sp.
Genscan	44%	41%	82%	73%	98%	88%
Rosetta	—	—	83%	83%	94%	98%
SLAM	—	—	78%	76%	95%	98%
TwinScan	—	—	86%	82%	96%	94%
SGP-1	—	—	70%	76%	94%	96%
EH	70%	66%	90%	85%	98%	94%
EH (nonhuman)	67%	63%	89%	84%	98%	93%

4.4.1 Performance on short single-gene human sequences

The Rosetta testing set was used recently by Alexandersson *et al.* [5] to compare several gene finders using human/mouse genome alignments to predict human genes. Table 4.3 compares ExonHunter with the results reported in their work. One could object to this test since many of the genes in the Rosetta set are also found in the databases of human ESTs and proteins. Therefore, we also evaluated the program without advisors based on these databases. This reduced set of advisors has very similar accuracy on the exon and nucleotide level; the change mostly affects the gene-level statistics.

Mouse homology information helps some of the tested programs to outperform the *ab initio* gene finder Genscan. Yet, ExonHunter has higher accuracy than the other tested programs at the exon and nucleotide levels, except for nucleotide specificity, even if we do not use human proteins and ESTs. This confirms that using more sources of evidence, such as the proteins, ESTs, and repeats we use, in addition to genomes, leads to better gene finding accuracy.

We also compare our accuracy with GenomeScan [173]. GenomeScan combines *ab initio* gene finder Genscan with evidence obtained from protein alignments. We have submitted the Rosetta set into the GenomeScan web server, together with the protein sequences used by ExonHunter. If ExonHunter is run with only protein and repeat advisors, it performs worse than GenomeScan. This suggests that ExonHunter perhaps does not use the protein information in an optimal way. ExonHunter with all advisors performs better; it is almost as good as GenomeScan at the exon and nucleotide levels, although still worse at the gene level.

We note that the results of Genscan and GenomeScan can be influenced by the fact that the training set for Genscan contains sequences with high similarity to 56 sequences in the Rosetta testing set (about 48% of the testing set). Our experience suggests that such a large overlap may artificially increase prediction accuracy due to overfitting.

Table 4.4: Comparison of GenomeScan and ExonHunter on the Rosetta set. We show the results of ExonHunter without advisors, with protein and repeat advisors only, and with all advisors. In general, GenomeScan and ExonHunter results are comparable at the exon and nucleotide levels, but GenomeScan outperforms ExonHunter at the gene level.

	Gene Sn.	Gene Sp.	Exon Sn.	Exon Sp.	Nucl. Sn.	Nucl. Sp.
Genscan	44%	41%	82%	73%	98%	88%
GenomeScan	76%	74%	92%	86%	99%	94%
EH (no adv.)	47%	46%	78%	74%	93%	91%
EH (proteins)	65%	62%	88%	82%	98%	93%
EH	70%	66%	90%	85%	98%	94%

4.4.2 Performance on longer human genomic sequences

Table 4.5 shows results of ExonHunter and several other gene finders on the ENCODE testing set. This set contains longer genomic regions of half a million bases or more each, with many genes per region. Most gene predictions shown in the table were submitted to the ENCODE gene prediction workshop in May 2005 [1] before the hand curated HAVANA annotation of the sequences was released. We have downloaded the predictions from the workshop website, and evaluated them with the Eval tool by Keibler and Brent [90]. We have added the Genscan prediction, which we generated, as well as the TwinScan prediction and RefSeq gene annotation from the UCSC genome browser [89] (version from May 2, 2005, downloaded by the organizers of the workshop). The row labeled *ExonHunter, old version* is the version we submitted to the ENCODE gene prediction workshop. Since then, we have made some improvements in ExonHunter, mainly by changing the HMM and its training. We have also made small changes in advisors and updated the protein and EST databases used by our program.

The HAVANA annotation of the ENCODE training set is relatively conservative, and it is possible that some of the false positives of the gene prediction programs are in fact true positives. Therefore, sensitivity is a more reliable measures than specificity, since it measures the fraction of reliable gene annotations matched by a gene finder. The first group of programs in Table 4.5 are *ab initio* gene finders, those that use only the query sequence to annotate genes. Although ExonHunter's HMM without advisors does not achieve the performance of the recently published gene finder GeneZilla [117], it is comparable with other *ab initio* gene finders. The next group of gene finders use alignments of the query sequence with one or several other genomes. Of these, the best performance is achieved by N-SCAN [76]. The high specificity of N-SCAN is partly explained by the fact that the authors mask the sequence for sequence repeats and pseudogenes. For this they probably use some existing pseudogene annotation created by analysis of protein alignments.

The gene finders in the last group use a variety of information sources, including EST and protein alignments. The most sensitive is AceView [159], based on EST alignments filtered by human annotators. Jigsaw [7] combines several existing gene annotations and alignments. By combining high-quality data, such as full length cDNAs and curated RefSeq genes from other genomes, they achieve very high sensitivity and specificity. The next three gene finders are as yet unpublished extensions of existing HMM-based gene finders, namely TwinScan [95] enhanced with EST information, Fgenesh [143] enhanced with RefSeq genes and protein alignments, and Augustus

Table 4.5: Comparison of ExonHunter and other gene finders on the ENCODE testing set. The first block contains *ab initio* gene finders, the second block gene finders using genomic alignments, and the last block gene finders using various sources of evidence, including EST and protein alignments. We order the programs within each block by their exon sensitivity.
Results reported for programs labeled by asterisks by the Eval tool are lower than preliminary results released by the ENCODE gene prediction workshop organizers. The differences are probably caused by incorrect format of the files produced by these gene finders. Unfortunately, official analysis of the workshop results were not available at the time of writing of this thesis.

	Exon Sn.	Exon Sp.	Nucl. Sn.	Nucl. Sp.	
GeneZilla [117]	62%	50%	86%	50%	
Genscan [37]	59%	37%	85%	44%	
ExonHunter (no adv.)	55%	39%	79%	52%	
Augustus [156]	52%	63%	78%	75%	
GeneID [78]	47%	59%	74%	78%	*
GeneMark [114]	45%	26%	77%	37%	*
N-SCAN [76]	67%	81%	85%	88%	
Augustus with mouse genome	63%	69%	88%	80%	
TwinScan [95]	52%	65%	77%	83%	
SAGA [44]	38%	51%	52%	81%	
AceView [159]	81%	54%	88%	76%	*
Jigsaw [7]	80%	89%	95%	92%	
TwinScan with ESTs	76%	88%	87%	92%	
Fgenesh++	75%	69%	91%	77%	
Augustus+ [155]	74%	76%	94%	82%	
ExonHunter	69%	51%	93%	67%	
ExonHunter, supported genes	68%	63%	90%	74%	
Ensembl [56]	67%	72%	89%	90%	*
RefSeq [133]	64%	83%	86%	98%	*
ExonHunter, old version	64%	42%	90%	59%	

[156] enhanced with EST, protein and mouse genome alignments. ExonHunter has somewhat lower sensitivity and much lower specificity than these three programs. Ensembl [56] and RefSeq [133] are annotation pipelines that also rely mostly on strong evidence provided by full-length cDNA and protein alignments. Some gene annotation in RefSeq are curated by hand. Unfortunately due to file format issues, we were not able to compare these programs reliably with ExonHunter.

The lower exon sensitivity of ExonHunter is probably caused by the use of simple methods for aligning proteins to the query sequence in our advisors, as suggested already by the comparison with GenomeScan on the Rosetta set. In particular, we were forced to trim protein alignments on both sides by a wide margin to avoid false positives in introns. More accurate alignments of proteins and the query sequence, for example those constructed by GeneWise [20] or a similar pair HMM program, could remove the necessity of trimming and provide us with more precise information about locations of exons, introns, and their boundaries. Also note that Jigsaw, the most sensitive fully automated method, relies on high quality sources of information, such as cDNA sequences and RefSeq genes from other genomes, whereas we have not used such information sources in this experiment. The advisor framework allows us to include such information very easily, and we will do so in future versions.

The low specificity of ExonHunter is because ExonHunter does not penalize unsupported genes. If a long region does not contain evidence from any of the sources, ExonHunter will predict genes in this region by the *ab initio* methods implemented in our HMM. In contrast, TwinScan, for example, uses a special character in its conservation sequence to mark the positions not covered by any alignment and those in alignment gaps (see Section 1.5.1). This character is emitted by different states with different probabilities, and the emission probability of this character is presumably lower in coding region states than in intron or intergenic states. Thus, TwinScan predictions in long regions not covered by any alignment tend to have fewer genes and those genes tend to have shorter coding regions than predictions of the HMM underlying TwinScan. Our decision not to penalize unsupported genes makes ExonHunter possibly more useful for finding candidate novel genes that can be later verified by laboratory methods such as RT-PCR. Still, it would be useful to increase the specificity by designing more advisors that provide evidence that some region is *not* likely a part of a protein coding region. Besides the repeat advisor we currently use, such evidence could include the locations of pseudogenes and non-coding RNA genes (genes that are not translated to proteins, but function in the cell as RNA molecules).

Although high gene density produced by ExonHunter may be desirable when looking for targets of laboratory experiments, sometimes the user may need more conservative annotation. The super-advisor prediction, summarizing evidence from all advisors, can be used as an indicator of the evidence support of individual predicted genes and exons. In a simple experiment, we have computed for each gene the portion of its coding regions that have the probability of the predicted label strictly higher in the super-advisor prediction than in the prior. Then we have deleted all genes from the ExonHunter prediction that have this portion smaller than one half. The result is shown in the row labeled *ExonHunter, supported genes* in Table 4.5. This simple method increases exon specificity from 51% to 63%, while the exon sensitivity decreases only by one per cent. We could probably obtain even better results by more sophisticated filtering.

Table 4.5 does not show gene level statistics, as the file format of most predictions did not allow easy gene level evaluation by the Eval program. In this experiment, ExonHunter achieved 29% gene sensitivity and 10% specificity, compared to 17% gene sensitivity and 6% specificity achieved by Genscan.

Table 4.6: Performance of various combinations of advisors on the three sequences from the ENCODE testing set. All combinations of advisors include the repeat advisor. Advisors are grouped by species from which the evidence comes, and by the type of evidence. Information based in mouse is the best source of evidence, followed by human. ESTs provide more information than proteins or genomes. The combination of proteins, ESTs, genomes, and repeats performs better than any of them alone.

Advisors used	Exon sensitivity	Exon specificity
No advisors	63%	58%
Repeats	63%	64%
Fruit fly	63%	64%
Chicken	68%	65%
Mouse	76%	66%
Human	70%	67%
Proteins:		
– all	69%	66%
– mouse	69%	65%
– human	69%	66%
ESTs:		
– all	72%	67%
– mouse	72%	68%
– human	67%	66%
Genomes:		
– all	71%	62%
– chicken	68%	65%
– mouse	71%	62%
All advisors	76%	67%

4.4.3 Contribution of individual advisors

Table 4.6 shows the performance of ExonHunter with various subsets of advisors on the three sequences from the ENCODE testing set. The repeat advisor on its own substantially increases exon specificity, by preventing many spurious exon predictions. The fruit fly genome and its proteins do not influence the prediction very much, but chicken is more useful. Information originating from mouse improves ExonHunter's accuracy more than the information from human proteins and ESTs. Part of the reason is that the set of mouse advisors includes the mouse genome advisor, which has quite high sensitivity. In addition, we see that mouse ESTs work significantly better than human ESTs, since human and mouse are less conserved by evolution in untranslated regions than in coding regions, and thus mouse ESTs indicate the location of coding regions more reliably than do human ESTs. Overall, ESTs are a more useful source of evidence than proteins. This is because they are more abundant, or perhaps because they are used more effectively by the advisors. Chicken genome alignments also seem to be a good source of evidence, as their use leads to accuracy very close to the use of protein advisors. Finally, the combination of all advisors performs better than proteins, ESTs, genomes, or repeats alone.

Table 4.7: Performance of ExonHunter on the fruit fly testing set. The first block of the table contains *ab initio* gene finders, the second block gene finders using additional information. Advisors increase the prediction accuracy compared to the HMM alone.

	Gene Sn.	Gene Sp.	Exon Sn.	Exon Sp.	Nucl. Sn.	Nucl. Sp.
Genscan	24%	19%	60%	42%	95%	69%
Augustus	34%	43%	64%	74%	86%	97%
ExonHunter, no advisors	39%	34%	73%	63%	96%	84%
N-SCAN	42%	47%	75%	73%	94%	93%
ExonHunter	44%	39%	76%	68%	97%	92%

4.4.4 Performance on the fruit fly genome

Table 4.7 shows the performance of ExonHunter on the testing set from the fruit fly genome. We compare our performance with *ab initio* gene finders Genscan [37] and Augustus [156]. We have run Genscan with parameters trained on human sequences: these are the only animal parameters available in the Genscan distribution. Augustus predictions were produced by Augustus authors, with an HMM trained on fruit fly sequences. We have also downloaded the N-SCAN predictions [76] from the UCSC genome browser [89]. N-SCAN uses multiple alignments of the fruit fly genome, the mosquito genome, and two other fruit fly genomes, *Drosophila yakuba* and *Drosophila pseudoobscura*.

ExonHunter without advisors is more accurate than Genscan. This is because Genscan was used with parameters trained on human sequences. Both Augustus and N-SCAN have higher specificity than ExonHunter, even when we use advisors. However, ExonHunter without advisors is more sensitive than Augustus, and ExonHunter with advisors is slightly more sensitive than N-SCAN.

Advisors improve the prediction accuracy of ExonHunter, but not as dramatically as was the case for the human genome. When constructing advisors for the fruit fly genome, we have followed the same procedures that were developed for the human genome. Additional adjusting could perhaps further increase the accuracy, but this experiment demonstrates that the advisor framework can be adapted for gene finding in a new species with modest effort and still improve the results of *ab initio* methods. Our experiment on the fruit fly genome was made easier by the availability of good annotation providing a wealth of training data. To adapt ExonHunter for a newly sequenced species, we would need to estimate the parameters from very little data or from some related and better characterized species.

4.5 Summary

In this chapter we have presented the implementation of our gene finder, ExonHunter. It is based on a hidden Markov model, extended with several novel features not used in previous HMM gene finders. ExonHunter combines HMM predictions with several sources of evidence represented in the advisor framework introduced in Chapter 2. We have discussed how to express evidence from sequence repeats and protein, EST and genome alignments in the advisor framework. All advisors issue a probability distribution over a binary partition at positions for which direct evidence is available, and vacuous advice at all other position. Changes in this binary partition allow us to tailor advisors to a particular source of data. For example, intron advisors based on human ESTs use a different partition than do their counterparts based on mouse ESTs. The possibility to express incomplete information gives us flexibility in advisor design.

In tests on human genomic sequences, ExonHunter, by combining multiple sources of evidence, achieves higher sensitivity than methods that use only alignments with other genomes as a source of evidence. In parallel work with ours, several authors have created methods for combining evidence that perform better than ExonHunter on the ENCODE testing set. Jigsaw [7], the only one of these methods published so far, relies on highly accurate sources of information, such as full-length alignments of cDNA molecules. AceView [159] achieves very high sensitivity by including human experts in the annotation process. In contrast, ExonHunter works autonomously and does not require high quality information sources, such as cDNA alignments. Therefore, ExonHunter might be appropriate for newly sequenced species where less information is available.

Our experiments suggest several venues of possible improvement. First, ExonHunter sensitivity might be increased by the use of better alignments between protein databases and the query sequence, which would indicate exon boundaries, instead of only approximate locations of exons. We might also include sources of high quality information, such as databases of full-length cDNAs. ExonHunter's specificity is low because of our design decisions, but perhaps could be increased by adding more evidence in regions that do not encode proteins and by post-filtering techniques.

Finally we show that ExonHunter can be adapted for gene finding in a new species by testing it on genomic sequences from the fruit fly *Drosophila melanogaster*.

Chapter 5

Conclusion

In this thesis, we have studied two fundamental problems of genomic sequence analysis: gene finding and homology search. We have concentrated on the use of sequence homology and other sources of evidence in gene finding.

In Chapter 2, we have presented a flexible framework for combining multiple sources of evidence in gene finding. The goal of gene finding is to label the nucleotides of the query DNA sequence with a label indicating its role in the correct gene structure. We have observed that a particular source of evidence typically does not support only one label being correct at a fixed position, but rather supports a subset of the labels. This subset may be different for different evidence sources, or even for different positions of the sequence for the same source. Therefore, we turn each source of evidence into an advisor—a collection of partial probability distributions that assign probability to appropriate sets of labels at each sequence position.

Our advisor framework allows us to express many sources of evidence in a natural way, and it handles missing information transparently. However, the partial probability distributions provided by individual advisors cannot be easily combined by standard methods. Therefore, we have developed a new method for combining evidence from multiple advisors, and have studied several its variants. In the final step of our algorithm, we combine the combined prediction of all advisors with a hidden Markov model characterizing sequence features typical for gene structure elements.

The most abundant source of evidence are databases containing sequences of known proteins, mRNAs, and genomes. To use this evidence in gene finding, we need efficient and sensitive tools for finding sequence similarities between the query DNA sequence and a given database. In Chapter 3, we have constructed spaced seeds that significantly increase the number of homologous protein coding regions discovered by a homology search program. We have achieved this result by designing probabilistic models of sequence conservation in homologous coding regions and optimizing spaced seeds with respect to these models. We have also generalized spaced seeds and several other commonly used seeding approaches to a common framework, called vector seeds, that allows us to combine their advantages.

Finally in Chapter 4, we have described the many details of of our gene finder, ExonHunter. ExonHunter combines evidence from protein, EST, and genome alignments and from predicted sequence repeats. We have tested ExonHunter on human and fruit fly genomic sequences. In the tests on human test sets, ExonHunter outperforms gene finders that use only genomic alignments as a source of information. A comparison with other very recent gene finders suggests that performance of ExonHunter might be improved by more accurate protein alignments and by the use of full length cDNA libraries.

While our method allows the incorporation of a wide range of information sources, we do not require a comprehensive set of sources to be always available. When no additional information is available, the system performs as a typical *ab initio* gene finder. Adding more information helps to improve the prediction accuracy. Even information from distant species, such as using chicken sequence information to help find genes in the human sequence, increases exon sensitivity and specificity. The method easily transfers to other species, since it does not require special species-specific data sets.

Our research suggests several directions for future work. Our advisor framework considers each sequence position independently. This assumption allows us to use efficient algorithms for inference and makes the problem conceptually simpler. On the other hand, it creates problems in practice, which we address by a simple heuristics. It would be interesting to construct a system for combining information that would combine the flexibility of advisors to provide partial probabilistic statements with the ability to encode long-range dependencies between sequence positions.

Although we have created the advisor framework specifically for gene finding, perhaps it could be used also for other sequence annotation tasks where HMMs are successfully used, such as transmembrane protein topology or protein secondary structure prediction.

In our work on spaced seeds for homology search, we have concentrated on modeling properties of protein coding regions. Perhaps our models could be used to study performance of other components of homology search, or perhaps similar models can be build for other special classes of homologous sequences.

Bibliography

[1] A. Abbott. Competition boosts bid to find human genes. *Nature*, 435(7039):134, 2005.

[2] M. D. Adams, S. E. Celniker, R. A. Holt, C. A. Evans, J. D. Gocayne, P. G. Amanatides, S. E. Scherer, P. W. Li, R. A. Hoskins, R. F. Galle, et al. The genome sequence of Drosophila melanogaster. *Science*, 287(5461):2185–2185, 2000.

[3] M. D. Adams, J. M. Kelley, J. D. Gocayne, M. Dubnick, M. H. Polymeropoulos, H. Xiao, C. R. Merril, A. Wu, B. Olde, R. F. Moreno, et al. Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–1656, 1991.

[4] P. Agarwal and V. Bafna. Detecting non-adjoining correlations within signals in DNA. In *RECOMB 1998: Proceedings of the 2nd Annual International Conference on Research in Computational Molecular Biology*, pages 2–8. ACM Press, 1998.

[5] M. Alexandersson, S. Cawley, and L. Pachter. SLAM: cross-species gene finding and alignment with a generalized pair hidden Markov model. *Genome Research*, 13(3):496–502, 2003.

[6] J. E. Allen, M. Pertea, and S. L. Salzberg. Computational gene prediction using multiple sources of evidence. *Genome Research*, 14(1):142–148, 2004.

[7] J. E. Allen and S. L. Salzberg. JIGSAW: integration of multiple sources of evidence for gene prediction. *Bioinformatics*, 21(18):3596–3603, 2005.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[9] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3392, 1997.

[10] Arabidopsis Genome Initiative. Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. *Nature*, 408(6814):796–815, 2000.

[11] J. L. Ashurst, C. K. Chen, J. G. Gilbert, K. Jekosch, S. Keenan, P. Meidl, S. M. Searle, J. Stalker, R. Storey, S. Trevanion, L. Wilming, and T. Hubbard. The vertebrate genome annotation (Vega) database. *Nucleic Acids Research*, 33(Database issue):D459–465, 2005.

[12] V. B. Bajic and S. H. Seah. Dragon gene start finder: an advanced system for finding approximate locations of the start of gene transcriptional units. *Genome Research*, 13(8):1923–1929, 2003.

[13] B. T. Bartell, G. W. Cottrell, and R. K. Belew. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1994)*, pages 173–181. Springer-Verlag, 1994.

[14] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy, S. Griffiths-Jones, K. L. Howe, M. Marshall, and E. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 30(1):276–280, 2002.

[15] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10(7):950–958, 2000.

[16] L. E. Baum. 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, 1972.

[17] L. E. Baum and J. A. Eagon. 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, 1967.

[18] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 28(1):15–18, 2000.

[19] Bioinformatics Solutions Inc. PatternHunter, 2002. `http://www.bioinformaticssolutions.com`.

[20] E. Birney, M. Clamp, and R. Durbin. GeneWise and Genomewise. *Genome Research*, 14(5):988–995, 2004.

[21] B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, 2003.

[22] M. Borodovsky and J. McIninch. GENMARK: Parallel gene recognition for both DNA strands. *Computers and Chemistry*, 17(2), 1993.

[23] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[24] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[25] B. Brejová, D. G. Brown, M. Li, and T. Vinař. ExonHunter: A comprehensive approach to gene finding. *Bioinformatics*, 21(Suppl 1):i57–i65, 2005. Proceedings of the 15th International Conference on Inteligent Systems for Molecular Biology (ISMB 2005).

[26] B. Brejová, D. G. Brown, and T. Vinař. Optimal DNA signal recognition models with a fixed amount of intrasignal dependency. In *Algorithms and Bioinformatics: 3rd International Workshop (WABI)*, volume 2812 of *Lecture Notes in Bioinformatics*, pages 78–94. Springer, September 2003.

[27] B. Brejová, D. G. Brown, and T. Vinař. Optimal spaced seeds for hidden Markov models, with application to homologous coding regions. In *Combinatorial Pattern Matching, 14th Annual Symposium (CPM 2003)*, volume 2676 of *Lecture Notes in Computer Science*, pages 42–54. Springer, 2003.

[28] B. Brejová, D. G. Brown, and T. Vinař. The most probable labeling problem in HMMs and its application to bioinformatics. In *Algorithms in Bioinformatics, 4th International Workshop (WABI)*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 426–437. Springer, 2004.

[29] B. Brejová, D. G. Brown, and T. Vinař. Optimal spaced seeds for homologous coding regions. *Journal of Bioinformatics and Computational Biology*, 1(4):595–610, 2004.

[30] B. Brejová, D. G. Brown, and T. Vinař. Vector seeds: An extension to spaced seeds. *Journal of Computer and System Sciences*, 70(3):364–380, 2005.

[31] B. Brejová and T. Vinař. A better method for length distribution modeling in HMMs and its application to gene finding. In *Combinatorial Pattern Matching, 13th Annual Symposium (CPM 2002)*, volume 2373 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2002.

[32] D. G. Brown. Optimizing multiple seeds for protein homology search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):29–38, 2005.

[33] D. G. Brown and A. K. Hudek. New algorithms for multiple DNA sequence alignment. In *Algorithms in Bioinformatics, 4th International Workshop (WABI 2004)*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 314–325, Bergen, Norway, September 2004. Springer.

[34] D. G. Brown, M. Li, and B. Ma. A tutorial of recent developments in the seeding of local alignment. *Journal of Bioinformatics and Computational Biology*, 2(4):819–842, 2004.

[35] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[36] J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *Journal of Computer and System Sciences*, 70(3):342–363, 2005.

[37] C. B. Burge. *Identification of Genes in Human Genomic DNA*. PhD thesis, Department of Mathematics, Stanford University, March 1997.

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

[39] S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. *Fundamenta Informaticae*, 56(1,2):51–70, 2003.

[40] M. Burset and R. Guigó. Evaluation of gene structure prediction programs. *Genomics*, 34(3):353–357, 1996.

[41] M. Burset, I. A. Seledtsov, and V. V. Solovyev. SpliceDB: database of canonical and non-canonical mammalian splice sites. *Nucleic Acids Research*, 29(1):255–259, 2001.

[42] D. Cai, A. Delcher, B. Kao, and S. Kasif. Modeling splice sites with Bayes networks. *Bioinformatics*, 16(2):152–158, 2000.

[43] A. Califano and I. Rigoutsos. FLASH: a fast look-up algorithm for string homology. *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB 1993)*, 1:56–64, 1993.

[44] S. Chatterji and L. Pachter. Multiple organism gene finding by collapsed Gibbs sampling. In *RECOMB 2004: Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology*, pages 187–193. ACM Press, 2004.

[45] S. F. Chen and R. Rosenfeld. A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, 8:37–50, 2000.

[46] T. Chen. Gene-finding via tandem mass spectrometry. In *RECOMB 2001: Proceedings of the 5th Annual International Conference on Research in Computational Molecular Biology*, pages 87–94. ACM Press, 2001.

[47] W. Chen and W. K. Sung. On half gapped seed. In *Proceedings of 14th International Conference on Genome Informatics (GIW)*, volume 14, pages 176–185. Universal Academy Press, Tokyo, 2003.

[48] K. P. Choi, F. Zeng, and L. Zhang. Good spaced seeds for homology search. *Bioinformatics*, 20(7):1053–1059, 2004.

[49] K. P. Choi and L. Zhang. Sensitivity analysis and efficient method for identifying optimal spaced seeds. *Journal of Computer and System Sciences*, 68(1):22–40, 2004.

[50] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

[51] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[52] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–935, 2004.

[53] Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–522, 2002.

[54] M. Csűrös. Performing local similarity searches with variable length seeds. In *Combinatorial Pattern Matching, 15th Annual Symposium (CPM 2004)*, volume 3109 of *Lecture Notes in Computer Science*, pages 373–387. Springer, 2004.

[55] M. Csűrös and B. Ma. Rapid homology search with two-stage extension and daughter seeds. In *11th International Computing and Combinatorics Conference (COCOON 2005)*, 2005. To appear.

[56] V. Curwen, E. Eyras, T. D. Andrews, L. Clarke, E. Mongin, S. M. Searle, and M. Clamp. The Ensembl automatic gene annotation system. *Genome Research*, 14(5):942–950, 2004.

[57] M. Das, C. B. Burge, E. Park, J. Colinas, and J. Pelletier. Assessment of the total number of human transcription units. *Genomics*, 77(1-2):71–78, 2001.

[58] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statististical Society Series B*, 39(1):1–38, 1977.

[59] L. Ding, A. Sabo, N. Berkowicz, R. R. Meyer, Y. Shotland, M. R. Johnson, K. H. Pepin, R. K. Wilson, and J. Spieth. EAnnot: a genome annotation tool using experimental evidence. *Genome Research*, 14(12):2503–2509, 2004.

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

[61] ENCODE Project Consortium. The ENCODE (ENCyclopedia Of DNA Elements) project. *Science*, 306(5696):636–640, 2004.

[62] E. Eyras, A. Reymond, R. Castelo, J. M. Bye, F. Camara, P. Flicek, E. J. Huckle, G. Parra, D. D. Shteynberg, C. Wyss, J. Rogers, S. E. Antonarakis, E. Birney, R. Guigo, and M. R. Brent. Gene finding in the chicken genome. *BMC Bioinformatics*, 6(1):131, 2005.

[63] J. Flannick and S. Batzoglou. Using multiple alignments to improve seeded local alignment algorithms. *Nucleic Acids Research*, 33(14):4563–4567, 2005.

[64] R. Fletcher. *Practical Methods of Optimization*. Wiley, 1987.

[65] L. Florea, V. Di Francesco, J. Miller, R. Turner, A. Yao, M. Harris, B. Walenz, C. Mobarry, G. V. Merkulov, R. Charlab, I. Dew, Z. Deng, S. Istrail, P. Li, and G. Sutton. Gene and alternative splicing annotation with AIR. *Genome Research*, 15(1):54–66, 2005.

[66] L. Florea, G. Hartzell, Z. Zhang, G. M. Rubin, and W. Miller. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Research*, 8(9):967–974, 1998.

[67] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[68] B. J. Frey, Q. D. Morris, W. Zhang, N. Mohammad, and T. R. Hughes. GenRate: a generative model that finds and scores new genes and exons in genomic microarray data. *Pacific Symposium on Biocomputing (PSB)*, pages 495–506, 2005.

[69] M. A. Frohman, M. K. Dush, and G. R. Martin. Rapid production of full-length cDNAs from rare transcripts: amplification using a single gene-specific oligonucleotide primer. *Proceedings of the National Academy of Sciences of the United States of America*, 85(23):8998–9002, 1988.

[70] T. Fulton, S. Kasif, and S. Salzberg. Efficient algorithms for finding multi-way splits for decision trees. In *International Conference on Machine Learning (ICML)*, pages 244–251, 1995.

[71] M. S. Gelfand. Computer prediction of the exon-intron structure of mammalian pre-mRNAs. *Nucleic Acids Research*, 18(19):5865–5869, 1990.

[72] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 93(17):9061–9066, 1996.

[73] C. Genest and J. V. Zidek. Combining probability distributions: a critique and an annotated bibliography. *Statistical Science*, 1(1):114–148, 1986.

[74] W. Gish and D. J. States. Identification of protein coding regions by database similarity search. *Nature Genetics*, 3(3):266–272, 1993.

[75] O. Gotoh. Homology-based gene structure prediction: simplified matching algorithm using a translated codon (tron) and improved accuracy by allowing for long gaps. *Bioinformatics*, 16(3):190–202, 2000.

[76] S. S. Gross and M. R. Brent. Using multiple alignments to improve gene prediction. In *RECOMB 2005: Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology*, volume 3500 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2005.

[77] A. J. Grove and J. Y. Halpern. Probability update: Conditioning vs. cross-entropy. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 208–214. Morgan Kaufmann Publishers, 1997.

[78] R. Guigó. Assembling genes from predicted exons in linear time with dynamic programming. *Journal of Computational Biology*, 5(4):681–702, 1998.

[79] R. Guigó, P. Agarwal, J. F. Abril, M. Burset, and J. W. Fickett. An assessment of gene prediction accuracy in large DNA sequences. *Genome Research*, 10(10):1631–1632, 2000.

[80] J. Y. Halpern and R. Fagin. Two views of belief: belief as generalized probability and belief as evidence. *Artificial Intelligence*, 54(3):275–317, 1992.

[81] P. M. Harrison, D. Milburn, Z. Zhang, P. Bertone, and M. Gerstein. Identification of pseudogenes in the Drosophila melanogaster genome. *Nucleic Acids Research*, 31(3):1033–1037, 2003.

[82] S. Hashem. Optimal linear combinations of neural networks. *Neural Networks*, 10(4):599–614, 1997.

[83] J. Henderson, S. Salzberg, and K. H. Fasman. Finding genes in DNA with a Hidden Markov Model. *Journal of computational biology : a journal of computational molecular cell biology*, 4(2):127–131, 1997.

[84] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915–10919, 1992.

[85] S. Henikoff, J. G. Henikoff, and S. Pietrokovski. Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, 15(6):471–479, 1999.

[86] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998.

[87] X. Huang, M. D. Adams, H. Zhou, and A. R. Kerlavage. A tool for analyzing and annotating genomic sequences. *Genomics*, 46(1):37–45, 1997.

[88] I. P. Ioshikhes and M. Q. Zhang. Large-scale human promoter mapping using CpG islands. *Nature genetics*, 26(1):61–63, 2000.

[89] D. Karolchik, R. Baertsch, M. Diekhans, T. S. Furey, A. Hinrichs, Y. T. Lu, K. M. Roskin, M. Schwartz, C. W. Sugnet, D. J. Thomas, R. J. Weber, D. Haussler, and W. J. Kent. The UCSC genome browser database. *Nucleic Acids Research*, 31(1):51–54, 2003.

[90] E. Keibler and M. R. Brent. Eval: a software package for analysis of genome annotations. *BMC Bioinformatics*, 4(1):50, 2003.

[91] U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, 2004.

[92] W. J. Kent. BLAT–the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, 2002.

[93] W. J. Kent and A. M. Zahler. Conservation, regulation, synteny, and introns in a large-scale C. briggsae-C. elegans genomic alignment. *Genome Research*, 10(8):1115–1125, 2000.

[94] D. Kisman, M. Li, B. Ma, and L. Wang. tPatternHunter: gapped, fast and sensitive translated homology search. *Bioinformatics*, 21(4):542–544, 2005.

[95] I. Korf, P. Flicek, D. Duan, and M. R. Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17(S1):S140–148, 2001. ISMB 2001.

[96] A. Krogh. Two methods for improving performance of an HMM and their application for gene finding. In *Proceedings of the fifth International Conference on Intelligent Systems for Molecular Biology (ISMB 1997)*, pages 179–186, 1997.

[97] A. Krogh. Using database matches with for HMMGene for automated gene detection in Drosophila. *Genome Research*, 10(4):523–528, 2000.

[98] A. Krogh, B. Larsson, G. von Heijne, and E. L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *Journal of Molecular Biology*, 305(3):567–570, 2001.

[99] A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in E. coli DNA. *Nucleic Acids Research*, 22(22):4768–4768, 1994.

[100] G. V. Kryukov, S. Castellano, S. V. Novoselov, A. V. Lobanov, O. Zehtab, R. Guigo, and V. N. Gladyshev. Characterization of mammalian selenoproteomes. *Science*, 300(5624):1439–1443, 2003.

[101] G. Kucherov, L. Noé, and Y. Ponty. Estimating seed sensitivity on homogeneous alignments. In *4th IEEE International Symposium on BioInformatics and BioEngineering (BIBE 2004)*, pages 387–394, 2004.

[102] G. Kucherov, L. Noé, and M. Roytberg. Multiseed lossless filtration. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):51–61, 2005.

[103] G. Kucherov, L. Noé, and M. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. In *Algorithms in Bioinformatics, 5th International Workshop (WABI 2005)*, 2005. To appear.

[104] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[105] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden Markov model for the recognition of human genes in dna. *Proceedings of the fourth International Conference on Intelligent Systems for Molecular Biology (ISMB 1996)*, 4:134–142, 1996.

[106] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. Integrating database homology in a probabilistic gene structure model. *Pacific Symposium on Biocomputing (PSB)*, pages 232–234, 1997.

[107] L. I. Kuncheva. Combining classifiers: Soft computing solutions. In S.K. Pal and A. Pal, editors, *Pattern Recognition: From Classical to Modern Approaches*, pages 427–452. World Scientific, 2002.

[108] L. Lam and C. Y. Suen. Optimal combinations of pattern classifiers. *Pattern Recognition Letters*, 16(9):945–954, 1995.

[109] J. Li and W. Miller. Significance of inter-species matches when evolutionary rate varies. In *RECOMB 2002: Proceedings of the 6th Annual International Conference on Research in Computational Biology*, pages 216–224. ACM Press, 2002.

[110] M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly sensitive and fast homology search. *Journal of Bioinformatics and Computational Biology*, 2(3):417–439, 2004.

[111] M. Li, B. Ma, and L. Zhang. Superiority and complexity of the spaced seeds. In *SODA 2006: Proceedings of the 10th annual ACM-SIAM symposium on Discrete algorithms*, 2006. To appear.

[112] L. P. Lim and C. B. Burge. A computational analysis of sequence features involved in recognition of short introns. *Proceedings of the National Academy of Sciences of the United States of America*, 98(20):11193–11198, 2001.

[113] P. Lio, N. Goldman, J. L. Thorne, and D. T. Jones. PASSML: combining evolutionary inference and protein secondary structure prediction. *Bioinformatics*, 14(8):726–733, 1998.

[114] A. V. Lukashin and M. Borodovsky. GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.

[115] R. B. Lyngsø and C. N. S. Pedersen. Complexity of comparing hidden Markov models. In *Algorithms and Computation, 12th International Symposium (ISAAC)*, volume 2223 of *Lecture Notes in Computer Science*, pages 416–428. Springer, 2001.

[116] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, March 2002.

[117] W. H. Majoros, M. Pertea, and S. L. Salzberg. TigrScan and GlimmerHMM: two open source ab initio eukaryotic gene-finders. *Bioinformatics*, 20(16):2878–2879, 2004.

[118] J. D. McAuliffe, L. Pachter, and M. I. Jordan. Multiple-sequence functional annotation and the generalized hidden Markov phylogeny. *Bioinformatics*, 20(12):1850–1850, 2004.

[119] S. McGinnis and T. L. Madden. BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*, 32(Web Server issue):W20–25, 2004.

[120] I. M. Meyer and R. Durbin. Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, 18(10):1309–1318, 2002.

[121] K. Murakami and T. Takagi. Gene recognition by combination of several gene-finding programs. *Bioinformatics*, 14(8):665–675, 1998.

[122] F. Nicolas and E. Rivals. Hardness of optimal spaced seed design. In *Combinatorial Pattern Matching, 16th Annual Symposium (CPM 2005)*, volume 3537 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2005.

[123] L. Noé and G. Kucherov. Improved hit criteria for DNA local alignment. *BMC Bioinformatics*, 5:149, 2004.

[124] L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33(Web Server issue):W540–543, 2005.

[125] P. S. Novichkov, M. S. Gelfand, and A. A. Mironov. Gene recognition in eukaryotic DNA by comparison of genomic sequences. *Bioinformatics*, 17(11):1011–1018, 2001.

[126] Numerical Algorithm Group, Oxford, U.K. NAG C library, mark 5, 1999. `http://www.nag.co.uk/numeric/`.

[127] G. Parra, P. Agarwal, J. F. Abril, T. Wiehe, J. W. Fickett, and R. Guigó. Comparative gene prediction in human and mouse. *Genome Research*, 13(1):108–117, 2003.

[128] V. Pavlović, A. Garg, and S. Kasif. A Bayesian framework for combining gene predictions. *Bioinformatics*, 18(1):19–27, 2002.

[129] N. Pavy, S. Rombauts, P. Dehais, C. Mathe, D. V. Ramana, P. Leroy, and P. Rouze. Evaluation of gene prediction software using a genomic data set: application to Arabidopsis thaliana sequences. *Bioinformatics*, 15(11):887–889, 1999.

[130] J. S. Pedersen and J. Hein. Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics*, 19(2):219–227, 2003.

[131] M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall, 1993.

[132] F. P. Preparata, A. M. Frieze, and E. Upfal. On the power of universal bases in sequencing by hybridization. In *RECOMB 1999: Proceedings of the 3rd Annual International Conference on Research in Computational Molecular Biology*, pages 295–301. ACM Press, 1999.

[133] K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI reference sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, 33(Database issue):D501–504, 2005.

[134] J. Quackenbush, J. Cho, D. Lee, F. Liang, I. Holt, S. Karamycheva, B. Parvizi, G. Pertea, R. Sultana, and J. White. The TIGR Gene Indices: analysis of gene transcript sequences in highly sampled eukaryotic species. *Nucleic Acids Research*, 29(1):159–164, 2001.

[135] L. R. Rabiner. A tutorial on Hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[136] M. G. Reese, G. Hartzell, N. L. Harris, U. Ohler, J. F. Abril, and S. E. Lewis. Genome annotation assessment in Drosophila melanogaster. *Genome Research*, 10(4):483–501, 2000.

[137] RIKEN Genome Exploration Research Group Phase II Team and the FANTOM Consortium. Functional annotation of a full-length mouse cDNA collection. *Nature*, 409(6821):685–690, 2001.

[138] H. Roest Crollius, O. Jaillon, A. Bernot, C. Dasilva, L. Bouneau, C. Fischer, C. Fizames, P. Wincker, P. Brottier, F. Quetier, W. Saurin, and J. Weissenbach. Estimate of human gene number provided by genome-wide analysis using Tetraodon nigroviridis DNA sequence. *Nature Genetics*, 25(2):235–238, 2000.

[139] S. Rogic, A. K. Mackworth, and F. B. Ouellette. Evaluation of gene-finding programs on mammalian sequences. *Genome Research*, 11(5):817–822, 2001.

[140] S. Rogic, B. F. Ouellette, and A. K. Mackworth. Improving gene recognition accuracy by combining predictions from two gene-finding programs. *Bioinformatics*, 18(8):1034–1035, 2002.

[141] M. S. Rosenberg, S. Subramanian, and S. Kumar. Patterns of transitional mutation biases within and among mammalian genomes. *Molecular Biology and Evolution*, 20(6):988–993, 2003.

[142] G. M. Rubin, M. D. Yandell, J. R. Wortman, G. L. Gabor Miklos, C. R. Nelson, I. K. Hariharan, M. E. Fortini, P. W. Li, R. Apweiler, W. Fleischmann, et al. Comparative genomics of the eukaryotes. *Science*, 287(5461):2204–2205, 2000.

[143] A. A. Salamov and V. V. Solovyev. Ab initio gene finding in Drosophila genomic DNA. *Genome Research*, 10(4):516–522, 2000.

[144] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26(2):544–548, 1998.

[145] T. Schiex, A. Moisan, and P. Rouzé. EUGÈNE: An eukaryotic gene finder that combines several sources of evidence. In *Computational Biology. Selected papers from First International Conference on Biology, Informatics, and Mathematics*, volume 2066 of *Lecture Notes in Computer Science*, pages 111–125, 2000.

[146] A. Schneider, G. M. Cannarozzi, and G. H. Gonnet. Empirical codon substitution matrix. *BMC bioinformatics*, 6(1):134, 2005.

[147] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13(1):103–107, 2003.

[148] G. Shafer. *A mathematic theory of evidence*. Princeton University Press, 1976.

[149] D. D. Shoemaker, E. E. Schadt, C. D. Armour, Y. D. He, P. Garrett-Engele, P. D. McDonagh, P. M. Loerch, A. Leonardson, P. Y. Lum, G. Cavet, et al. Experimental annotation of the human genome using microarray technology. *Nature*, 409(6822):922–927, 2001.

[150] J. E. Shore and R. W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26:26–37, 1980.

[151] A. Siepel and D. Haussler. Computational identification of evolutionarily conserved exons. In *RECOMB 2004: Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology*, pages 177–186. ACM Press, 2004.

[152] A. F. A. Smit, R. Hubley, and P. Green. RepeatMasker. `http://www.repeatmasker.org`, 2002.

[153] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[154] E. E. Snyder and G. D. Stormo. Identification of protein coding regions in genomic DNA. *Journal of molecular biology*, 248(1):1–18, 1995.

[155] M. Stanke, O. Schöffmann, B. Morgenstern, and S. Waack. Gene prediction in eukaryotes with a generalized hidden Markov model that takes hints. Submitted for publication, 2005.

[156] M. Stanke and S. Waack. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*, 19(S2):II215–II225, 2003. ECCB 2003.

[157] Y. Sun and J. Buhler. Designing multiple simultaneous seeds for DNA similarity search. *Journal of Computational Biology*, 12(6):847–851, 2005.

[158] D. M. J. Tax, M. van Breukelen, R. P. W. Duin, and J. Kittler. Combining multiple classifiers by averaging or multiplying? *Pattern Recognition*, 33:1475–1485, 2000.

[159] D. Thierry-Mieg, J. Thierry-Mieg, M. Potdevin, and M. Sienkiewicz. Identification and functional annotation of cDNA-supported genes in higher organisms using AceView. Unpublished. `http://www.aceview.org/`.

[160] D. Torrents, M. Suyama, E. Zdobnov, and P. Bork. A genome-wide survey of human pseudogenes. *Genome research*, 13(12):2559–2567, 2003.

[161] A. Ureta-Vidal, L. Ettwiller, and E. Birney. Comparative genomics: genome-wide analysis in metazoan eukaryotes. *Nature Reviews Genetics*, 4(4):251–252, 2003.

[162] S. A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, 1991.

[163] V. Veeramachaneni, W. Makalowski, M. Galdzicki, R. Sood, and I. Makalowska. Mammalian overlapping genes: the comparative perspective. *Genome Research*, 14(2):280–286, 2004.

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

[165] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[166] J. Vomlel. Integrating inconsistent data in a probabilistic model. *Journal of Applied Non-Classical Logics*, 14(3):365–386, 2004.

[167] E. Wagner and J. Lykke-Andersen. mRNA surveillance: the perfect persist. *Journal of Cell Science*, 115(15):3033–3038, 2002.

[168] J. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[169] T. D. Wu and C. K. Watanabe. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics*, 21(9):1859–1865, 2005.

[170] J. Xu, D. G. Brown, M. Li, and B. Ma. Optimizing multiple spaced seeds for homology search. In *Combinatorial Pattern Matching, 15th Annual Symposium (CPM 2004)*, volume 3109 of *Lecture Notes in Computer Science*, pages 47–58. Springer, 2004.

[171] Y. Xu, J. R. Einstein, R. J. Mural, M. Shah, and E. C. Uberbacher. An improved system for exon recognition and gene modeling in human DNA sequences. In *Proceedings of the second International Conference on Intelligent Systems for Molecular Biology (ISMB 1994)*, pages 376–384, 1994.

[172] Z. Yang. Maximum-likelihood estimation of phylogeny from dna sequences when substitution rates differ over sites. *Molecular Biology and Evolution*, 10(6):1396–1401, 1993.

[173] R. F. Yeh, L. P. Lim, and C. B. Burge. Computational inference of homologous gene structures in the human genome. *Genome Research*, 11(5):803–806, 2001.

[174] M. Q. Zhang and T. G. Marr. A weight array method for splicing signal analysis. *Computer Applications in Biosciences*, 9(5):499–509, 1993.

[175] Z. Zhang and W. J. Henzel. Signal peptide prediction based on analysis of experimentally verified cleavage sites. *Protein Science*, 13(10):2819–2824, October 2004.