A Plasmodium falciparum Genefinder

Honours research project

Anthony Ian Wirth (20712)

Author address:

Department of Mathematics and Statistics, University of Melbourne, Parkville VIC, 3052

Email address: awirth@ms.unimelb.edu.au

Acknowledgements

Terry Speed (WEHI) for his supervision.

Robert Heustis (QIMR), Allan Saul (QIMR), Alan Cowman (WEHI) and Mauro DeLorenzi (WEHI) for their advice on P. falciparum.

Chaitanya Rao (Melb. Uni.) and Andrew Wirth (Melb. Uni.) for proofreading this report.

My family and Alex for their support during the year.

ABSTRACT. An probabilistic model of the Plasmodium falciparum genome is presented. The model is used in a computer program to predict the location of protein coding regions. Its input is not limited to single genes but may consist of arbitrary fragments of P. falciparum DNA. The model is constructed as a generalized hidden Markov model (GHMM) whose progenitor, the hidden Markov model, is also described. The GHMM has states for intergene, exon and intron sections of both strands of the P. falciparum DNA. Functional site detection is only implemented at a basic level: with state prediction based primarily on nucleotide composition. Without an algorithm for parameter estimation, maximum likelihood estimates based on frequency counts from annotated training sequence are used to estimate the parameters of the model. An effort is made to clear up a few of the unresolved probabilistic issues in GHMM application. The model is able to predict whether individual nucleotides are in exons with remarkable accuracy, but due to the lack of motif detection has difficulty inferring overall exon structure. Nevertheless, the program is so far the first integrated computational genefinder for P. falciparum.

This report, and the research that is described in it, is dedicated to the memory of my step-grandfather, Paul Rosta, who always hoped I would do something medically related.

Contents

CHAPTER 1

A brief introduction to genes and proteins

Life is controlled by enzymes, biological catalysts. To understand how enzymes are produced we need to look into the cell, into the world of macromolecules. The information for the production of all enzymes is stored on chromosomes, single DNA molecules. Deoxyribonucleic acid (DNA) is a macromolecule with two sugar-phosphate chains wound around in the shape of a double helix. Attached to each sugar molecule is a "base" molecule which is one of Adenine (`A`), Cytosine (`C`), Guanine (`G`) or Thymine (`T`). Each building block of a sugar, a phosphate and a base is called a nucleotide. The bases attached to the two helix chains bond in pairs, so that an `A` on one chain is always matched with a `T` on the other (and vice versa), and a `C` is always lined up with `G` (and vice versa). These are known as complementary base pairs and hence the two strands are considered to be complements of each other (see Figure 1.1).

Proteins are also macromolecules, consisting of a chain of amino acids, of which there are twenty useful for life. Most proteins are in fact enzymes. The segments of DNA which are involved in the production of proteins are called genes, with each gene responsible for a single protein. In humans approximately 1% of the genome is genes.

The other key macromolecule used in the production of proteins is ribonucleic acid (RNA). Like DNA, RNA has a sugar/phosphate backbone, but unlike DNA has only one strand and has a slightly different type of sugar. Instead of the base thymine, RNA uses uracil (`U`) which has essentially the same function as thymine.

The production of proteins begins when the two strands of DNA are separated and a new RNA molecule, the complement of one DNA strand, is made by the enzyme RNA polymerase. This new RNA is called messenger RNA (mRNA) and the process creating it is called transcription. At this point the process in eukaryotes (organisms whose cells have a nucleus: humans, yeast,

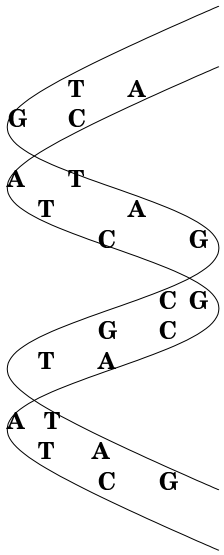

FIGURE 1.1. The DNA double helix

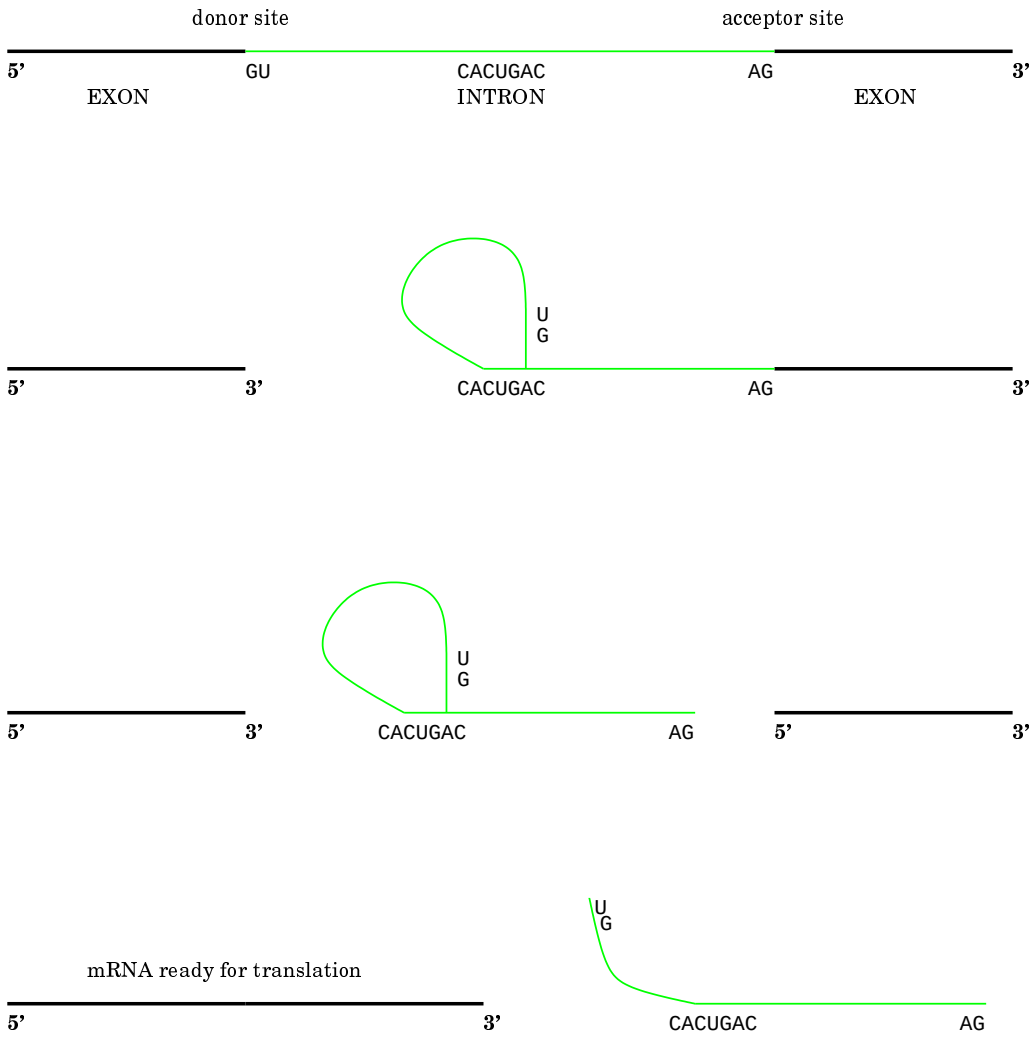

FIGURE 1.2. Stages of the splicing process, adapted from [**Lew87**]. The bases labelled in the middle are called the branch point: in this report they will be ignored.

Plasmodium falciparum etc.) undergoes a few extra steps from that in prokaryotes (those without cell nuclei: bacteria etc.). Firstly, an "upside down" guanine base, called a cap, is attached to the start of the mRNA. Secondly, a long chain of adenine nucleotides (up to a few hundred), called a poly-A tail, is added to the end of the mRNA. Thirdly and most interestingly, a complex of molecules called a spliceosome chops loops of RNA out of the mRNA chain, in a process is known as splicing. The parts of DNA whose mRNA complements are removed are known as introns, the other segments of the gene are called exons. It should be noted that not all genes in eukaryotes have introns and that the exact purpose of introns is not known. The loops having been removed, the rest of the mRNA (that coded by exons) is joined together (see Figure 1.2). Henceforth the eukaryote protein production process parallels the prokaryote process.

There is, however, one other peculiarity of eukaryotic DNA: there are sequences of bases which are repeated many, many times. For example, there is one sequence of about 300 base pairs (bp) which is found almost a million times in the human genome. In fact humans contain more repeat DNA than genes. The origin of such repeats in unclear: one theory is that they could be a result of

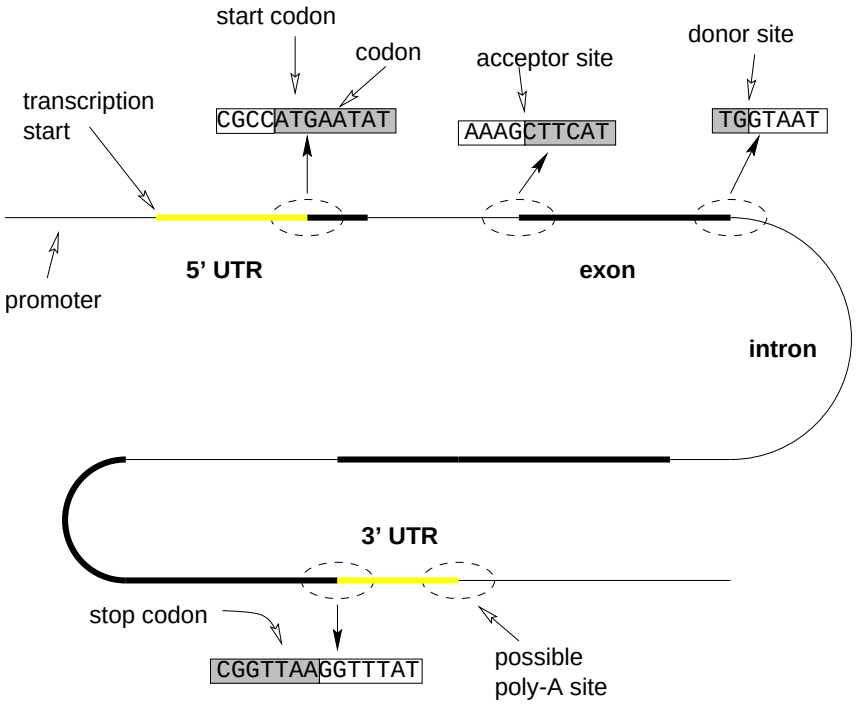

FIGURE 1.3. The general structure of the gene, adapted from [**Kro98**].

retrovirus infection long ago. Retroviruses such as HIV can attach a DNA copy of their RNA onto the host cell's DNA. Somehow the cell manages to deactivate any genes with interspersed repeats.

At this point the mRNA contains the information for the protein. Each triplet of mRNA bases (known as a codon) codes for a single amino acid. Since there are $4^3 = 64$ possible codons and only 20 amino acids, there is obviously some redundancy in the genetic code. This code is common to essentially all life forms. With the help of ribosomes which govern the procedure, transfer RNA (tRNA) molecules translate each codon into an amino acid. Translation always begins with the first `AUG` codon in the mRNA. As the mRNA is being read, the ribosome binds the new amino acids into a single protein chain. Three of the RNA codons (`UAA`, `UAG`, `UGA`) have no tRNA to match them to an amino acid, so when they are read in, translation ceases and the new protein is released.

As a matter of notation, we always refer to the DNA strand that "matches" the mRNA except that `T` is used instead of `U`—this the complementary strand to the one that is actually transcribed into mRNA. For example the start codon in DNA is deemed to be `ATG`. The section of the first exon that is not translated but only transcribed to mRNA is called 5' untranslated region (UTR). (5' essentially stands for left.) Similarly the part of the last exon that is not translated is known as 3' UTR (see Figure 1.3). 5' of the 5' UTR area is a sequence of nucleotides known as a promoter which is involved in binding the polymerase to the DNA before transcription starts. At the end of the 3' UTR is the poly-`A` site which is a signal for all the `A`s to the added to the end of the mRNA. The point in the DNA where an exon ends and the intron starts is known as a splice donor site whereas the end of the intron is known as the splice acceptor site. Almost always the first two bases of each intron are `GT` and the last two are `AG` [**SB97**]. An excellent introduction to the above ideas can be found in [**GW91**] with a detailed description in [**Lew87**].

CHAPTER 2

Plasmodium falciparum

2.1. What is P. falciparum?

Until the onset of HIV/AIDS malaria was arguably the most important Third World disease [**HS87**]. It is caused by protozoa of the genus Plasmodium. In general malaria results in headaches, spiking fever and chills with complications including convulsions and anaemia [**Pow89**]. These symptoms may disappear and reappear periodically. Infection with the species Plasmodium falciparum may cause death within two weeks if untreated. P. falciparum usually enters the body through a bite from a female anopheline mosquito. Despite an attempt at eradication in the 1950s and 60s the disease in still prevalent in tropical and sub-tropical areas of Africa, Asia and South and Central America with little natural immunity. Recent figures suggest that there are up to 2.7 million deaths each year and 300–500 million cases worldwide [**D$^+$96**].

As well as being the most lethal species, Plasmodium falciparum is also most resistant to current drugs [**SW98**]. Its genome consists of fourteen chromosomes with lengths ranging from 0.8 to 3.5 megabases (Mb) making a total of between 25 and 30 million base pairs (roughly 1% of the amount of human DNA). According to Su and Wellems [**SW98**], there is one gene every three to five kb, 2kb may well be a more accurate figure for the gap [**Cow98**]. Most genes have no introns, but in those that do the introns tend to be less than 1 kb. The 5' UTR section has between 500 and 1000 base-pairs (bp) whilst the 3' UTR usually has fewer than 400 bp and often has a large poly-`A` region. The genome has an unusually high `AT` content: 82% overall and perhaps over 90% in the intergenic regions, compared with roughly 50% in many bacteria and in humans.

2.2. Why do we need a genefinder for P. falciparum?

The principal justification for computational genefinding is efficiency. Finding the structure of genes and *attempting* to infer their function is much faster and cheaper using a computer than carrying out experiments in a wet lab. Of course lab experiments still need to be conducted to verify the computer predictions and to firmly establish biological function. Despite this Fickett [**Fic96**] believes that experimental methods have their limitations in "isolating, expressing and determining the functions of genes". Recent figures estimate that 2Mb of human DNA is being sequenced each day, that this rate will persist over the next seven years [**REKH97**], and that 30 complete microbial genomes, with average size exceeding 1Mb [**SDKW98**] will be sequenced in the next two years. With such volumes of data emerging speed is essential in determining the proteins produced by the DNA and in identifying candidate disease genes. Lukashin and Borodovsky [**LB98**] cite the case of ten bacterial genomes sequenced at the end of 1997 whose genes were found in the main by computational methods.

So far the only paper on Plasmodium falciparum (Pf) genefinding listed in the web databases [**Gel97**] and [**Li98**] has been that of Saul and Battistutta [**SB88**]. This paper describes a method of obtaining a coding score for each nucleotide position but is not a fully integrated genefinder (see section 3.4). Nevertheless, the idea could be used in a neural network genefinder.

Su and Wellems [**SW98**] report that "Programs designed to recognize genes from their structural characteristics in other organisms have often not given correct predictions with P. falciparum." Although 30 to 40% of Pf ORFs are similar to sequences in GenBank, the similarity in sequence may not extend to similarity in the biological function of the genes. (An ORF or *open reading frame*, is a sequence of codons with no internal stop codons that has the potential to code

for a protein.) Anecdotal evidence suggests that GENSCAN, the leading genefinder for human DNA, performs poorly on Pf DNA [**DeL98**]. GENSCAN's performance tends to be better when trained with maize DNA instead of human, possibly because P. falciparum has some evolutionary connection to plants. In summary, progress on a computational genefinder has been limited and with the completion of sequencing likely to be less than two years away, innovation is required as soon as possible.

CHAPTER 3

How do you find genes in DNA?

There are three main approaches to the gene finding process:

1. Searching databases for similarities to known sequences
2. Examining variations in nucleotide composition
3. Recognizing signals for cellular processes

Before using these methods this it is essential to scan for and remove repeats from eukaryote DNA. Repeats can clog up database searches as they score too many hits: it is best to mask them before further analysis. Moreover, it is rare for repeats to exist in active genes, so removing them can generally only aid genefinding.

3.1. Database homology searches

The oldest (and perhaps most reliable) method of gene finding is searching for a homolog to the sequence in some database. Two sequences are homologs if they have some common (evolutionary) ancestor. So strictly speaking, the search is for similar and *hopefully* homologous sequences. There are several different types of database search possible [**BK98**] of which the most common is to translate the DNA sequence in all six reading frames and then compare the output with protein databases or databases of functional motifs. The drawback with this approach is that even for human DNA only half of new proteins found are actually in databases. On the other hand the main benefit of database search methods is the possibility of establishing the biological function of the DNA sequence due to homology to a known protein, even from another organism.

3.2. Base composition variation

Many computational genefinders distinguish coding and non-coding regions by noticing statistical differences in nucleotide use between the two regions. Interestingly the cellular processes do not seem to use this concept in recognizing genes. Fickett and Tung [**FT92**] carried out an in depth examination of measures of protein coding potential, reaching the conclusion that reading-frame-specific dicodon usage gave the best discrimination. Most subsequent programs have followed this finding. It is worth noting that other possible measures include monitoring sequence periodicity and homogeneity, and the occurrence of potential ORFs [**Fic96**], which could all be integrated in some kind of neural network. The principal disadvantage of this approach is that the prediction of the location the boundaries between coding and non-coding regions tends to be imprecise. Despite this, measures of protein coding potential provide a method of classifying short fragments of DNA which are devoid of genetic signals (see next section).

3.3. Functional site recognition

The best way to find genes would be to model exactly the processes inside the cell. It would be nice if we could mimic with statistical models the mechanisms of transcription and RNA processing. They are however much too complex to model, so we look for signals in the DNA that direct these processes. The most common transcription signals used are the initiator (cap) signal for transcription and the `AT`-rich `TATA` box signal some 30bp 5' of the start of transcription. Unfortunately these signals only occur in 70% of human promoters [**BK98**]. In fact, regardless of the presence of such signals, it is difficult to predict precisely promoter position. At the 3' end

of the gene, the poly-A signal, twenty to thirty base-pairs 3' of the translation end, has a simple consensus sequence AATAAA, although this signal only appears in less than half of all 3' UTRs.

The use of consensus words is widespread in sequence analysis. A consensus word is obtained by aligning a set of sample sequences which have some common biological function. At each sequence position we select the nucleotide which is most common and from that form the consensus word. We then try to match the consensus word with a new sequence to find possible functional sites. This is quite a crude method as it gives no information about the frequency of other possible bases at each position and strict comparisons to consensi tend to give "only a very rough functional mapping of a sequence" [**Gel95**].

The main signals used to indicate the beginning and end of translation are the Kozak signal and the stop codon. The Kozak signal, located just upstream of the first (coding) ATG, points to the start of translation whereas the absence of an in frame stop codon shows that the coding region hasn't ended yet. These signals are so weak that prediction of the initial and terminal ends of the translated region is considerably worse than the prediction of splice sites. As it happens P. falciparum doesn't use the Kozak signal [**Cow98**].

Splicing signals indicate the boundary between exons and introns. At the very least the minimal GT and AG consensi can be used to predict splice sites. As well as larger consensus words there are a number of matrix models for splice site recognition. To create a position-specific scoring matrix (PSSM) we note in our aligned training sample the normalized frequency $f(b(x), x)$ of each base $b(x)$ in each position x. Therefore under the functional site model M a sequence O has probability

$$\mathbf{P}[O|M] = \prod_x f(b(x), x)$$

assuming independence of positions. Of course the next step is to come up with some model of "non-sites" $\bar{M}$ and assign a probability to sequences at non-sites. A simple choice for Pf would be to use the general distribution of bases, (p_A, p_C, p_G, p_T), at each position. Given a sequence O a suitable score for it is the log likelihood ratio of site to non-site being

$$\log \frac{\mathbf{P}[O|M]}{\mathbf{P}[O|\bar{M}]} = \sum_x \log \frac{f(b(x), x)}{p_{b(x)}}.$$

There may be biological reasons for the independence assumption in the PSSM above to be false. New models, specifically the Maximal Dependence Decomposition method [**BK97**], make use of the dependencies between base positions around splice sites. Unless incorporated into an integrated genefinder, neither type of scoring model performs especially well. Other methods involve formal grammars, perceptrons and neural networks. Importantly one can use a hidden Markov model (HMM) to try to match a new sequence to an aligned family. Profile HMMs [**KBM+94, DEKM98**], which were the first widespread type of biological HMMs, also have the ability to model insertions and deletions in sequences (potentially) belonging to the family.

3.4. Putting it all together

An integrated gene finder uses all of the above methods to predict the entire exon-intron structure of a sequence. A brute force approach is to consider all candidate start, stop and splice sites and to assign a score to all legal parses, but the computational blow-out is enormous. To make the problem tractable impractical restrictions (on exon length, for example) need to be imposed. Nevertheless a sensible integrated program will clearly perform much better than models based on only one of the methods in the previous paragraphs. If, for example, the program guesses the presence of a splice site using a matrix method, then it knows that it should use the exon nucleotide composition model on one side and the intron model on the other. The Hidden Markov Model is able to incorporate many submodels each returning probabilities of various events and combine them in a meaningful way. In the next chapter we examine the basics of this type of model in detail, and following that we see how a generalization of Hidden Markov Models is used to predict gene structure in Pf DNA sequences.

The principal faults of contemporary (integrated) gene finders are that they are unable to cope with alternate splicing patterns in the one gene, cannot deal with overlapping genes and that they tend to be too sensitive to small sequencing errors. All the same Fickett [**Fic96**] points out that

> It is remarkable that current algorithms work as well as they do, given that they make use of only a rather small fraction of available biological knowledge. The 'understanding' of genes implicit in any of the current generation of programs could be written down in two pages, but, of course, the underlying biology of even fairly general cases is far more complex than this.

CHAPTER 4

Hidden Markov Models

Before examining my program for Pf genefinding it is necessary to look briefly at Hidden Markov Models in general. A detailed summary which is the basis of this chapter is provided in [**Rab89**]. Hidden Markov models can be used to model real world signals, whether they are characters from some alphabet, speech patterns or temperature readings. There are three reasons for modelling signals:

1. A model for the signal can be used to derive the theory for a signal processing program which in turn will provide useful output from signals;
2. It is possible to learn something about the source of the signal, even if it cannot be seen;
3. Signal models work well in making predictions and recognitions. This has to be the most compelling reason for their creation.

The application of statistical models such as Gaussian, Poisson and Markov processes requires the estimation of the statistical properties of the signal and the parameters of the random process.

4.1. What is a Hidden Markov Model?

Hidden Markov Models are an extension to discrete-time Markov processes. Continuous time HMMs also exist but they are not relevant to the task of genefinding. Recall that the key components of a (time-homogeneous and first-order) Markov process are: a set of states $S = \{S_1, \ldots, S_N\}$, an initial state distribution vector $\pi = (\pi_1, \ldots, \pi_N)$, and a matrix $A = \{\{a_{ij}\}\}$ of state transition probabilities. Let the state at time t be denoted by q_t so that

$$a_{ij} = \mathbf{P}[q_{t+1} = S_j | q_t = S_i] = \mathbf{P}[q_{t+1} = S_j | q_t = S_{i_0}, q_{t-1} = S_{i_1}, q_{t-2} = S_{i_2}, \ldots].$$

This type of model is useful when it is possible to observe directly the state of the system. In the gambler's ruin problem, for example, the current state of the system, the amount of money the gambler has, can be observed. In the genefinding problem, the states might be exon, intron and intergene, but it is not possible to see this sequence of states by looking at the DNA. Instead we see the *signal*—the sequence of nucleotides—and we would like to model this signal as having been produced by a Markov process on the above states.

To extend the model, consider an alphabet of symbols $V = \{v_1, \ldots, v_M\}$ that are used in the observed signal $O = O_1 \ldots O_T$. For DNA this will be {A,C,G,T}. To conserve space later let O_a^b stand for $O_a \ldots O_b$ and Q_a^b stand for $q_a, \ldots q_b$. As the Markov process goes on, as well as moving between states, the system emits a symbol after it enters each state. The symbol emitted is determined by the symbol emission distribution of the state. So the other new component of the HMM is the matrix of state-symbol probabilities $B = \{\{b_{ik}\}\}$ where

$$b_{ik} = \mathbf{P}[O_t = v_k | q_t = S_i].$$

In general time-homogeneity is not required and the process can be described thus:

1. $\{(q_t, O_t)\}$ is a Markov process;
2. $\mathbf{P}[q_{t+1}, O_{t+1} | \text{past}] = \mathbf{P}[q_{t+1} | q_t] \mathbf{P}[O_{t+1} | q_{t+1}] = a_{q_t q_{t+1}} b_{q_{t+1} O_{t+1}}$.
   Note that dependence on the past could be generalized so that

   $$\mathbf{P}[q_{t+1}, O_{t+1} | \text{past}] = \mathbf{P}[q_{t+1} | Q_{t-k}^t] \mathbf{P}[O_{t+1} | O_{t-l}^t, Q_{t-m}^{t+1}],$$

   but the focus of most of this chapter will be on the standard first-order HMM.

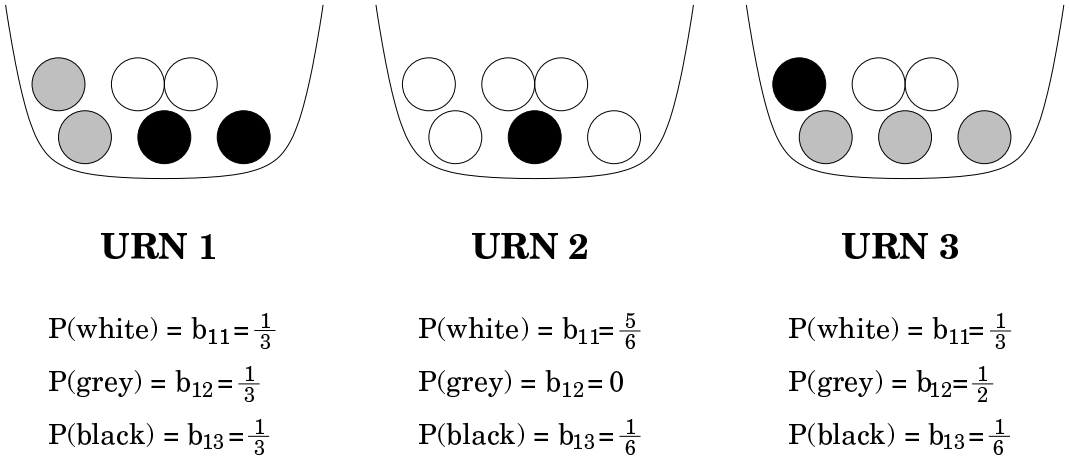

FIGURE 4.1. Urn and ball model of HMM

Rabiner [**Rab89**] uses the following example to help explain the workings of the HMM. Consider a set of three urns with various coloured balls in them (Figure 4.1). The urns are hidden in a room by a ghost. The ghost randomly selects an urn, using the Markov property, then randomly picks a ball from the selected urn. The ghost shows you the ball and you note its colour. The ghost then replaces the ball in the urn, selects the next urn using the Markov property on the urn sequence, picks out a ball from the new urn and so forth. The sequence you observe is obviously the sequence of ball colours. The generation of a HMM observation sequence can be expressed more formally:

1. Use π to choose an initial state q_1.
2. $t \leftarrow 1$
3. Use the distribution of symbols for state q_t—that is $(b_{q_t 1}, b_{q_t 2}, \ldots, b_{q_t M})$—to generate O_t.
4. Move to a new state q_{t+1} according to transition distribution $(a_{q_t 1}, a_{q_t 2}, \ldots, a_{q_t N})$.
5. $t \leftarrow t + 1$, if $t \leq T$ goto step 3, else exit.

Again, this procedure will work without the time homogeneity assumption.

Semi-Markov processes, where the time between transitions is a random variable based on the states in the transition, and Kalman filters, where we are trying to estimate a (continuous) state sequence from a series of observations, are precursors to the HMM. There will no doubt be a number of abuses of this notation in the following sections. Occasionally terms such as a_{iq_t} and b_{iO_t} will be used, and although not strictly correct their interpretation should be clear.

4.2. What can we do with HMMs?

Let λ stand for the parameter set of a particular (time-homogeneous) HMM: $\lambda = (\pi, A, B)$. The three main problems to solve for the HMM (with the genetic interpretation attached) are:

1. Given a sequence of symbols $O = O_1 \ldots O_T$ what is $\mathbf{P}[O|\lambda]$?
   What is the probability of a sequence of nucleotides?
2. Given O how do we select the optimal sequence of states $\widehat{q_1}, \ldots, \widehat{q_T}$ that produced that model? Except in certain degenerate cases there is no one "best" model and the definition of optimal may not be clear.
   Given a nucleotide sequence, what is the best description of the sequence in terms of functional regions in the DNA?
3. How do we adjust the parameters of the model to maximize the likelihood $\mathbf{P}[O|\lambda]$?
   How can we train the model to recognize the features of DNA?

The next three sections seek to provide sensible answers to all these questions.

4.3. Calculating the probability of the observed sequence

Let $Q = q_1, \ldots, q_T$ represent the sequence of states that the HMM visits. First we calculate the probability of an observation sequence given the state sequence. Clearly,

$$\begin{aligned}\mathbf{P}[O|Q,\lambda] &= \prod_{t=1}^{T} \mathbf{P}[O_t|q_t,\lambda] \\ &= \prod_{t=1}^{T} b_{q_t O_t},\end{aligned}$$

since in a standard HMM the observations are independent conditional on the corresponding state(s). The probability of the state sequence follows directly from the Markov property:

$$\mathbf{P}[Q|\lambda] = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t}.$$

Since we know

$$\mathbf{P}[O,Q|\lambda] = \mathbf{P}[O|Q,\lambda]\mathbf{P}[Q|\lambda],$$

we can compute the required quantity by summing $\mathbf{P}[O,Q|\lambda]$ over all possible state sequences,

$$\begin{aligned}\mathbf{P}[O|\lambda] &= \sum_{\text{all } Q} \mathbf{P}[O,Q|\lambda] \\ &= \sum_{\text{all } Q} \left[\pi_{q_1} b_{q_1 O_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t O_t} \right]. \qquad (1)\end{aligned}$$

Whilst it seems we have solved the first problem, there are some unfortunate practical problems with our solution. Since we are summing over all state sequences, of which there are $\mathbf{O}(N^T)$ (except in degenerate cases), and each choice of state sequence needs about $2T$ calculations, the number of required calculations is $\mathbf{O}(TN^T)$, which is (worse than) exponential in the sequence length.

A more efficient approach involves the Forward procedure. We define the set of forward variables $\alpha_t(i)$ to be

$$\alpha_t(i) \equiv \mathbf{P}[O_1^t, q_t = S_i|\lambda],$$

which is the probability of observing the sequence up to and including time t and being in state S_i at that time. A neat algorithm allows us to calculate the forward variables efficiently—where the model λ is implicit:

1. Initialization

$$\alpha_1(i) = \mathbf{P}[O_1, q_1 = S_i] = \pi_i b_{iO_1} \qquad (1 \le i \le N).$$

2. Induction

$$\begin{aligned}\alpha_{t+1}(j) &= \mathbf{P}[O_1^{t+1}, q_{t+1} = S_j] \qquad (1 \le t \le T-1, \quad 1 \le j \le N) \\ &= \sum_{i=1}^{N} \mathbf{P}[O_1^{t+1}, q_{t+1} = S_j, q_t = S_i] \\ &= \sum_{i=1}^{N} \mathbf{P}[O_{t+1}|O_1^t, q_{t+1} = S_j, q_t = S_i]\mathbf{P}[q_{t+1} = S_j|O_1^t, q_t = S_i]\mathbf{P}[O_1^t, q_t = S_i] \\ &= \sum_{i=1}^{N} b_{jO_{t+1}} a_{ij} \alpha_t(i).\end{aligned}$$

3. Termination

$$\begin{aligned}\mathbf{P}[O_1^T] &= \sum_{i=1}^{N} \mathbf{P}[O_1^T, q_T = S_i] \\ &= \sum_{i=1}^{N} \alpha_i(T).\end{aligned}$$

For each pair (j,t), $\mathbf{O}(N)$ calculations are required to calculate $\alpha_t(j)$, and therefore the algorithm as a whole requires $\mathbf{O}(N^2T)$ calculations. Importantly, for a fixed number of states, the number of computations is linear in T instead of exponential. We now have a computationally efficient way of calculating solving problem 1.

Even though their application is not as immediate as the forward variables it is sensible at this point to define backward variables. Let the backward variables $\beta_t(i)$ be defined as:

$$\beta_t(i) \equiv \mathbf{P}[O_{t+1}^T | q_t = S_i, \lambda].$$

That is, the probability of the observing the particular sequence *after* time t given that the system is in state S_i at the time. For $t = T$ the backward variables are defined to be $\mathbf{P}[\Omega | q_t = S_i, \lambda] = 1$.

Having initialized $\beta_T(i)$ in this way we can inductively define the other values—again λ is implicit.

$$\begin{aligned}\beta_t(i) &= \mathbf{P}[O_{t+1}^T | q_t = S_i] \qquad (i \le t \le T-1, \quad 1 \le i \le N) \\ &= \sum_{j=1}^{N} \mathbf{P}[O_{t+1}^T, q_{t+1} = S_j | q_t = S_i] \\ &= \sum_{j=1}^{N} \mathbf{P}[O_{t+2}^T | O_{t+1}, q_t = S_i, q_{t+1} = S_j] \mathbf{P}[O_{t+1} | q_{t+1} = S_j, q_t = S_i] \mathbf{P}[q_{t+1} = S_j | q_t = S_i] \\ &= \sum_{j=1}^{N} \beta_{t+1}(j) b_{jO_{t+1}} a_{ij}.\end{aligned}$$

Unlike the forward variables, there is no termination step for the backward variables. There is no reason to have one as we are not using them directly to calculate some useful quantity. Note again that the number of calculations required is $\mathbf{O}(N^2T)$.

4.4. Selecting the optimal sequence of states

As mentioned earlier there is no canonical definition of optimal. A first attempt at a definition is the sequence of states that at each value of t are individually most likely given the observations. This definition maximizes the number of correctly guessed states. We use a new set of variables,

$$\gamma_t(i) = \mathbf{P}[q_t = S_i | O, \lambda],$$

which don't have a common name in the literature so we may as well call them *gamma* variables. Therefore $\gamma_t(i)$ is the probability of being in state S_i at time t given the (whole) observation sequence O. As it happens, the gamma variables can be expressed in terms of the forward and

backward variables.

$$\begin{aligned}\gamma_t(i) &= \mathbf{P}[q_t = S_i|O] \\ &= \frac{\mathbf{P}[q_t = S_i, O]}{\mathbf{P}[O]} \\ &= \frac{\mathbf{P}[O_{t+1}^T|q_t = S_i, O_1^t]\mathbf{P}[q_t = S_i, O_1^t]}{\mathbf{P}[O]} \\ &= \frac{\beta_t(i)\alpha_t(i)}{\mathbf{P}[O]} \\ &= \frac{\beta_t(i)\alpha_t(i)}{\sum_{i=1}^N \beta_t(i)\alpha_t(i)}.\end{aligned}$$

Finally note that with our definition of optimal,

$$\widehat{q_t} \equiv \underset{1\le i\le N}{\operatorname{argmax}}\, \mathbf{P}[q_t = S_i|O,\lambda] = \underset{1\le i\le N}{\operatorname{argmax}}[\gamma_t(i)],$$

where $\widehat{q_1}, \ldots, \widehat{q_T}$ is the predicted state sequence. Assuming that the forward and backward variables have already been calculated, this is an efficient $\mathbf{O}(NT)$ procedure (the quantity $\mathbf{P}[O]$ only needs to be calculated once, and could be excluded anyway). However this definition of optimality has a serious drawback: it is possible that the "optimal" sequence of states will involve a state transition which has zero probability. A better definition will look at the state sequence as a whole rather than optimizing the choice of states individually. With this in mind, we consider the optimal sequence of states to be that which maximizes $\mathbf{P}[Q|O]$, and thus $\mathbf{P}[Q, O]$. The procedure for finding the optimal sequence is called the Viterbi algorithm.

First define the *delta* variables as:

$$\delta_t(i) = \max_{Q_1^{t-1}} \mathbf{P}[Q_1^{t-1}, q_t = S_i, O_1 \ldots O_t|\lambda]. \tag{2}$$

So $\delta_t(i)$ represents the probability of the highest probability path including state S_i at time t and observing O_1 to O_t. The induction step will involve selecting the previous state q_{t-1} which maximizes (2). To be able to keep track of these values we need to store this value of q_{t-1} for each value of q_t in a matrix $\Psi = \{\{\psi_t(i)\}\}$. To recover the optimal state sequence we use a similar approach to the forward algorithm.

1. Initialization

$$\delta_1(i) = \mathbf{P}[q_1 = S_i, O_1] = \pi_i b_i(O_1) \qquad (1 \le i \le N)$$

2. Induction

$$\begin{aligned}\delta_t(i) &= \max_{Q_1^{t-1}} \mathbf{P}[Q_1^{t-1}, q_t = S_i, O_1^t] \qquad (2 \le t \le T, \quad 1 \le i \le N) \\ &= \max_{1\le j\le N} \max_{Q_1^{t-1}} \mathbf{P}[Q_1^{t-2}, q_{t-1} = S_j, q_t = S_i, O_1^t] \\ &= \max_{1\le j\le N} \max_{Q_1^{t-1}} \big[\mathbf{P}[O_t|Q_1^{t-2}, q_{t-1} = S_j, q_t = S_i, O_1^{t-1}] \\ &\qquad \mathbf{P}[q_t = S_i|Q_1^{t-2}, q_{t-1} = S_j, O_1^{t-1}]\mathbf{P}[Q_1^{t-2}, q_{t-1} = S_j, O_1^{t-1}]\big] \\ &= b_i(t) \max_{1\le j\le N} a_{ji}\delta_{t-1}(j).\end{aligned}$$

$$\psi_t(i) = \underset{1\le j\le N}{\operatorname{argmax}}\, a_{ji}\delta_{t-1}(j).$$

3. Termination

$$\begin{aligned} p^* &\equiv \max_{Q_1^T} \mathbf{P}[Q_1^T, O_1^T] \\ &= \max_{1\le j\le N} \max_{Q_1^{T-1}} \mathbf{P}[Q_1^{T-1}, q_T = S_j, O_1^T] \\ &= \max_{1\le j\le N} \delta_T(j). \end{aligned}$$

Hence

$$\widehat{q_T} = \operatorname*{argmax}_{1\le j\le N} \delta_T(j).$$

4. Backtracking

$$\widehat{q_t} = \psi_{t+1}(\widehat{q_{t+1}}).$$

At the end of this exercise we have the sequence of states $\widehat{q_1}, \ldots, \widehat{q_T}$ which maximizes $\mathbf{P}[Q, O]$. The Viterbi algorithm is essentially the same as the forward algorithm with two exceptions. Firstly, there is no backtracking needed in the forward procedure. Secondly instead of adding terms in the induction and termination steps the Viterbi algorithm selects the highest term. Yet again, this is a $\mathbf{O}(N^2T)$ time algorithm.

4.5. Adjusting model parameters to maximize likelihood

Given a training sequence, or more generally a set of training sequences, we would like to find the maximum likelihood estimates of the HMM parameters. Unfortunately the likelihood function of the parameter space for most HMMs is multimodal and so it is difficult to find the global maximum of this function. The maximization can only be done locally: the algorithms find the "locally best" set of parameters for the hidden Markov model. Various approaches are available including the Baum-Welch method (presented below), Expectation Maximization and gradient techniques, which are all iterative processes [**Rab89**]. Importantly none of these parameter estimation procedures change the topology of the model: the number of symbols, the number of states and which ones have null transitions.

The reestimation procedure in the Baum-Welch forward-backward algorithm (as described by Rabiner) uses yet another matrix of variables,

$$\xi_t(i,j) \equiv \mathbf{P}[q_t = S_i, q_{t+1} = S_j | O, \lambda],$$

which are the probabilities of being in state S_i at time t and S_j at time $t+1$ given both the model and the observation sequence. For the moment it is okay to leave λ out of the calculations and derive a formula for the *xi* variables.

$$\begin{aligned} \xi_t(i,j) &= \mathbf{P}[q_t = S_i, q_{t+1} = S_j | O] \qquad (1 \le t \le T-1, \quad 1 \le i,j \le N) \\ &= \mathbf{P}[q_t = S_i, q_{t+1} = S_j, O]/\mathbf{P}[O]. \end{aligned}$$

Therefore

$$\begin{aligned} \xi_t(i,j)\mathbf{P}[O] &= \mathbf{P}[O_{t+2}^T | O_1^{t+1}, q_t = S_i, q_{t+1} = S_j]\mathbf{P}[O_{t+1} | O_1^t, q_t = S_i, q_{t+1} = S_j] \cdot \\ &\qquad \mathbf{P}[q_{t+1} = S_j | O_1^t, q_t = S_i]\mathbf{P}[O_1^t, q_t = S_i] \\ &= \beta_{t+1}(j) b_{jO_{t+1}} a_{ij} \alpha_t(i). \end{aligned}$$

And so

$$\begin{aligned} \xi_t(i,j) &= \frac{\beta_{t+1}(j) b_{jO_{t+1}} a_{ij} \alpha_t(i)}{\mathbf{P}[O]} \\ &= \frac{\beta_{t+1}(j) b_{jO_{t+1}} a_{ij} \alpha_t(i)}{\sum_{i=1}^N \sum_{j=1}^N \beta_{t+1}(j) b_{jO_{t+1}} a_{ij} \alpha_t(i)}. \end{aligned}$$

It should be clear from the definitions that

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j).$$

Now, thinking intuitively, if we fix i and sum $\gamma_t(i)$ over t from 1 to $T-1$ we obtain the expected number of transitions out of state S_i given the model and the observation sequence. (If we include $t = T$ this also represents the average number of times the system is in state S_i.) Similarly, fixing i and j and summing $\xi_t(i,j)$ over t from 1 to $T-1$, we have the expected number of transitions from S_i to S_j. Letting $\bar{\lambda} = (\bar{\pi}, \bar{A}, \bar{B})$ stand for the new estimates of the model parameters, the reestimation formulas should by intuition be:

$$\begin{aligned}
\bar{\pi}_i &= \text{expected number of times in state } S_i \text{ when } t = 1 \\
&= \gamma_1(i).
\end{aligned}$$

$$\begin{aligned}
\bar{a}_{ij} &= \frac{\text{expected \# of transitions from } S_i \text{ to } S_j}{\text{expected \# of transitions from } S_i} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}.
\end{aligned}$$

$$\begin{aligned}
\bar{b}_{ik} &= \frac{\text{expected \# of times in state } S_i \text{ emitting symbol } v_k}{\text{expected \# of times in state } S_i} \\
&= \frac{\sum_{t=1:O_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}.
\end{aligned}$$

Baum and colleagues proved that if this iterative procedure is used, $\mathbf{P}[O|\bar{\lambda}] > \mathbf{P}[O|\lambda]$ unless $\bar{\lambda}$ is a fixed point of the reestimation procedure and therefore a local maximum. (See section 4.6 for a proof based on [**BPSW70**].) There are $\mathbf{O}(N^2T)$ calculations required for $\bar{A}$ and $\bar{B}$, and since all of the variables used in reestimation also require $\mathbf{O}(N^2T)$ calculations, each iteration of the reestimation process is $\mathbf{O}(N^2T)$, which for a particular model topology is linear in symbol sequence length. Note that for complex models we are maximizing the likelihood in many dimensions and there are likely to be many local maxima, so it may be necessary to use many different starting points for the iterations in order to obtain values close to the global maximum likelihood estimates.

4.6. A proof of the maximization technique

Treat the observation sequence O as being fixed and define $P(\pi, A, B)$ to be $\mathbf{P}[O|\lambda]$ and $p(Q, \pi, A, B)$ to be $\mathbf{P}[O, Q|\lambda]$. If we let X represent the set of all possible state sequences ($X = S^T$), then

$$P(\pi, A, B) = \sum_{Q \in X} p(Q, \pi, A, B).$$

Also define an auxiliary function $R(\pi, A, B, \pi', A', B')$ to be

$$\sum_{Q \in X} p(Q, \pi, A, B) \log \left[p(Q, \pi', A', B') \right].$$

For convenience we can rewrite the arguments (π, A, B) and (π', A', B') to the functions P, p and R as λ and λ' respectively.

LEMMA. *If* $R(\lambda, \bar{\lambda}) \geq R(\lambda, \lambda)$ *then* $P(\bar{\lambda}) \geq P(\lambda)$ *with strict inequality unless* $R(\lambda, \bar{\lambda}) = R(\lambda, \lambda)$ *and* $p(Q, \lambda) = p(Q, \bar{\lambda})$ *for all* $Q \in X$.

PROOF.

$$\begin{aligned}
\log\left(\frac{P(\bar{\lambda})}{P(\lambda)}\right) &= \log \sum_{Q\in X}\left(\frac{p(Q,\bar{\lambda})}{P(\lambda)}\right)\\
&= \log \sum_{Q\in X}\left(\frac{p(Q,\lambda)}{P(\lambda)}\right)\left(\frac{p(Q,\bar{\lambda})}{p(Q,\lambda)}\right)\\
&\geq \sum_{Q\in X}\left(\frac{p(Q,\lambda)}{P(\lambda)}\right)\log\left(\frac{p(Q,\bar{\lambda})}{p(Q,\lambda)}\right) \qquad (3)\\
&= \frac{1}{P(\lambda)}\sum_{Q\in X} p(Q,\lambda)[\log p(Q,\bar{\lambda}) - \log p(Q,\lambda)]\\
&= \frac{1}{P(\lambda)}[R(\lambda,\bar{\lambda}) - R(\lambda,\lambda)] \geq 0 \qquad \text{by hypothesis.}
\end{aligned}$$

The inequality at (3) is an application of Jensen's inequality to the concave function log with discrete probability measure $p(Q,\lambda)/P(\lambda)$ on X which is strict unless $p(Q,\bar{\lambda})/p(Q,\lambda)$ is constant on X. □

Therefore if we find $\bar{\lambda}$, the value of λ' that maximizes $R(\lambda,\lambda')$, we have found a parameter set $\bar{\lambda}$ that increases the value of $P(\lambda)$. Since

$$p(Q,\pi,A,B) = \pi_{q_1} b_{q_1 O_1}\prod_{t=2}^{T} a_{q_{t-1}q_t} b_{q_t O_t} \qquad \text{(recall (1))}$$

we wish to maximize

$$R(\lambda,\lambda') = \sum_{Q\in X} p(Q,\lambda)[\log \pi'_{q_1} + \textstyle\sum_{t\leq T}\log a'_{q_{t-1}q_t} + \sum_{t\leq T}\log b'_{q_t O_t}]$$

subject to the stochastic constraints:

$$\sum_{i=1}^{N}\pi'_i = 1; \qquad \sum_{j=1}^{N} a'_{ij} = 1, \quad \sum_{k=1}^{M} b'_{ik} = 1 \qquad (1\leq i\leq N).$$

Using Lagrange multipliers it is quite easy to show that the only critical point of $R(\lambda,\lambda')$ as a function of λ' is

$$\begin{aligned}
\pi'_i &= \frac{\sum_{q_0=i} p(Q,\lambda)}{\sum_{Q\in X} p(Q,\lambda)}\\
a'_{ij} &= \frac{\sum_{q_{t-1}=i,q_t=j} p(Q,\lambda)}{\sum_{q_{t-1}=i} p(Q,\lambda)}\\
b'_{jk} &= \frac{\sum_{q_t=i,O_t=v_k} p(Q,\lambda)}{\sum_{q_t=i} p(Q,\lambda)}.
\end{aligned}$$

Since the second derivative of R with respect to all of the parameters in λ' is negative, the critical point is the maximum value of R—there being no boundary surfaces—and thus becomes the new parameter set $\bar{\lambda}$. In the light of the Lemma it is clear that transforming λ to $\bar{\lambda}$ increases the value of the function $P(\lambda) = \mathbf{P}[O|\lambda]$ unless $\lambda = \bar{\lambda}$ is a fixed point of the transformation. Indeed this transformation of λ coincides with the reestimation formulas on page 15. We have therefore developed procedures to solve all three HMM problems which are linear in the sequence length.

4.7. A computational consideration

There is one considerable practical difficulty with all of the calculations above. Some of the variables are probabilities of very "unlikely" events and so will have very small values: beyond the range of most computer floating-point representations. Rabiner suggests a method of rescaling for the forward and backward variables, but I find this tedious. For the Viterbi algorithm however he suggests the logical choice of storing the log of the probability in the computer. This works fine as all of the arithmetic operations in this algorithm are multiplications. In the forward procedure amongst others, addition presents a bit of a problem. Rather than using Rabiner's rescaling ideas I decided to obtain logs of sums of probabilities like this ...

If $x = \log(X)$ and $y = \log(Y)$ and WLOG we assume $Y < X$ then

$$\begin{aligned}\log(X+Y) &= \log(e^x + e^y)\\ &= \log(e^x(1+e^{y-x}))\\ &= x + \log(1+e^{y-x}).\end{aligned}$$

Whilst relatively time consuming for the CPU, this method of calculation allows us to express everything with log probabilities. Note that if $Y \ll X$ then the CPU will round off $e^{y-x} = Y/X$ to zero and hence $\log(X+Y)$ will be deemed to be $\log(X)$ which is good enough in such circumstances. As it happens this technique has already been applied [**DEKM98**].

4.8. What is a Generalized Hidden Markov Model?

The current design of the HMM imposes a geometric distribution on state duration. Given that the (hidden) Markov process is time-homogeneous this can be shown easily:

$$\mathbf{P}[q_2 = S_i, \ldots, q_d = S_i, q_{d+1} = S_{j\neq i} | q_1 = S_i] = a_{ii}^{d-1}(1-a_{ii}).$$

This may not be a sensible restriction, and it may be useful to add an explicit duration distribution

$$p_i(d) \equiv \mathbf{P}[q_2 = S_i, \ldots, q_d = S_i, q_{d+1} = S_{j\neq i} | q_1 = S_i]$$

to each state. In doing so we remove self-transitions from the state transition matrix A. Therefore the steps for the generation of a symbol sequence are modified to:

1. The initial state q_1 is randomly selected using π.
2. $r \leftarrow 1$
3. Using the state duration density function $p_{q_r}(\cdot)$ calculate the number of symbols d_r to be generated in this state.
4. Given d_r use the state symbol distribution to generate a sequence of symbols $O_{s_r+1}^{s_r+d_r}$. Note that we define s_r to be $d_1 + \cdots + d_r$. These symbols could be generated by some complex rule internal to the state; for example a Markov chain.
5. Using the state transition distribution $a_{q_r(\cdot)}$ move to a *new* state q_{r+1}.
6. $r \leftarrow r+1$, if $s_r < T$ goto step 3, else perform some truncation exercise (if required) and exit.

Note that if the observations O_k are independent and the state duration densities are geometric then we end up with the standard HMM. This new type of model is called an explicit state duration hidden Markov model, a hidden semi-Markov model or perhaps a generalized hidden Markov model (GHMM).

Whilst the reestimation process detailed by Rabiner for this class of models is of limited use (due to an upper bound on state duration length) it is still worth examining the change in his definition of the forward variables:

$$\alpha_t(i) \equiv \mathbf{P}[O_1^t, S_i \text{ ends at time } t | \lambda].$$

If indeed r states have been visited, so that

$$t = s_r \text{ and } q_r = S_i, \tag{4}$$

then

$$\alpha_t(i) = \sum \pi_{q_1} p_{q_1}(d_1)\mathbf{P}[O_1^{d_1}|q_1] \cdot a_{q_1 q_2} p_{q_2}(d_2)\mathbf{P}[O_{s_1+1}^{s_2}|q_2] \cdots a_{q_{r-1} q_r} p_{q_r}(d_r)\mathbf{P}[O_{s_{r-1}+1}^{s_r}|q_r],$$

where the sum is over all sets of states and durations consistent with the constraints in (4). Evaluating these sets of states and durations is messy, so I don't use Rabiner's new definition of $\alpha_t(i)$. Instead my model uses some techniques to bring back some of the simplicity of the original HMM. Of course one could generalize even further by making the duration in a state depend on the previous symbols emitted so far.

CHAPTER 5

The model

5.1. Background to my work

Although Hidden Markov Models have been used in speech recognition for over twenty years, their application to computational biology is relatively recent. In 1993 Haussler and colleagues realized that profiles used for multiple sequence alignment could be expressed as HMMs [**HKMS93**]. In 1994 Anders Krogh and members of that team developed the first HMM for finding genes in E. coli DNA called ECOPARSE [**KMH94**]. Since ECOPARSE there have been many more HMM genefinders released, the first with generalized hidden Markov models being Genie [**KHRE96, REKH97**]. The two that are the main inspiration for mine are GenMark.hmm [**LB98**] and especially GENSCAN [**BK97, Bur97**].

5.2. The building blocks

I decided to build my Pf gene finder as a generalized hidden Markov model. Since a GHMM can use arbitrary rules for generating symbols within states, I could model the regular grammar structure of the genome using a Markov process on the states without worrying at first about the exact nucleotide generation rules. Figure 5.1 shows the state structure that I chose for the Pf DNA. The most obvious classification of the states is according to strand. The top half of the diagram—the states with the + suffix—represent functional regions on the forward strand whilst those in the bottom half—with the - suffix—represent features on the reverse strand. The intergenic state is "neutral" as it is specific to neither strand. Note that the flow through the reverse strand states is the opposite to that through the forward strand states. This stems from the fact that the GHMM will read the DNA on one strand only and so will see features on the reverse strand back-to-front. Since the GHMM can only follow one state path at a time this model has no way of representing genes on the two strands which overlap.

Due to my lack of experience in genetics I decided to keep the model simple. The gene searching is (almost) entirely nucleotide-composition based, with functional site recognition limited to use of minimal consensi. The three types of state are intergene, exon and intron so there is no modelling of untranslated regions, promoters or poly-`A` sites. In particular the exon states of this model include only the *protein-coding* or *translated* sections of the initial and final real exons. The UTR sections are considered to be intergenic: the first exon state starts with `ATG` and the last exon state ends with a stop codon. My model doesn't employ any specialized splice site recognition but the intron states must start with `GT` and end with `AG`.

It is imperative that my model keeps track of reading frame. By this I mean that if the model leaves an exon state and enters an intron state, it should remember how much of the last codon it had processed. When the system enters the next exon state it generates/reads the first nucleotide of the state in relation to where it was up to in the previous exon state. In this way it maintains a reading-frame consistent with that of the ribosome which sees mRNA from which the "intron mRNA" has already been excised. To achieve this, the model has three separate intron states `int 0`, `int 1` and `int 2`. Concentrating on the forward strand for now, the system moves to state `int 0+` if it had just finished generating/reading a codon in the previous exon state, to `int 1+` if it had emitted/read the first base of a codon, and to `int 2+` if two bases of the last codon had been emitted/read when it switched states. Unless it moves to the terminal exon state, the system must go to state `exon i+` after `int i+`, $\texttt{i} \in \{0, 1, 2\}$. So in state `exon i+` the system starts with the $\texttt{i}+1^{\text{th}}$

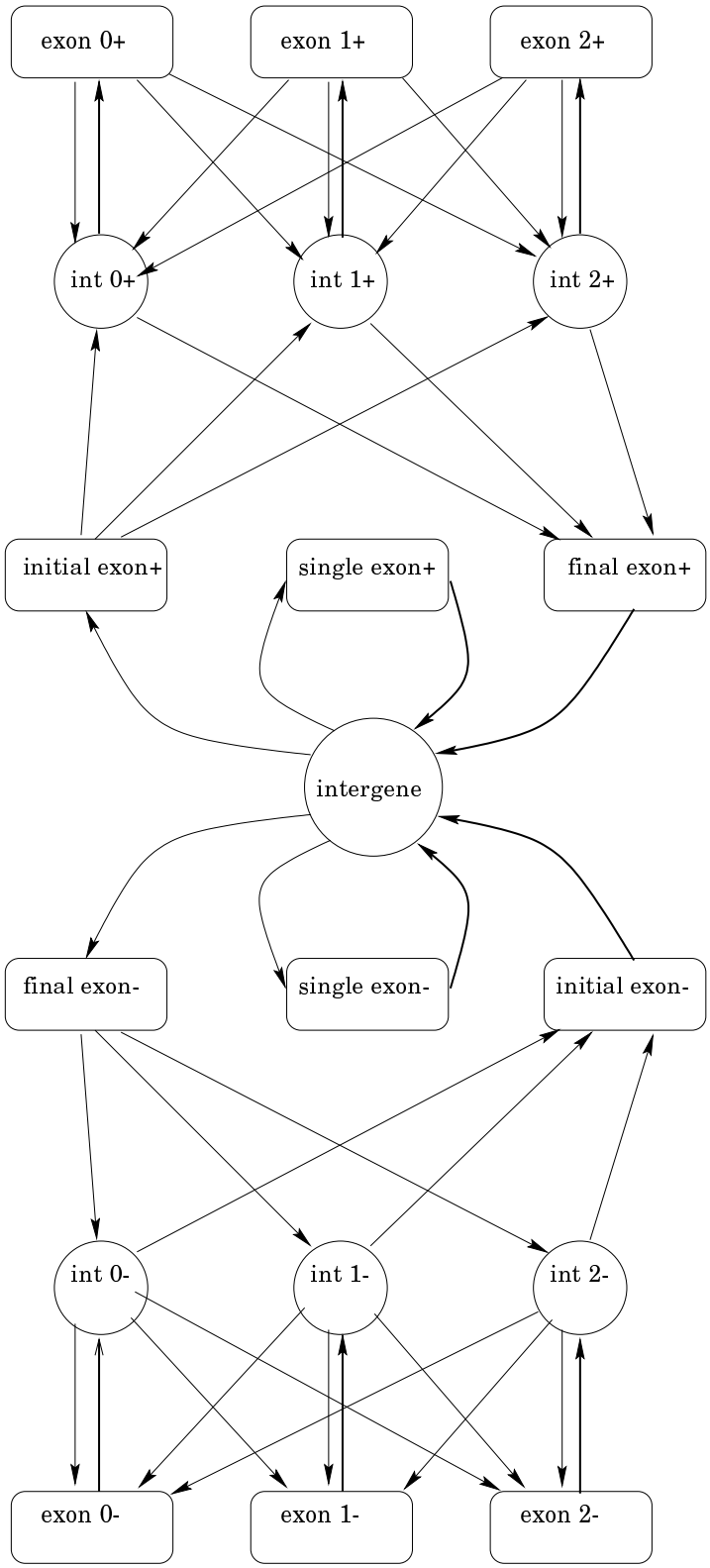

FIGURE 5.1. State space of my model

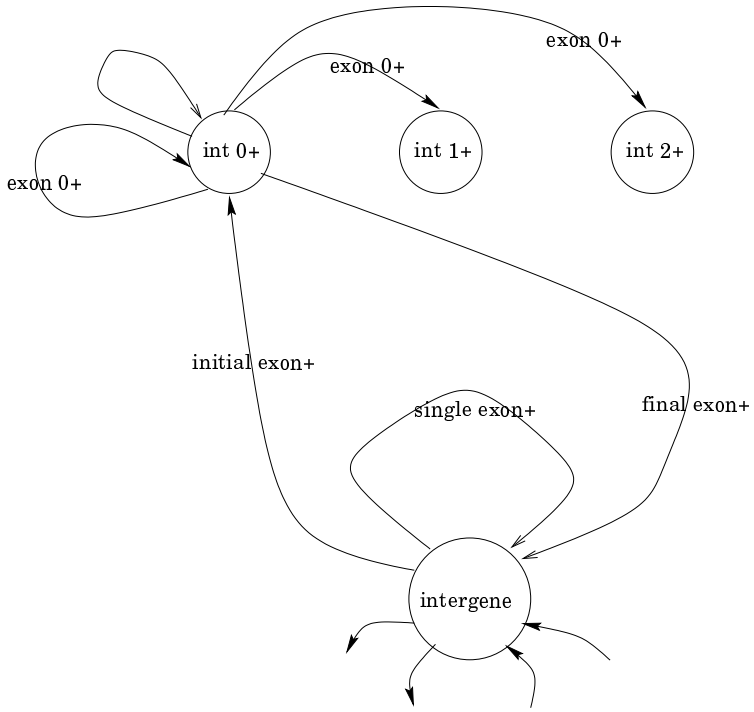

FIGURE 5.2. Remodelling the state system as a standard HMM—only some states and transitions shown

base of the codon. Although the program reads (the complements of) the symbols of the reverse strand back-to-front the GHMM still defines the states to be those that would be seen if read in correct order. Therefore state `int i-` must follow state `exon i+`, and the right-most nucleotide in state `exon i-` is the $i+1^{\text{th}}$ base of the right-most codon, when read from 3' to 5'.

5.3. Simplifying the calculations

As shown in section 4.8, the calculations for the forward variables become rather tricky in a GHMM. In order to both speed up the algorithm and improve understanding, I decided to mimic Burge's state simplifications in GENSCAN. If we classify the states into coding (C) and non-coding ($\bar{C}$) states, it should be clear that any legal sequence of states will alternate between the C and $\bar{C}$ states. To simplify the algorithms we assume that the $\bar{C}$ states behave as if they were part of an ordinary HMM. That is, they emit exactly one symbol on each entry to the state, have self-transitions, and therefore have geometric duration distributions. The C states still have arbitrary length and symbol emission distributions. A discussion of the reasonableness of this assumption for the $\bar{C}$ states is presented in section 5.6.2. There is no truncation problem for the $\bar{C}$ states as they only generate one symbol at a time, but for C states it is be useful to assume that they end at the boundaries of the sequence, positions 1 and L, and to artificially regard positions 0 and $L+1$ as being part of $\bar{C}$ states in all sequences. (Henceforth L is used to denote sequence length to prevent ambiguity with the thymine base.)

The main gain from this separation of the states is that we can simplify the recursions for the forward, backward, gamma and other variables. We can, *in some sense*, consider the model as a standard HMM with seven states, where more interesting events occur on the transitions (see Figure 5.2). The general approach for each recursion is to consider the different C states that could end just prior to the current $\bar{C}$ state. For example, if we are trying to evaluate

$$\alpha_{24}(\texttt{int 1+}) = \mathbf{P}[q_{24} = \texttt{int 1+}, O_1^{24}],$$

then we consider all the C states that could possibly end at position 23. Given that `int 1+` is an intron state, the C states that could have come before it are: `initial exon+`, `exon 0+`, `exon 1+`, and `exon 2+`. For each of these C state types we list all of the possible start positions: start codons, reverse complement stop codons, acceptor sites and reverse complement donor sites and position 1. (Actually this isn't quite how I implemented it, see below for the truth.) For example, it is possible that an initial exon started at position 11 and ended at position 23—especially if the sequence looked something like that in Figure 5.3. If so, then it is clear that the evaluation of $\alpha_{24}(\texttt{int 1+})$ will involve adding terms like

$$\alpha_{10}(\texttt{intergene}) \cdot a_{\texttt{intergene, int 1+}} \cdot \mathbf{P}[O_{11}^{23}|\texttt{initial exon+}, \text{duration } 13] \cdot \mathbf{P}[\text{duration in state } \texttt{initial exon+} = 13],$$

since the initial exon must be preceded by the intergene state.

5.4. Formalizing the ideas

The following description contains one (small) lie; the truth is explained in Appendix A which should be read after, though not necessarily immediately after, this chapter. In the TEXbook Knuth suggests that deliberate lying at first sometimes makes concepts easier to follow later. Before analyzing the sequence, a list is made of all possible start and stop for the various C states on both the forward and backward strands using the minimal consensi. To deal with exons extending off the edge of the sequence, positions 1 and L are added to the appropriate lists. A matrix of forward variables $\alpha_t(i)$ is constructed for the $\bar{C}$ states with

$$t \in \{0, 1, \ldots L+1\} \quad \text{and} \quad i \in \{N, I_0^+, I_1^+, I_2^+, I_0^-, I_1^-, I_2^-\},$$

where N stands for intergene and the other state labels follow as simple abbreviations. For each $\bar{C}$ state i and position t a list $E_{i,t}$ is made of all the possible non-empty exons (C states) ending at position $t-1$ which could be followed by state i at position t. Recall that since the complementary strand is read in reverse order, exons in the reverse strand will "end" with `CAT`, or just before `CT` on the forward strand, being the reverse complements of the start codon and acceptor site. For every element $\varepsilon \in E_{i,t}$ we define the following parameters:

- x: The start position of state ε.
- λ: The length of the state ε so that $x + \lambda = t$
- c: The type of C state that ε is.
- j: The $\bar{C}$ state that immediately precedes ε.

In the example above, $i = I_1^+$, $t = 24$, $x = 11$, $\lambda = 13$, $c =$ `initial exon+`, $j = N$. The exons have length distributions $\Lambda_c(\cdot)$ and symbol emission probabilities

$$\Theta_c(a, b) = \mathbf{P}[O_a^b | O_1^{a-1}, Q_a^b = c].$$

The $\bar{C}$ states behave just like the standard HMM and so with one exception we need no new notation for them. We need to distinguish between self-transitions which represent adjacent nucleotides in the same $\bar{C}$ state and self-transitions where there is an intervening exon, such as I_0^+, `exon 0+`, I_0^+. In future only the former case will be termed self-transitions. The probability of a self-transition will be denoted by a_{ii}^* so that

$$a_{ii}^* + \sum_{j \in \bar{C}} a_{ij} = 1.$$

Note that a_{ij} is the probability that the next $\bar{C}$ *state* after this one is S_j.

```
                    10                  20
1 2 3 4 5 6 7 8 9 X 1 2 3 4 5 6 7 8 9 X 1 2 3 4 5 6 7
C A A T A T T T G A A T G G T A G G A A T A A G T G G
```

FIGURE 5.3. Sample DNA sequence

To execute the forward algorithm we still use a three stage process. Recall that

$$\alpha_t(i) \equiv \mathbf{P}[O_1^t, q_t = i].$$

Again, excuse obvious abuses of notation, and please see Appendix A for the justification of the induction step:

1. Initialization

$$\alpha_0(i) = \pi_i \qquad (i \in \bar{C}).$$

2. Induction

$$\alpha_t(i) = \alpha_{t-1}(i)a^*_{ii}b_{iO_t} + \sum_{\varepsilon \in E_{i,t}} \alpha_{x-1}(j)a_{ji}b_{iO_t} \cdot \Lambda_c(\lambda)\Theta_c(x, t-1)$$
$$(i \in \bar{C}, \quad 1 \le t \le L).$$

3. Termination

$$\alpha_{L+1}(i) = \alpha_L(i) + \sum_{\varepsilon \in E_{i,t}} \alpha_{x-1}(j)a_{ji} \cdot \Lambda_c(\lambda)\Theta_c(x, L) \qquad (i \in \bar{C}),$$
$$\mathbf{P}[O] = \sum_{i \in \bar{C}} \alpha_{L+1}(i).$$

Note that the induction step in this version of the algorithm is essentially identical to the standard HMM except for the presence of the terms $\mathbf{P}[O_x^{t-1}|c, \lambda]$ and $\Lambda_c(\lambda)$, and the fact that the $\bar{C}$ states that immediately precede state i at position t are not at position $t-1$, except for self-transitions of course.

The Viterbi algorithm, as was commented in the previous chapter, is very similar to the forward procedure, so it is appropriate to list it here. The *psi* variables now store the best previous $\bar{C}$ state and its position, so the exons are implicit in the parse.

1. Initialization

$$\delta_0(i) = \pi_i \qquad (i \in \bar{C}).$$

2. Induction

$$\delta_t(i) = \max\left\{\delta_{t-1}(i)a^*_{ii}b_{iO_t}, \max_{\varepsilon \in E_{i,t}}\{\delta_{x-1}(j)a_{ji}b_{iO_t} \cdot \Lambda_c(\lambda)\Theta_c(x, t-1)\}\right\},$$
$$\psi_t(i) = \operatorname*{argmax}_{(i,t-1),(j,x-1) \in E_{i,t}}\left\{\text{ditto}\right\} \qquad (i \in \bar{C}, \quad 1 \le t \le L).$$

3. Termination

$$\delta_{L+1}(i) = \max\left\{\delta_L(i), \max_{\varepsilon \in E_{i,t}}\{\delta_{x-1}(j)a_{ji} \cdot \Lambda_c(\lambda)\Theta_c(x, L)\}\right\} \qquad (i \in \bar{C}),$$
$$\psi_{L+1}(i) = \operatorname*{argmax}_{(i,t-1),(j,x-1) \in E_{i,t}}\left\{\text{ditto}\right\},$$
$$\mathbf{P}[\hat{Q}, O] = \max_{i \in \bar{C}} \delta_{L+1}(i),$$
$$\widehat{q_{L+1}} = \operatorname*{argmax}_{i \in \bar{C}} \delta_{L+1}(i).$$

The backward procedure, which can be used to determine the probability that a particular segment of the nucleotide sequence is an exon, is presented below. We change slightly the definition of the backward variables; now used only for $\bar{C}$ states:

$$\beta_t(i) = \mathbf{P}[O_t^L|O_1^{t-1}, q_t = i].$$

Moreover, we now treat $E_{i,t}$ as the set of C states that *begin* at position $t+1$ and redefine x to be the last position of the ε and j to be the $\bar{C}$ state that comes "after" it.

1. Initialization

$$\beta_{L+1}(i) = 1 \qquad (i \in \bar{C}).$$

2. Induction

$$\beta_t(i) = \beta_{t+1}(i)a^*_{ii}b_{iO_t} + \sum_{\varepsilon \in E_{i,t}} \beta_{x+1}(j)a_{ij}b_{iO_t} \cdot \Lambda_c(\lambda)\Theta_c(t+1, x)$$
$$(i \in \bar{C}, \quad 2 \le t \le L).$$

We can calculate the probability that an exon of type ε of type c has emitted symbols O_x to O_y—with $\bar{C}$ states i and j flanking it—to be

$$\mathbf{P}[Q_x^y = c, O] = \alpha_{x-1}(i)a_{ij}\Lambda_c(\lambda)\Theta_c(x, y)\beta_{y+1}(j).$$

Importantly this probability calculation is made in light of the whole sequence, and so is not just a local property of the DNA sequence. This probability can be used as a method of assessing the "quality" of potential exons, and thus it is possible to create a list of the best few exons even if they are not in the optimal parse. Again a justification of the formula is provided in Appendix A.

The complexity analysis of these algorithms is rather too involved for this report. Nevertheless the algorithms are no longer linear in the sequence length as the number of exons that must be considered in all of the induction steps is $\mathbf{O}(L^2)$. See section 6.1 for a little more about this.

5.5. What data does my model learn from?

To train my model a large amount of Plasmodium falciparum sequence was required. Unfortunately at the time of the research no large contiguous sequence of *annotated* DNA (say hundreds of kb) was available. (If the sequence hasn't already been annotated, by experiment for example, then it is no use for model training or testing.) Nor is there a "standard" test and training suite for Plasmodium falciparum like the Burset/Guigó set for human and vertebrate DNA [**BG96**]. All that was available was a set of GenBank loci containing single genes or parts of genes. I obtained 264 non-redundant Pf GenBank extracts (collated on 27 May 1998) from Robert Heustis of the Queensland Institute of Medical Research (QIMR), Brisbane. I then removed all sequences with pseudo genes, partial genes, putative genes, those which represented the complementary strand and those with `N`s in the sequence (indicating uncertainty about the base), and two which seem to have been left out because I was scared of their detailed annotation. This cleaning process left 172 sequences for the training and testing process, whose GenBank locus and accession numbers are listed in Appendix B. The CDS key for all of these sequences was used to determine the location of exons in the gene.

5.6. Training the model

From what I have found in the literature there is no procedure similar to the Baum-Welch procedure for the estimation of the parameters in a GHMM. Given that my model has transitions considerably more complex than a standard HMM, this is hardly surprising. That it not to say that such a procedure does not exist; just that I haven't had the time to try to work it out. Taking my cue from GENSCAN, I decided to come up with empirical estimates of all required quantities from the training data.

5.6.1. Initial and transition probabilities. Determining the HMM parameters not related to the observations didn't require much forethought. A transition table showing the frequency of transitions between the various introns states was made. Also a count was made of the number of times a particular intron phase was the last intron in the gene (see Table 5.1). The proportion of genes in the GenBank extracts that had no introns in the GenBank sample was $132/172 = 0.767$, a value that is much higher than that in humans.

TABLE 5.1. Intron transition frequency table from all 172 GenBank sequences

	To phase			
From phase	0	1	2	TERMINAL
0	28	8	8	26
1	8	10	4	11
2	5	4	1	6

We denote the values in Table 5.1 by f_{ij} for $0 \le i \le 2$ and $0 \le j \le 3$, and use them to calculate a_{ij}. The quantities a_{ij} below should be multiplied by $(1 - a^*_{ii})$ the probability that a transition takes place at all. Without this the a_{ij} are values conditional on some transition occurring.

- for $i \in \{I_0^+, I_1^+, I_2^+\}$ and $j \in \{I_0^-, I_1^-, I_2^-\}$, $a_{ij} \equiv 0$ as we are not allowing state sequences to cross strands.
- for $i \in \{I_0^+, I_1^+, I_2^+\}$ and $j \in \{N, I_0^+, I_1^+, I_2^+\}$, $a_{ij} = \frac{f_{ij}}{\sum_{j=0}^{3} f_{ij}}$.
- for $i \in \{I_0^-, I_1^-, I_2^-\}$ and $j \in \{I_0^-, I_1^-, I_2^-\}$, a_{ij} can be worked out by "inverting" the intron transition frequency table, because the features are seen in reverse order. That is $a_{ij} = \frac{f_{ji}}{\sum_{j=0}^{3} f_{ij}}$
- a_{NN} is merely the proportion of genes that have no introns.
- for $j \in \{I_0^+, I_1^+, I_2^+\}$, a_{Nj} is $1 - a_{NN}$ multiplied by the proportion of initial introns that are of type j, multiplied by a half, because of the possible transitions to the reverse strand. The number of initial introns of **phase i** is $\sum_{j=0}^{3} f_{ij} - \sum_{j=0}^{2} f_{ji}$.
- similarly a_{Nj} with $j \in \{I_0^-, I_1^-, I_2^-\}$ is $1 - a_{NN}$ multiplied by a half multiplied by the proportion of exons of terminal exons that are of type j which is $\frac{f_{j3}}{\sum_{j=1}^{3} f_{j3}}$.
- for $i \in \{I_0^-, I_1^-, I_2^-\}$, a_{iN} is the proportion of introns of type i that are initial introns

$$\frac{\sum_{j=0}^{3} f_{ij} - \sum_{j=0}^{2} f_{ji}}{\sum_{j=0}^{3} f_{ij}}.$$

A crude calculation suggested that 0.6 was a reasonable value for π_N as the average length from start to stop codons in the GenBank sequences is approximately 2013 bp. Therefore

$$\pi_{I_i^+} = \pi_{I_i^-} = (1 - \pi_N) \cdot \frac{1}{2} \cdot \frac{\sum_{j=0}^{3} f_{ij}}{\sum_{i=0}^{2} \sum_{j=0}^{3} f_{ij}}.$$

5.6.2. State duration distributions. Since the $\bar{C}$ states are part of a standard HMM their state duration distributions are geometric. All that is required is to estimate a^*_{ii} for each state. Since

$$\mathbf{P}[q_2 = \cdots = q_d = S_i \neq q_{d+1} | q_1 = S_i] = (a^*_{ii})^{d-1}(1 - a^*_{ii}),$$

the maximum likelihood estimate of a^*_{ii} is the mean length of sequence in state i. The histogram of the lengths of the 84 introns (Figure 5.4) doesn't instil much confidence in the geometric length model. Perhaps a gamma distribution would have been more appropriate, but with so few introns it's very difficult to ascertain what would be a suitable length distribution. In any case the symbol emission probabilities probably drown out the influence of the length distribution in many cases so this is not such a deficiency. The mean intergene length was set to be 3kb based on general information in the literature.

I ended up using a geometric length distribution for C states anyway, so to some extent the point of using a GHMM was lost. However the GHMM allowed me to fix the boundaries of the C states where appropriate consensi were present. Separate length distributions were created for single, initial, internal and terminal exons. Histograms of the observed lengths in the GenBank extracts are presented in Figure 5.5.

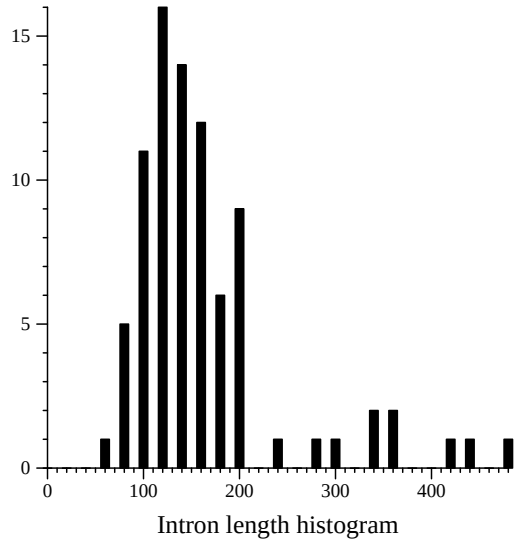

FIGURE 5.4. Lengths of the 84 introns in the GenBank sequences. *Note that in all histograms the final bin contains all sequences larger than the bin label, not just those that should go into that bin.*

TABLE 5.2. Trimer frequencies of introns in GenBank loci

Context	A	C	G	T
AA	1124	94	131	747
AC	233	37	22	65
AG	110	13	23	152
AT	2033	87	304	843
CA	100	46	30	271
CC	39	15	9	58
CG	12	5	4	23
CT	59	30	22	164
GA	196	23	21	84
GC	43	10	3	23
GG	44	5	12	6
GT	253	26	88	132
TA	676	194	116	2165
TC	132	59	10	129
TG	158	56	28	318
TT	806	187	146	1800

5.6.3. Nucleotide composition models. Following the examples in GenMark, GenMark.hmm and GENSCAN, I decided to use Markov chains to model the observation outputs. Recall that even in a HMM the observation distributions are not limited to being independent conditional on the state. For the non-coding states I used second-order homogeneous Markov chains to model the observations. That is

$$b_{ik} = \mathbf{P}[O_t = v_k | \text{past}] = \mathbf{P}[O_t = v_k | O_{t-1} O_{t-2}, q_t = i].$$

To get maximum likelihood estimates of these second order probabilities I needed to count trimers in the intron training sequences. A 16×4 matrix of contexts and symbols was assembled (see Table 5.2. As mentioned in section 3.2 it would be better to use fifth-order Markov chains

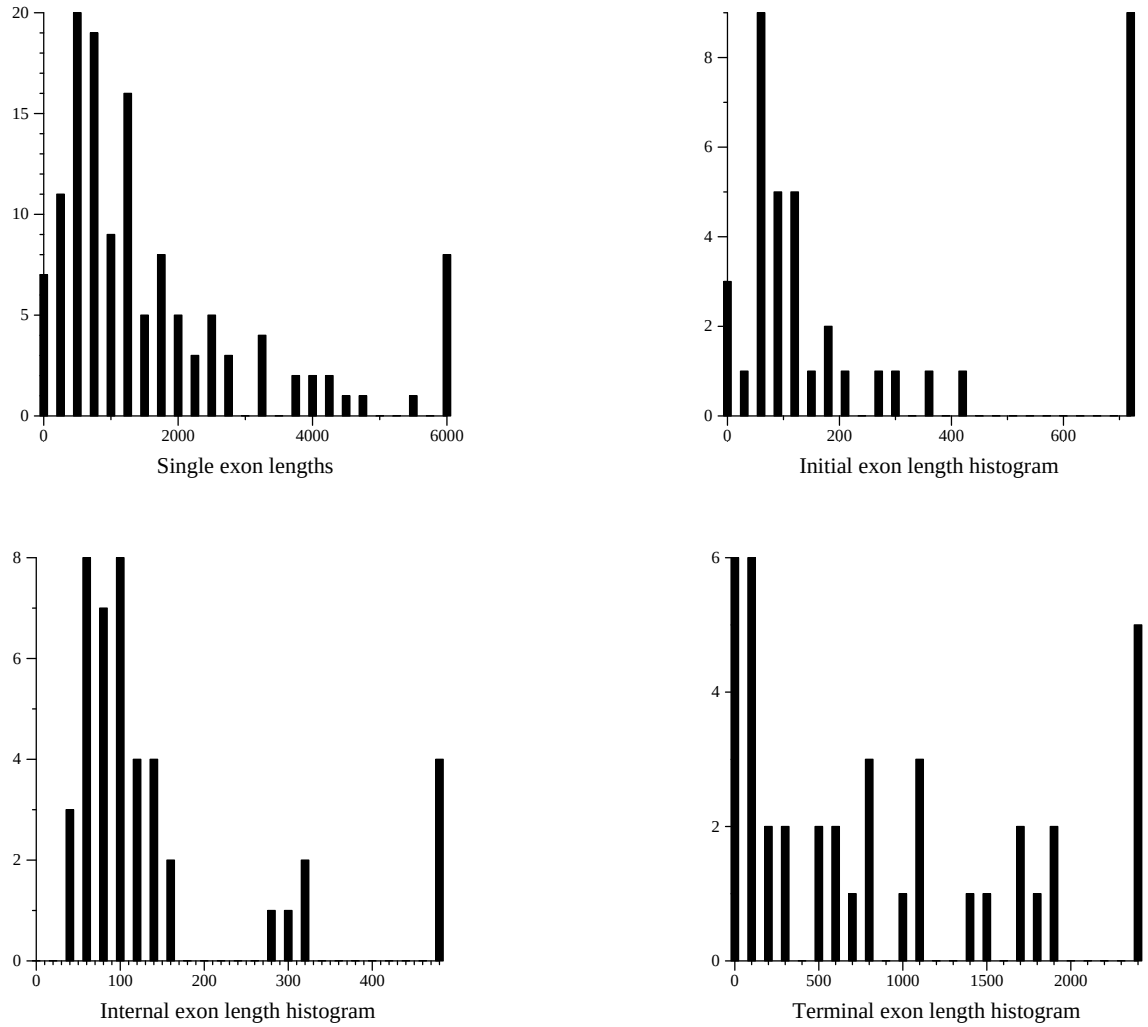

FIGURE 5.5. Exon Lengths from the GenBank sequences

(i.e. hexamer-based) to model nucleotide composition. Unfortunately with so little intron data a fifth-order matrix would have been too sparse (too many 0 entries).

To generate the intergene matrix I used a large contig of "bulk" DNA which actually may have contained many genes. Nevertheless being "bulk" DNA it was thought to model the intergenic regions of the genome reasonably well. The contig was approximately 996 kb and was provided by Mauro DeLorenzi of the Walter and Eliza Hall Institute of Medical Research (WEHI). There would have been more than enough data for a fifth-order model for intergene, but to keep the programming simple I decided to use the same order model for all $\bar{C}$ states.

For the C states I a used fifth-order inhomogeneous Markov chain to model the nucleotide emissions. Viz.

$$\Theta_c(a,b) = \prod_{t=a}^{b} \mathbf{P}[O_t | O_{t-5}^{t-1}, q_t = c].$$

The model was inhomogeneous as three different "transition" matrices were used; one for each reading frame. These matrices were created by counting hexamers in each of the reading frames from the GenBank training sets. One small problem was that some hexamers never appeared, and thus would have had an emission probability of zero. To prevent this problem a pseudocount

of one was added to each frequency. Whilst there were no null frequencies in the intron and intergene matrices, pseudocounts were also added as a safety measure and for consistency.

Creating the reverse strand matrices for the $\bar{C}$ states was relatively easy. For example, to get the entry for position (TA, G), I used the forward value from position (CT, A) being the reverse complement of the trimer. For the backward strand C state matrices I needed to make sure I had the correct reading frame. It turns out that in general if there are R reading frames $0, \ldots, R-1$ (of course we really only care about the case $R = 3$) then reading frame $R-2$ forwards is reading-frame $R-2$ backwards, but for the others reading-frame i forwards is $R-3-i$ backwards and vice versa.

Two points ought to be made about the creation and use of the frequency tables. As it happened, I decided that in the training process the first hexamer used from any exon to count towards the table would be from the first six positions of the coding region, and the last would be the last six. So if the coding region was of length λ then $\lambda - 5$ hexamers would be used. Similarly for the trimers of the $\bar{C}$ states. *However, when I actually applied the Markov chains in the algorithms the first symbol of each state was actually predicted by the five previous symbols in the sequence regardless of the fact that they were not even part of that state.* So the context for the second symbol consisted of the previous symbol in that state and others which were not part of the same state (see below).

```
                          10                  20
          1 2 3 4 5 6 7 8 9 X 1 2 3 4 5 6 7 8 9 X 1 2 3  4 5 6 7
(DNA)     C A A T A T T T G A A T G G T A G G A A T A A  G T G G
(state)   N N N N N N N N N N E E E E E E E E E E E E E  I I I I
```

The nucleotide at position 12 (T) is predicted by the sequence TTGAA even though only the base before it (A) is actually part of the exon. This obviously presents a problem at the left boundary of the observation sequence. My solution was to copy the bases at positions $L-4$ to L into positions -5 to -1 assuming that this wouldn't affect the calculations too much. Of course a better approach would be to change the order of the Markov chain at the left boundaries to be the maximal context that can be obtained within the state and/or predict the bases at the start of an exon in the context of the last few bases of the previous exon (if there was one).

CHAPTER 6

Does my model work?

6.1. Implementation

The Pf genefinder was written in the C language [**KR88**] on a Pentium II running the Debian 2.0 operating system. Its inputs are a file containing the exon/intron statistics (lengths, frequency and transition tables) extracted from the training set, a file containing the statistics generated from the "bulk" DNA, and the DNA sequence for examination. The program returns the log-probability of the sequence and a (possibly empty) sequence of putative CDS start and stop positions, and a list of exons whose probability exceeds some reasonable threshold. These are calculated using the forward procedure, the Viterbi algorithm, and the forward and backward procedures respectively. Sample output is shown in Figure 6.1. Since I don't know anything about repeats in Pf DNA my program pretends that there aren't any. Advice from people at WEHI suggests that this is a reasonable assumption.

(a)

```
P(O) = -7665.662894
```

(b)

```
0.019247 Exon from 240 to 275 of type 2
0.025364 Exon from 187 to 275 of type 2
0.026834 Exon from 240 to 290 of type 2
0.034322 Exon from 215 to 275 of type 2
0.035362 Exon from 187 to 290 of type 2
0.035921 Exon from 234 to 275 of type 2
0.047851 Exon from 215 to 290 of type 2
0.050080 Exon from 234 to 290 of type 2
0.090377 Exon from 395 to 485 of type 2
0.096387 Exon from 212 to 275 of type 2
0.119723 Exon from 752 to 4893 of type 2
0.123714 Exon from 198 to 275 of type 2
0.134381 Exon from 212 to 290 of type 2
0.172479 Exon from 198 to 290 of type 2
0.779357 Exon from 395 to 500 of type 2
0.810937 Exon from 629 to 700 of type 2
0.857805 Exon from 810 to 4893 of type 2
0.981480 Exon from 5098 to 6519 of type 2
```

(c)

```
198
290
395
500
629
700
810
4893
5098
6519
```

FIGURE 6.1. Sample output from my P. falciparum genefinder. (a) The log probability of the sequence (LOCUS PFAPOLA) is about -7666. (b) The exons with probabilities > 0.01 are listed in order of increasing probability—type 2 stands for internal exon. (c) The putative list of exon start and stop positions can be compared with the GenBank annotation: 752 4893 5098 6523.

		Reality	
		coding	non-coding
Prediction	coding	TP	FP
	non-coding	FN	TN

FIGURE 6.2. The 2×2 table

The first implementation was very slow, taking over a minute to analyse a 10kb sequence. Clearly some improvements were necessary. Since the logarithm and exponential functions are considerably slower than even floating point arithmetic it is sensible to execute them as rarely as possible. Logarithms of probabilities are used throughout the program, so I decided to create a lookup table of the logarithms of the transition and symbol emission probabilities. However, not all of these probabilities are used in any particular parse, so the logarithms are only calculated on a need-to-know basis. Secondly, the number of exons examined is $\mathbf{O}(L^2)$, because all possible start and stop positions are checked. It is therefore imperative to speed up the exon probability loop. To this end, I created for each C state a lookup table of the log probability of the symbol sequence from position 1 to position t given that it is in that state, that is $\log \mathbf{P}[O_1^x | Q_1^x = c] = \Theta_c(1, x)$ for all $t \leq L$. This table can be used in the following manner:

$$\begin{aligned}\log \mathbf{P}[O_{x+1}^y | O_1^x, Q_{x+1}^y = c] &= \log \mathbf{P}[O_{x+1}^y | O_1^x, Q_1^y = c] \\ &= \log \mathbf{P}[O_1^y | Q_1^y = c] - \log \mathbf{P}[O_1^x | Q_1^y = c].\end{aligned}$$

6.2. Standard performance analysis techniques

The definitive reference on the assessment of the performance of gene finding programs is that of Burset and Guigó [**BG96**]. They list three tasks which a genefinder should perform and by which they should be assessed: determining whether individual nucleotides are part of exons, predicting overall exon structure and predicting the protein product of the sequence.

6.2.1. The nucleotide level. The first step is to examine the predictions at the nucleotide level. We examine the binary variable "codingness" for each nucleotide, being equivalent to its presence within an exon. We can construct a 2×2 table to represent the association between the predictions of "codingness" and reality, and borrow some terminology from the analysis of 2×2 tables. *True positives* (TP) are those nucleotides which are correctly predicted to be part of exons, whereas *false positives* (FP) are those incorrectly predicted to be part of exons. Similarly there are true negatives (TN) and false negatives (FN)—see Figures 6.2 and 6.3. Whilst the table itself provides the most information it would be convenient to have a single measure to compare programs. The two best known measures are sensitivity (Sn) and specificity (Sp) which

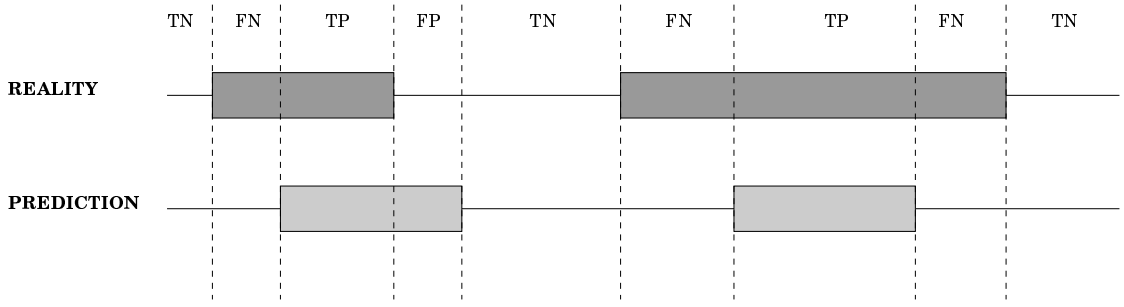

FIGURE 6.3. Comparison between prediction and reality at the nucleotide level. Shading denotes coding region.

are defined as

$$Sn = \frac{TP}{TP + FN} \qquad Sp = \frac{TN}{TN + FP}.$$

Sensitivity is the proportion of actual coding nucleotides correctly predicted as such, whilst specificity is the proportion of non-coding nucleotides correctly predicted as such. Burset and Guigó note that the proportion of non-coding nucleotides is usually much greater than coding nucleotides. Hence $TN \gg FP$ which would always make Sp artificially large and so this definition is unsuitable for genefinder analysis. Despite the fact that in my test data (detailed below) the number of coding bases both predicted and real is larger than non-coding, I used Burset and Guigó's modified measure of specificity

$$Sp = \frac{TP}{TP + FP},$$

being the proportion of bases predicted as being in exons that actually are in exons. It seems sensible however to adopt the new definition in [**BG96**] as it is now standard in the field. Like power and size, sensitivity and specificity should always be used together. By predicting every nucleotide as coding we achieve perfect sensitivity but terrible specificity, and vice versa if we predict only a couple of nucleotides as coding.

A single overall measure is still desirable for comparisons. Originally the correlation coefficient (CC) served this purpose where

$$CC = \frac{(TP \cdot TN) - (FN \cdot FP)}{\sqrt{(TP + FN) \cdot (TN + FP) \cdot (TP + FP) \cdot (TN + FN)}}.$$

A quick check shows that $CC^2 = \frac{\chi^2}{2L}$ where χ^2 is the statistic used to test for independence of the binary variables in the contingency table, and $2L = TP + FP + TN + FN$. One flaw with this measure is that it is undefined if any one of the terms on the denominator is zero, which might happen if the input is a fragment of an exon, a fragment of non-coding DNA, or if the program predicts either of those cases. Burset and Guigó detail one further measure, but settle on the approximate correlation (AC). The approximate correlation is defined in terms of the Average Conditional Probability (ACP) thus

$$AC = (ACP - \frac{1}{2}) \times 2;$$

where the ACP is the average of the terms

$$\frac{TP}{TP + FN}, \frac{TP}{TP + FP}, \frac{TN}{TN + FP}, \frac{TN}{TN + FN}$$

that are defined. Burset and Guigó claim that ACP is a probability (measure). Exactly what ACP is a probability of is not made explicit; and why the weights of the different components should all be equal is not clear either. It is a quantity between 0 and 1 but doesn't seem to be a probability measure and its justification seems empirical rather than theoretical. Nevertheless, since $0 \leq ACP \leq 1$, $-1 \leq AC \leq 1$ which puts AC on the same scale as CC. The justification for AC is that it is always defined (for non-empty sequences) and that in 90% of cases that Burset and Guigó examined $|AC - CC| < 0.05$ even though $|AC| \geq |CC|$.

6.2.2. The exon level. At the exon level we compare the end points of predicted exons with real exons. Historically a prediction is only considered correct when both the end points of the prediction and reality match up exactly. Measures of sensitivity and specificity can be made using this criterion. This approach is however rather stringent, so exons could be considered correct if there is a certain amount of overlap or if one of the predicted end points is accurate. It is also useful to calculate the number of missing exons (ME)—those which are not overlapped by any predicted exon—and wrong exons (WE)—predicted exons which don't overlap any real exons (see Figure 6.4). Note that AC and CC cannot be defined at the exon level as it is not clear what a true negative is.

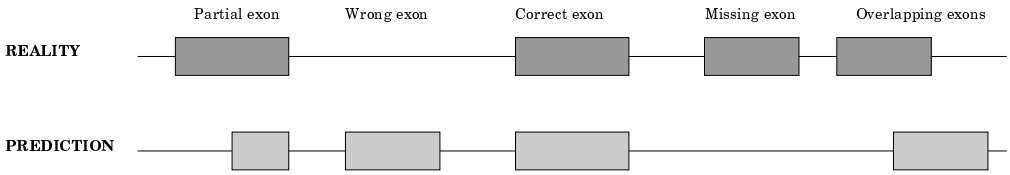

FIGURE 6.4. Comparison between prediction and reality at the exon level

Exon level evaluation adds information to the nucleotide level analysis: it is possible for a program to score well at the nucleotide level but poorly at the exon level. This will usually occur if it is good at distinguishing coding and non-coding regions using base composition, but is unsuccessful at finding signals such as splice sites and translation initiation and cessation sites. Therefore one would guess that my program would perform well at the nucleotide level but not at the exon level.

6.2.3. Protein product analysis. Using various global alignment programs it is possible to assess the final protein product of the prediction. Historically few program authors carried out this step but it is essential because it confirms whether the program can pick the correct reading frame, and because it is what the researcher will use to establish the function of the gene. A crucial step in analyzing protein output is introducing mutations into the test sequence. The sequencing of DNA is likely to have errors, albeit few, and so it is important to test program robustness. As one would expect, programs which use ORF reading-frame constraints, such as mine, tend unfortunately to be more sensitive to such mutations.

6.2.4. How do you summarize the results? If we run tests over many DNA sequences, it is not clear how to combine all the results. Do we calculate the measures for each sequence and then average them, or do we imagine concatenating the many test sequences and then calculate the measure for this enormous sequence? (Clearly the latter approach makes no sense for protein analysis.) Both approaches have been advocated in the literature, although Burset and Guigó prefer the former ("by gene") to the latter ("by base"). They believe that since people wish to know the expected performance for a single given sequence then only the "by gene" approach makes sense. I feel, however, that for the nucleotide level measures in particular, combining the results on a "by base" basis will provide a better estimate. Probably the best idea is to use both methods for program comparison.

6.3. So does it work?

To assess my program I needed some test sequences that were independent of my training sequences. Again these test sequences had to have been annotated already. As it happened I only had the 172 GenBank extracts that I originally used for training. The solution was to run leave-one-out cross-validation trials. That is, I took the first sequence out of the 172 as a test sequence and used the other 171 for training. I then repeated this process for every extract so that I had 172 trials each with a test sequence independent of the training set. The results of the cross-validation trials, in which I applied the stringent "exact endpoints" test to the exons, are listed in Tables 6.1 and 6.2.

TABLE 6.1. Count results of cross-validation trials

		At the exon level there were:
67		correct exons
156	(61%)	actual exons overlapped
159	(64%)	predicted exon overlapped
33	(13%)	missed exons
22	(8%)	wrong exons

	At the nucleotide level there were:
323261	true positives
12866	false positives
89211	true negatives
8142	false negatives

TABLE 6.2. Statistics from cross-validation trials

	Exon		Nucleotide			
	Sn	Sp	Sn	Sp	AC	CC
"by base"	0.26	0.27	0.98	0.96	0.86	0.86
"by gene"	0.24	0.25	0.97	0.97	0.87	0.87
GENSCAN	0.78	0.81	0.93	0.93	0.91	0.92

The most noticeable feature is the difference between the nucleotide level performance and the exon level performance. This shows that the base composition modelling worked very well in distinguishing coding from non-coding regions, but the lack of signal information made it hard to get the end points exactly right. GENSCAN [**BK97, Bur97**] is a genefinder primarily for human DNA and was tested and trained on an entirely different set of data, not using cross-validation. GENSCAN's statistics were summed "by gene" and it clearly benefits from its signal models. In terms of missed exons and wrong exons, my model performs almost as well as GENSCAN (9% and 5% respectively). Indeed my results for missing and wrong exons are better than any of those in the Burset and Guigó trials. This fact and the very high scores at the nucleotide level may be a feature of the Pf genome. Colleagues at WEHI [**Cow98**] say that they can find genes in P. falciparum by eye: this technique is effective but naturally tiring and slow. The difference in AT density, as well as features Saul [**SB88**] employed in his coding score make it relatively easy for trained humans to find genes in Pf DNA. It may be that some of the "wrong exons" found by my program are indeed correct: perhaps the experimenters made errors when determining the CDS annotation for the GenBank entries. In any case, the results are reasonably encouraging given that my model is considerably simpler than Burge's and not nearly as much is known about the Pf genome as the human genome.

It would have been good to test my gene parser on a large contig which included many genes as this is how it would be used in practice. One week prior to the submission of this report, chromosome 2 was released with annotation [**G$^+$98**]. However, the annotation was not in a form which made it possible for me to run tests by the submission date, and in many cases exons were predicted by a computer program rather than by experiment. Whilst this demonstrates the use of genefinding programs, if I were to test my program on this sequence I would be comparing the two programs against each other but not against "the truth".

The program output provides a way of looking at parses of the sequence other than the one deemed optimal by the Viterbi algorithm. The list of exons of high probability gives some idea of what might be near optimal parses and from a researcher's point of view may be worth examining. I haven't made any precise study of this, but I noticed that some exons in the optimal parse have very low probabilities (less than 10^{-5} in one case), and yet the real exon, amongst others, receives a much higher score. It still puzzles me a little how such large discrepancies can occur ... perhaps it is because the Viterbi algorithm looks at joint probabilities whereas the forward-backward procedure returns marginal probabilities.

It would have been interesting to examine the protein product of the genes predicted by my program, and whether it can predict correct reading frame. Mauro DeLorenzi of WEHI has been urging me to do this for some time. As witnessed in the example of Figure 6.1, my program is reluctant to categorize exons as being initial or terminal, preferring *possible* nearby splice sites to the real start and stop codons. This may cause the program to miss the correct $\bar{\mathsf{C}}/\mathsf{C}$ switch site by only a few bases, which would hardly affect nucleotide level results but would be a serious error in a practical sense if the error caused a shift in reading frame. Clearly a (better) functional site model would alleviate this problem. I could also have tested the program's robustness to sequence errors which would have provided more information about its ability to infer the correct reading frame.

6.4. What now?

There are a number of improvements that I should make to my Pf gene finder if it is to perform adequately.

1. Some functional site models, such as those in GENSCAN, need to be incorporated. Burset and Guigó feel that functional site models are less sensitive to the choice of training data than composition-based models as they rely on the data used by the cell machinery in recognizing genes rather than some other artifacts of the genome. Krogh [**Kro98**] details how this could be achieved. Briefly, if any functional site is predicted, a new state is invoked which generates the nucleotides in the vicinity of the site instead of the usual exon, intron and intergene states. It is thought that the bases immediately 5' and 3' of the functional sites have distributions quite distinct from those in the "middle" of the states. This might involve abandoning the $\bar{\mathsf{C}}$, C, $\bar{\mathsf{C}}$ state sequence, but Burge [**Bur97**] does not make it clear how this was done in GENSCAN. Whether or not this is done, and notwithstanding the next point, some effort should be made to read the first few bases of a non-initial exon in the context of the 3' end of the previous exon.
2. To improve robustness to sequencing errors, reading-frame constraints should be relaxed. At various stages in the parse, particularly at splice sites, some factor should be introduced which allows for the sequence to shift frame—with small probability of course. In the current setup, a single exon whose length is not a multiple of three is not even considered as a legal parse.
3. The untranslated regions of P. falciparum genes are (as a proportion of the gene) quite large compared to human DNA for example. This suggests that it is vital to add new states which model the 5' and 3' UTR sections. Moreover, in order to produce reliable trimer/hexamer tables for the intergene state a large source of intergenic DNA is will be required. If this can be found in a large annotated contig with the genes delineated a picture of the length distribution of the intergenic region can be obtained.
4. The geometric length distribution should be dropped. The whole point of the GHMM was to allow for more unusual length distributions. In doing so the $\bar{\mathsf{C}}$, C idea will also have to be abandoned. This will require developing better algorithms for GHMMs perhaps including a reestimation algorithm. (On the other hand, if I do retain the geometric duration distribution then it might be worth trying a HMM for simplicity.)

An article detailing chromosome 2 of P. falciparum and its annotation was published on 6 November 1998. The computer program used in annotating the genome was GlimmerM, a version of Glimmer [**SDKW98**] for eukaryotes. If it turns out that GlimmerM's performance is considerably superior to mine, then I really ought to move on to a different problem. At the moment it is hard to tell, as details of GlimmerM have not yet been published, though I have made contact with Steven Salzberg of The Institute for Genomic Research (TIGR), the senior author.

In any case, the final point in the list above is still one that I needs following up. Somehow I feel the literature on GHMMs has not fully described the procedures related to them and that more needs to be said about what can and cannot yet be done.

CHAPTER 7

Conclusions

For my honours research project I constructed a genefinder for Plasmodium falciparum, using a generalized hidden Markov model. Currently there is no published integrated genefinder for P. falciparum, but references to GlimmerM have been made. Whilst my model was inspired considerably by GENSCAN, a number of significant modifications were made. In keeping with my knowledge of the biology of the gene, many of the more subtle features of the genome such as promoters and poly-A sites were not considered. Moreover, functional site detection was minimal: state transitions took place only if the required `ATG`, `GT` etc. were present. My contribution, I feel, was a more complete probabilistic explanation of the features of the model. Indeed, the principal advantage of the GHMM is that it provides a satisfying way of incorporating many features into a single model. Neural networks perhaps suffer from difficulty in interpreting the meaning of the various scores that are combined. The Markov assumption of the genome region transitions seems reasonable.

My genefinding program performed very well at the nucleotide level but, not surprisingly, considerably less well at the exon level. Clearly further work in developing splice site and translation initiation and termination models for P. falciparum will resolve such problems. A comparison with both Saul's program and GlimmerM, specifically examining prediction of reading frames—a criterion which he (justifiably) thinks is essential—useful in gauging the significance of my work. Before doing this, I would need in particular to ascertain more precisely how Saul's model was applied.

Whilst I was happy with what progress was made, the fixed deadline prevented me from delivering a fuller analysis of my program from a practical point of view. This is just the end of the beginning.

APPENDIX A

Tying up a few loose ends

A.1. The lie exposed

We can condition $\alpha_t(i) = \mathbf{P}[O_1^t, q_t = i]$ on the previous $\bar{C}$ state. In doing so, there are two possibilities: the previous $\bar{C}$ state is of type i at position $t-1$ i.e. a self-transition, or the previous $\bar{C}$ state was further back and there was some intervening C state (the type of which can be inferred from the $\bar{C}$ states: see Figure 5.1). Viz.

$$\begin{aligned}\alpha_t(i) &= \mathbf{P}[O_1^t, q_t = i] \\ &= \sum_{\text{previous } \bar{C} \text{ state}} \mathbf{P}[O_1^t, q_t = i, \text{previous } \bar{C} \text{ state}] \\ &= \mathbf{P}[O_1^t, q_t = i, q_{t-1} = i] + \\ &\quad \sum_{\varepsilon \in E_{i,t}} \mathbf{P}[O_1^t, q_t = i, \text{previous } \bar{C} \text{ state is of type } j \text{ at position } x-1 \text{ with intervening } C \text{ state } \varepsilon],\end{aligned}$$

recalling that exon ε is of type c, length λ, starting at position x, ending at position $t-1$ with $\bar{C}$ state j preceding it. At this point it is convenient to rewrite the right hand side as $\aleph + \sum_{\varepsilon \in E_{i,t}} \beth_\varepsilon$ and determine $\aleph$ (aleph) and $\beth_\varepsilon$ (beth) separately.

Now,

$$\begin{aligned}\aleph &= \mathbf{P}[O_t | O_1^{t-1}, q_t = i, q_{t-1} = i]\mathbf{P}[q_t = i | O_1^{t-1}, q_{t-1} = i]\mathbf{P}[O_1^{t-1}, q_{t-1} = i] \\ &= b_{iO_t} a_{ii}^* \alpha_{t-1}(i),\end{aligned}$$

recalling that $b_{iO_t} = \mathbf{P}[O_t | O_{t-1}, O_{t-2}, q_t = i]$.

Similarly,

$$\begin{aligned}\beth_\varepsilon &= \mathbf{P}[O_1^t, q_t = i, Q_x^{t-1} = c, q_{x-1} = j] \\ &= \mathbf{P}[O_t | O_1^{t-1}, q_t = i, Q_x^{t-1} = c, q_{x-1} = j]\mathbf{P}[q_t = i | O_1^{t-1}, q_t \neq c, Q_x^{t-1} = c, q_{x-1} = j] \\ &\quad \mathbf{P}[O_x^{t-1} | O_1^{x-1}, q_t \neq c, Q_{t-x}^{t-1} = c, q_{x-1} = j]\mathbf{P}[q_t \neq c, Q_x^{t-1} = c | O_1^{x-1}, q_{x-1} = j]\mathbf{P}[O_1^{x-1}, q_{x-1} = j] \\ &= b_{iO_t} \cdot 1 \cdot \Theta_c(x, t-1)\mathbf{P}[q_t \neq c, Q_x^{t-1} = c | O_1^{x-1}, q_{x-1} = j]\alpha_{x-1}(j). \qquad (\dagger)\end{aligned}$$

Referring the state space in Figure 5.1 we note that in a $\bar{C}$, C, $\bar{C}$ sequence the first two states and the length of the C state fix the last. Therefore at $(\dagger)$ in the formula above,

$$\mathbf{P}[q_t = i | q_t \neq c, Q_x^{t-1} = c, q_{x-1} = j]$$

equals 1. However, the interpretation of

$$\mathbf{P}[q_t \neq c, Q_x^{t-1} = c | O_1^{x-1}, q_{x-1} = j] = \mathbf{P}[q_t \neq c, Q_{x+1}^{t-1} = c | q_x = c, q_{x-1} = j]\mathbf{P}[q_x = c | q_{x-1} = j] \tag{5}$$

is not entirely obvious. If we are going to incorporate the values a_{ij} into the model then we cannot simplify $\mathbf{P}[q_t \neq c, Q_{x+1}^{t-1} = c | q_x = c, q_{x-1} = j]$ to $\mathbf{P}[q_t \neq c, Q_{x+1}^{t-1} = c | q_x = c]$. For example, the

next $\bar{C}$ state after an N state is I_i^+ iff state `init exon+` has length congruent to i modulo 3. So if we made this simplification then we want

$$a_{N,I_0^+} = a'_{N,\texttt{initexon+}} \sum_{\lambda \equiv 0 \mod 3} \Lambda_{\texttt{initexon+}}(\lambda) \tag{6}$$

to be true. This is most unlikely as the distributions Λ_c were not defined with the a_{ji} in mind.

The solution is a little tricky ... Firstly define $u_{jc} = \mathbf{P}[q_x = c | q_{x-1} = j]$ to be the sum of the a_{jk} terms for the $\bar{C}$ states k that could follow state c. (The a_{jk} terms are those obtained empirically, as described in section 5.6.1.) This makes u_{jc} like a transition probability. The next step is to create a biased length distribution $\Lambda'_c(\cdot)$ so that, for example, relation (6) is indeed true. Let

$$\mathfrak{L}_{ji} = \left\{ \lambda : \mathbf{P}[q_{x+\lambda} = i | q_{x+\lambda} \neq c, Q_x^{x+\lambda-1}, q_{x-1} = j] = 1 \right\};$$

that is, those lengths that lead to $\bar{C}$ state i after c. For example $\mathfrak{L}_{N,I_0^+} = \{3, 6, 9, \ldots\}$. In general we want the biased distribution to satisfy

$$\mathbf{P}[\text{next } \bar{C} \text{ state is } i \text{ given } \bar{C} \text{ state } j \text{ ends at } x-1] = \mathbf{P}[q_x = c | q_{x-1} = j] \mathbf{P}[\bar{C} \text{ state } j \text{ follows } C \text{ state } c].$$

That is,

$$a_{ji} = u_{jc} \sum_{\lambda \in \mathfrak{L}_{ji}} \Lambda'_c(\lambda). \tag{7}$$

One way to obtain this biased distribution is to introduce bias factors v_{ji} so that $\Lambda'_c(\lambda) = v_{ji}\Lambda_c(\lambda)$ for $\lambda \in \mathfrak{L}_{ji}$. The biased distribution must satisfy (7) so that

$$\begin{aligned} a_{ji} &= u_{jc} \sum_{\lambda \in \mathfrak{L}_{ji}} \Lambda'_c(\lambda) \\ &= u_{jc} \sum_{\lambda \in \mathfrak{L}_{ji}} v_{ji}\Lambda_c(\lambda) \\ &= u_{jc} v_{ji} \sum_{\lambda \in \mathfrak{L}_{ji}} \Lambda_c(\lambda). \end{aligned}$$

Therefore

$$u_{jc} v_{ji} = a_{ji} \Big/ \sum_{\lambda \in \mathfrak{L}_{ji}} \Lambda_c(\lambda).$$

The factor in $\beth_\varepsilon$ that we couldn't calculate was (5), which means the missing piece is

$$u_{jc}\Lambda'_c(\lambda) = u_{jc}v_{ji}\Lambda_c(\lambda) = \frac{a_{ji}}{\sum_{\lambda \in \mathfrak{L}_{ji}} \Lambda_c(\lambda)} \Lambda_c(\lambda) \equiv a'_{ji}\Lambda_c(\lambda),$$

and *not* just $a_{ji}\Lambda_c(\lambda)$ as I had supposed when I first wrote the computer program. Therefore

$$\alpha_t(i) = b_{iO_t} a^*_{ii} \alpha_{t-1}(i) + \sum_{\varepsilon \in E_{i,t}} b_{iO_t} \Theta_c(x, t-1) a'_{ji} \Lambda_c(\lambda) \alpha_{x-1}(j)$$

This modification extends to the Viterbi algorithm and the backward procedure. I feel much better now owning up to all that.

A.2. The probability of a particular exon

Again referring to the state space diagram, the knowledge of state j is redundant in fixing state i: the type of C state and its length determine the previous and subsequent $\bar{C}$ states. For

example, if the current C state is `exon 2-` of length 23 then the next $\bar{C}$ state *must* be state I_2^- and the previous state is I_1^- because $2 + 23 \equiv 1 \mod 3$. With this in mind ...

$$\begin{aligned}
\mathbf{P}[Q_x^y = c, O] &= \mathbf{P}[O_1^L, Q_x^y = c, q_{x-1} = i, q_{y+1} = j] \\
&= \mathbf{P}[O_{y+1}^L | O_1^y, Q_x^y = c, q_{x-1} = i, q_{y+1} = j] \mathbf{P}[O_1^y, Q_x^y = c, q_{x-1} = i, q_{y+1} = j] \\
&= \beta_{y+1}(j) \Theta_c(x, y) a'_{ij} \Lambda_c(\lambda) \alpha_i(x-1),
\end{aligned}$$

where most of the last line follows from the calculation of $\beth_\varepsilon$ in the previous section. Of course this calculation shouldn't be made if the appropriate consensi aren't at positions x and y.

APPENDIX B

GenBank loci

LOCUS	ACCESSION
A00661	A00661
A00663	A00663
A04562	A04562
A12418	A12418
A13449	A13449
A13455	A13455
A13457	A13457
A13459	A13459
A13463	A13463
A13469	A13469
A13471	A13471
A28741	A28741
A30787	A30787
A32124	A32124
A32873	A32873
D86573	D86573
D86574	D86574
PFADHFRTS	J03028
PFAABRA	J03902
PFAGAR	J03998
PFA412ANT	J04656
PFAGIPA	J05544
PFATPIX	L01654
PFAGAMSPEC	L04161
PFAS230A	L04162
PFATBP	L06060
PFAASPR	L06303
PFASPAM	L07944
PFAVAPA	L08200
PFARPI	L11172
PFAMRNA	L12043
PFADDGHOM	L15446
PFAPOLA	L18785
PFACPSI	L32150
PFADAAS	L46348
PFASA7	M10130
PFAGBPAB	M12897
PFAANT	M13021
PFAPP300A	M19462
PFAHSP70H	M19753
PFAACTIIA	M22718
PFAACTIA	M22719
PFAAMA1A	M27133
PFATUBB	M28398
PFAALD	M28881
PFAMEM12A	M28889
PFAMALX	M31305
PFAATUBII	M34390
PFAPHOSKIN	M59249
PFAGLURPA	M59706
PFABSA	M59961
PFAGBPH	M65160
PFAMESA	M69183
PFACBP	M77834
PFACYPROT	M81341
PFAHMGLP	M86518
PFAH2AA	M86865
PFAHGPTA	M88110 + Y00519
PFAERBPB	M93397
PFALDH	M93720
PFAVLAP	M94732
PFAENOL1	U00152
PFU01322	U01322
PFU01323	U01323
PFU03915	U03915
PFU07365	U07365
PFU07706	U07706
PFU08113	U08113
PFU10322	U10322
PFU14189	U14189
PFU14734	U14734
PFU15994	U15994
PFU16955	U16955
PFU16995	U16995
PFU25814	U25814
PFU34363	U34363
PFU37225	U37225
PFU38963	U38963
PFU40228	U40228
PFU41269	U41269

LOCUS	ACCESSION
PFU49381	U49381
PFU51645	U51645
PFU54642	U54642
PFU56663	U56663
PFEXP1G	X05074
PF22R	X06426
PF111A	X07453
PF111C	X07455
PF111E	X07457
PFS25	X07802
PFTUBAI	X15979
PFPOLII	X16561
PFA14C1	X17483
PFA17C1	X17484
PFA18C1	X17485
PFA25C4	X17486
PFA28C6	X17487
PFA52C11	X17489
PFA53C6	X17490
PFLSA1G	X56203
PFPFMDR1	X56851
PFRAP2	X58777
PFEF1	X60488
PFPFC2	X61921
PFGTUB	X62393
PFPOLD	X62423
PFCPK	X67288
PFGBPH2	X69769
PFCAATPAS	X71765
PFRAN1	X73954
PFGLPH	X74988
PFASPHEM	X75787
PFSERPIG	X77854
PFTBPM	X77914
PFTOPOII	X79345
PFCRK1	X80759
PFRNATRAN	X83551
PFDNAPK1	X83707
PFTOPOI	X83758
PFDNACPT	X84041
PFG27	X84904
PFRNACYC	X85956
PFPF70GEN	X91661
PFRAB6GTP	X92977
PFRAB11GN	X93161
PFRABGDI	X93166
PFORF9	X95373
PFMAP2	X98689
PFPRIMSSU	X99254
PFKAHRP	Y00060
PFS4548A	Z22145
PFERD2	Z26043
PFSTARP	Z26314
PFUNKORF	Z37724
PFGPXGN	Z68200
PFARF	Z80359
PFAARPS2	Y08924
PFU73195	U73195
PFU69552	U69552
PFU69551	U69551
AF003086	AF003086
AF008549	AF008549
PFU78753	U78753
PFU85963	U85963
PFCDPK2	X99763
AF003473	AF003473
PFJ002197	AJ002197
PFFESODMR	Z49819
AF023665	AF023665
PFPK4GENE	X94118
AF012551	AF012551
AF017139	AF017139
PFGCSGEN	Y14674
AF031144	AF031144
PFU84403	U84403
PFU94594	U94594
PFRNACRK3	X87839
PFY17045	Y17045
AF056936	AF056936
AF006204	AF006204
PFJ002233	AJ002233
AF030694.1	AF030694.1
AF030694.2c3	
AF030694.4c	
AF030694.5c	
AF030694.6	
AF030694.7c	
AF030694.8	
AF030694.9	
AF030694.10c	
AF030694.11c	
AF030694.12c	AF030694.12c

Bibliography

[BG96] Moisès Burset and Roderic Guigó, *Evaluation of gene structure prediction programs*, Genomics **34** (1996), 353–67.

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

[BK98] Christopher B Burge and Samuel Karlin, *Finding the genes in genomic DNA*, Current Opinion in Structural Biology **8** (1998), 346–54.

[BPSW70] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss, *A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains*, The Annals of Mathematical Statistics **41** (1970), no. 1, 164–71.

[Bur97] Christopher Burge, *Identification of genes in human genomic DNA*, Ph.D. thesis, Stanford University, March 1997.

[Cow98] Alan Cowman, 1998, Private Communication.

[D+96] John B. Dame et al., *Current status of the plasmodium falciparum genome project*, Molecular and Biochemical Parasitology **79** (1996), 1–12.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*, Cambridge, 1998.

[DeL98] Mauro DeLorenzi, 1998, Private Communication.

[Fic96] James W. Fickett, *Finding genes by computer: the state of the art*, Trends in Genetics **12** (1996), no. 8, 316–20.

[FT92] James W. Fickett and Chang-Shung Tung, *Assessment of protein coding measures*, Nucleic Acids Research **20** (1992), no. 24, 6441–50.

[G+98] Malcolm J. Gardner et al., *Chromosome 2 sequence of the human malaria parasite Plasmodium falciparum*, Science **282** (1998), 1126–32.

[Gel95] M. S. Gelfand, *Prediction of function in DNA sequence analysis*, Journal of Computational Biology **2** (1995), no. 1, 87–115.

[Gel97] M. S. Gelfand, *Functional analysis of nucleotide sequences reference database*, `http://www-hto.usc.edu/software/procrustes/fans_ref/index.html`, February 1997.

[GW91] Larry Gonick and Mark Wheelis, *The cartoon guide to genetics*, Updated ed., HarperPerennial, New York, 1991.

[HKMS93] David Haussler, Anders Krogh, I. Saira Mian, and Kimmen Sjölander, *Protein modelling using hidden Markov models: analysis of globins*, Proceedings of the Hawaii International Conference on system sciences (Los Alamitos, CA), IEEE Computer Society Press, 1993.

[HS87] John E. Hyde and Paul F.G. Sims, *Anomalous dinucleotide frequencies in both coding and non-coding regions from the genome of the human malaria parasite Plasmodium falciparum*, Gene **61** (1987), 177–87.

[KBM+94] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjölander, and David Haussler, *Hidden Markov models in computational biology: Applications to protein modelling*, Journal of Molecular Biology **235** (1994), 1501–31.

[KHRE96] David Kulp, David Haussler, Martin G. Reese, and Frank H. Eeckman, *A generalized hidden markov model for the recognition of human genes in DNA*, Procedings of the Fourth International Conference on Intelligent Systems for Molecular Biology (Menlo Park, CA), AAAI Press, 1996.

[KMH94] Anders Krogh, I. Saira Mian, and David Haussler, *A hidden Markov model that finds genes in E. Coli DNA*, Nucleic Acids Research **22** (1994), no. 22, 4768–78.

[KR88] Brian W. Kernighan and Dennis M. Ritchie, *The C programming language*, Second ed., Prentice Hall, Englewood Cliffs, NJ, 1988.

[Kro98] Anders Krogh, *An introduction to Hidden Markov Models for biological sequences*, Computational Methods in Molecular Biology (S. Salzberg, D. Searls, and S. Kasif, eds.), Elsevier, 1998.

[LB98] Alexander V. Lukashin and Mark Borodovsky, *Genemark.hmm: new solutions for gene finding*, Nucleic Acids Research **26** (1998), no. 4, 1107–15.

[Lew87] Benjamin Lewin, *Genes*, Third ed., Wiley, New York, 1987.

[Li98] Wentian Li, *Bibliography on computational gene recognition*, `http://linkage.rockefeller.edu/wli/gene/`, October 1998.

[Pow89] Robin D. Powell, *Malaria and babesiosis*, Tropical Medicine and Parasitology (Robert Goldsmith and Donald Heyneman, eds.), Appleton & Longe, Norwalk, Conn., 1989.

[Rab89] Lawrence R. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), no. 2, 257–85.

[REKH97] Martin G. Reese, Frank H. Eeckman, David Kulp, and David Haussler, *Improved splice site detection in Genie*, Journal of Computational Biology **4** (1997), no. 3, 311–23.

[SB88] Allan Saul and Diana Battistutta, *Codon usage in Plasmodium falciparum*, Molecular and Biochemical Parasitology **27** (1988), 35–42.

[SB97] Phillip A. Sharp and Christopher B. Burge, *Classification of introns: U2-type or U12-type*, Cell **91** (1997), 875–9.

[SDKW98] Steven L. Salzberg, Arthur L. Delcher, Simon Kasif, and Owen White, *Microbial gene identification using interpolated Markov models*, Nucleic Acids Research **26** (1998), no. 2, 544–8.

[SW98] Xin-Zhuan Su and Thomas E. Wellems, *Genome discovery and malaria research: Current status and promise*, Malaria: Parasite Biology, Pathogenesis, and Protection (Irwin W. Sherman, ed.), ASM, Washington, D.C., 1998.