Sequence alignment in bioinformatics

Ewan Birney
Sanger Centre
Wellcome Trust Genome Campus
Hinxton, Cambridge CB10 1SA,
England.
Email: birney@sanger.ac.uk

March 28, 2000

Contents

List of Tables

List of Figures

Preface

Many people have helped my scientific career which has led up to this thesis. At Cold Spring Harbor, Adrian Krainer and Akila Mayeda were great teachers for a young student in molecular biology. I have also benefited from the timely interjections of Jim Watson as I have bounced around institutions. At Oxford, Iain Campbell allowed me to work unhindered for a year, which I am very grateful for.

The most rewarding aspects of my work have come in the last three years. At the Sanger Centre I have been stimulated by, amused by and grown with the researchers around me: in particular, Ian Holmes, Dan Lawson, Alex Bateman and Kevin Howe have made my working life a pleasure. Michele Clamp and James Cuff were two friends who's scientific minds (not to mention their out of hours companionship) has made life at Hinxton more enjoyable.

Finally, Richard Durbin has been a wonderful supervisor of a potentially impossible student. He has taken some pretty ill formed ideas and formed sensible research from it: he has moderated my stubbornness and tolerated my distractions. Finally he has provided good humour and stimulating discussions. Many thanks Richard.

Chapter 1

Introduction

The cost of sequencing DNA is around 10 pence a base; one example of how inexpensive data has become in biology. Rather than investigating each aspect of the biology of an organism piecemeal, huge quantities of data can be gathered at once. For example, an entire genome can be sequenced for an achievable cost. The genome of an organism encodes in principle nearly all the information of its biology. So for an outlay which is relatively small compared to the worldwide annual expenditure on biomedical research a collection of data is being generated that will underpin nearly all aspects of human biology.

These simple economic facts have started a revolution in biology. Biological science used to be a science of craftsmen producing individual, tailored facts that painted a single picture: suddenly part of it is becoming a science of large scale data production. At the junction of the new and the old techniques lies a new science, *bioinformatics*, trying to organise the mass of data into large sets of coherent images that can be viewed along side the traditional biologist's work.

Bioinformatics is a science which simply did not exist in its current form ten years ago. Computer scientists and biologists could confidently expect never to cross paths in a professional capacity. Although there have been many interesting applications of both mathematics and computers to biological problems before the 90s, these have been in specialised fields with narrow applications, such as statistical genetics or modeling protein structure. For the larger questions in biology, biological experiments were by far the best way to progress. However, with the advent of such a large amount of biological data the biologists are having to use computational and mathematical techniques to try to manipulate the data into interpretable information. Because of this there are quantitative scientists being recruited into the field. These researchers are often more at home with sciences with well defined rules and a mathematical foundation; they find that biology has few rules and many exceptions, but tantalizing applications for theories that have been developed in more quantative fields.

The pace of change and the clash of cultures makes bioinformatics a wonderful science. Everyone is always learning: new aspects of biology or computer science depending on their background, or new technologies which threaten to give us another mass of data for some biological problem. The people who participate can find radically new perspectives on old problems simply from the cross pollination between the two fields.

This thesis is focused on one area of bioinformatics, using dynamic programming to analyse sequences to extract useful information. Each chapter concentrates on the generation of one programmatic tool or resource. Each tool is presented by examining first the background and theory of its use, and then the programmatic issues which ensure efficient and useful applications. For tools which produce biological results, I present an evaluation of its effectiveness. Finally each chapter ends with a discussion of the success (or failures) of the tool and any future directions.

The rest of the introduction will describe the background theory that the other chapters rely on: in particular *probabilistic finite state machines* (PFSMs) and their applications to biology will be throughly examined. Chapter 2 describes a programming toolkit, *Dynamite* that automates the programming of these PFSMs. Chapter 3 is a real applicatios of Dynamite, in gene prediction, where I provide a principled way to combine *ab initio* and similarity based approaches. Chapter 4 describes the work I did on *Pfam*, a database of protein families and Chapter 5 describes how I applied Pfam on a genome wide scale.

I hope you enjoy reading the remainder of the work.

1.1 Information flow in biology

Figure 1.1 shows the basic flow of information between the genome (top of the figure) and the phenotype (bottom of the figure). Researchers are interested in the molecular function that leads to the phenotype. The genome provides the data which can be determined in an automatic fashion. This sequence information is an invaluable resource in its own right, providing information for molecular biologists to design and produce new experiments. Another use of the sequence data is to be able to predict the function of the sequence without any additional experiments. One role of bioinformatics is to somehow help bridge the gap between the sequence data and the functional data. This transformation of genome sequence to the functional information is understood in broad terms in the sense that we know the biological players, but nowhere near well enough to allow a mechanical deduction.

The five basic steps from the genomic sequence to function, being transcription, pre-mRNA splicing, translation, protein folding and the final function of the protein product mean that there is considerable manipulation of the information in genomic sequence before it is actually produces a biological effect. This transformation of information is understood extremely well for one step, translation. For transcription and pre-mRNA splicing there is a partial understanding of the biological process which provides this manipulation, and for protein folding and the function of the protein product our understanding is far from ideal.

The inability to deduce the function of a piece of genomic sequence simply by understanding the biological machinery which processes it has forced a different approach to the problem of deriving functional information from sequence. This is to make arguments based on homology i.e. that two sequences shared a common ancestor sequence at some point of evolution and since their divergence have kept many features the same, from sequence to structure to function.

By comparing two sequences, either genomic DNA, mRNA or protein sequences, some inference of whether they really share homology can be gained. This inference has to be some type of statistical test which provides a way of distinguishing non homologous pairs from homologous pairs. Given that a researcher accepts the inference, he or she can then assert that at least some of the protein structure is shared between the two genes. In addition, in many cases the functional information can be transfered. There are cases where the transfer of functional information by homology is plain wrong: for example the gamma lens crystallin has a clear homolog in alcohol dehydrogenase, but their functional roles, the former to refract light in the lens and the latter to metabolise alcohol are unrelated. However, the structural similarity is clear. Biology has simply co-opted this enzyme for a different role. Thankfully such case are rare, and transfer of functional information is in general

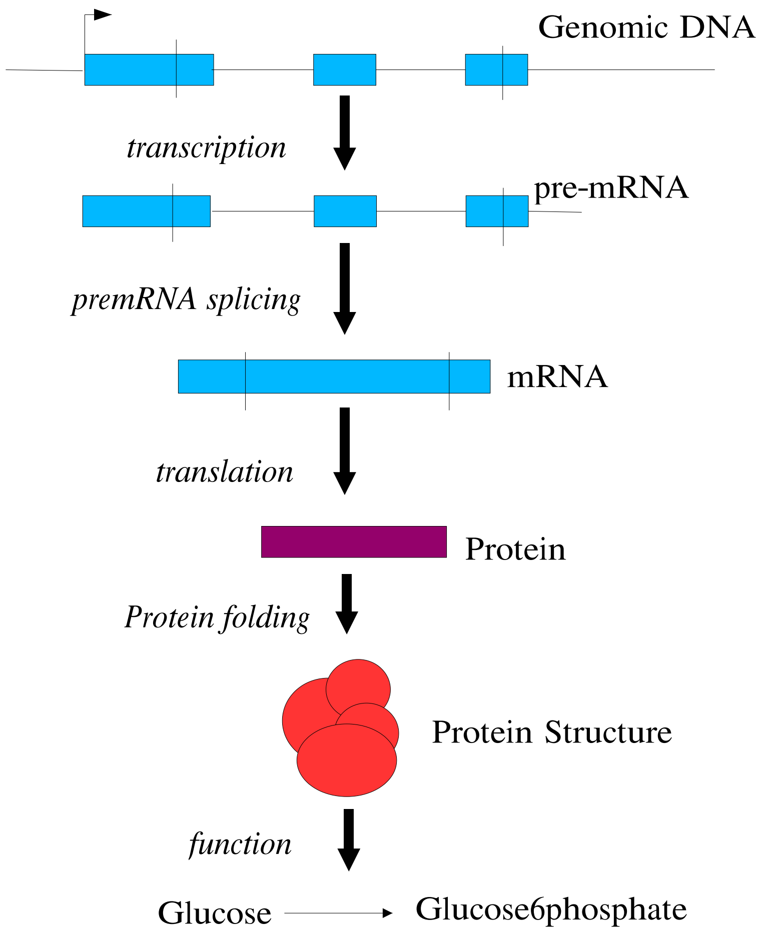


Figure 1.1: A figure showing the central dogma in biology. The top is genomic DNA: going down the page are the main processes which transform the information in the genomic DNA to the functional aspects of the organism. The exons are in blue, and the introns and intergenic DNA as thin lines. The start and stop codons are represented as thin vertical lines. The magenta rectangle represents a linear protein sequence, and the red circles its three dimensional structure.

reliable.

Much of bioinformatics is therefore interested in one of the two following problems. The first problem is providing computer programs which model the transformation of information from one biological entity to another, in other words, modeling in a computer one of the steps in figure 1.1. The second problem is providing computer programs which model the process of evolution between two homologous genes. Both these problems can be well described as Finite State Machines, which leads us to the next section.

1.2 Probabilistic Finite State Machines

Finite state machines (FSMs) are well known computer science constructs which have a wide range of uses from the abstract theory computation to a variety of practical problems, including speech recognition and process engineering. They are also a good fit to biological sequence analysis, as biological sequences are represented well by linear chains of letters. This section will briefly introduce finite state machines, and the following section will review some of the previous work in bioinformatics that uses them.

A finite state machine has a number of *states* connected by *transitions*. The machine starts in one state, moves from state to state and stops when it reaches a particular state. The action of moving between the states on the basis of the transitions produces a sequence of observable events, such as sounds, readings from a machine or biological sequence data [1]. An important point is that the choice of the next transition is not dependent on the previous transition (or, in some cases, only dependent on a limited number of proceeding states). One nice feature of finite state machines is that an infinite set of observations can be generated, despite the finite nature of the machine. The machine in Figure 1.2 could generate (ab) or (aab) or (aaabbbbb) but not (aabbaa): indeed the machine will generate any $a_n b_m$ string. This machine only distinguishes between strings which are valid ($a_n b_m$) or not valid.

The most common extension of Finite State Machines used in this field is to associate probabilities with the transitions, and their associated emissions. The effect of this is to assign a probability of certain strings emerging from the machine.

[1] for readers with a background in finite state machines, I will be consistently using the Mealy representation of FSMs, with emissions occurring on the transitions throughout

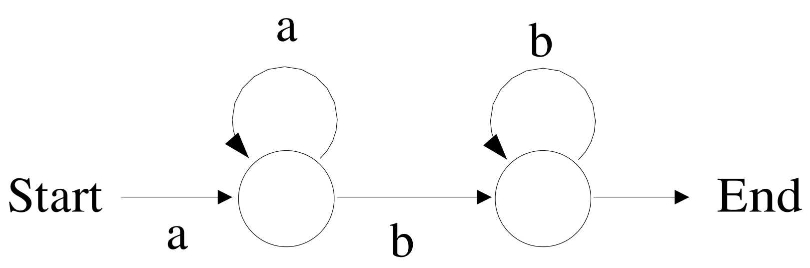


Figure 1.2: A finite state machine which generates a string of a's followed by a string of b's. The round circles represents states, and the arrows between them transitions. On each transition, potentially a letter can be *emitted* which becomes the observed data. The starting and stopping criteria of this state machine are shown by rectangles

For example, figure 1.3 shows the previous figure but now parameterised with probabilities. This machine would generate the string (aaaaaaaab) with a far higher probability than (aabbbbbbbbbb).

One final property of FSMs is that it is not always the case that one can deduce the state of the machine by the sequence of observed letters. The machine I have used as in examples so far is one in which the current state of the machine can be deduced from the sequence of letters: such a machine is called a deterministic finite state machine. However, it is very easy to construct machines in which this is not the case. An example is shown in figure 1.4. Many paths can be consistent with a particular observed string of letters in nondeterministic finite state machines, meaning that it is not clear which state path made a particular observed string.

For PFSMs one natural state sequence to consider is the one with the highest probability of generating observed symbols. This is known as the most likely state path which is calculated by an algorithm called the Viterbi path. The probability of the Viterbi path is called the Viterbi score. This probability is generally not the same as the total probability considering every path, which is sometimes called the sum of all paths or the Forward score. I will describe the algorithmical details for calculating these scores and the Viterbi path in the next chapter.

Nondeterministic, probabilistic finite state machines are the main types of ma-

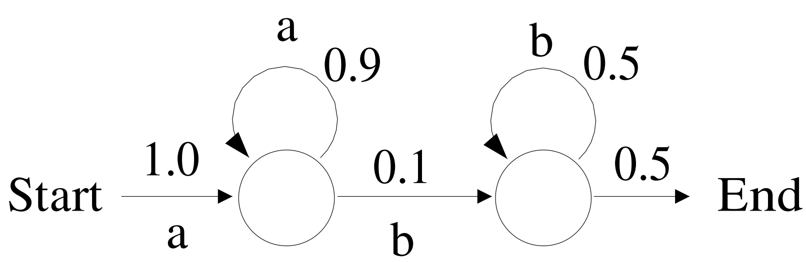


Figure 1.3: The finite state machine in figure 1.2 parameterised with probabilities. Each transition has a probability associated with it, including the transition leaving start, which has a probability of 1. The probabilities for all the transitions leaving a particular state sum to one

chines used in bioinformatics. They are equivalent to *hidden Markov models*. A hidden Markov model is a mathematical model which evolves in some dimension (often time, but need not be) via Markov rules, and has certain variables which are not observable. In the case of PFSMs, the hidden variables is the path information through the state machine, i.e. in figure 1.4 which sequence of states was used to generated the b letters. The two descriptions of the process as "probabilistic Finite State Machine" and "hidden Markov model" are entirely equivalent. I prefer using the PFSM formalism as for me the description of the process is both more intuitive and closer to the programmatic implementation.

1.3 PFSMs in bioinformatics

There are two main types of probabilistic finite state machine which are used in bioinformatics, corresponding to the two types of problems to solve. The first has each transition emit a single letter of an observed sequence. This type of model is the familiar hidden Markov model, which is used in speech recognition and other fields. In bioinformatics this type of model includes profile hidden Markov models for protein domains, simple models of protein sequence, and gene prediction hidden Markov models. The states represent theorized explanations of the observations (different phonemes that make up a word in speech recognition, and the different

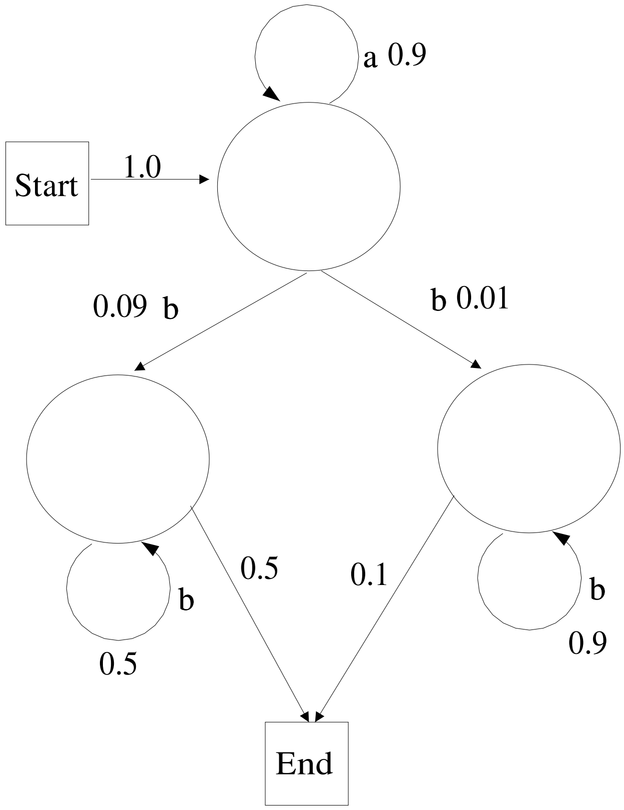


Figure 1.4: A non deterministic finite state machine, which produces the same type of language as the previous finite state machines, namely $a_n b_n$ but there are two classes of strings generated, one with short b tails, coming from the left hand side of the figure and one with long b tails, coming from the right hand side. The long b tail class is rarer than the short b tail class. This can be deduced by the probabilities of the machine. For a particular string of letters, one path through either the left or the right side of the machine will have a larger probability of producing that string of letters: however, which was used cannot be known for certain.

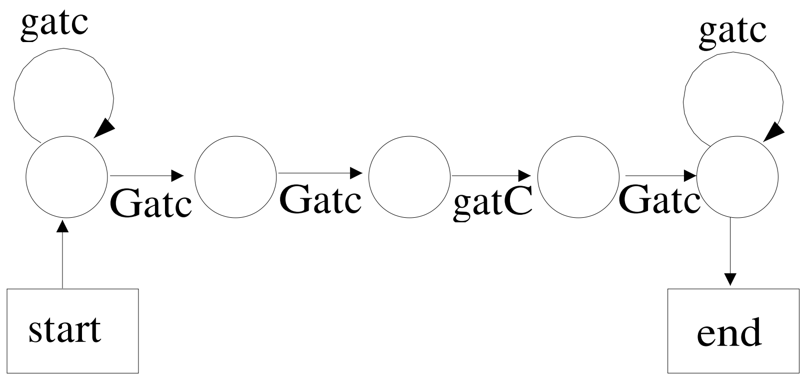
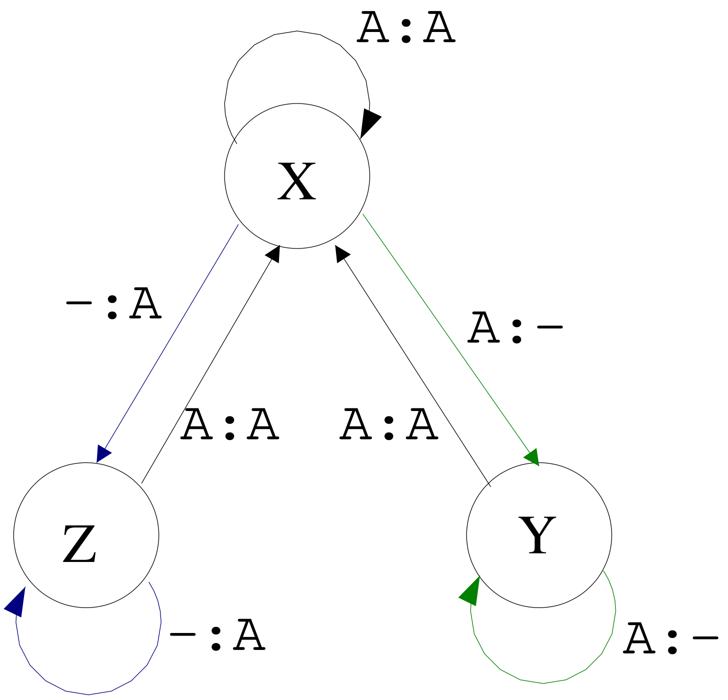


Figure 1.5: A figure showing a simple PFSM which models a small DNA motif, in this case a sp-1 transcription factor binding site, with a consensus of GGCG. The rectangular states show the start and end points. The two looping transitions emit bases at equal probabilities, generating random sequence either side of the motif. The other transitions emit bases with a strong G or C bias (indicated by capitalisation). This model is a very simple model: more complicated transitions can model more complex processes

biological features, e.ġexons and introns in gene prediction models). An example is show in figure 1.5 which models a motif in a DNA sequence.

The second type of model is one which emits letters from two different sequences of observations. Outside of bioinformatics this type of model is reasonably rare: the only mainstream application is for transducing grammars (grammars which convert one language to another). However, in bioinformatics, this model is ubiquitous, as it represents the alignment of one sequence to another. Figure 1.6 shows the common Smith-Waterman type PFSM which represents the alignment of two DNA sequences by a process allowing affine gaps scores.

In this alignment type PFSM, the states represented theorized explanations of evolution, for example the process of inserting an amino acid. In the PFSM structure, finding the most likely path incorporates finding the best position of the gap characters in the sequence, i.e. determining the alignment.




```
Seq1: ATGGGTGGT-----GGTT
Seq2: ATG--TGCTGGTATTGTT
State XXXYYXXXXZZZZZXXXX
```

Figure 1.6: Figure showing a PFSM of an alignment of two sequences. The PFSM has transitions which emit a pair of letters, one from each sequence, potentially with gap '-' residues which indicate the absence of a letter.

1.3.1 Complex FSMs

The PFSMs described so far have been relatively simple. More complex machines can be written in the same formalism. For example, figure 1.7 shows a model of a bacterial sequence containing a single coding gene. The top panel shows this drawn out as individual states, each transition emitting a single base. The lower panel shows this contracted to a smaller state diagram but with the transitions emitting more than one base at a time.

The lower panel is both a far more compact way of representing a the PFSM and is also closer to how the implementation in a programming language works. Notice that on the state after the main codon state (in brown) the transitions which enter the state have different lengths of emissions: in one case they are 3 base pair codons, and in the other case one base pair DNA sequence. Having different length emissions leading to the same state is a common occurrence in bioinformatics. This is why I prefer the Mealy representation of PFSMs which has the emissions on the transitions, as I have done throughout this work. The other representation of PFSMs is the Moore representation, which has the emissions occurring on the states, and is perhaps more commonplace. The two representations are inter-convertible: in my hands, Mealy machines are clearer representation of PFSMs than Moore machines for the problems I am interested in.

1.4 Previous use of PFSM in bioinformatics

There have been a large number of applications of probabilistic finite state machines in bioinformatics over the last decade. This includes both "standard" PFSMs such as hidden Markov models for modeling DNA sequence and protein sequence, and alignment PFSMs. The following sections goes into more details on specific examples of PFSMs in bioinformatics.

1.4.1 DNA composition models

One of the earliest explicit uses of hidden Markov models (and hence PFSMs) in bioinformatics was to model DNA composition. Gary Churchill provided a number of HMMs to deduce properties of DNA sequence, such as GC content and eukaryotic isochores [22]. He calculates the marginal distribution of what state (with some

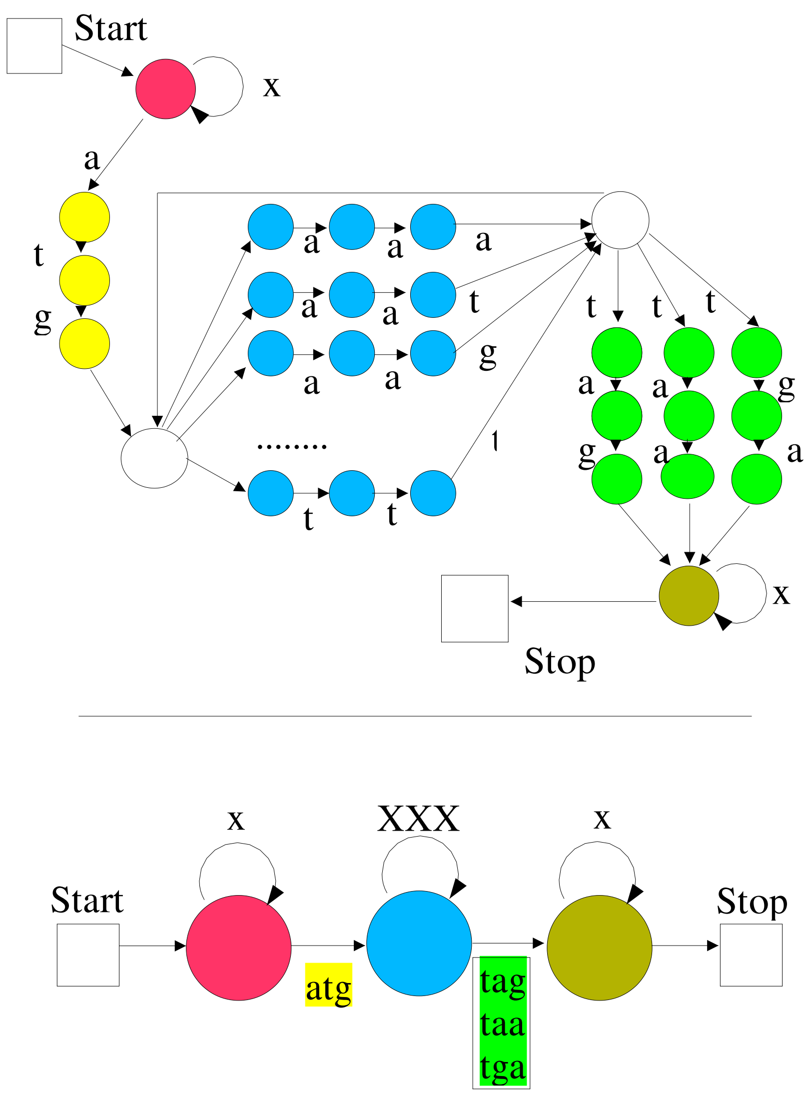


Figure 1.7: Figure showing two representations of the same PFSM, which models a piece of bacterial sequence that contains a single protein coding gene. The colours are consistent between the two views. On the top panel, PFSM equivalent to a zero order HMM is written out, with one or zero letters emitted on each transition. x stands for any base, and the ... region indicates that the three state block is repeated over all 61 codons. The lower panel is the same PFSM equivalent to a second order HMM is shown. The XXX indicates any triplet, each one of which will have a separate probability.

defined model) produced a particular sequence: the defined models he uses are inspired by known biological phenomena, such as GC islands in the yeast genome and isochore switching in eukaryotes.

1.4.2 profile hidden Markov models

A number of groups have employed a type of PFSM which models protein sequences, called *profile-hidden Markov models* due to its similarity to profile analysis which had been previously employed by a number of groups in bioinformatics. Anders Krogh and colleagues were the first group to provide a clear definition and implementation of this sort of model, [50]. They defined a restrictive architecture of hidden Markov model which is based around repetitions of 3 states, called Match (M), Insert (I) and Delete (D). This choice of architecture was deliberately modeled after previous work on profile analysis [5, 38, 13, 57], which they acknowledge in the paper. The rest of the work details how they applied standard HMM techniques to this problem, including parameter training using expectation maximisation and using the Viterbi algorithm to discover the best path through a number of sequences, providing a multiple alignment. The paper also solves a number of practical problems in using this type of HMM. Firstly they introduce additional states to model the fact that a conserved region might be embedded inside a protein sequence which is otherwise unrelated to the HMM. These additional states (called "dummy states") are added at the start and the end of the HMM. Secondly they provide a useful statistic to estimate the significance of the match of a HMM to a sequence. They found that the likelihood of the sequence given the model over all paths (called Negative Log Likelihood, NLL score) is strongly correlated with the length of the sequence. Therefore they provided a way of correcting for this correlation, reporting a Z-score of the number of standard deviations away from a windowed length bin of sequences, discarding outliers. The final Z score is used as a statistic, and they suggest using a Z-score of 5 standard deviations as being a sensible cutoff, but suggested caution in trusting automatic cutoffs. They asses their work by modeling three families: the globins, the protein kinases and the EF hands. In each case they were able to recover nearly all known examples of the domain and find potentially novel examples of the domains in the protein database. Also they show that the multiple alignments which were generated by the HMM are sensible.

Another strong group in the profile hidden Markov model field is Sean Eddy and colleagues [31, 29]. The profile-HMM software package written by Sean Eddy, called HMMER has become the most widely used profile-HMM package in bioinformatics. A number of important enhancements to profile-HMMs are incorporated into the package which makes it ideal for practical use. Firstly, the problem of estimating the model size (in terms of the number of match, insert, delete nodes used) is solved using an exact method which estimates the posterior probability of whether a column in an alignment should be modeled at a match or insert state. Secondly the additional states which allow a profile HMM to model a domain rather than a entire sequence was extended to allow for multiple copies of a domain to be modeled with additional random sequence between the copies. This changes the architecture of the HMM, and at the same time, two transitions between insert to delete and delete to insert were dropped in the new architecture (informally called "plan7"). These changes made HMMER a far better package for modeling protein domains in sequences. Thirdly the statistical reporting of whether a reported match represents a particular domain was put into a better framework. Firstly the basic score was reported as a log likelihood ratio (LLR) rather than NLL, considering the ratio to an alternative, null, model. Secondly it was assumed that the distribution of random scores from the Viterbi log likelihood score followed an Extreme Value Distribution (EVD). This assumption is based on the work on the statistics of pairwise sequence alignments in which the Extreme Value Distribution has been shown theoretically for ungapped locally alignments and shown empirically to also fit well for gapped alignments [59, 2]. The fitting to an EVD is done by a separate calibration step specific for each profile HMM. The derived expectation values (e-values) for the number of random sequences expected to get a particular score or higher work well in practice, allowing a single cutoff to be applied across all profile HMMs automatically.

A number of other supporting papers to the basic profile hidden Markov modeling technique have been published. These include: derivations of sensible priors using Dirchlet mixtures [75] for the amino acid distributions in the match and insert states, justification of the previous profile techniques in terms of large prior information [4], incorporation of more motif like training techniques which drops the number of parameters to be trained in the EM process [40] and novel sequence weighting mechanisms to derive the best model for a particular family [32]. In addition, the success of profile HMMs have provided a more detailed study into their efficacy in entirely automated model training [66] and a number of innovations in the implementation of the algorithms [39]. Finally a number of databases of profile HMMs have been developed, such as Pfam [7], SMART [74] and Prosite Profiles [43] and TIGRFAM. Chapter 5 details my contribution to the Pfam project.

Protein profile HMMs can probably be seen as the biggest success in the deployment of PFSMs in bioinformatics.

1.4.3 PFSM interpretations of pairwise sequence alignment

Pairwise sequence alignment was one of the first applications in sequence analysis, developed some 30 years ago. A number of researchers have used probabilistic models to interpret pairwise sequence alignment with considerable success. Martin Bishop and Elizabeth Thompson provided one of the earliest probabilistic interpretations [11]. Their motivation was to place the alignment method characterised by Needleman and Wunsch into a formal model. Although the paper does not mention Finite State Machines as such in it, their method for evaluating the likelihood of divergence between two sequences is easily recognisable as the sum over all paths for a PFSM. They show that for tRNA sequences this method gives sensible and interpretable results.

A very similar approach was presented almost a decade later by Philip Bucher and Kay Hoffmann [16] in a paper which was more focused on providing a sensitive search method using pairwise alignment. Again they recast a standard algorithm, this time the local alignment method of Smith Waterman, into a probabilistic formalism. One of the main differences however is that they did not provide an explicit probabilistic parameterisation of the method. Instead they reinterpreted the Smith-Waterman scoring scheme in terms of a probabilistic framework: their recursions have a heavy use of terms like $z^{S_{ij}}$, where z is the base of the log which is implicit in the scoring matrix and S_{ij} is the score at position i in one sequence and j in another sequence. In other respects the recursions are very similar to a sum over all paths type calculation. The power terms are equivalent to probabilities in a more standard framework, but the advantage of this work is that they can use arbitrary gap penalty settings and incorporate local alignment behavior and still provide a sum over all paths. The success of their method was shown with a number of examples. A better judgment of their success came from an independent assessment of a number of methods from a separate group, [1] in which their method was a clear leader.

Jun Zhu and colleagues provided one of the most complete interpretations of pairwise alignment in a Bayesian framework, called the "Bayes Aligner" [88, 89]. Their intention was to remove the nuisance variables of specifying the parameters for the substitution matrix and the gap penalties. Both of these cases are hard to remove as one cannot find easy integrals which would provide analytical answers for summing over an entire distribution of parameters. For the substitution matrix they simply summed over a series of different matrices: they suggested taking matrices from a well defined series, such as the PAM matrices. For gaps they took a non standard approach. Rather than specifying gap parameters they considered all possible alignments with up to k gaps, (k 20 typically). With careful manipulations of the equations of the definitions of the joint probability of seeing two sequences over all possible parameters they provided methods to sum over all given matrices and over all given number of gaps up to some maximum. These methods are very similar to the standard sum over all paths recursions. The authors provide a number of different posterior distributions, in particular the posterior distribution of the matched residues, over all gap numbers and all gap matrices make for interesting and visually appealing pictures. One consequence of looking at the posterior distribution of probabilities for matched residues is that they show there are a number of alternative paths between well conserved blocks. For example, the authors show an alignment for the two GTPases in which two clear well-aligned blocks are apparent, but there are a number of alternative routes between them. The posterior distribution over the PAM matrix choice is also interesting, with a bimodal distribution for this example of either 80 or 140 PAM distances, suggesting that evolution is not a homogeneous process across a sequence.

The difference in notation and change towards finding alignments of a fixed number of blocks rather than alignments with specific gap parameters make understanding the Bayes Aligner as a standard PFSM challenging. By examining the recursions carefully, it is clear that the Bayes Aligner has a PFSM similar to that shown in figure 1.8. This type of machine has been previously published as the "generalised gap penalty" model, though not with probabilistic parameterisation [3]. A benefit of this architecture is that gaps can contain unaligned but otherwise "matched" residues, representing regions where residues occurs in both sequences but they are

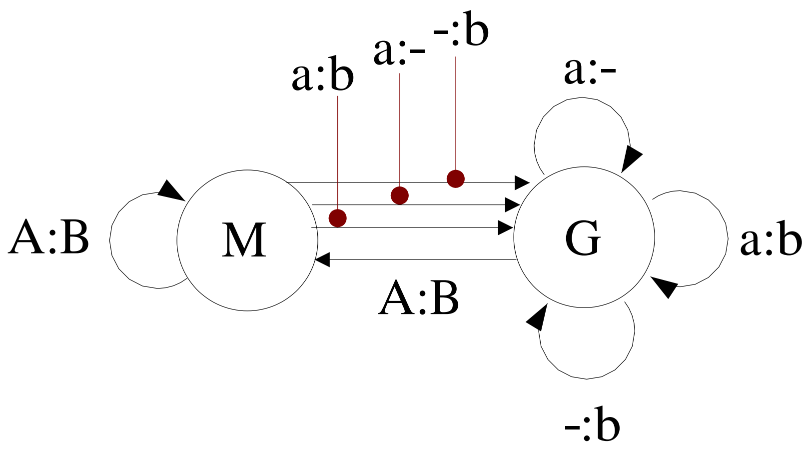


Figure 1.8: The two state gap model common to both the "generalised gap penalty" method and the Bayes aligner. The two states are called Match (M) or Gap (G). Transitions leading to the match state emit two aligned residues, indicated by the A:B pairs. Transitions leading to the Gap state emit one of three different alignment pairs; gapped A sequence, a:-, gapped B sequence, -:b and dual emission of both A and B, but not aligned, a:b.

not alignable. Both the "Bayes aligner" and the "generalised gap penalty" papers emphasise that this is possibly a better model of protein evolution. The use of a fixed number of gaps rather than a gap penalty is harder to interpret in a standard PFSM framework. However, they use priors over the gap numbers which are not equiprobable for each number of gaps but rather take into account the complexity of the number of different possible alignments for each gap number (this is best described in the 1997 paper [88]). This prior can be thought of as a prior over gap parameters in a more standard PFSM, and the recursions provided to sum over all possible gaps become almost identical to approximating the integral over gap parameters by sampling at a number of points in the gap parameters distribution. It is reassuring to find a mapping from the techniques used in the "Bayes Aligner" to more standard PFSM techniques.

Ian Holmes and Richard Durbin published a paper that investigated alignment accuracy [44]. They built up a framework to investigate alignment accuracy around a probabilistic model of sequence evolution. Given this framework, they showed that the natural choice of parameters for PFSM to align the sequences, being the probabilities from the model which generated the sequences, were good parameter choices. They also derived an analytical approximation for the expected accuracy of alignment for a set of parameters, emphasising that no alignment process can be expected to be perfectly accurate. Finally they developed a novel algorithm to maximise the accuracy of an alignment, (as opposed to maximising the likelihood of an alignment). This maximal accuracy alignment is a type of posterior decoding using the posterior distribution of state labels of the two sequences. The algorithm has since been applied to profile-HMMs to good effect.

1.4.4 Probabilistic models of RNA

RNA sequence is commonly used as a biological entity in its own right, as well as being used as an intermediate in the processing of a gene. RNA sequences which are biologically active form specific three dimensional structures which provide the function of the RNA sequence. The process of the one dimensional sequence of RNA folding into the three dimensional structure is dominated by base pairing interactions between the RNA bases, and the majority of those are found in stem-loop structures in which a series of consecutive bases pair up with another consecutive region of base pairs, producing a free single stranded region between the two base paired regions.

Two groups published papers on applying probabilistic models to RNA sequences at around the same time. Sean Eddy and Richard Durbin developed a probabilistic model for these RNA stem loop structures [30]. This model was based around the idea of parsing the RNA stem loop structures into tree which represented the stem loop pairings. These trees could model stem loops with bulges and single stranded RNA, but could only model nested stem loop structures. RNA pseudoknots, in which one stem loop is interposed between another in a non nesting manner could not be modeled. This tree structure was then used as a framework in which a number of states are placed, in particular states which emitted two base paired letters, states which emitted single stranded RNA letters, states which emitted additional letters relative to the consensus (either base paired or not) and states which did not emit letters to model absence of a conserved position. Having built up this model they then detailed algorithms which given a model and a sequence could find the most likely path through the model, being the equivalent to the Viterbi algorithm and the equivalent the Baum-Welch expectation maximisation technique for estimating model paramters.

In the paper their describe both the methods and the application to a number of problems: tRNA detection and the alignment of the structural RNA U100 sequences. They show that there are considerable benefits in providing a probabilistic model of RNA sequences, both in the detection of "atypical" examples of RNA molecules in which the stem loop pairing was well conserved although the primary sequence was not and the deduction of the stem loop patterns from the RNA sequences directly. In the paper they also discuss the fact that this covariance model represents a type of stochastic context free grammar (PFSMs can be be considered a type of stochastic regular grammar) which places this work in the context of language theoretical models.

The other group, Sakakibara and colleagues published a very similar paper in which the correspondance to Stochastic Context Free Grammars was made more clear [72]. Again they used tRNA sequences to illustrate the power of their method, showing that with remarkably few sequences to train the model they are able to easily distinguish tRNA sequences from background.

Interestingly Eleanor Rivas and Sean Eddy extended this work recently to encompass certain types of pseudoknots [70]. In their paper they use a modified type of Feynman diagram to enumerate the different possible combinations during the dynamic programming recursions. Although the algorithm is parameterised on the basis of experimental energies to find the minimum energy fold, they indicate that a probabilistic parameterisation is possible. One intriguing feature of this work is that is was commonly thought that RNA pseudoknot prediction required a higher grammar than a context free grammar and hence NP complete (non deterministic polynomial; e.g. only solvable by heuristic or brute force approaches). The solution provided in this paper indicates that one can solve certain types of higher grammars in polynomial time.

1.4.5 Genome Mapping

An interesting use of a hidden Markov model in bioinformatics was in Genome Mapping. Donna Slonim and colleagues developed a hidden Markov model which represented the position of markers on a genome sequence [76]. The attraction of using a hidden Markov model in this case was that there was a natural representation of the uncertainty of the observations, modeling the possibility for errors to occur in the laboratory work. Using the HMM they could derive a likelihood for a particular ordering of markers along the genome consistent with known radiation hybrid data. They provided an efficient search routine to try to find the maximum likelihood set of markers which involves a generation of a sparse map of only a few, reliable, markers followed by greedy incorporation of additional markers.

1.4.6 Gene Prediction Methods

Representing gene prediction methods as PFSMs started in bacterial genomes, with a number of papers describing hidden Markov models to find open reading frames in bacteria [52, 12]. Anders Krogh and colleagues designed a PFSM similar, though far more complex to the one shown in figure 1.7 which finds protein open reading frames in bacterial DNA.

PFSMs for eukaryotic gene finding were a clear application. David Kulp and colleagues developed a "generalised" hidden Markov model, called Genie [53] . The generalised aspect of the HMM is that they allow transitions between states to be any variable length: this additional freedom allows Genie to accurately model exon length, which has a distribution of lengths that is not well fitted by a geometric decay. In addition they can use any type of "sensor" component which will generate a probability of seeing some observed sequence. The attraction is that they can then use sensors such as neural networks for splice sites. One of the problems they encountered was in how to assess the alternative model for this sensor to provide a likelihood ratio. They note that finding the alternative model for discriminative sensors such as neural networks is difficult, and suggest a way of estimating it by deducing the implicit alternative model from the training criteria of the network. Interestingly, Anders Krogh has developed a framework in which the training of neural networks embedded inside HMMs can be achieved, using so called *hidden neural networks* (HNNs) [69]. In this case the training of the HNNs provides the way to integrate the score coming from the neural network with the generative HMM type model.

Chris Burge developed a fully featured, integrated HMM for gene finding called Genscan [17]. Genscan provides a complete model of genomic DNA, including forward and backward strands and multiple genes. The HMM also contains the ability to model explicit length distributions, which he describes as a semi-Markov model, but does not include the more general "sensor" formalism used in Genie. Genscan performed significantly better than the next best program when released (sensitivity of 0.93 and specificity of 0.93 vs the next best being 0.77 and 0.88 on his test set), and remains one of the best gene prediction programs currently.

Anders Krogh developed a HMM gene prediction model which has a number of novel features [51]. This introduced a concept of optmising the *labeling* of a sequence rather than the total likelihood in using a PFSM, which he calls a Class HMM (CHMM). The gene model has a number of states to model exon regions: when summed over all paths the expected distribution of the total time spend in the states nicely humped, as expected. In contrast a Viterbi path provides only an exponential decay. To deduce a gene prediction from the gene model, he did not use a Viterbi decoding but a method which attempts to find the most probable labels which are still consistent with the model, called 1-best decoding. As the parameters to provide this labeling are not the same as the maximum likelihood, he provides a different training scheme for the parameters, called conditional maximum likelihood. He shows that by using conditional maximum likelihood training and the 1-best decoding method he can improve in particular the accuracy of the gene prediction model without changing the sensitivity (accuracy at the base level goes from 78% to 94% with coverage remaining around 80% on his test set).

1.4.7 Other techniques

There are a large number of other techniques which use PFSMs in bioinformatics. Building and interpreting phylogenetic trees have used probabilistic methods for a long time, as there are not many other ways of attacking the problem. Jeff Thorne, Nick Goldman, David Jones and colleagues have published a number of papers on combining PFSMs with phylogenetic trees [84, 36, 55]. These papers show that there is an increase in the likelihood of the tree when a PFSM of the sequence is included in the model. Many of the processes in large scale sequencing can benefit from HMM techniques, such as the decoding of trace data as discrete bases [23].

The following chapters describe my research into PFSMs in bioinformatics.

Chapter 2

Dynamite

2.1 Introduction

Dynamic programming is a ubiquitous algorithm in computer science which has been applied to many different fields. In its broadest sense, dynamic programming is the recursive decomposition of an optimisation problem into smaller sub problems until known initial conditions are reached. To compute the solution of a problem, the known conditions are used to initialise the first sub problem, and then the recursions which provide the solution of the next larger sub problem from the solution of the previous, smaller sub problem allow the propagation of this information until the complete solution is known.

Dynamic programming as a technique allows the solution of such problems as finding the shortest path between two nodes in a weighted graph [25], and other variations of this algorithm. This in turn provides a basis for the solution of a number of well known problems, such as finding the minimal string edit cost, the parsing of regular grammars, or, alternatively stated, finding the path through a Finite State Machines consistent with a given emitted sequence. These Finite State Machines include probabilistic Finite State Machines (PFSMs) also known as hidden Markov models, where the two main algorithms that occur when using HMMs, the Viterbi algorithm and the forward-backward algorithm, are both types dynamic programming.

Dynamic programming methods are used in a wide variety of applied fields, from passive sonar detection, through oil exploration techniques, speech recognition and biological sequence analysis [48].

2.1.1 The use of dynamic programming in bioinformatics

The application of dynamic programming to biological sequence was an early development in the field [64] and continues to be a key technique in the arsenal of sequence analysis, including common algorithms such as finding the optimial local alignment of two sequences [77, 37]. Even as more complex problems were tackled, such as representing family information and recognising structural folds, new methods were commonly simple variations of the basic dynamic programming algorithm developed previously for pairwise comparisons of sequences.

The introduction of probabilistic finite state machines in bioinformatics was inspired by the common use of dynamic programming algorithms to solve sequence analysis problems. A number of different methods which use PFSMs have been already developed, as described in the previous chapter. The next section will outline how PFSMs use dynamic programming.

2.2 PFSMs and dynamic programming

Dynamic programming techniques are used to solve two of the crucial problems found in PFSMs: firstly finding the optimal parse through the PFSM given the data, called the *Viterbi* algorithm, and secondly finding the likelihood that observed data was generated by the PFSM, summing over all possible paths (often called the Forwards score). Both these methods essentially rely on the same feature, which is the Markov rules of the PFSM: the probability of moving to the next state is only dependent on the current state of the machine, and no other properties of how the machine got to this state.

2.2.1 Finding the maximum likelihood path

The Viterbi algorithm finds the most likely path through a probabilistic finite state machine given observed data. Assume that P_{ki} is the total probability of the best path which passes through state k and ends at data point i (see figure 2.1 for a pictorial representation). Consider the S possible states which could have been immediate predecessors of the k state, which we will call *sources* of this state. Each source s has to have an offset, o_s in the data dimension (o_s can be zero but not negative). Each ks pair describes a single transition in the PFSM, which will

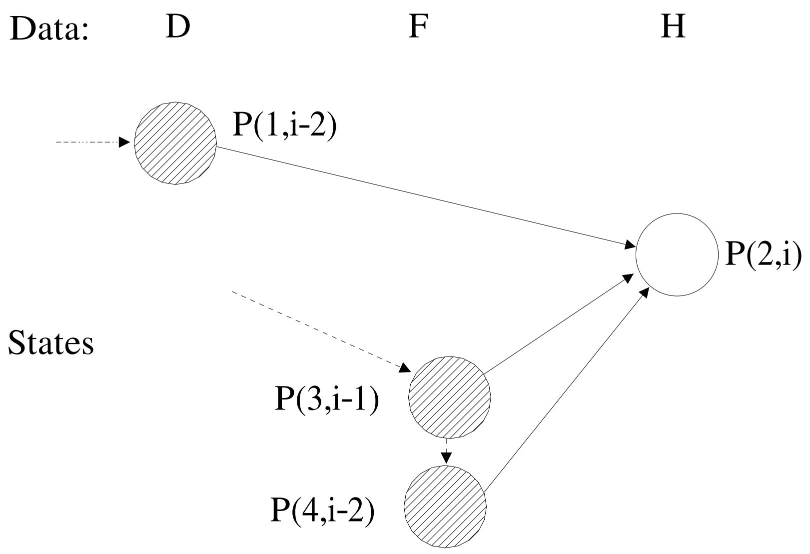


Figure 2.1: A diagram illustrating the basic Viterbi recursion. The process is calculating the best path that goes through state 2, data position i. Three possible states (shaded) could be the sources of the best path passing through this position: the best one is chosen by comparing the total probability of the best path to each source times the probability of the transition to state 2. The actual best path to each of these sources, indicated by the dashed lines is not need to calculate the best path through state 2. Note that transitions from state 1 has an offset of two, emitting two symbols, whereas the transitions from the other two states have an offset of 1, emitting a single observation.

have a probability dependent both on the nature of the transition and on the data observations by the transition. From the Markov rules, the following recursion holds:

$$P_{ki} = \max_{\text{sources } s} (P_{s(i-o_s)} T_{sk}(d_{i-o_s}, \ldots, d_i)) \tag{2.1}$$

If o_s is zero we adopt the convention that $(d_{i-o_s}, \ldots, d_i)$ is the null string. The Markov rules are being used here to discard all the possible path information in how the best path reached state s at the data position $i - o_s$, and only be interested in its total probability $P_{s(i-o_s)}$.

Given this recursion rule, and the starting and ending conditions, the problem of finding the best path can be solved using dynamic programming. It starts at the first state with the first data point and proceeds to apply equation 2.1 iteratively to for all states at each data position: the process is terminated when it has exhausted the data and has reached the ending state. At this point the probability of the most likely path can be simply read off.

The actual state labels which generated the best path can be found in a number of ways once the probability of the best path is known. One can keep a note for each ki position about which source was used. This is conceptually simple but requires considerable additional bookkeeping of the "backpointers" for each ki position. A more efficient method is to work backwards up the path using the recursion rules to reconstruct the preceeding ki position from the correct position of the best path, starting from the end point and working. This only requires storing the best probability for each ki position. However this storage requirement is prohibitive for large problems: in such cases a divide-and-conquors method which has a longer running time but considerably less memory requirements can be used. These methods are described in more detail in a later section (2.4).

The discussion so far has been for a single sequence being compared to a PFSM. When an alignment style PFSM is used, two sequences are being aligned on the basis of the PFSM. The Viterbi algorithm is identical to that described previously except that there are two data dimensions, i and j which form a matrix of data points. The states form an additional third dimension, which is generally far smaller than the sequences of observations.

2.2.2 Finding the total probability of observations

Often one wants to know the total probability of seeing some observations regardless of which state path was taken for a particular PFSM. The calculation to find the total probability over all paths, often called the "forward" score, is closely related to the Viterbi algorithm. All it requires in effect is to replace the Maximum in equation 2.1 with a sum. Consider AP_{ki}, the total probability over all paths to a particular state k and data observation i. Again a recursion rule can be set up that provides AP_{ki} in terms of its predecessors

$$AP_{ki} = \sum_{\text{sources } s} (AP_{s(i-o_s)} T_{sk}(d(i-o_s), \ldots, d_i)) \tag{2.2}$$

Again this rule can be applied from the starting conditions to the ending conditions using dynamic programming. The final result however is the total probability over all paths of observing the data given the model, rather than the probability of the most likely way of producing it.

2.3 Dynamite

I realised early on in my research that I was likely to be developing a number of dynamic programming methods in this field. The implementation of some of these methods are time consuming to program and debug. I wanted some type of toolkit to help develop these methods, allowing me to concentrate on the actual biological problems and not the implementation.

Usually programming toolkits are based around libraries of resuable code. However, in this case I knew that that a library based solution would not provide efficient execution of the method, as the part of the method that I wanted to be flexible, the definition of the dynamic programming recursion, was the part in which the vast majority of the CPU time is spent. If the compiler is unable to optimise this part of the method the execution time can suffer by an order of magnitude or more.

For these reasons I choose to build a compiler which would work off a high level description of the dynamic programming problem. The compiler would produce C code for the specific dynamic programming problem. This C code would need to be linked to a library of supporting routines and a piece of driving code where the programmer actually called the generated dynamic programming routine. This separation of compiler, libraries and driving code is common in other computer science applications, such as the yacc compiler system for programming languages. The entire ensemble of language, compiler and libraries I called *Dynamite* [9].

By using a compiled language the generated code can take advantage of specific features of the hardware it is using: for example, for symmetric multiprocessor machines which use a threading model, multi-threaded code can be generated, or for distributed memory multiprocessor machines, message passing code can be generated. The separation of the specification of the program from the generation of efficient code allows people who are only interested in the design of dynamic programming algorithms to take advantage of large compute hardware without changing any code. From the other perspective of the computer hardware designer, people who are only interested in the efficient execution of the algorithm on specialised hardware can work on a more general form of the problem, being confident that their work can be applied to many different methods with a minimum of effort.

2.3.1 The Dynamite language

In looking at how Dynamite is actually structured, it is probably worth reminding the reader of the basics of dynamic programming as it is actually implemented in a programming language for this class of problems (Figure 2.2). One has two objects (often sequences), called the query and the target in Dynamite, which provide the two axes of a matrix of numbers. This matrix is usually divided up into cells, each cell containing a collection of numbers that represent the best score (or the summed score) of the problem to the ith position in an object compared to the jth position of the other object, ending in that state.

To calculate a cell one needs to use the

- cells from previous ijth positions (only some cells might be required).
- data elements in the query and target
- additional resources (such as a protein comparison scoring matrix)

In cases where we have to calculate the path, the most efficient approach is to do a traceback through the matrix of numbers, deducing how the best score was made. When memory limitations prevent allocating and using the entire matrix in memory, a linear space divide and conquor version of the algorithm must be employed.

2.3.2 formal definition of a dynamic programming recursion

Dynamite is a language like yacc or lex (the UNIX tools to automate compiler generation and regular expression parsing), which generates C code from a definition of the problem. The Dynamite language must provide the following features

- A definition of the dynamic programming recursions suitable for the state machine one is considering.
- The ability to supply the data structures to provide information to calculate the recursion.

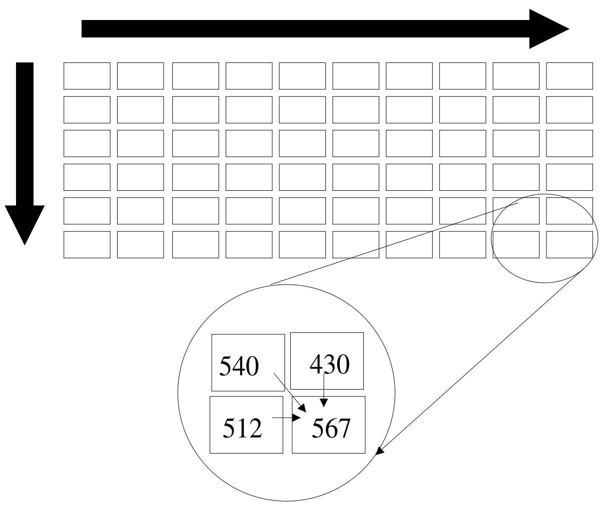


Figure 2.2: A diagram explaining dynamic programming as it is usually implemented for sequence alignment. A matrix of cells is laid out, one axis for one sequence, the other axis for another sequence. The cell represents the best score in each of the states for a particular prefix of sequence A compared to a particular prefix of sequence B (for cases where there is more than one state, each cell stores more than one number). Cells are calculated via recursion rules which indicate how the score for the extension of the alignment to the next cell is modified, usually on the basis of properties of the sequence at that point.

- The interface between the definition of the state machine and the data structures to indicate where in the state machine which calculations occur.

An example dynamite definition, in fact for the Smith-Waterman algorithm, is written out below

```
%{
#include "dyna.h"
%}

matrix ProteinSW
query  type="PROTEIN"  name="query"
target type="PROTEIN"  name="target"
resource type="COMPMAT" name="comp"
resource type="int"    name="gap"
resource type="int"    name="ext"
state MATCH offi="1" offj="1"
        calc="AAMATCH(comp,AMINOACID(query,i),AMINOACID(target,j))"
        source MATCH
                calc="0"
                endsource
        source INSERT
                calc="0"
                endsource
        source DELETE
                calc="0"
                endsource
        source START
                calc="0"
                endsource
        query_label  SEQUENCE
        target_label SEQUENCE
endstate
state INSERT offi="0" offj="1"
        source MATCH
                calc="gap"
                endsource
        source INSERT
                calc="ext"
                endsource
```

```
        query_label  INSERT
        target_label SEQUENCE
endstate
state DELETE offi="1" offj="0"
        source MATCH
                calc="gap"
                endsource
        source DELETE
                calc="ext"
                endsource
        query_label  SEQUENCE
        target_label INSERT
endstate
state START !special !start
        query_label  START
        target_label START
endstate
state END !special !end
        source MATCH
                calc="0"
                endsource
        query_label  END
        target_label END
endstate
endmatrix
```

The `matrix` to `endmatrix` lines provides the Dynamite definition. The first 5 lines indicate the data structures which are used in calculating the recursion. The first two, *query* and *target* are the two objects which define the two axes of the matrix. The other, *resource*, lines indicate additional data structures needed to do the calculation - in this case, a protein comparison matrix and the gap penalties.

The rest of the text is used to define the finite state machine and its interface to the data structures. Each state is defined by the `state` to `endstate` lines, and contains within it the definition of the transitions which end at this state in each `source` to `endsource` block. Each source block describes a single transition from a particular state to the state it is within.

The interface to the data structures is provided by the calc lines, which can be found in one of two places - in a source block which provides the definition of the calculation for this transition and optionally in the state block, which indicates a calculation that is the same for all transitions which end on this state and is added to all transitions regardless of where they originated.

The calc lines are built from a subset of C which allows the usual C arithmetic (`+,-,/,*`), memory dereferencing (array `[]` deferencing, `*` as a pointer reference), structure deferencing (the `.` and $->$ operators) and function calls. This is implemented using a small yacc grammar which is fully parsed by the dynamite compiler.

2.4 Implementations provided by Dynamite

Dynamite produces a number of implementations which provide different calculations based on the same Finite State Machine definition. The first ones are those which provide the Viterbi decoding of the alignment process in either quadratic (section 2.4.1) or linear (section 2.4.4) memory. There is also a convenience function which decides whether to call the large or small memory model for a particular query and target object, returning back an alignment, which shields the user from having to switch between the two implementations.

Dynamite also produces some functions to provide the score of two objects either by the Viterbi path (section 2.4.2) or the Forwards algorithm, which calculates the sum over all paths (section 2.4.3). These implementations work faster and in linear memory, as they do not need to calculate the alignment.

A common use for these types of comparisons is a database search. When dynamite is made aware of how to loop over a database of objects (see section 2.4.8) a number of different database searching modes can be used. The serial search (section 2.4.5) provides a standard database search. As the database search can be done in any order, the problem is very amenable to coarse grain parallelization, in which each pair of comparisons required for the database search is packaged off to a separate execution stream. Dynamite can generate a multi-threaded implementation, suitable for symmetric multi processor machines (section 2.4.6) for database searching.

Finally as Dynamite is a true compiler, not merely a large macro system, in the sense that it completely parses the definition of the state machine into a internal representation that fully describes the machine, very complicated target machine architectures can be considered. One example of this is the generation of a system which is compatible with specialised hardware that runs on Digital Signalling Processing chips (section 2.4.7).

2.4.1 Viterbi quadratic memory alignment

The easiest implementation of dynamic programming is to calculate an explicit matrix of numbers recursively, as in figure 2.2. The final score can be read off from the appropriate END state. The alignment which generated the high score can then be deduced by recursively inferring which previous state the high scoring path must have originated from (this is commonly called the trace back routine).

2.4.2 Viterbi score only, linear memory

When the alignment is not required the score of the Viterbi path can be easily calculated in linear space (this is a natural consequence of the fact that only a small portion of the matrix is required to calculate the next portion of the matrix). The only issue with this implementation is that as this will be used in the database searching method, care should be taken to minimise its execution time. For the Dynamite compiler, this means taking care of two issues

- The C generated by Dynamite must be suitable for optimisation by the C compiler. Generally this means providing a large inner loop of execution statements some of which can be done concurrently.
- The memory layout should be such that concurrent retrievals of information are close in memory. Dynamite ensures that the matrix memory is laid out optimally.

.

2.4.3 Forwards score only, linear memory

The forwards algorithm, which sums the probability over all paths, can also be run as an implementation which provides a score. Its implementation is almost identical to the Viterbi method, but instead of a max at each position a sum in probability space is provided. The only minor twist to this is that the matrix numbers are in a log space representation: the sum function therefore has to move the numbers to probability space before adding them. This can be done efficiently by precomputing a table of mappings in log space for addition of different probability values.

2.4.4 Recursive linear memory alignment

Storing the explicit matrix of numbers rapidly becomes a resource problem since the memory requirment grows as the product of the length of the sequences being compared. As DNA sequences can easily be over 100,000 positions long, the explicit form can quite easily be impossible to calculate on standard workstations.

A solution for this problem has been known for some time, which involves a recursive algorithm that calculates the place where highest scoring path crosses the half way point in the length of one of the sequences (Figure 2.3) [62]. This can be calculated in linear memory (proportional to one of the sequences). Once the half point is known the problem can be split into two problems on each side of the half point. These sub problems themselves are also alignment problems, which can be solved by the application of the same method, i.e. finding the half point so as to partition each problem into two smaller sub problems. The recursion is terminated when the remaining matrix is small enough to fit into memory explicitly.

This implementation is notoriously hard to program. The first problem is that the recursive alignment routine can only be called once the start and end points of the alignment are known. Secondly, when the alignment is local or, even worse, the alignment has complex boundary conditions, such as a loop, very annoying book-keeping has to employed to ensure that the correct alignment start/end points are used. Many Dynamite users were attracted to the toolkit due to the implementation of the linear space alignment which they did not feel confident to program themselves for non trivial dynamic programming situations.

Dynamite solves the linear space alignment in a reasonably standard way. The first stage is to break a more complex alignment problem which potentially involves loops into a series of simple alignment problems which has no loop (see the section on special states, 2.5.1, to see how these come about). Then these alignments are solved using a recursive function. Both the splitting of the complex alignment to simple, non looping alignments and the recursive routine use a concept called the

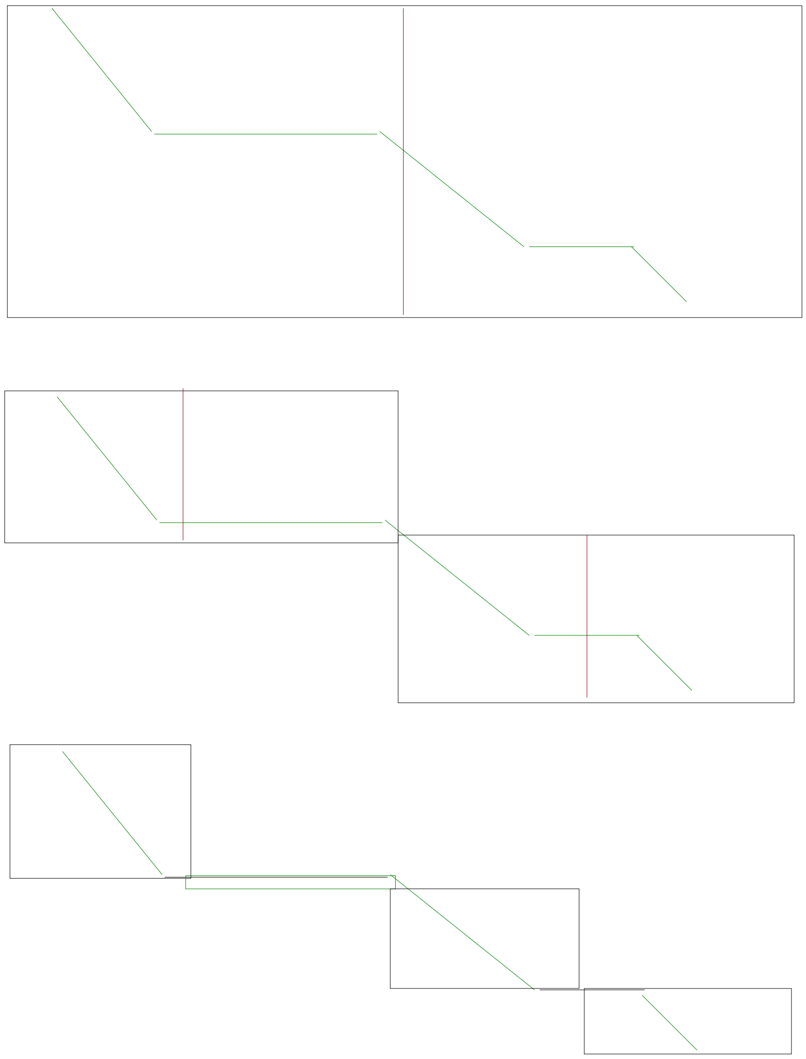


Figure 2.3: The first 3 rounds of the divide and conquer method, illustrated pictorially. The green line represents the best alignment, and the black boxes the matrix calculated at each iteration. The red line is the midpoint at which the position of the best path is stored for each state in each cell, and then propagated on to the end of the calculation. Each iteration produces two sub problems which can then be solved using the same method, until the sub problems are small enough to solve using conventional quadratic memory

shadow matrix. A shadow matrix is information which is attached to the score of a particular matrix number, and is propagated on with the score. By using a shadow matrix the start point of the match can be propagated to the end of the alignment in the first stage of the alignment process. In the full recursive alignment, the shadow matrix is used to store the position of each path at the half-way point. At the end of the alignment process, the position of the half way point can be read off from the end point's shadow matrix.

This use of a shadow matrix is more expensive than the better method of running the dynamic programming recursions in two directions, one forward to the halfway line and one backwards from the end point to the halfway line. The shadow matrix method has to make the same number of calculations as this forward/backward sweeps, but incurs the additional expense of propagating information in the shadow matrix. The benefit is that is a far easier implementation to code as it does not involve inverting the dynamic programming recursions.

2.4.5 Serial database search

The ability to generate a database search of one database of objects vs another database of objects is a common use of the dynamic programming algorithms. In these cases generally only the score of the match is required, and so one can use the Viterbi (2.4.2) or Forwards (2.4.3) score methods. Although the programming of serially looping through a database of objects, applying one of the scoring functions seems simple enough, robust routines require a little more thought:

- The database searching routines must not leak memory as they will be called multiple times during the search.
- Efficient database searching requires that a minimum of objects remain in memory during the database search, and that objects can be retrieved later on for processing of the alignments.
- The database searching routines should be able to either provide a Viterbi or a forwards score.
- Error reporting of problems during the database search should provide an indication of where in the database search they occured, and where possible still provide partial results.

Coupled with considerations of minimising the execution time of the database search, writing a good serial database searching routine is not that trivial. Dynamite provides an immediate, robust and fully featured database searching routine for this reason.

2.4.6 pthreads Database search

Database searches are often the time limiting factor for research, taking sometimes around a week to run for some complex models. The advent of multi-processing boxes provides a way to reduce the actual amount of time taken for a database search to complete by using multiple processors on one particular search. One common model for programming multiple processor boxes has been pthreads a POSIX standard in which each execution stream is represented as a different thread, but each thread has access to the same memory.

The database search is an *embarassingly parallel* problem, meaning that the problem can both be easily broken into sub problems and that these sub problems require minimal communication between them. This makes the conceptual writing of a pthreads implementation simple: each thread does a comparison of one object to another object separately and then places the score in a common results structure.

Even though the conceptual programming is quite simple, there are many barriers to writing pthreaded code. Firstly the pthread syntax, which involves calling specialised library functions to create threads and other structures which allow coordination between threads has to be understood. Secondly, each thread must not share memory with other threads which would cause program failure or worse incorrect results. Finally the threads must be coordinated to allow sequential access to common resources, such as the database streams.

Dynamite provides an efficient, well designed pthreads implementation.

2.4.7 OneModel BioXL/G port

Database searching by dynamic programming is so compute intensive that a number of companies have designed specialised hardware to accelerate it. This hardware is generally hard coded with a small number of algorithms: the lack of flexibility in the algorithm means that every development of a new algorithm must be implemented by a small team of people who understand the hardware.

The creation of Dynamite as a programming language that represents dynamic programming in theory allows the possibility of Dynamite generating code suitable for the specialised hardware. One specialised hardware company (Compugen) has developed hardware flexible enough to run many different versions of the Viterbi score database search method. Along with this hardware they provide an API to "program" the machine for a new algorithm, called *OneModel*. This system is in some ways similar to Dynamite, but importantly, reads information from simple arrays of integers, and not directly from C structures as the data must be moved to the specialised hardware and no general C compiler is available for the hardware.

The fact that Dynamite represents the entire definition, including the interface to the user defined calculation methods (the calc lines in the Dynamite definition) allows Dynamite to generate code which can use the OneModel API. To do this, Dynamite has to generate C code which queries the user defined C code to extract the information required for the OneModel API. For example, the calc line written below might be found in a protein profile HMM matching a protein sequence.

```
source MATCH
  calc="position_specific_gap(query,i) +
        insert_emission(query,i,AMINOACID(target,j))"
  endsource
```

would generate the following code (written as pseudo code)

```
foreach i position in the query {
   offset = 0;
   model[i][offset] = position_specific_gap(query,i)
   offset++
   foreach A over amino acids (26) {
     model[i][offset+A] = insert_emission(query,i,A)
   }
   offset = offset + 26
   // the next calc line stored
}
```

Notice that the calc line has been split logically into two separate expressions. The compiler knows that the `position_specific_gap` expression is invariant with respect to changes in the target information. However the `insert_emission( query, i, AMINOACID(target,j))` expression uses information from the target information. The compiler then has to loop over every possible input of that information. In this case the compiler parses `AMINOACID(target,j)` as having 0-26 as its possible values and generates a loop to iterate over those values storing the information in the model line. The generation of this code requires that the Dynamite compiler can manipulate these calc lines in a detailed manner.

The compiler then has to generate the necessary definition for the specialised hardware to indicate that for this particular transition in the state machine it needs to use the numbers in `model[query-position][1+aminoacid in target]`.

2.4.8 Software engineering details of Dynamite

Dynamite is a toolkit which provides considerable help to the algorithm designer for implementing new algorithms. The aim is not only that this toolkit is a prototyping environment but also that it provides functions for the final implementation. For this to be achievable the code generated and used by Dynamite must be sensible.

The Dynamite generated code passes strict ANSI requirements, making it portable to any architecture with an ANSI C compiler. It has been tested on a wide variety of UNIX machines and Windows NT. To allow the Dynamite generated code to be used with other libraries, a unique prefix can be appended to each external name, preventing namespace clashes when it is linked with other libraries. Care has been taken to allow new implementations of database searching functions to be seamlessly integrated into existing programs which use dynamite with only the bare minimum of changes.

The Dynamite run-time environment is also built with the same considerations of code correctness and external namespace protection. Parts of the Dynamite run-time environment are hard-coded into the Dynamite compiler, but other parts are made more flexible to allow different libraries to provide, for example, database streaming functionality. This allows other user-defined types to take advantage of the database search functions generated by the Dynamite compiler without having to understand the innards of the compiler.

2.5 Innovations in Dynamite

Dynamite has a number of innovations which extend the paradigm of sequence alignment algorithms represented as finite state machines.

2.5.1 Special states

Special states are best considered in the framework of repetitive hidden Markov models. A quick introduction to using Dynamite for repetitive HMMs is given first, and then an explanation of special states in this model.

When Dynamite is used for a hidden Markov model, it is best suited to those hidden Markov models in which there is a repetitive series of states which are connected in a standard pattern. Such hidden Markov models include the profile HMM architecture suggested by Anders Krogh and colleagues [50]. In speech recognition, these sorts of hidden Markov models are called time-dependent hidden Markov models, and in general, when people consider using large hidden Markov models it is very common to constrain the architecture into a left-to-right, repetitive architecture.

Although the main state definition of Dynamite can easily satisfy these repetitive HMMs, the boundary conditions can be quite intricate. The boundary conditions are sometimes particular parameterisations of the start conditions and end conditions with respect to the position in the repetitive nature of the hidden Markov model. As boundary conditions become more complex, generally people want additional "boundary" states to model aspects of the problem near the boundary conditions. These additional states become yet more involved when looping architectures are employed. In such architectures the repetitive block of states can be reentered multiple times. The boundary conditions now apply to both the entering and leaving the repetitive block at the start and end of the observations, and between each occurrence of the block.

Dynamite handles all boundary conditions, from simple to complex using special states. Special states are states that are only present once in the HMM: they are not repeated. This means that there is a break in symmetry between the query and target dimensions: in Dynamite the special states are only applicable to the query dimension. Many special states can occur, and they can be connected to both standard states and other special states (see section 2.5.3 to see how null cycles are avoided). By using special states, many different complex boundary conditions can be created. Examples are given in the dynamite definitions of a variety of dynamic programming algorithms in Appendix B.

2.5.2 Labels

When designing dynamic programming code for biological applications one might be changing the dynamic programming recursions whilst still attempting to solve the same biological problem. One annoyance of this is that the data structure which represents the result of the dynamic programming parse will be different for different precise recursion definitions: to generate sensible biological results one needs to endlessly change the code to that looks are the data structures emerging from the dynamic programming code.

Dynamite provides a way of embedding the biological interpretation of the dynamic programming result into the dynamite definition file. This is done by defining two text *labels* for each transition (source block in the dynamite file), one label for the query dimension, one label for the target dimension. For convenience you can also define a state default for the labels. Dynamite then generates the code to convert the initial description of an alignment path as a state path through the machine to a set of aligned regions for each dimension, with the appropriate labels.

As the programmer is free to reuse the same label in many positions, radically different finite state machines can generate the same set of labels. By having the downstream code only read the label based alignment, one can change actual dynamic programming code at will and use just one set of postprocessing code for the downstream analysis of the alignment. This feature is heavily used in the GeneWise algorithm, where 7 different architectures were tried whilst designing the algorithm and 5 different architectures are actually distributed in the Wise2 package (see Chapter 3). The fact that very different dynamic programming recursions are calculating the alignment is completely hidden from the bulk of the GeneWise code.

Interestingly the concept of labels being attached to the dynamic programming recursion is more than a programming tool. A recent set of algorithms to cope with the mismatch between the biological interpretation of a model and the model architecture have been developed by Anders Krogh [51]. These algorithms use a concept also called labelling to indicate the biological interpretation of a state, and using this concept provide a number of algorithms to replace the standard training and path finding techniques with ones that are optimised for making the correct label assignment, not the correct state assignment. It is interesting to investigate whether the labels as defined in Dynamite will have a broader use than just programmatic convience but also be involved in the algorithmical aspects of Dynamite in the future.

2.5.3 Compile time error detection by the Dynamite compiler

As the Dynamite language is parsed and loaded into an internal data structure inside the Dynamite compiler a number of errors can be found at this abstract level. These include, of course, syntactic errors when the Dynamite definition is incorrect, but can also include semantic checks after the Dynamite source code has been parsed. These semantic checks ensure the integrity of the language and so catch some errors at compile time, errors which are impossible to catch with a lower level language. The semantic checks include

dangling transitions Ensures that each transition starts and ends on a valid state.

start/End Ensures that there is a single start state and a single end state, and that there is at least one path from start to end.

null cycles Ensures that there are no null cycles by making sure that each transition advances in at least one data dimension

inappropriate indices The programmer is free to use the `i` variable as the index in the query dimension and the `j` variable as the index in the target dimension. The dynamite compiler detects when `i` is being used to index into the target data structure or the `j` into the query data structure. This nearly always indicates an error, but it could conceivably have some use, so only a warning is issued.

type checking of function/macro calls Each function call in the Dynamite compiler is type checked (see below)

The type checking in the dynamite compiler is particularly flexible, and can apply to both function calls and macros as used by the C programming language. This makes the type checking useful, as for performance reasons, many of the ways of accessing the data will be via macros which return integers, and so mistyping errors cannot be detected by the C compiler.

An example of this type checking is given below. Imagine we have declared `CODON_HMM_MATCH(hmm,i,codon)` as taking a HMM as the first argument and a codon as the third argument, `CODON_SEQ(seq,j)` as returning the codon at position j, and `BASE_SEQ(seq,j)` as returning the base at position j. The following calc line would be a valid calculation, and be parsed without error

```
CODON_HMM_MATCH(hmm,i,CODON_SEQ(seq,i))
```

However the next line would cause a mistype error

```
CODON_HMM_MATCH(hmm,i,BASE_SEQ(seq,i))
```

This line generates the following error from the Dynamite compiler

```
In parsing calc line for state [MATCH] source [MATCH]
        Mis-type in argument 3 of CODON_HMM_MATCH:
        wanted [CODON] got [BASE]
Warning Error
```

This is a case where in the C code, the macros which accessed the base or codon information would return identical results: therefore this error can only be picked up by the Dynamite compiler.

Even for a user such as myself, who has a clear knowledge of Dynamite and how to use it, these compile time checks catch many problems early on in the development of an algorithm. A particular benefit are cases where the definition would produce an incorrect answer, not merely a run-time error. These sorts of problems are hard to track down in the logic of the C code.

2.5.4 An optimiser for dynamic programming

As Dynamite is a complete compiler, holding a representation of the dynamic programming code in an internal data structure, there is the possibility of rearranging the data structure to generate code which executes faster. One should be wary of such optimisers: Dynamite is not generating assembly code for a specific class of machines, but rather C code for a generic class of machines. By and large it is better to let the C compilers do the optimisation, as they are both written by experts in optimisation and also have more knowledge about the target machine they are writing for. However Dynamite has more knowledge than the C compiler in how the code is actually used. In particular it knows that certain data structures are read-only from the point of view of the dynamic programming.

I have only implemented one optimisation which exploits the read only nature of parts of the dynamic programming code. I developed a standard common sub-expression analyser which could find identical `calc` lines which are executed more than once in the inner loop. These sub expressions were then analysed as to which loop they remain invariant under, and the actual expression moved outside the loops they remain invariant under (this is called a "loop hoist" in the compiler optimisation field, and is a very basic optimisation). In some cases, the expressions which I promoted included complex data structure dereferencing: the standard C compiler could not validly perform the same optimisation as it does not know that that the data structure is read only for the duration of this function.

This optimisation showed dramatic results when the C compiler optimisation was not used, sometimes showing a two fold speed up. However when the C compiler optimisation was used, the effect dropped to 5% or less speed up. Considering the sophistication of the C compilers, I feel that 5% speed up is still significant.

Currently there are more experienced researchers in compiler optimisation looking at the possibilities of providing better optimisations in the Dynamite compiler.

2.6 Example Dynamite programs

The next three chapters discuss in detail a number of Dynamite generated algorithms. I have used Dynamite for many more algorithms than those detailed in this thesis. Here are three other algorithms which were written in Dynamite that illustrate its flexibility.

2.6.1 Dna Block Aligner, DBA

When aligning a eukaryotic promoter region with the corresponding region from a related species it is clear that standard Smith-Waterman alignments of the two sequences do not reflect the known biology of promoters. Promoters are known to contain a number of small motifs which bind DNA binding proteins, some as basal transcription machinery, some as transcription factors. These motifs can be found at vastly different distances and still function identically. When aligning two homologous promoter or regulatory DNA sequences, for example from mouse and human, we should expect large intervening sequences to occur between regions that are conserved.

There were no available programs which would easily provide this sort of alignment information. Rather than post processing existing algorithms (such as BLAST or SIM1) we investigated designing our own alignment systems to represent evolution in regulatory regions. The architecture we settled on is shown in figure 2.4. We presumed that the two sequences would share motifs that could be interrupted by small gaps, but that between these motifs very large gaps, indistinguishable from background, were likely to be present. Because we were interested in measuring the level of conservation between different sites, we split the blocks into four different types of blocks which matched differing levels of conservation. Thus the alignment process would align the two sequences and also classify the conserved blocks simultaneously.

The parameterisation of the FSM was done by setting parameters by hand for the conservation levels and the two gap probabilities (in blocks and between blocks). The parameters chosen gave good results on the test cases we used. The model was parameterised against a null model which was as if there were no conserved blocks in the two sequences. This null model could be applied directly to the scoring scheme, as there are no hidden variables to calculate, meaning that we could provide a single program that would provide the log-odds ratio of the alignment.

An example output is shown in figure 2.5

2.6.2 Deriving an alignment from a 3D superposition of protein structures

When the 3D structures of two protein sequences are determined by X-ray or NMR techniques it is usually easy to decide visually whether they are related or not (unlike only knowing their linear amino acid sequence, where it is extremely hard, even with computational help). To help understand how two structures are related, a common technique is 3D superposition, such as rigid body rotation and translation to minimise the root mean squared distance of the alpha carbon atoms between the

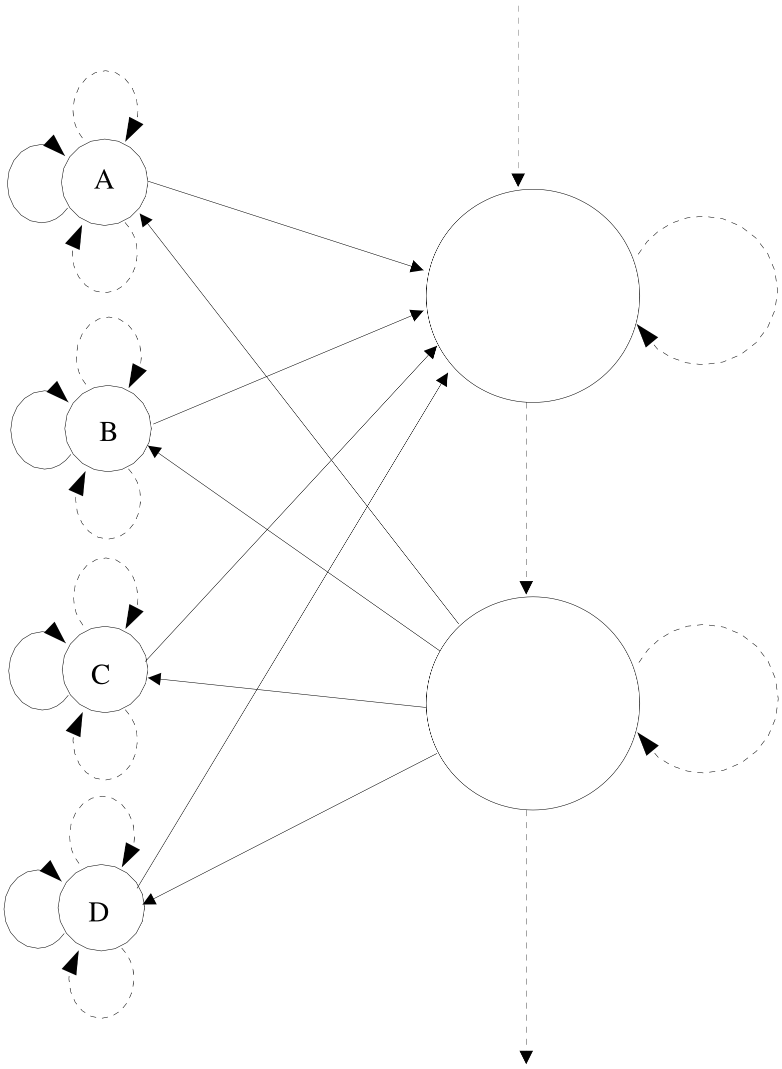


Figure 2.4: A figure illustrating the Dna Block Aligner PFSM. The conserved blocks are on the left hand side of the page: each block is parameterised with a different level of expected DNA conservation, shown as solid transitions. Small gaps are allowed, showed as dashed transitions, one for each sequence. Separating the blocks are two states which represented unaligned DNA sequence. Each state emits only for one sequence. It is possible to pass through the entire PFSM without entering a conserved block.

```
                             score = 72.52
EM:MMENDOBA 110      ACTCCCTCCACTCTTTCCACCATTCACCACATCCCCCCACACACACTCT
                  A  ACTCC T   CTCTTTCC  CATTC   A    CCCC AC C  ACTC
EM:HSKER101 57       ACTCCTTTGCCTCTTTCCG-CATTCCATAACCACCCCAACCCCTACTCC

EM:MMENDOBA 158      ATGGGGACGGAGTTAGGAATAC-CTGGACTCTCACCC
                  A  A  GGGA GG GTT GG ATAC CTGGA T  CA CC
EM:HSKER101 106      AC-GGGAGGGGGTTGGGCATACCCTGGATTTCCATCC


EM:MMENDOBA 359      TTAGAGGGTTAAGCGGATGTGGCTAAGGGTGAGTCATCTAGGAGTAAAC
                  C  TT GAGGGTTAAGCGGATGTGGCTAAGG TGAGTCATCTAGGAGTAAAC
EM:HSKER101 322      TTGGAGGGTTAAGCGGATGTGGCTAAGGCTGAGTCATCTAGGAGTAAAC

EM:MMENDOBA 408      AGGAGCCTTACCTGTAGGAGGGGCCA
                  C  A GAG C T CCT T GGAGG GCCA
EM:HSKER101 371      AAGAG-CCTTCCTTTGGGAGGAGCCA


EM:MMENDOBA 468      GGGGGGGGGGGGGGGGGGGTTAGCAGGTGCACCTGGAAGAAGATGCCAG
                  A  GGG G  GGGGG    G GT A CAGGTGCAC  GG A AA ATGCCAG
EM:HSKER101 402      GGGTGTAGGGGGCCCAGAGTGACCAGGTGCACTAGG-AAAAAATGCCAG

EM:MMENDOBA 516      GAGAGGATCAGAAGGAAATCTTGTTGGAAGCTGCTCTTTTGTAAGCA
                  A  GAGAGG  CAG A GA   CTTGTT G AGC  CTC  TT T  GCA
EM:HSKER101 451      GAGAGGGCCAG-A-GAGGACTTGTTAGTAGCGACTCACTTCTGGGCA


EM:MMENDOBA 594      GGAATCCAGGAAGGGAG-GGA
                  D  GGAATCCAGGAA GGAG GGA
EM:HSKER101 561      GGAATCCAGGAAAGGAGGGGA


EM:MMENDOBA 647      GCCTCTGACTGTTCCTGGGACTGGGATGGATTCACTGGAAAACACAAGA
                  B  GCCTCTG CT TTCCTGGGAC  GGA G  TTCACT G A ACA AA A
EM:HSKER101 623      GCCTCTGGCTATTCCTGGGACCAGGAAGTTTTCACT-GGA-ACATAACA

EM:MMENDOBA 694      CGT-TTTTCTCATTCCCCCCAC
                  B  C T TTT C CA TC CCCCAC
EM:HSKER101 672      CTTTTTTACACA-TC-CCCCAC
```

Figure 2.5: An example output of the DBA algorithm, aligning an intron from the keratin gene from mouse and human respectively. This intron is known to be conserved between the two species. Each alignment shows a particular pass through one of the conserved blocks (the intervening, unaligned sequence is not shown).

two structures.

Mapping the 3D superposition back to the one dimensional sequence alignment is not as trivial as it might seem. The superposition often does not assign every alpha carbon of one sequence unambiguously to one from the other sequence, in particular in loop regions, where gaps are common. Although visually one can usually derive a sensible looking alignment from the 3D superposition, on a large scale one needs a program.

The problem can be solved by a rather simple Dynamite comparison which compares two sequences in a manner similar to Smith-Waterman but the comparison method is related to distance between the carbon alpha atoms in the superposition. In this case it was crucial that Dynamite could handle some arbitrary C type calculation in the comparison method, as this would be read from a precalculated matrix that was read in.

Although this method was very ad hoc, the key feature was that I was able to develop the algorithm within minutes of having the problem described to me: and once we had settled on the algorithm, the code required no debugging. This drastically dropped the development time from weeks to under a day. This program was used in the evaluation of the 1998 CASP results [46].

2.7 Comparing two transmembrane proteins

Transmembrane proteins are known to have different residue conservations across the protein, corresponding to the different environments of intracellular, transmembrane and extracellular [47]. In particular the transmembrane domain almost reverses the usual amino acid conservation rules, with hydrophobics often substituting many times for other hydrophobics, but small polar amino acids being highly conserved. Matching two transmembrane proteins on the basis of standard globular matrices is clearly not optimal.

It is easy to construct a PFSM to represent the matching process between two different transmembrane segements as long as one accepts that the length distribution of the different regions will be of an exponential form. This PFSM is shown pictorially in figure 2.6. Finding parameters for this PFSM is harder, as one needs to provide probabilities for 3 different 20 by 20 matrices.

To generate these probabilities I took a pragmatic approach to take trusted trans-

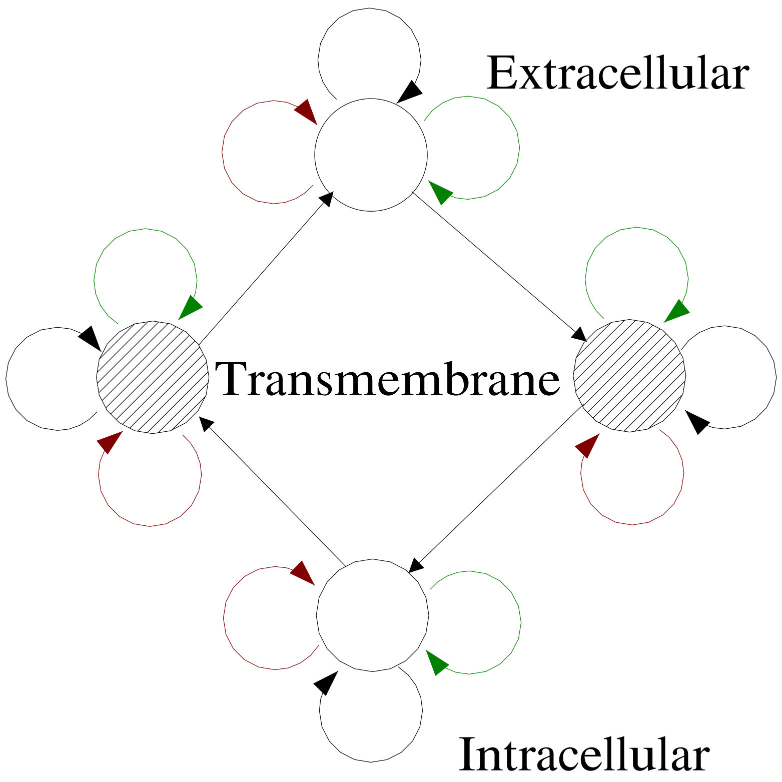


Figure 2.6: A transmembrane matching PFSM. The four states represent intracellular, transmembrane, extracellular and transmembrane environments. The transmembrane state is duplicated to enforce the topology rules of transmembrane proteins. Each state has three looping transitions, the black representing matching of two residues, the green a gap on sequence and the red a gap on another sequence. The transition leading into the state is a matching residue. Many aspects of this PFSM is unrealistic, including the exponential length distribution of the transmembrane regions and the linear (as opposed to the more realistic affine) gaps in the sequence

```
                          +++++++++++++++++++++++ ++++++  +++++  ++
CB21_MAIZE     1          MAASTMAISSTAMAGTPIKVGSF-GEGRIT--MRKTV--GK
CB23_LYCES     1          MA-S-MAA--TASSTTVVKATPFLGQTKNANPLRDVVAMGS
                          ++ + +++  +++++++++++++++++++++++++++++++

                          +++++++++++++++++++++++++++++++++++++++++
CB21_MAIZE     37         PKVAASGSPWYGPDRVKYLGPFSGEPPSYLTGEFPGDYGWD
CB23_LYCES     38         ARFTMSNDLWYGPDRVKYLGPFSAQTPSYLNGEFPGDYGWD
                          +++++++++++++++++++++++++++++++++++++++++

                          +++++++++++++++++================------ -
CB21_MAIZE     78         TAGLSADPETFAKNRELEVIHSRWAMLGALGCVFPELLS-R
CB23_LYCES     79         TAGLSADPEAFAKNRALEVIHGRWAMLGALGCIFPEVLEKW
                          +++++++++++++++++================--------

                          -----------------------------============
CB21_MAIZE     118        NGVKFGEAVWFKAGSQIFSEGGLDYLGNPSLIHAQSILAIW
CB23_LYCES     120        VKVDFKEPVWFKAGSQIFSDGGLDYLGNPNLVHAQSILAVL
                          -----------------------------============

                          ===============++++ +++++++++++++ +++++++
CB21_MAIZE     159        ACQVVLMGAVEGYRIAGGP-LGEVVDPLYPGGS-FDPLGLA
CB23_LYCES     161        GFQVVLMGLVEGFRINGLPGVGEGND-LYPGGQYFDPLGLA
                          ===============+++++++++++ ++++++++++++++

                          +++++++++++++++==================--------
CB21_MAIZE     198        DDPEAFAELKVKELKNGRLAMFSMFGFFVQAIVTGKGPLEN
CB23_LYCES     201        DDPTTFAELKVKEIKNGRLAMFSMFGFFVQAIVTGKGPLEN
                          +++++++++++++++==================--------

                          ------------------------
CB21_MAIZE     239        LADHIADPVNNNAWAYATNFVPGN
CB23_LYCES     242        LLDHLDNPVANNAWVYATKFVPGA
                          ------------------------
```

Figure 2.7: An example output of the transmembrane aligner. The two sequences are chloroplast sequences from maize and tomato. The aligment is shown with + indicating the intracellular environment, = the transmembrane regions and - extracellular environment

membrane alignments and make counts of residue matches in the 40-60% identity range for each region. These counts were then converted into probabilities using a simple pseudocount addition followed by dividing by the total counts. The resulting matrices were good enough to make sensible looking alignments for transmembrane proteins, as shown in figure 2.7.

This method was a prototype to illustrate that using Dynamite more biological knowledge could be integrated with the standard protein comparison methods. This small prototype did not answer many of the harder questions which the work poses, such as what is the best architecture of the PFSM: exponential length distributions are particularly inappropiate for the transmembrane segments which require a minimum number of amino acids to span the membrane. Other researchers are currently looking at integrating protein structure rules with protein alignments using Dynamite, and their work looks promising.

2.8 Other Dynamic Programming toolkits

Dynamite is certainly not the first finite state machine toolkit, and is unlikely to be the last. Here is a brief review of some of the other toolkits available and a comparison of their features with those of Dynamite.

2.8.1 UNIX pattern matchers

There are a number of UNIX pattern matchers which provide implementations of finite state machines designed for text manipulation. These include the UNIX tool `grep`, Perl regular expressions and the compiler tool `lex`. All these methods have flexible pattern matching engines optimised for text processing, in which generally a number of fixed letters are required at certain points in the match. There is little or no ability for being able to score the matches.

The only large application of text pattern matching to bioinformatics is the collection of Prosite patterns [43], convertible to grep expressions (with minor post-processing required for end effects). This pattern library is useful for a number of biological applications, in particular where there are strict rules due to ligand binding to specific residues. However, for most cases, probabilistic grammars (such as hidden Markov models) are more useful.

2.8.2 Dong and Searls

Dong and Searls described a system that allowed specification of quite general dynamic programming finite state machines in 1994 [27]. Their application had a graphical front end in which the user could draw out a finite state machine like in figure 1.2. The application generated PROLOG code which then solved the dynamite programming using the in-built Prolog parsing method, which is a depth first search through the possible paths.

The application was conceived as a prototyping tool and an illustration of how cleanly FSMs represented the alignment process. In the paper they mentioned work to make the front end also generate efficient, compilable programs, however this was not to my knowledge implemented.

The educational and prototyping nature of the application is also clear in the logical design. There was no easy way to connect the finite state machine to the machinery to actually calculate the score at any point: a very restrictive set of functions were provided. For example, this restriction completely prevented their system to describe the repeated HMMs so prevelant in bioinformatics. In contrast, Dynamite allows a large subset of C as the interface between the FSM machine definition and the calculation. In addition, improvements in Dynamite such as special states and labelling greatly extend Dynamite's usefulness.

2.8.3 Lefebvre

A context free grammar generation suite was developed by Lefebvre primarily for RNA folding problems [54]. This grammar suite used yacc like rules to indicate how to build the parser. As it uses a context free grammar, it embraces a larger set of grammars than Dynamite. Like Dynamite it generates C code. However, the integration of more arbitrary C style functions is hard, and it only generates the alignment parsing code, and not other implementations. To its credit, it recognises when it has a grammar which is a regular grammar, and generates the faster, regular grammar code.

2.9 Discussion

I believe the value of Dynamite can be justified by the uses to which it has been put. The algorithms that are presented in the next chapter include some extremely large finite state machines: without Dynamite I am sure I would still be debugging the algorithmical code. The fact that designing a new algorithm takes under an hour, and, with the labelling system, an improved algorithm can usually be easily plugged into an existing system, has meant I have experimented with many different variant algorithms to solve particular biological problems. Suddenly from spending 3 or 4 months designing and coding a particular algorithm one can play around with a number of alternatives, making 3 or 4 different versions in a single day. This freedom allows one to take a different approach in the research of new methods.

I have not only used Dynamite internally, but also distributed and encouraged other people to use it. There is a considerable difference between the author using his toolkit and someone else picking it up and using it effectively (I would like to thank those early users who have helped me so much in this process). I believe that Dynamite has a lot to offer researchers other than myself. Intriguingly Dynamite has even been used in other fields than bioinformatics, such as econometrics.

The ability for Dynamite to automatically bridge the gap between the algorithm designer and specialised solutions such as pthreads code or specialised hardware is a great boon to both camps. It removes the need for the specialist implementor to meet the algorithm designer and understand his new algorithm. To allow this technology to be widely used, Dynamite is distributed with very few restrictions to anyone who wants it. In particular commercial firms can take the Dynamite compiler and integrate it with their own proprietary system without any restriction.

A good example of this ability to separate the implementation from the definition is the OneModel port of dynamite. Recently researchers have provided a message passing interface (MPI) code port of dynamite; this supports parallelisation where there is not shared memory. In addition, I am thinking of improving the performance of some of the alignment methods using different reduced space implementations [39]. By implementing this algorithm in Dynamite it will apply to all 12 different algorithms in my standard package automatically.

A number of biologically focused researchers have downloaded Dynamite and started to use it to investigate new algorithms. As Dynamite shields them from the actual implementation, and encourages experimentation of the algorithm they can focus far more on the biological model to use. For some 30 years we have been using the same basic biological model of protein evolution in sequence alignment - with the restraint of implementation removed, how much better can we do? I am looking forward to seeing the results of these endeavors.

Chapter 3

GeneWise

3.1 Introduction

Most eukaryotic organisms exhibit *pre-mRNA splicing*. In this process, the transcribed RNA from the DNA sequence is processed by a complex protein and RNA mediated system which excises parts of the mRNA sequence (these sequences are called *introns*), and joins the remaining mRNA sequences in order (these sequences are called *exons*). The resulting, mature, mRNA sequence is exported from the nucleus where it usually provides the template for the translation of a single protein sequence.

The reasons why eukaryotic organisms indulge in such a complex mechanism between the genomic DNA sequence and the translatable mature mRNA sequence are unclear: people have hypothesised everything from the fact that this exon/intron structure is beneficial to the evolution of genes through to the idea that introns are simply selfish elements that became fixed in eukaryotic genomes. An important issue is that the processing of the mRNA provides another level of control of gene expression, one of the most common being that a single mRNA can generate more than one mature product due to *alternative splicing*. Regulation via alternative splicing is, for example, responsible for a key decision in sex determination pathway in Drosophila melanogaster [20].

One point in all this is very clear: pre-mRNA splicing greatly increases the computational complexity of understanding an organism from its genomic DNA sequence alone. This chapter presents some new approaches to predicting the splicing structure of genes. Firstly I will review some of the biochemistry of splicing and the current approaches to solving the gene structure prediction problem.

3.1.1 The biochemistry of pre-mRNA splicing

The process by which freshly transcribed mRNA gets processed to mature mRNA is one which has been extensively researched since the 1970s when splicing was discovered. A number of experimental systems, in particular *in vitro* human extracts and yeast genetics have provided a detailed view of the splicing process [58]. The anatomy of the spliced RNA is shown at the top of figure 3.1. Each exon/intron boundary is called a *splice site* and are named relative to the intron, the 5' splice site (also called the donor site) being at the start of the intron and the 3' splice site (also called the acceptor site) at the end of the intron. All pre-mRNA splicing goes through two steps. The first step has the 5' end of the intron attack a Adenosine residue inside the intron, forming an unusual 5' to 2' phosphodiester bond. The branched mRNA structure which this forms is called the lariat and is easily distinguished biochemically from other mRNA species. The second step has the free 3' end of the exon attack the 5' end of the downstream exon. This excises the intron and splices the two exons together.

The splicing process can be divided into two distinct phases: the recognition of a particular intron and flanking exons defining the correct splice points and the actual biochemical splicing. Sadly the latter phase (the biochemical splicing) is far better understood than the recognition. Some of the major players in the splicing process are small nuclear ribonuclear particles, or snRNPs. Each snRNP is a large particle made from one particular RNA molecule and a number of protein molecules. Each snRNP is characterised by the RNA molecule it contains, which are called U1, U2 etc, leading to U1 snRNP, U2 snRNP etc. The intron is recognised by a poorly understood set of factors, which include SR proteins and other proteins not associated with snRNPs [45]. These proteins and the U1 and U2 snRNPs somehow both recognise the splice junctions as being valid, and also correctly pair up the 5' and 3' of the intron. There is a suggestive partial base paring between part of the U1 snRNA with the 5' splice site, however the role of this base pairing in intron recognition is unclear. This recognition process leaves the U1 snRNP at the 5' splice site of the intron and the U2 snRNP at the branch site. Next the tri-snRNP, U4/5/6 snRNP, which is made from the U4, U5 and U6 snRNP joins the U1 and

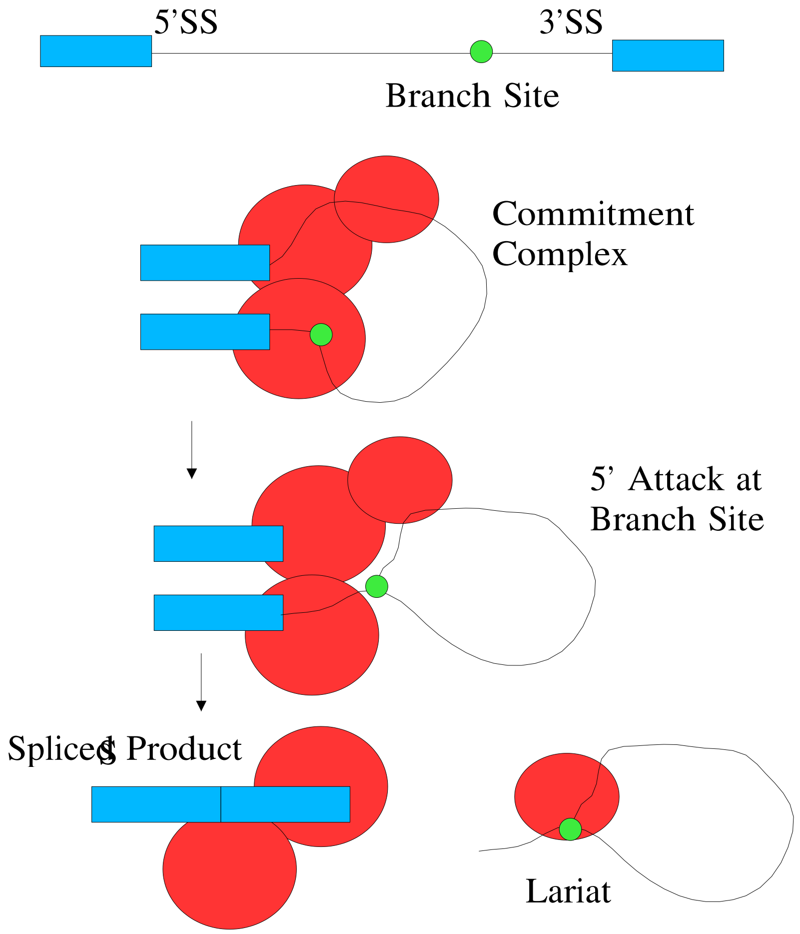


Figure 3.1: A diagram showing some of the major processes in splicing. The top panel shows the exon/intron structure. The exons are in blue and the line joining them represents the intron. The 5' and 3' splice site are shown, and the green spot represents the branch site. The next three panels shows the splicing process. The red circles represent protein complexes. The first shows the commitment complex which is a loose association of proteins forming on the two ends of the intron and elsewhere. The second shows the first catalytic step which is the creation of a 5'2' linkage to the branch site. Finally, the second catalytic step is the joining of the two exons and the release of the intron in the closed "lariat" form.

U2 snRNPs, making a complex which can be fractionated called the *spliceosome*. There are a number of rearrangements of the complex (crucially the association between the U4 and U6 snRNPs change) as the spliceosome catalyses the two steps of the reaction. Intriguingly, the U6 RNA itself has been implicated in the actual catalytic process, which leads to interesting hypotheses of the relationship between pre-mRNA splicing and the self splicing introns (in particular group II self splicing introns).

Although the biochemical catalysis of the splicing process is well worth further investigation, the computationally interesting part is the recognition of the introns and exons. One can think of at least two problems the recognition machinery must be able to handle: firstly it needs to distinguish real splice sites and hence real exons and introns from other regions in the mRNA which "by chance" resemble splice sites. Secondly it needs to be able to pair the correct splice sites together, as one could easily recognise correct splice sites, but in fact skip over an exon, erroneously splicing the mRNA. These problems are exasperated when one considers alternative splicing where a single primary transcript has multiple valid splicing patterns in either different tissues, or in many cases, the same tissue at the same time.

Splice Site Recognition

By looking at many introns and exons, a number of sequence features in the mRNA could be deduced. Firstly the splice sites both on the 5' splice site and the 3' splice site have some conserved residues. The strongest feature of these conservation is that the 5' splice site has a conserved GT dinucleotide and the 3' splice site has a conserved AG dinucleotide, giving rise to the so called "GT-AG" rule. However it is obvious that more information than these dinucleotides are required to specify an intron: indeed when naive or complex approaches for modeling the 5' and 3' splice sites are used they are not sufficient to distinguish real splice sites from random noise in the sequences. Another biochemical feature is the branch site adenosine which is attached to the 5' end of the excised intron. Early examination of vetebrate branch site sequences suggested a weak consensus around the A (an adenosine is always used), however, further analysis of a number of introns did not confirm this feature. In contrast, in *S. cerevisiae* and *S. pombe*, strong branch point consensi were defined. In human introns, additional information must be used by the recognition machinery. At the 3' splice site, there is a clear run of cytosine and uracil nucleotides in the RNA before the splice site and after the branch site. (these are encoded as cytosine and thyamine nucleotides in the DNA sequence). This *poly pyrimidine* tract seems like a clear signal to localise the 3' end of the intron, and factors which are biochemically implicated in the splicing process, such as U2AF, have a *in vitro* affinity for poly-pyrimidine RNA (as do, it must be admitted, other factors which are not involved in RNA splicing) [49].

U12 splicing

In recent years a number of introns have been found which do not contain the GT-AG dinucleotides but instead have AT-AC dinucleotides at either end of the intron. Are these introns were spliced with the same machinery? Elegant experiments showed that for a number of AT-AC introns, a parallel splicing system was used, with a new series of snRNP factors, each corresponding to one of the "traditional" set of factors, i.e., U11 for U1, U12 for U2 etc. The two systems were named after U2 or U12 snRNPs, as it is these snRNPs which have been implicated as the key components which distinguish the two pathways.

On further investigation, it became clear that whether an intron is spliced by U2 or U12 dependent systems is not associated with its dinucleotide splice site endings, but instead on the presence or absence of a branch site consensus, (U12 dependent splicing has a strong branch point signal like the yeast sequences). This emphasised that the recognition of the splice sites does not occu by straightforward base pairing of snRNA species to the splice site consensi [18]. The discovery of U12 splicing has provided biochemists with a more amenable splicing system as the U12 system seems to be simpler in terms of the number of additional factors it uses.

Splicing enhancers

Because it was clear that the splice site consensi were not sufficient to explain splice site recognition, a number of groups attempted to find additional *cis*-acting motifs (in the RNA sequence) and *trans*-acting factors (which would bind the RNA) which are involved in this recognition. A group of non snRNP splicing factors were discovered, called SR proteins which have a number of similar sequence features [10] and biochemical features. These proteins as a group have been shown to be necessary factors for splicing and their concentration can influence alternative splicing patterns both *in vitro* and *in vivo* [81].

Many different attempts to classify how these proteins work has led to an understanding that they bind small, weakly conserved RNA sequences, often found in exons. The binding of these factors in many cases stimulates the splicing of the upstream intron (for example, see [56]). An analogy has been made with the transcriptional enhancers involved in promoter recognition. In both cases the sequence motifs are not enough to uniquely indicate the splice site or promoter. The motifs are often additive in their action, and can occur in a number of orientations relative to the splice sites.

3.1.2 Current computer approaches to predicting splicing patterns

Prediction of genes solely from sequence data has been a topic of research since the 1970s [79]. The first computer programs to model splicing were based on a mixture of rules and intuition about how to represent and combine the signals from a number of different features (for example, the knowledge of the splice sites and the fact that protein coding genes have no stop codons). From these rule based methods came three different approaches which attempted to provide a more formal basis of designing a gene finding program.

Neural Networks

A number of artificial neural networks were used to characterise genomic features involved in splicing. Soren Brunak and colleagues made a feed forward network which took a window of 19 base pairs around a potential 'G' nucleotide to classify splice sites [14]. Using this network they discovered a number of errors in the nucleotide databases.

Ed Uberacher and colleagues also used a neural network to predicted exon features [85]. Rather than providing the network with a direct representation of the DNA sequence they presented the network with a number of features calculated on a 10 base pair window, the most important of which were different types of 6 base-pair codon frame measures. The network then integrated these signals to provide an overall exon prediction.

Although these methods were certainly effective at the tasks they designed for, it is less clear how complete gene predictions can be made by using neural networks, as presenting all base pairs in a piece of genomic DNA to a network is clearly impractical. The importance of neural networks in gene prediction has diminished over this decade mainly for this reason.

Linear Discriminant Functions

Other machine learning techniques have been used apart from neural networks. Solovyev, Salamov and Lawrence exploited Linear Discriminant Analysis to solve the gene prediction problem [87, 86]. Linear Discriminant Analysis finds the best hyperplane in a series of feature dimensions which separates two classes of data. In these cases the authors built up Linear Discriminant functions for many different genomic features, such as 5' and 3' splice sites, the starting ATG and poly-A addition site. These features in each case were trained by compiling training sets of real features and "pseudo-features" which were pieces of random genomic DNA which met some criteria of being a possible, but not real, splice site or poly-A. For example, for the 5' splice site, they found sequences containing GT which were not splice sites.

These individual features could be used on their own as feature prediction in their own right, but this would not help people get a complete prediction of the gene. To counter this, the authors provide a final dynamic programming based parsing of a sequence on the basis of these pre-computed features. This dynamic programming is of features, not of individual bases, and so is unlike the dynamic programming outlined in chapter two of this thesis. The score of the entire gene prediction is the combination of the Linear Discriminant values for the individual elements in it, and the best parse has be consistent with a number of rules, the most important one being that it does not contain stop codons in the translation.

3.1.3 PFSMs in Gene Prediction

The use of PFSMs in gene prediction has already been discussed in the introduction chapter (section 1.4.6). PFSMs are a good fit to the gene prediction problem. The following sections will briefly discuss the different PFSM based gene prediction methods with an emphasis on their biological gene model.

Genie

Genie [53] provides a generalised framework which represents the gene model in which the actual components can be swapped in and out. The components were envisaged to be a mixture of simple ones, such as 5th order Markov chains (hexamer frequencies) for exon regions, and complex ones, such as neural networks for splice sites. The integration of neural networks, which do not come with an easy probabilistic interpretation into a probabilistic correct model was a challenge. To allow these complex features to be correctly expressed, the framework had to cope with features of variable length being "produced" by the model. To allow this, the Markov rules were relaxed to allow the duration of a particular set of observations to be taken into account in the definition of the probability of its emission. This extension can be integrated into all the common place manipulations of the hidden Markov models at the expense of additional running time and memory requirements.

Genscan

Genscan provides a very similar framework model as Genie does but rather than making a clear distinction between the framework model and its components, concentrates on providing a single, integrated model for the entire process [17]. Like Genie, the Markov rules need to be relaxed, though in Genscan's case this is primarily to allow the exon length distribution to be more accurately modeled. Genscan provids a complex representation of 5' splice sites, using a mixture between a decision tree and position specific weight matrices. Another important feature was that the hidden Markov model was parameterised for four different GC content levels. In human sequence it has long been known that the gene structures are different in different isochores: in euchromatin introns tend to be shorter whereas in hetrochromatin introns tend to be longer. Euchromatin and hetreochromatin also have different GC content. The reparameterisation on the basis of different GC content provides an effective way of modeling the observation of different intron lengths in the different isochores without requiring the user to define the isochore beforehand: whether the GC content is directly linked to intron length is less obvious.

Genscan's effectiveness is that it is entirely focused on providing a single, integrated hidden Markov model for ab initio gene prediction. Because of its simplicity in approach it has become the gene prediction method to beat, even though other methods have a roughly similar measurable performance.

HMMgene

HMMgene is the third major hidden Markov model approach to gene prediction [51]. In this case a number of interesting additions to the HMM approach were applied to tackle the problem of modeling exon length distribution. The exons are modeled as being the production of a series of identical states which leads to a broad normal like length distribution for the exon. As each state is providing the same "biological" model, the straight forward Viterbi path is inappropriate: the distribution of exon lengths arising from Viterbi parsing would be an exponential decay.

HMMgene solves these problems by providing a best parse over biological "labels", where each label is the biological interpretation of the state. The exon states share the same label, making this parse of this model not the same as Viterbi. HMMgene provides a non exact method (N-best paths) to capture the best label parse given particular data. One pleasing consequence of this is that as desired the expected length distribution of exons is nicely humped.

3.1.4 Performance of ab initio Gene Prediction programs

The performance of *ab initio* gene prediction programs is hard to assess. The main problem is that in long pieces of genomic DNA of the type one one would like to using for testing, it is hard to know for sure that one has experimentally verified all genes. A gene prediction which does not overlap with experimental proof can easily be a mistake in the experimental verification, not a false positive. Despite these problems a number of assessments of gene prediction accuracy have been developed. Guigo and Burset provided one of the earliest assessments [19], which has since been followed by a number of assessments more relevant to genomic DNA [15]. The conclusion of the large scale studies is that gene prediction programs have a coverage of around 90% of exons but at an accuracy of around 50%: in other words, one can expect around half the exon predictions to be incorrect in large genomic datasets.

3.2 Combining Homology with Gene Prediction

The central idea behind this work is very simple: by combining the fact that one knows that a gene produces a protein which is homologous to another known protein with the gene structure rules one should be able to make far better gene prediction methods. Looking at this problem from another perspective, it also frees people who are interested in finding proteins which are homologous to their protein of interest from being dependent on a good gene predictor to analyse genomic DNA.

Both the process of evolution between two homologous protein sequences and the gene structure rules are well described by PFSMs. This suggested that a single PFSM which performs both processes simultaneously should be possible. To provide this PFSM I need a description of the two models and a theory to provide the combination of the models. Intuitively one expects that the resulting model would be a PFSM with similar properties to the two combined models, though considerably larger in size. By relying on the other probabilistic models as a starting point, the parameterisation could be solved by taking the previous models and making sure that we had a principled way of combining the models.

The implementation of the combined probabilistic model looked daunting, as its sheer complexity, along with the size of the sequences which I would be using suggested a programming nightmare. Thankfully, by using Dynamite as a way of implementing the algorithm a robust, timely implementation of the Viterbi algorithm could be provided.

3.3 Combining Probabilistic Models

By solving the general problem of how to combine two PFSMs I could then use that for this specific case. The type of PFSMs which I wish to combine are two alignment type PFSMs, each providing a mapping from one of set of letters to another set of letters. Consider two machines S and T, where S maps a sequence of letters from the alphabet $\mathcal{A}$ to sequence of letters from the alphabet $\mathcal{B}$ and T maps letters from $\mathcal{B}$ to $\mathcal{C}$. The following notation is used to describe the machines. (This process is shown diagrammatically on the top panel of figure 3.2).

S has states $1 \ldots n_s$. The transition from state i to state j, emitting a finite string of letters a in one sequence and b in another sequence has probability S_{ijab}.

a is a finite string of letters drawn from the alphabet $\mathcal{A}$ with 0 representing a non-emitting spacer. b is a single letter drawn from the alphabet $\mathcal{B}$ with the additional string 0 representing a non-emitting spacer. There is a requirement for b to be a single letter to allow the merging process to work. Similarly the T machine is defined with states $1 \ldots n_t$, and a transition from k to l emitting a single b and any finite length of c letters is T_{klbc}.

We wish to construct the state machine U which will map a sequence of $\mathcal{A}$ letters to a sequence of $\mathcal{C}$ letters, considering all possible sequences of $\mathcal{B}$ intermediates. We propose that U has $n_s n_t$ states, each of which can be characterised by a pair of states in each of the original machines. i and j will be used for states from the S machine and k and l as indexes from the T machine. Thus the transition $U_{(i,k)(j,l)ac}$ is the transition from the state (i,k) in U derived from i in S and k in T to the state (j,l) derived from j and l states respectively, emitting a in one sequence and c in another sequence. We need to construct the definitions of the probability for each of these transitions in U in terms of the transitions defined in S and T. The following equations provide that definition.

For neither a nor c being 0

$$U_{(i,k)(j,l)ac} = \sum_{b \neq 0} (S_{ijab} T_{klbc}) \tag{3.1}$$

For a being 0 but not c

$$U_{(i,k)(j,l)0c} = \delta_{ij} T_{kl0c} + \sum_{b \neq 0} (S_{ij0b} T_{klbc}) \tag{3.2}$$

For c being 0 but not a

$$U_{(i,k)(j,l)a0} = \delta_{kl} S_{ija0} + \sum_{b \neq 0} (S_{ijab} T_{klb0}) \tag{3.3}$$

The first equation (3.1) sums over all possible b intermediates for this transition, using the underlying transitions from the S and T machines. The two other cases are when no letter is generated for one of each sequence. In each case the blank could have been generated from either S or the T machines. For the case of generating a blank in the a stream of letters, if it was generated by the S machine then that means that there is an intermediate b sequence which has to account for the c string. However, if the blank was generated by the T machine then there is no possible b

letter, and furthermore this case can only occur when the transition remains silent in the S machine index (hence the δ_{ij}). The symmetrical argument applies for c emitting 0 but a. We need not worry ourselves about the case when S emits a $a, 0$ pair and T emits a $0, c$ pair as this contains no b sequence, and so is not permitted.

The combined machine is shown pictorially on the bottom panel of 3.2.

The requirement that the two machines emit at most a single b letter per transition is so that we can do the summations in equations 3.1, 3.2 and 3.3 over all b. If b was longer than a single letter it would be possible to have S machine produce a b string which was out of phase with the b string that the T machine would produce. I can see no clean way to provide the derivative U machine transitions in this case. One could claim that longer b strings could be allowed as long as the two machines were emitting "in phase" strings, but this simply means that there is a new alphabet, $\mathcal{B}'$ where each "in phase" string is mapped to a single b' letter.

Notice that the U machine represents a sort of model "product" of S and T. This means that for even modestly sized machines, the product will be quite large. However the combined machine allows us to ignore the identity of intermediate sequences in standard calculations of, for example, what is the likelihood of two sequences, A and C being generated by the combined machine, and what is the most likely path through both S and T. The additional bulk of the combined machine is easily justified when one considers the alternative is listing all possible B sequences, which, depending on the architecture of the machine, might in fact be infinite.

This theory has been built up for a first order PFSM in the model. To extend this theory to higher order PFSMs is easy as one can utilize the fact that any PFSM can be represented as a first order PFSM at the cost of more states in the machine. This provides us with a principled way of combining any two machines. However, notice that the expansion of a non first order machine to a first order machine is also a "product" type operation. If one does want to merge two non first order machines, the number of states required to model all the independent paths through each machine will rapidly become impractical.

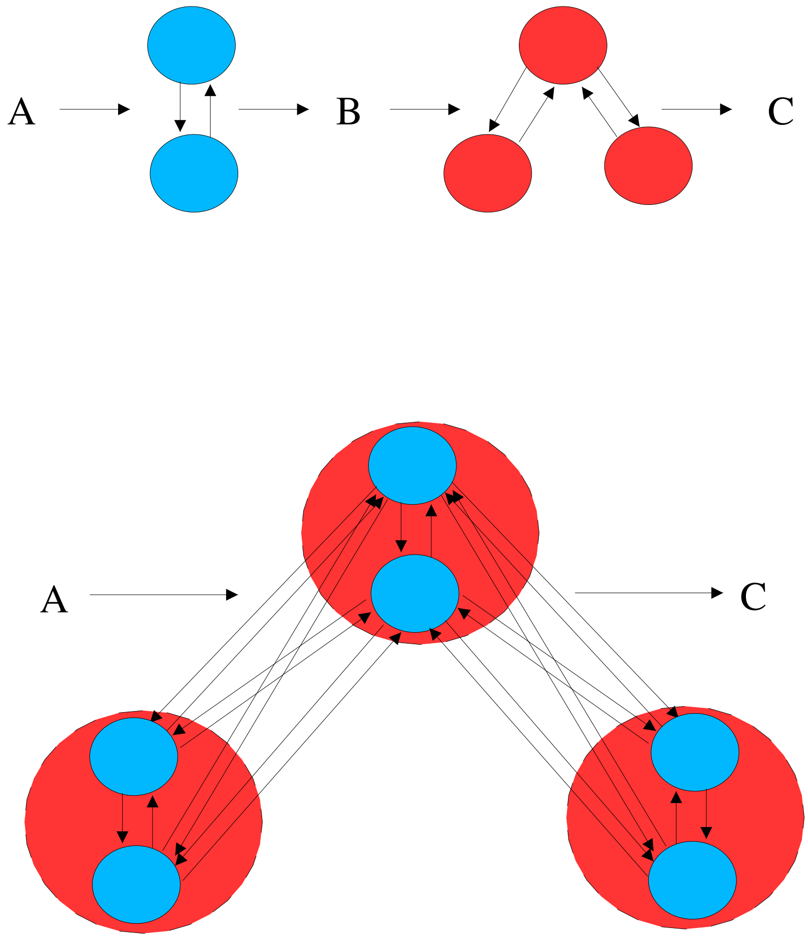


Figure 3.2: A diagram showing the combination of two models. The top panel shows the two PFSMs acting sequentially, first transducing the A letters to B letters and then transducing the B letters to C letters. The lower panel shows the merged machine.

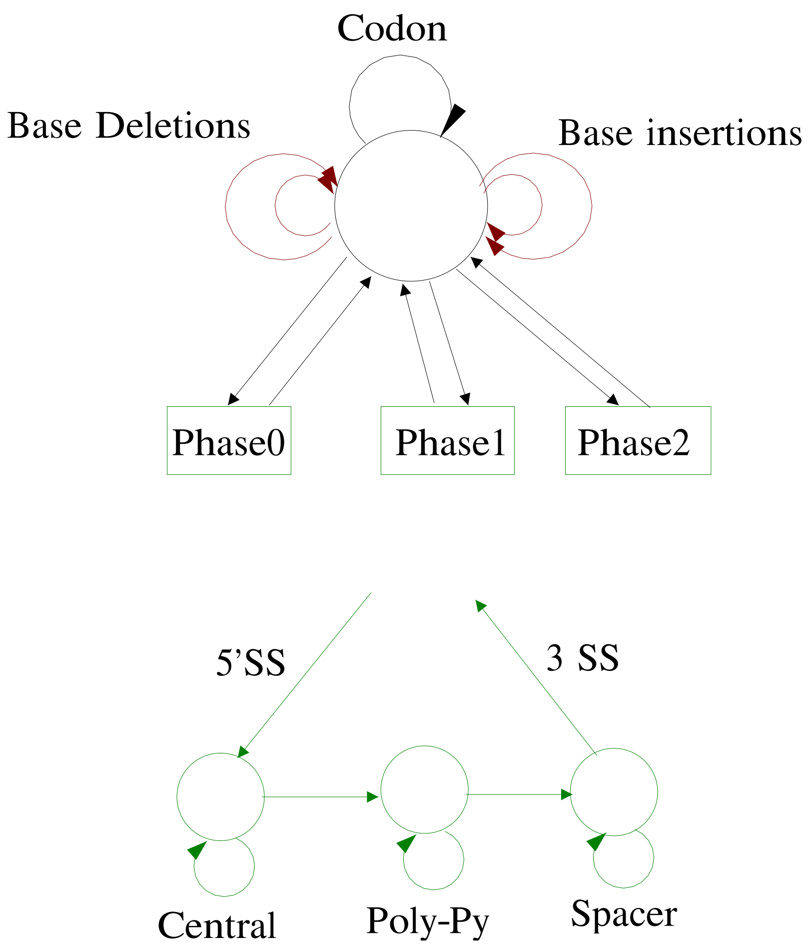


Figure 3.3: The gene model used in GeneWise. The central black state represents the codons (there is no explicit start or ending of the codons). Sequencing error are modeled as codons of one or two bases long (deletions) or four or five bases long (insertions) shown in red. There is a separate model for each phase of introns, each of which have three states, shown in green. The transition to the first state emits a fixed length 10 base pair region representing the 5'SS. The transition from the last state emits a 6 base pair fixed length region representing the 3'SS. The poly-pyrimidine tract is modeled as a separate state

3.4 GeneWise model

The GeneWise model was created to be the integration of two separate models, a gene prediction model and a protein homology model, using the ideas outlined in (3.3). The genomic sequence is equivalent to the $\mathcal{A}$ sequence, the predicted protein sequence of the gene is the $\mathcal{B}$ and the homologous protein sequence to which it is being compared to is $\mathcal{C}$. The aim is to compare genomic sequence directly to the homologous protein sequence considering all possible intermediates of the predicted protein.

To be able to use the model combination theory effectively, high order Markov dependencies have to be removed from the two models. The protein model, which is a probabilistic Smith-Waterman model is first order. Gene prediction models however are generally of a higher order and it this model which has simplifications to make the merging process achievable. There are two approximations made to make the model usable. Firstly amino acids which are split by introns are ignored. This is similar to what happens in most gene prediction programs, which make the same approximation to avoid having a high order Markov dependency in the PFSM. Secondly high order Markov dependencies in the coding sequence model are shortened. Most gene prediction programs use 5th order Markov chains to model coding regions, whereas I use a simpler model that emits triplets (equivalent in some sense to a 2nd order Markov chains). This has the nice feature that there is a simple mapping to the letters of the intermediate sequence, the predicted protein.

The gene prediction model is shown in figure 3.3. It has a single state representing exons which emits a series of independent codons. The intron states are differentiated by which phase the intron occurs in (phase being the place in the codon which the intron interrupts). For phase 1 and phase 2 introns, the fact this interrupts a codon is ignored, and is not scored. Each intron is considered to be made from 5 sections: the 5' splice site, a central intron section, a poly-pyrimidine tract, a spacer following the poly-pyrimidine tract and the 3' splice site. As the 5' and 3' splice sites are considered to be ungapped motifs, they can be represented by single transitions which "emit" 10 or 6 base pairs respectively.

Sequencing error is represented in a naive manner in which the insertion or deletion of bases is considered to produce codons of one or two (in the case of deletion) or four and five (in the case of insertion) base pairs. In these cases, the type of codon is ignored.

The homology model is deliberately modeled to be the inclusive model of the Smith-Waterman protein alignments and Krogh-Eddy protein profile HMM. The homology model has a number of repetitive nodes, each node being 3 states, called Match, Insert or Delete, see figure 3.4. The Match State for protein sequences represents matching a protein residue in the observed data, whereas the Insert state represents inserting an additional protein residue in the observed sequence relative to the given homology information. The Delete state represents a node which is skipped out (silent), i.e.homology information which does not have an equivalent in the observed sequence.

This homology model can also be represented in a more compact form as in the bottom panel of figure 3.4. In this form the node structure of Match, Insert, Delete is represented, and it is assumed to repeat for the length of the model (length being in nodes). This representation emphasises the similarity to the traditional Smith-Waterman method, and as a transducer type PFSM.

Given these two models, the combination using the rules outlined above is simple. The combined model should have 10×3 states, expanding each homology model state into 10 separate gene finding states. This process is shown pictorially in figure 3.5. However not all these states are actually required in the comparison as we know that some transitions are forced to zero. This is because it is impossible to get an intermediate protein sequence letter with no genomic DNA sequence: in other words the combination $0b$ in the previous notation does not occur. Applying this to equation 3.2 means that we can remove a number of states.

Because transitions which emit $0c$ are all directed to the "Delete" state of the homology model, this means that all the transitions which are directed to the intron states of the delete state in the combined model have probability zero, as $i \neq j$ for these states. The result is that we can remove them. This is intuitively correct, as the Delete state models positions in the homologous protein sequence which have no sequence in its protein counterpart. With no protein sequence in the predicted protein there is no possibility of an intron at this position. Notice that the Delete state still has transitions to the Match and Insert introns as these transitions do emit a protein sequence.

The inter-intron transitions can also be removed. The intron states all have transitions which emit $a0$ producing genomic sequence with no corresponding protein

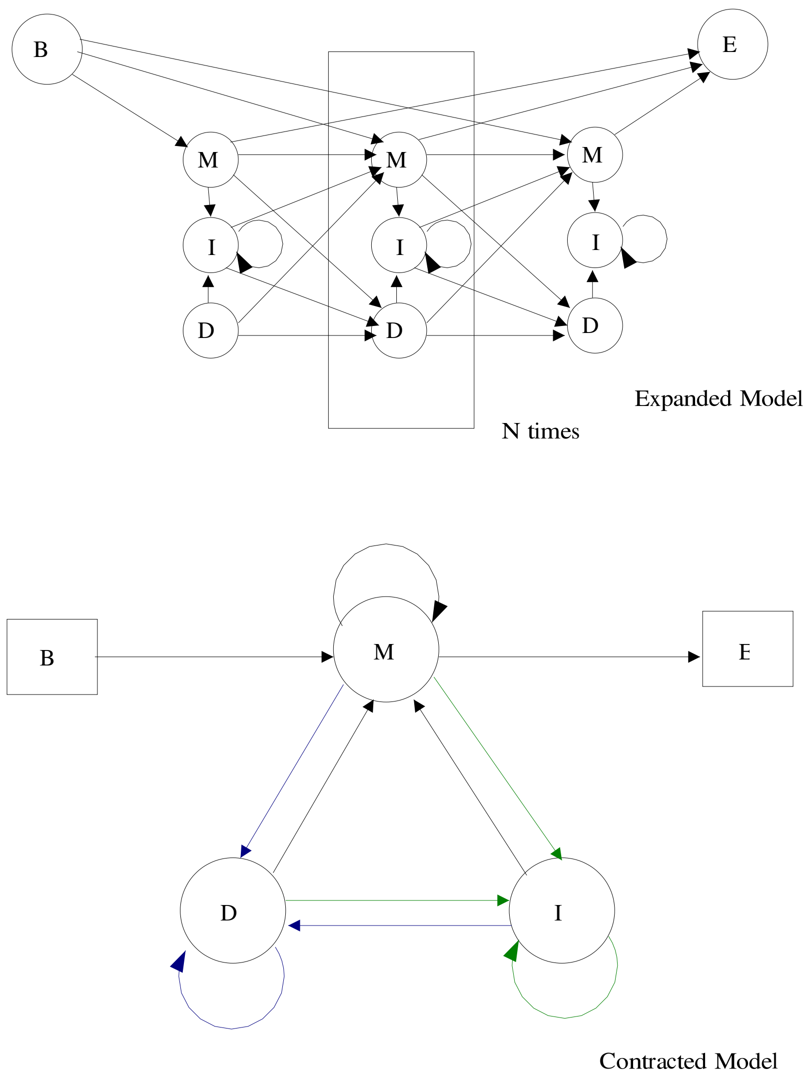


Figure 3.4: Two representations of profile Hidden Markov Models. The top panel shows a standard "expanded" hidden Markov model, in which each state in the model is shown. The central portion of the model can be repeated N times. The bottom panel shows a contracted representation of the model, where each state represents one of the canonical states in the HMM. The transitions between the states are coloured to show how they progress in terms of the model and the sequence. Black transitions advance in both the model and the sequence, blue transitions in only the model and green transitions only in the sequence.

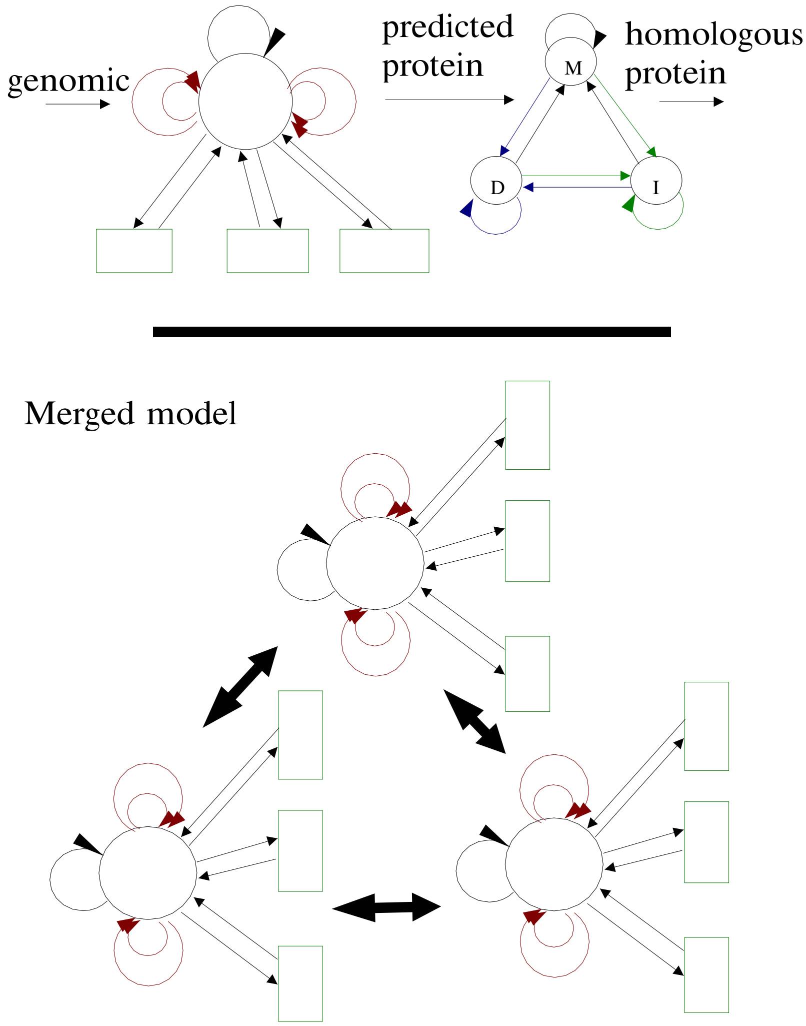


Figure 3.5: A figure showing the merging process between the gene prediction model and the homology model. The models used have the same outline as those in Figure 3.3 and 3.4. They are written out in the top panel. The lower panel shows the merged model. The large black arrows indicate that between each set of four states, all possible transitions exist. This model will be later pruned to a small set of states: see the text for more details

sequence. For these transitions the sum in equation 3.3 is zero as all the transitions $ab, b \neq 0$ are zero. The other, δ_{kl} term is also zero as movement between different introns implies $k \neq l$. Again this marries well with the observation that one cannot change from being in a "Match" intron to an "Insert" intron in the middle of an intron.

The pruned model, called GeneWise 21:93 is shown in figure 3.6. The name reflects the number of states (21) and number of transitions (93) used in the model. Like the HMM figure, this figure represents the repetitive structure of a single node, which is repeated for the length of the homology model.

This model is written down as a Dynamite model, which is provided in Appendix B. The Dynamite compiler then generated the Viterbi and database searching versions of this algorithm automatically.

It was clear from the start of this work that GeneWise21:93 was an overly ambitious model, and probably not useful for practical work. Using Dynamite I experimented with a number of different machines, and a good compromise between speed, sensitivity and compromise on the correct model was the GeneWise 6:23 model, shown in figure 3.7. Compared to the GeneWise21:93 model, what I have removed is the following

- The poly-pyrimidine states are removed, providing a saving of 12 states and 36 transitions

- The Match/Insert information is lost in the introns. This means that for the cases where a Match to Insert or Insert to Match transition occurs in the protein in the same position as an intron, it will not be scored correctly. As both introns and Match/Insert switches are rare, this should not be a significant problem

- The sequencing error transitions are removed from the Match/Insert and Delete to Match and Delete to Insert transitions. This means that sequencing error which falls at a switch in the protein state will not be modeled, and will probably occur a base pair before or after the real position

Heavy use of the GeneWise6:23 model has shown excellent results (section 3.9 presents results on this), and has become the workhorse for GeneWise methods. I

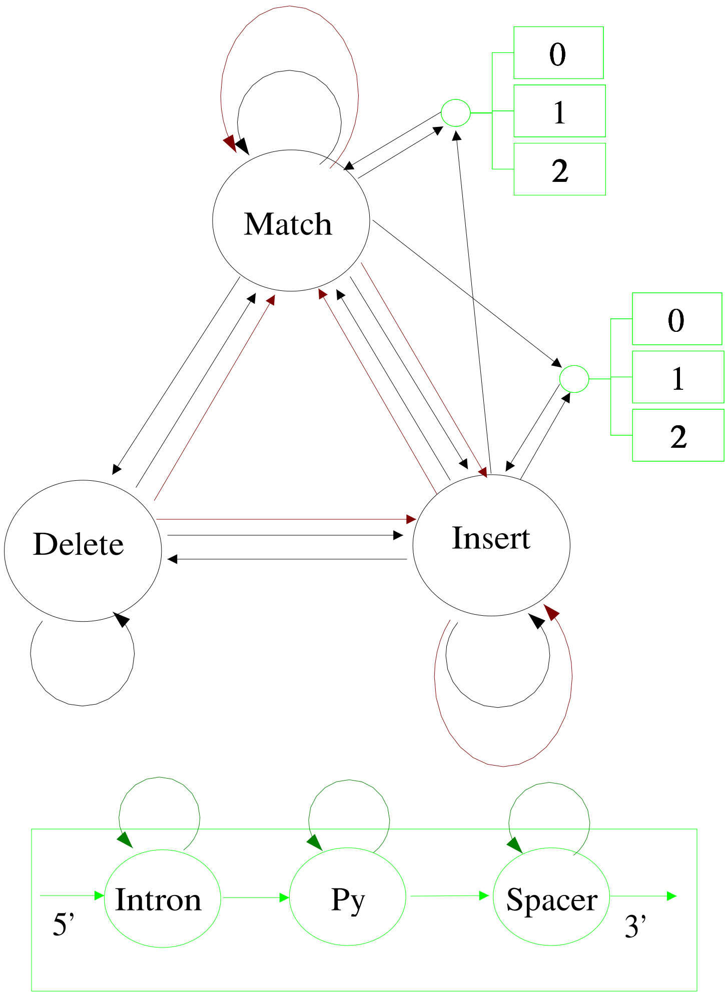


Figure 3.6: GeneWise21:93 Algorithm. The dark circles represent states, and the arrows between them transitions. Black transitions are standard protein transitions, red transitions are frameshifting transitions and green transitions are intronic transitions. Introns are each built of three states, listed at the bottom of the figure

made even further reductions in GeneWise4:21, in which the different phases of the introns were merged into a single state, resulting in only 4 states and 21 transitions.

3.5 Parameterisation

Simplicity was the byword for the parameterisation problem. I expected most of the power of this method to come from the application of accurate protein profile-HMMs to the gene prediction model (i.ethe protein homology model would force the gene model to take certain parses as the gene prediction was a better fit to the protein homology). The gene model would only be used to provide good edge detection of exons, principly splice sites. Therefore the approach was to take the established profile hidden Markov models from Sean Eddy's HMMER package for the protein homology model. When we used a single sequence, not a profile HMM, we parameterised it as if it was a protein profile HMM, providing parameters deduced from the standard Smith-Waterman parameters routinely used. For the gene model, we wanted to make it simple. The approach was to make maximum likelihood estimates of the fixed length motifs of the splice sites from known splice sites, and parameterise almost all the rest of the gene model as if it was background. However the devil is in the details of parameterisation, as the next sections will illustrate.

3.5.1 Splice Site Models

Splice sites have a long history of being modeled in some complex ways. These range from neural networks, through position weight matrices and more complex, decision tree type models such as the one used in Genscan. As the first attempt at a splice model we designed a system which was a) a generative probabilistic model (so that we could fit it in well with our scoring scheme) and b) could capture a lot of the supposed complexity in splice sites (in particular the 5' splice site) using a decision tree structure very similar to Genscan's but with more branches.

However, this method did not perform well, perhaps due to our training system or perhaps because we did not require a complex splice site model (see the path interpretation detailed below). We were also impressed by work from David Haussler and colleagues that showed both that decision tree methods could be considered a mixture of position weight matrices, which in many ways simplified their possible training and that these mixtures could not detect that much more information in

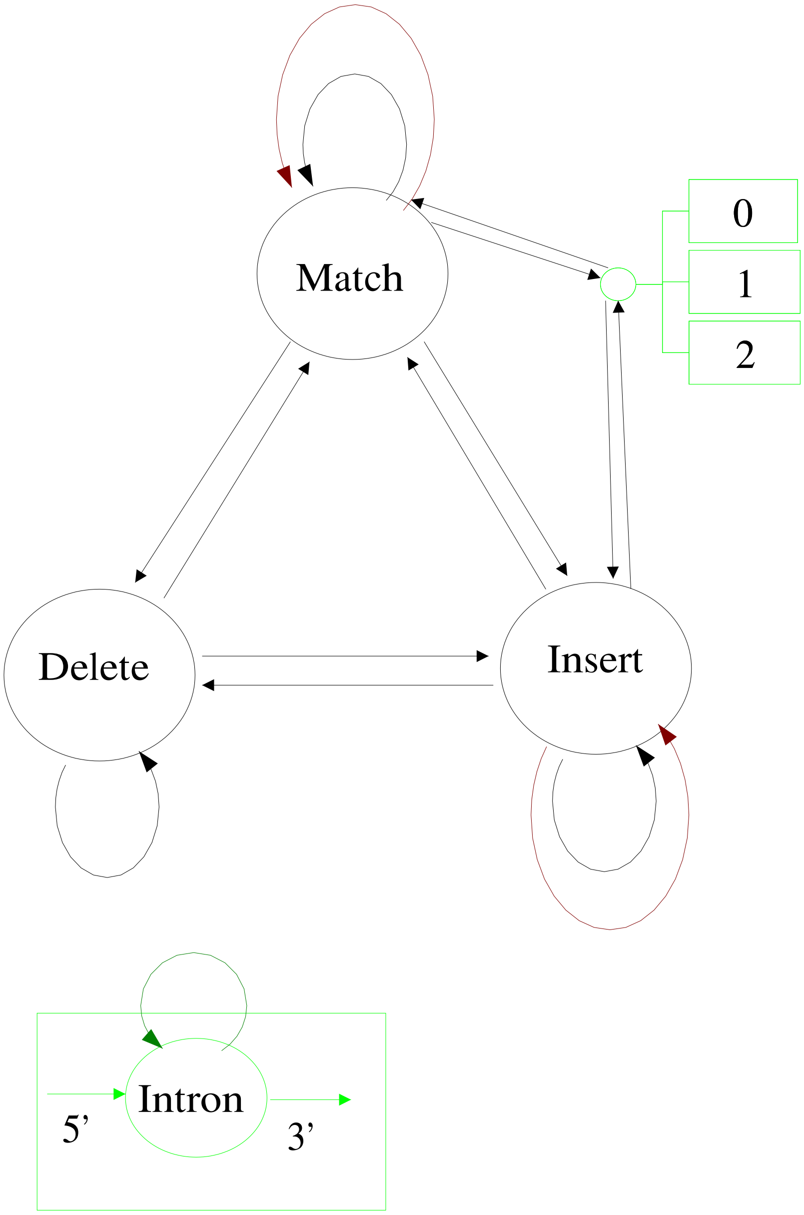


Figure 3.7: GeneWise6:23

the splice sites than simple position weight matrices. Despite all the mystique surrounding splice site models, it seemed that simpler was better! We therefore settled for a very simple, ungapped model of splice sites which could be estimated from observed splice sites by simply lining up splice sites on the splice junction and observing residue counts at each position. To deduce the probabilities of the model, we added a simple pseudo count method which represents a single, unbiased Dirchlet prior distribution on the multinomial of residue emission probabilities. This simple model outperformed the old model considerably, and we have found no reason to revisit the splice site models since.

3.5.2 Codon emission probabilities

The emissions of codons in the Match and Insert transition in the model are due to three different effects:

- The amino acid distribution of the protein homology model
- The codon bias of the organism
- The substitution of the base pairs due to possible sequencing error

We considered this process to be the transformation of the vector of 20 amino acid probabilities in the homology model to the 64 possible codons. The codon probability given a particular amino acid is decomposed as coming from two possibilities.

- There was no substitution error, in which case the probability is 0 for codons which are not translated to this amino acid, or $P(codon|aminoacid)$ for the codon bias in this organism.
- There was a substitution error, meaning that the observed codon is actually a different codon in the real DNA sequence. In this case, we considered every possible single base pair substitution, but not double hits inside one codon.

Due to every possible base being substituted, in most cases this means that codons combine information from a number of different amino acid positions. The effect of the substitution error therefore was to smudge out the amino acid distribution over a number of different codons, mainly the ones which encoded the amino acid, but also "nearby" codons, which are related by a single sequencing error. An upshot of this is that stop codons do have some small probability associated with them, but this probability is greater when the homology positions are more likely to emit amino acids which are a single base substitution away. For example, strong trytophan emitting positions (codon TGG) have a relatively large chance of matching TAG and TGA stop codons, compared to other positions. Default parameters for substitution error was one error in ten thousand base pairs, the quoted accuracy for genome sequencing projects.

3.5.3 Insertion or Deletion errors

GeneWise's approach to handling insertion or deletion errors in the sequencing process is deliberately naive. This is because handling the insertion/deletion errors correctly is difficult. When one considers the theory outlined in 3.3, it would be attractive to consider sequecing error to be the action of another machine, which substitutes, inserts or deletes bases of the DNA sequence before it is used in the gene prediction process. This sadly breaks the rule that the intermediate $\mathcal{B}$ sequence is only emitted as single bases. The sequencing error machine would emit bases (the intermediate sequence) as single bases, but the gene prediction machine takes triplets or larger strings to form the amino acid sequence. This means that I cannot use the theory in providing the combined machine. However, I can use it to guide the design of some of the ad hoc solutions to it. The model merging theory suggests that the sequencing error transition should occur at every place where DNA sequence could be emitted.

GeneWise considers sequencing error to be a 1,2,4 or 5 base codon. The base composition of the deletion or insertion is ignored completely, which is a gross approximation. For example, if one observes a TTTT in a putative sequencing error, a strong to phenalanine emitting position (codon, TTT) is a far more likely position to emit this than Glycine (codons GGN). Ignoring the base composition was really to prevent excessive calculation. Deriving the potential probability for base deletions is relatively easy to do, as one is considering only one or two bases, and one can build up a look up table for each position of all combinations. However one cannot take this approach for 4 or 5 base pairs as the tables become too large. An alternative is to call a function which would on-the-fly calculate the probability of 4 or 5 base pair insertion of particular bases to a probability distribution of amino acids. However such a function would be called at every cell in the dynamic programming matrix, making it an extremely expensive solution.

3.5.4 Intron parameterisation

The gene prediction model is non standard - we have no nice parameterisation such as the HMMER HMMs which we have for the protein homology. Therefore we had to parameterise the entire model ourselves. Again, simplicity was our watch word, and we took the entirely simple way of collecting a data set of known genes, checking their accuracy and tabulating counts of particular features. These counts were

- splice site regions (to calculate position weight matrices, as above)
- codons in coding regions (to calculate coding bias)
- bases in introns (to calculate intron bias)
- length of introns (to calculate transition parameters in introns)

The only issue here was deciding on the parameterisation of poly-pyrimidine tract regions. These tracts cannot be defined biochemically. Rather than going for a full blown training technique, we simple chose regions which by eye looked like the poly-pyrimidine tract. As it happens, the model with the poly-pyrimidine tract is not the most common model used, so this parameterisation was not important in the final analysis.

3.6 Path scoring

Having chosen and parameterised the probabilistic model, one might have thought that all the difficulties were over. Indeed, I expected that the Viterbi path through this combined model would provide highly accurate gene prediction using the protein homology to drive the process. Sadly there were a number of problems to confront first.

3.6.1 Flanking Regions

One issue I did not account for at first were the flanking regions of genomic DNA outside of a homology region that we were interested in predicting. These regions could of course contain genes, either as part of the gene we wished to predict which was outside the region of homology, or entirely different genes. These genes might not have any similarity to the protein homology model we were using, but they do have gene features which would score well against the gene prediction portion of the GeneWise model. These additional fragments often caused mispredictions at the end of the homology region. In particular if the homology region ended near an intron, as introns have a very broad length distribution, GeneWise could ignore the correct end of the intron and simply choose the best start or end splice site respectively for the start or end of the region within the rest of the sequence. For large clones, this gave rise to "comedy" gene predictions which spanned almost the entire length of the clone. They would start with a first very large intron leaving at the best 5' splice site in the clone, joining to the homology region and continuing to near the end of the homology region when another large intron that would jump to the best 3' splice site in the remainder of the DNA sequence. Unsurprisingly, the biologists who saw these early results were not impressed.

The solution is to somehow make the flanking regions less attractive to the homology model. The most principled way of doing this is to provide flanking models which represent the content of genomic DNA in the absence of the homology model. These regions would then score at least as highly as homology + gene prediction model in the absence of homology, and in general much better, causing the homology model to be kept in its correct place. The most natural way to build the flanking models is to duplicate the gene prediction model in the absence of the homology model. (A practical issue of this is that these regions are "special" states in the dynamite model). This is what was done in the GeneWise21:93 model.

A major drawback to this approach is that now every genomic DNA will score well against this model, even if the genomic DNA does not contain a gene with homology to a particular HMM or protein sequence. Thus using this model for the detection of the presence of homology requires the path information of where the most likely path went through the model, in particular if it crossed into the homology part of the larger, complete model of both flanking regions and homology model.

Ideally one also wants the likelihood score of just the homology portion. Although there are ways to computationally achieve this without requiring the calculation of the entire path, it is an additional computational step in an already expensive operation.

For GeneWise6:23 we provided the reverse solution, by toning down the gene prediction parts of the model so that any potential benefit of producing a erroneous intron would be more than outweighed by the additional penalties for mis-aligning a homology region. As GeneWise6:23 does not have a polypyrimidine tract model, a considerable gene prediction signal is removed. In addition no intronic bias (where the base composition of the intron is different to the intergenic DNA sequence) was provided. The only remaining gene signals were the actual splice sites, and in tests, these were not sufficient to cause this error in practical use.

3.6.2 Coding region scoring

Further analysis of the gene prediction errors from GeneWise showed a common error in mispredicting splice sites internally even with reasonable homology across this region of the gene. Further analysis of this phenomenon showed that in these regions, the homology information was reasonably consistent with a number of splice sites but the gene prediction information weighed heavily towards a particular site. In nearly every case, if one had taken the best homology predicted splice site one would have made the correct decision. (One issue to note here is that it is perfectly possible that we are being misled by alternative splicing - i.e that there are a number of correct splice sites in this region, and that the "disagreement" between the two models indicates alternative splicing. It is hard to assess this as clean datasets indicating every possible alternative splice transcript of a set of genes have not been developed yet.)

Again the problem here seems to be that the gene prediction information is misleading the protein homology information. One of the largest effects that provides this misleading information is the gene model of the coding regions: rare codons are heavily penalised and hence the most likely path in ambiguous regions of the homology matching are dominated by the fact that certain amino acids are better represented in random DNA sequence.

A very empirical solution to this problem is to score the coding regions as if they were scored as protein, in particular to compare the probability of the codon with the probability of the codon occurring in a random protein. The logic behind this is that one wants to find the best homology path considering a random model of a protein coding gene: we call this a *synchronous* null model, as one imagines a gene prediction model for the null model which is used in sync with the combined model, entering and leaving coding regions at the same places as the combined model.

The effect of using this *synchronous* model is to change the best scoring path. Empirical evidence shows that this scoring technique does produce better results.

3.7 Using the GeneWise algorithm

How one uses the GeneWise algorithm is not trivial - GeneWise is not going to provide a complete "answer" for the genomic analysis of a region. Indeed if the homology region does not extend across the entire region, which is the most common case, GeneWise will not even provide a complete gene prediction. In defense of GeneWise, when the prediction is provided, it is highly accurate (3.9).

Using GeneWise in a practical manner is going to require the integration of GeneWise with other tools. Providing an easy route for this integration is the role of the software which surrounds the GeneWise algorithm. GeneWise is the figure piece of the Wise2 package, a software package which contains most of the algorithms which I have developped. The Wise2 package provides access to the GeneWise algorithm in two principle ways

- as command line programs, to be used in the UNIX paradigm of reading and creating files, potentially these files being pipes into or from other programs.
- as a callable API (Application Programming Interface) accessible in either C or Perl.

Both systems provide the results of GeneWise in standard formats, such as EMBL feature table format, AceDB interchange files and GFF (Gene Feature Finding) format. The appendix C has a copy of the Wise2 package user documentation, which lists fully all the options available in the package, and has good discussions about how to integrate GeneWise into a genomic analysis package.

```
                    Alignment 1 Score 32.55 (Bits)

RRM               1 LFVGNLPPDVTEEDLKDLFS                       KFGPIV
                    L V+N+P+   + DL+++F+                       +FG+I
                    LHVSNIPFRFRDPDLRQMFG                       QFGKIL
HS41P2        11790 ccgtaactctcgcgcccatgGTAAGTC  Intron 1   CAGctgaac
                    tatcatctgtgacatgattg<0-----[11850:15370]-0>atgatt
                    gtcttttcccgctccgggtg                       gtcaca

RRM              27 SIKIVRDIIEKPKETGKSK                       GFAFVEF
                    +++I+ +          SK                       GF+FV+F
                    DVEIIFN-------ERGSK                       GFGFVTF
HS41P2        15389 gggaata       gcgtaGTAAGCT  Intron 2   CAGgtgtgat
                    atattta       aggca<0-----[15425:25099]-0>gtgttct
                    taaactt       atctg                       acgcatc

RRM              53 ESEEDAEKALEALNGKELGGRKLRV
                    E++ DA++A E+L+G++++GRK++V
                    ENSADADRAREKLHGTVVEGRKIEV
HS41P2        25121 gaaggggagagatcgaggggcaagg
                    aagcacagcgaatagcttaggatat
                    gttttacgcggaacccgagctacgg
```

Figure 3.8: An example of a alignment from GeneWise. This output format is designed for human readability, there are other formats designed to be parsed. The alignment of the RRM profile HMM against the human sequence HS41P2 is shown running left to right in a block. At each position the most probable amino acid in the HMM match emission is shown matching to the predicted amino acid from the DNA sequence. The codon which provides that amino acid is shown below, running from top to bottom. Introns are shown as breaks in the alignment. The splice site bases are shown in full, but the central portion of the intron is omitted for clarity.

3.8 Example of using GeneWise

A good example of using GeneWise in a real life situation is with the clone dJ41P2, a clone on chromosone 22. Preliminary halfwise analysis (see Chapter 6) which uses genewise showed the presence of a gene on the reverse strand with a RNA Recognition Motif domain internal to it, shown in figure 3.8. Further analysis showed that it was a member of the SR protein family[10]. A small multiple mutliple alignment of this family was made, and from this a profile-HMM of the larger gene family, which includes more of the surrounding sequence of the domain specific for this family. Rerunning GeneWise with the extended HMM enlarges the gene prediction.

3.9 Evaluation of GeneWise

An ongoing evaluation of GeneWise was necessary to assess how well it works, and how different parts of the algorithm contribute to the final result. Assessing gene prediction algorithms is notoriously tricky, principly because gathering good datasets together is painful. In addition, GeneWise needs a source of homology information as well as the DNA sequence: the assessment should also quantify the success of the algorithm at differing levels of similarity.

A dataset to fulfill these criteria was generated in the following manner. I took each Pfam family and found a human protein which both had a genomic sequence and was spliced. The human protein had to be referenced in SWISSPROT, which indicates a certain amount of manual curation. Using this protein sequence, I then selected five more protein sequences, one from each of the following similarity bands: 98-90%, 90-75%, 75%-50%, 50%-40%, 40%-30% identity, based on the Pfam alignment. In addition we could also use the Pfam HMM as a source of homology information.

One common source of error in GeneWise is misplacing the relative positions of protein insertions relative to introns. A protein insertion in the homology model which is close to an intron will be hard for GeneWise to find.

There are a number of criteria we would like to assess GeneWise on:

- the accuracy of GeneWise - when GeneWise claims that a base pair is part of a gene, how often is this so;
- the coverage of the gene, both in absolute terms and, in addition, in terms of how far we expect the homology to cover;

What is reported for each similarity band are the following figures:

- the total number of base pairs predicted as coding regions,
- the accuracy of these coding region predictions,
- the number of bases which should have been predicted as coding, and this represented as a proportion of the total base pairs predicted,
- the coverage of the correct coding region vs the entire gene.

	Total	Acc	Missing	Prop. Missing	Coverage
90	20059	99.93	40	0.001	98
75	18258	99.58	209	0.01	90
50	14101	99.91	437	0.03	70
40	12972	99.86	388	0.03	64
30	6334	97.46	601	0.09	30
HMM	6775	99.72	505	0.07	34

Table 3.1: Table showing performance of the standard 6:23 algorithm

	Total	Acc	Missing	Prop. Missing	Coverage
90	20059	99.91	31	0.001	98
75	18258	99.72	193	0.01	92
50	14101	99.31	435	0.02	72
40	12972	99.01	375	0.03	64
30	6334	97.02	530	0.09	34
HMM	6775	99.91	503	0.07	36

Table 3.2: Table showing performance of the standard 21:93 algorithm

	Total	Acc	Missing	Prop. Missing	Coverage
90	18056	99.92	26	0.001	98
75	16837	99.48	237	0.01	92
50	13241	99.87	271	0.02	73
40	12585	98.78	281	0.02	69
30	6827	96.41	355	0.05	36
HMM	7069	97.89	590	0.08	38

Table 3.3: Table showing performance of the 6:23 algorithm with Viterbi scoring

	Total	Acc	Missing	Prop. Missing	Coverage
90	20083	99.81	40	0.00	98
75	18320	99.46	170	0.01	91
50	14891	99.27	455	0.03	74
40	12966	99.68	418	0.03	64
30	6157	97.32	625	0.10	20
HMM	6814	99.15	505	0.07	34

Table 3.4: Table showing performance of the 4:21 algorithm with standard parameters

There are a number of points to draw from these results. Firstly GeneWise is extremely accurate. This should be expected: after all GeneWise is using similarity information to drive the gene prediction process, and there is a huge amount of information in the similarity data. In addition this accuracy is maintained down to really very low similarity measures: even at 30% identity, one is still getting 98% accurate coding sequence prediction.

As similarity decreases two things are lost: firstly the coverage of the gene drops to about one half of what is achievable from 90% similar sequences. Secondly the amount of missing coding sequence, i.ėregions predicted as introns but actually coding sequence grows dramatically, increasing by a factor of ten between from 75% identical sequences to 30% identical sequences. This means that at 30% identity, the exons are shorter by about 15%.

The differences between the different algorithms are marginal. There is slight difference in the balance between CDS prediction by 21:93 (Table 3.9) compared to the 6:23 method (Table 3.9). Indeed, even the 4:21 method which has a very limited gene model has almost identical accuracy levels to the 6:23 model. These results emphasise that it is the quality of the protein homology model which is driving the accuracy of the gene prediction, not the gene prediction machinery.

3.10 Other evaluations of GeneWise

There have been two other critical evaluations of GeneWise by independent researchers. These evaluations by other parties have provided very useful feedback,in particular in default configurations and in more real life situations.

3.10.1 Guigo and Agarwal

Roderic Guigo and Pankaj Agarwal evaluated GeneWise [41] alongside two other programs: Procrustes [35] and Genscan [17]. GeneWise came out with a positive write up, with the one drawback of the computational expense of the program. The essential figure gave GeneWise an accuracy of 99% when used with proteins selected from a BLAST output of less than 1e-20, very much in line with the results presented in this chapter. They considered programs both in the small DNA sequence case and in artificial, large genomic DNA sequence where they embedded known genes into random DNA sequences. The figures for the protein comparison programs,

Similarity:	strong			moderate		
Program	Sn	Sp	CC	Sn	Sp	CC
GenScan	0.91	0.66	0.77	0.91	0.61	0.74
GeneWise	0.99	0.99	0.99	0.68	0.98	0.81
Procrustes	0.92	0.96	0.94	0.66	0.79	0.75

Table 3.5: Table summarising results from Guigo and Agarwal on GeneWise's comparisons to other algorithms. Strong similarity is a protein BLAST hit of under 10^{-50} p-value. Moderate similarity is a protein BLAST hit of under 10^{-6} p-value

GeneWise and Procrustes, remained the same. A summary of one of the tables in the paper is reproduced here (with the authors' permission).

The principle figure they use to assess gene predictions is the correlation coefficient, which is a combination of the sensitivity (coverage) and the selectivity (accuracy) of the method. Because GeneWise does not build gene predictions outside of the homology portion of the match, the sensitivity drops considerably, as it does in my analysis. As the primary statistic they quote is the correlation coefficient, performance does not look ideal. However, examination of the tables (summarised in 3.5) shows that the accuracy measurement tallies well with the results presented in this chapter. If one considers that GeneWise's role is to produce highly accurate, but partial, gene predictions which should then be used by further analysis, then it is doing its job admirably.

3.10.2 The Drosophila annotation experiment

Martin Reese and colleagues used the 2.9 megabase region around the ADH region of Drosophila for an assessment of gene prediction tools, called GASP1. A number of *ab initio* prediction methods and some combined homology/prediction methods were assessed. I used here the Halfwise method with will be described later (in Chapter 5), but for the purposes of the comparison, it was basically GeneWise using HMMs from Pfam. GeneWise scored surprisingly poorly, with an accuracy of 82% and a sensitivity of 12%. The low sensitivity can be explained on the basis that Pfam HMMs were used as the source of homology information, and not protein sequences. The poor accuracy is more worrying.

Closer examination of the data revealed a number of reasons why GeneWise scored with such a low accuracy. Firstly I did not remove retroviral proteins from the Pfam database, nor did I use masked DNA sequence. This was deliberate as I wanted to see the behaviour of the search when potential retroviral proteins could be matched. Although I provided sets both with and without retroviral hits to the assessors, only the set with retroviral hits were considered. As retroviral genes were not considered to be real genes, unsurprisingly, this caused a large loss in sensitivity. With the hits removed, GeneWise's sensitivity on a base pair level was raised to 91%.

There were still a number of GeneWise predictions which did match up to any of the "standard" datasets, either the experimentally verified genes (std1) or the broader, well annotated set (std3). These gene predictions generally had high bits score, which indicates that there was a strong hit to homologous proteins. These look like real genes to me, and I have not been able to ascertain whether they were real genes but missannotated, or known pseudogenes.

3.11 Discussion of GeneWise

GeneWise's performance should be critically measured on two key criteria. Firstly, how accurate is it in objective tests? The results presented in this chapter, along with the two independent assessments show that GeneWise is very accurate. This is not surprising given the added homology information it brings to the problem, but it is good to see it independently verified. This level of accuracy, being around 99% means that GeneWise predictions can be used in an automatic fashion by annotation systems. The second criteria is how useful is it in the practice? Here the response to GeneWise has been less forthcoming. GeneWise's main strength is producing accurate gene predictions when there is a close homolog: these were in many cases the sort of genes which human annotators found easy and rewarding to annotate. However, with the ever increasing data flow, people have grown accustomed to using GeneWise's prediction as a starting point to eventually annotating genes: in some people's experience this has reduced the time to annotate from 2 hours per gene to under 15 minutes.

GeneWise is now used in many locations world wide. The principle users are the Sanger Centre and Celera, a private company in the US, and it has some devoted followers at other places. A common complaint is that GeneWise is very CPU expensive. In many ways I am unapologetic about the cost: GeneWise is focused on getting the correct answer, not on speed. In my view spending an additional hour's worth of CPU time is a considerably better use of resources than an additional hour of a human annotator. However the CPU expense of GeneWise does limit its use in large scale projects. Chapter 5 details how I coped with this problem at the Sanger Centre.

I believe that the key to GeneWise was the analysis of the problem from theoretical principles. I did not use complicated machine learning techniques, nor invented any radically new algorithms - instead I simply looked at the problem as being the application of two probabilistic Finite State Machines. That view of the problem suggested a solution based on the principled merging of the two Finite State Machines. As we have a coherent theory for how the algorithm works we can easily add or remove parts of the model.

Chapter 4

Pfam: a protein family database

4.1 Introduction

Protein sequences are the most conserved pieces of genetic information in genomes. For example, the protein sequence of actin has remained conserved at around 70% identity over the last half billion years of eukaryotic evolution. Some more striking examples are the clear similarities of eukaryotic and prokaryotic ribosomal proteins, indicating a single evolutionary ancestor which one assumes was present before the split of prokaryote and eukaryote lineages.

Protein sequences are synthesised as linear polypeptide chains, which then fold up to provide the protein with a three dimensional shape (see section 1.1 in the Introduction). However, the folding and sometimes the function of the protein sequence can often be split into continuous regions: each region able to fold by and large independently. These regions are commonly called *domains*. During evolution different domains have been combined to produce proteins of unique functions from a large set of components. This reuse of domains as building blocks for proteins is clearly favoured by evolution compared to reinvention: the majority of proteins have been found to be multidomain [82]. Over a large timescale, one often finds that the only similarities between distantly related proteins are within one or two domains rather than along entire protein sequences.

Of course, this being biology, not everything conforms to this scheme. Firstly there are quite a few proteins which *are* conserved as complete units during evolution. For example, Ribosomal protein L11 is conserved in eubacteria (RL11_ECOLI), Archea (RL11_METJA) and Eukaryotes (RL12_HUMAN) over its entire length.

These can be considered to be single domain proteins where the domain is never reused in a different context, so conceptually such cases fit without problem into a domain orientated approach to proteins. Another class to consider includes cases where the region conserved during evolution is not contiguous or colinear. An example of these are Swaposins, in which the homologous portions of the sequence seemed to have been circularly permutated during evolution [67]. More commonly one finds one complete domain inserted into the middle of the sequence of another domain. Although there are a number of cases in which discontinuous domains are present, the majority of domains are continuous in sequence.

A consequence of domain evolution is that if one focuses on defining and characterising domains, one can provide a very informative decoding of the protein sequence, even if one has never seen a protein with that particular collection of domains before. Some of the most impressive predictions using sequence analysis have come from careful domain analysis of proteins. For example, Bhattacharya and colleagues found the DNA cytosine methylase in humans by starting from the identification of a methyl binding domain [8]. Domain analysis has also been critical for effective structure prediction. In the structure prediction exercise, CASP2, target T2 was predicted correctly by only one group [61]; one of the main differences of their analysis compared to other groups was that they considered the domain structure of the protein (there were two copies of one domain) before attempting to predict its structure.

4.1.1 protein profile HMMs of domains

One question in domain analysis is how to represent a protein domain in a computationally tractable manner. After some 15 years of work in the field a rough consensus of how to do this sensibly is using *profile* analysis [38, 6, 83] which, when describing these profiles as probabilistic models gave rise to *profile hidden Markov models* (profile HMMs) [50, 28]. Profile HMMs attempt to model protein domains using the standard HMM paradigm: the sequence of amino acids are the observations and the HMM produces these observations. To greatly simplify the learning of the form of the HMM to produce the amino acids, the architecture of the HMM is constrained to be a repeated set of 3 states, joined in a left to right manner, as shown in figure 3.4 in the previous chapter. This architecture of HMMs was inspired by the established technique of *profile* analysis which provided a number of successes in analysing protein sequences, albeit using an ad hoc derivation of the parameters. The big improvement which profile HMMs provided was a mathematical framework of how to parameterise and use these profiles.

The adopting of profile HMMs as a sensible way of modeling protein domains has become widespread in bioinformatics.

4.2 The Pfam database

The acceptance of a standard way of building models of domains has allowed researchers to make databases of these models. These databases can be used as a resource for analysing proteins in a tractable manner, domain by domain. The number of proteins domains is considered to be perhaps 4,000 in total, which is not a large number compared to the around 100,000 protein genes in a single mammalian genome. This size is small enough that a curated database can cover most domains found in biology. Three researchers, Erik Sonnhammer, Sean Eddy and Richard Durbin set about to produce such a database using the profile HMM software written by Sean Eddy, HMMER. The database is called Pfam (standing for Protein FAMily) [78] and started as system based around flat files kept in a Unix directory structure, with a number of shell scripts to maintain consistency and use the database.

The basic structure of Pfam has four files for each family. The *seed* file is a protein multiple alignment of a representative set of protein members. These members are chosen for their ability to produce a good profile HMM. The *HMM* file is the profile-HMM, built using HMMER, from the seed alignment. The *full* file is a complete alignment of all members of the domain that can be found with the profile-HMM in the complete protein database, aligned with the aid of the HMM. Finally the *desc* file contains the documentation for the family.

4.2.1 Requirements for Pfam as database

My primary role in Pfam ended up being to provide a stable environment to expand the collection and work with Pfam. As a database there were a number of challenges to solve

- Physical data integrity. Pfam had traditionally relied on the users of the database to know precisely how to manipulate the different parts. This had already caused a number of problems in people inadvertently deleting, renaming or losing data. One particular problem was that the implicit building rules of Pfam, that the HMM is built from the seed, and the full alignment from the HMM, could be ignored. This allowed a number of families to be stored incorrectly.

- Sequence data integrity. Pfam is a database of multiple alignments, built on top of a sequence database. Pfam needed to be kept synchronised with the underlying sequence database. The particular challenge to this was when the underlying sequence database changed all the sequences in Pfam could potentially change, having knock-on effects on the seed alignment.

- Internal data integrity. One early rule adopted in Pfam is that no two domains could overlap on any sequence. This rule had to be implemented by a program that checked that new data conformed to this rule, and there had to be a way of ensuring this check occurred every time.

- Interactions with external programs. Unlike more standard databases, the main ways of querying and using Pfam is not via ad-hoc structured queries. Instead the HMMs are used to search protein databases, and multiple alignments are viewed and edited in specialised programs. This means that the database has to interact well with these external programs, whose input is file based.

- Productivity of the database curators. As Pfam expands, a greater load is placed on the database curators to maintain the database. By providing systems to automate some tasks considerable work can be removed from the curators, allowing curators to be more productive.

4.3 The Pfam Database Management System

The first task with Pfam was to provide stronger physical integrity of the data. The database was very file based, strongly suggesting a file orientated solution. I took the pragmatic decision to store the data as a series of RCS (Revision Source

Utility	Description
pfco	Check out a family to local area
pfci	Check in a family back to the database
pfupdate	Check out a family without a lock
pfabort	Abort a checked out family
pfinfo	Provide information on a family
pfnew	Check in a new family to the database
pfmove	Rename a pfam family
pfadduser	Give another user access to the Pfam database
pfhelp	Help on pfam database management system

Table 4.1: Pfam Database Management System utilities

Control) files, indexed by using a Unix directory system and a lightweight Berkeley style database. The users would not access this system directly - instead a collection of utilities provided access to the database (Table 4.1).

The *pfco* utility allows users to "checkout" a family from the database: this places a lock (using the inbuilt locking facilities of RCS) on this family, preventing anyone else from editing the family. The user is then be free to make changes to the family locally. Once he or she is happy with the family, the changes can be committed back into the database using the *pfci* utility. Alternatively, if the user wants to discard the edits, the *pfabort* utility removes the lock on the family without committing any changes.

To make a new family, the *pfnew* utility creates a new database index, assigns a new accession number and checks in the family.

A number of other supporting utilities are provided. *pfupdate* provides the ability to get a local copy of a family without placing a lock on it. *pfinfo* provides information about a family directly to the terminal, and, in a different mode, will provide overview information on the entire database, indicating which families had been locked by which users. *pfmove* gives the ability to rename a family (I had foolishly decided to make the primary index on the human readable name, and not the pure identifier of the family, its accession number. This was a design error and why *pfmove* had to be written to cope with renaming a family). *pfadduser* gives a way to add a new user with read/write permissions to the Pfam database.

4.3.1 Triggers on data entry

One clear role for the database management system was to force data integrity. This is achieved by running a number of programs which ensures the integrity of the data before it can be checked back into the database (as the database management system is file based, there are not many inherent integrity checks in the way the database is structured). There are three main integrity checks before allowing data in the database.

- The file modification times have to satisfy the rule that the seed file must be older than the HMM file which in turn must be older than the full file. This rule ensures that the family has been built correctly.
- As the database system is file based, rules about data integrity can not be directly represented in the database. To ensure that the internal structure is correct, a utility called *pqc-format* was written which checked the integrity of the desc, seed and full files.
- The overlap rule has to be satisfied. A utility called *pqc-overlap* was written which compares the current family to all families in the database, flagging overlaps. This utility writes a file containing all the overlaps in the family directory. This way the pfci or pfnew scripts can check that both the overlap check has been run and that it has reported no overlaps.

4.4 Productivity tools

When Pfam started, most of the aspects of building families required the user to remember a string of connected commands. This both took a significant amount of the curator's time and also created a large hindrance for new people coming joining the Pfam group. To reduce these problems, I developed a number of productivity tools for the Pfam database.

The main tool is automake2 (table 4.2), which removes the mechanical process of scanning the HMMER output file to find the sequence regions that have been matched, building a fasta file of these regions and building then an alignment using the Fasta file and the HMM. In addition, automake2 writes some statistics about

tools	Description
automake2	Takes HMMER output, optional manual cutoffs and builds new FULL file, placing certain information in the DESC file
HMMResults modules	Objects representing the output of a HMMER search, used in automake2 and the Pfam Web site

Table 4.2: Pfam Productivity tools

the distribution of scores near the cutoff into the Desc file. These were previously maintained by hand and prone to error.

A key piece of functionality for automake2 is the ability to scan HMMER output files. The HMMER search reports a bit score and expectation value for each sequence, and a bit score and expectation value for each domain: by careful manipulation of the cutoff at both the sequence and the domain level better performance in finding some domains, in particular repeats, could be managed. The HMMResults set of modules encapsulates the parsing of HMMER files and filtering with this two level threshold on the results for furthering processing.

4.5 Underlying Sequence database update

All sequences in alignments must be valid sequences in the underlying sequence database of Pfam. About every 6 months we change the underlying sequence database to a newer version of a non redundant database set from the SWISSPROT + SPTREMBL databases. When this occurs, all the seed alignments need to be checked to ensure that the sequences that they contain are consistent with the new database. When sequences change these changes have to be reflected in the alignment and manually verified. Once the seed alignments are consistent with the database, HMMs need to be rebuilt and searched against the new database, from which the new full files can be made.

The main problem in all this is that the underlying sequences can change. A sequence can change in a number of different ways

- The sequence can be radically different from previously given, often because the annotators have recognised a frameshift in the DNA sequence or a gene misprediction.

- The sequence may have had a large addition or deletion of a region outside the region of interest for the domain. Although the domain sequence in the seed alignnment therefore is still in the sequence, the coordinates have changed.

- The sequence can have some small changes due to the recognition of sequencing error in the DNA sequence or merging of a number of polymorphic forms of the sequence (only one "reference" sequence is provided for polymorphic sequences in SWISSPROT).

- The sequence could have been merged with another logically equivalent sequence, often given rise to small changes, but more importantly, making it harder to track in the database as one has to use secondary accession numbers to track merges.

- The sequence can be deleted all together.

These changes in sequence cause major problems in the remapping of the seed alignments. As some seed alignments have changed and so the HMMs are potentially different, one would like to know how many full matches have been lost during the migration: however, due to changes in sequence, and, in particular sequence merging, tracking sequences across two versions of the database is far from trivial.

Just before I took over managing the Pfam database a single researcher spent around an entire month doing the move of Pfam to a new version of the sequence database. Since the number of families in Pfam has increased by a factor of four; it was clear that such labour intensive work could not be maintained. To alleviate this, I provided an automated system to take the brunt of the work in the database move.

After some less successful attempts, I developed a robust system in which the automatic moving process is good enough to run unattended. Researchers can then briefly examine by eye the families which have underlying sequence changes, and in most cases accept the final result: only a few cases require manual intervention.

The key part of the system was the *ReSeed* module which takes a multiple alignment on one underlying database and attempts to map it to a new database. For each sequence it uses the following rules:

- find the new sequence, potentially using secondary accession numbers;

- if the old and new sequences are identical, accept;
- if it is possible to find the subsequence used in the alignment in the new sequence, even if at a different sequence position, simply change the start/end points in the new sequence;
- otherwise, align the old and new sequences using the Smith-Waterman algorithm. Using the coordinates of the old sequence, find the corresponding coordinates in the new sequence;
- Finally align all the changed fragments back to the unchanged portion of the alignment using the old HMM as a guide.

This procedure is robust to changes in the new sequence, including insertion and deletions of residues. The tolerance towards indels is due to using the Smith-Waterman algorithm to map old domain coordinates to new coordinates rather than fixed length matching, and following that by using the HMM to guide the alignment of the new fragment onto the new alignment.

4.6 Middleware Layer

As the sophistication of Pfam grew a number of scripts were written which accessed the Pfam database for a variety of reasons, for example, calculating statistics or automating the insertion of new structure links into the database. This growing body of scripts became a large burden to write and maintain: many of the scripts had similar code segments, for example to loop over the entire database. More worryingly, when we made changes to the layout of the database it was not clear which scripts would be affected, and how. This cloud of interactions with the Pfam database was effectively stifling any potential improvements of the actual database as changes would invariably have knock on effects on a number of scripts.

To counter this I built a middleware layer that shielded the actual database access from the script writer. Instead, access to the database was provided as a single object which hid all the implementation details of the database. The methods of the object allowed the retrieval of database entries as objects: these entries had further methods to retrieve parts of the entry, for example the seed and full alignment.

Figure 4.1 shows the main objects and how they interrelate. The key feature of the middleware design was sensible separation of the Pfam specific information from more general annotation information and the actual multiple alignments. Access to the HMM itself was not provided by the middleware layer.

The middleware layer is written in Perl and based on Bioperl. Bioperl [1] is a open software project to produce stable objects in Perl for bioinformatics. For the Pfam middleware layer I reused a number of objects from Bioperl, in particular the Sequence object (Bio::Seq) and the alignment object (Bio::SimpleAlign).

The middleware layer has been used to support scripts which interacted with the database and also the web site. A fully functional web site is an important part of a public bioinformatics resource: the web site at the Sanger centre was written with an eye for extendibility by maintainers over the years, with the middleware layer providing insulation of the html rendering code from the database access.

4.7 Some Example families

As well as being the database administrator for Pfam, in charge of maintaining the integrity of the database as a system, I maintained a number of actual families in the database. These ranged from simple families which I have had a long term interest in through to difficult families which required many manipulations to maintain their biological correctness. I list my contributions to the data curation in Appendix D.

4.7.1 The RNA Recognition Motif

I have had a long association with RNA binding proteins, as they were the original reason I became interested in computational biology. One of the principle RNA binding domains is the RNA Recognition Motif [10]. This domain of around 80 amino acids is one of the most abundant in biology, and certainly the most abundant domain involved in RNA processing. A number of crystal structures are known for this domain ([63, 65, 42, 26]) and there is a large series of biochemical experiments into the action of certain family members (for example [73, 21]).

The seed alignment of the RRM family is shown in figure 4.2. It is quite large for a Pfam seed alignment, indicating the large number of protein sequences required to

[1] Bioperl is coordinated from http://bio.perl.org. In Janurary 2000 I became the official coordinator for bioperl.

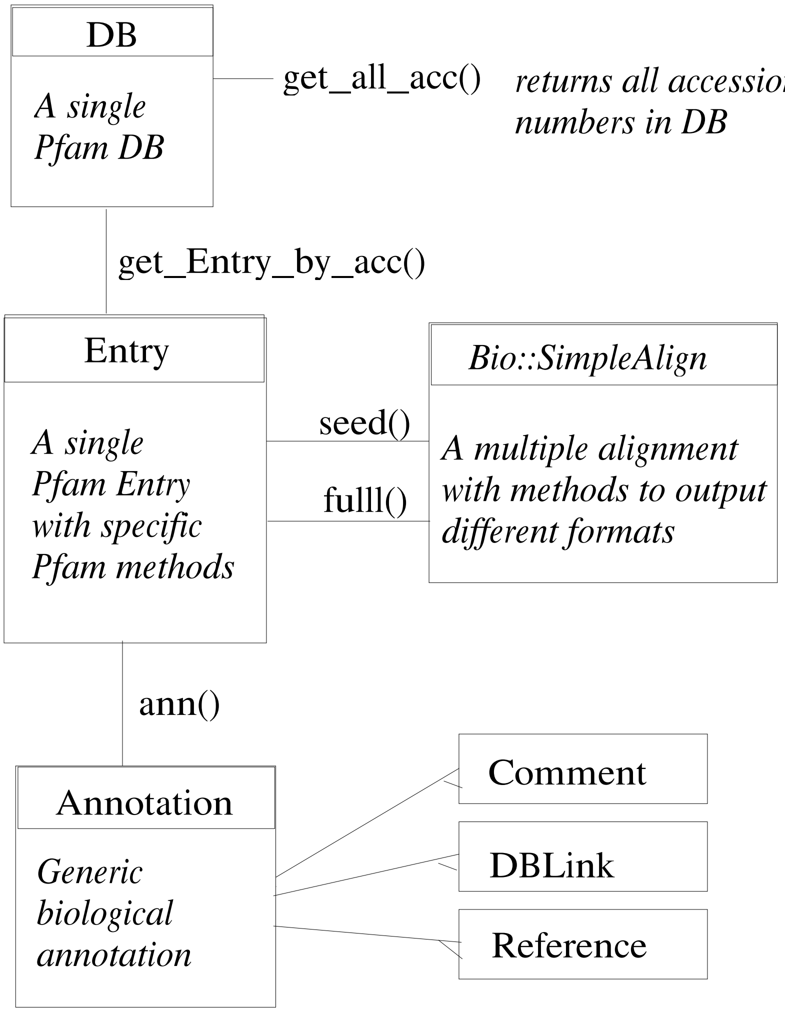


Figure 4.1: A diagram showing the essential objects in the Pfam middleware. Each rectangle indicates an object. The lines between rectangles indicates a method which produces the object. Bio::SimpleAlign comes from the bioperl project. Most methods are not shown for simplicity.

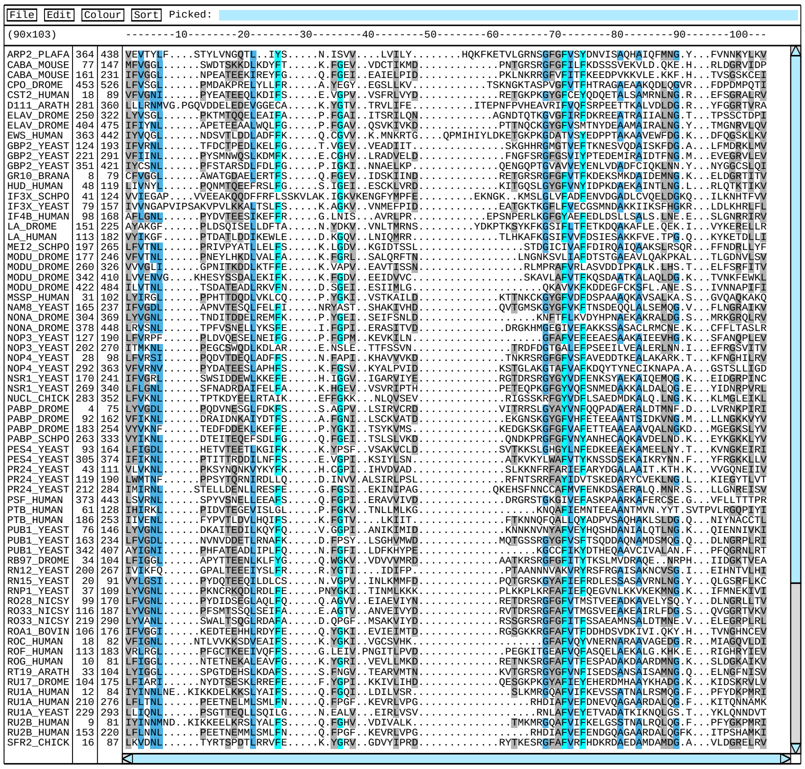


Figure 4.2: Figure showing the RRM seed alignment. The protein sequences are on the left hand side of the figure. In the alignment columns, conserved columns are shaded.

make an effective profile HMM. Included in the alignment are a number of "atypical" RRMs which help to educate the HMM about some of the more distant members of the domain. This is a particular problem for this family, as some of the most conserved features of the domain from typical families are the surface facing aromatic residues involved in RNA binding [65]. Despite their strong conservation in typical families, there are a number of RRMs which do not have these aromatic residues but have all the structural core residues and bind RNA, for example hnRNPL. Without the inclusion of these atypical RRMs many structually correct domains would be missed by the HMM.

4.7.2 Protein complexes

I have taken an interest in a number of protein complexes. One problem of these common complexes is that often researchers dismiss them as uninteresting and do not spend as much time checking their annotation in genome projects. A common mistake is to misname proteins due to naming differences between bacterial and eukaryotic proteins. When the top hit is a bacterial sequence, they might give the bacterial name to a eukaryotic protein, although the naming convention for the complex in eukaryotes is different. For example, in the $F_O F_1$ ATP synthase complex, the D subunit in bacteria corresponds to the Oligomycin sensitive subunit in Eukaryotes (also called the O subunit), and there are Delta and Epsilon subunits in Eukaryotes which corresponds to the Epsilon subunit in bacteria. Just to add more confusion, in Eukaryotes there is a D subunit for Vacuolar ATPases which are related to the $F_O F_1$ ATPases in some aspects but not others. This means that there are three possible types of "ATP synthase D" subunits, all different - the Bacterial D subunit, the Eukaryotic Delta subunit and the Vacuolar D subunit. The confusion here is inherent in the naming of the protein complexes, and there is very little that can be done now to solve this problem in the general literature. Pfam on the other hand can provide an accurate single annotation automatically: there are three separate Pfam families for these "D" subunit types and the description makes it clear what the different naming conventions mean in different organisms.

I have worked on the ATP synthase, the Signal Recognition Particle and a number of the electron transport components. The seed alignment construction and HMM building in these families is trivial: in every case the separation of signal from noise is considerable. However, tracking down the precise annotation for these families is a harder task, as it requires finding, retrieving and reading the relevant papers. The reward is knowing that the problems in naming and classification can be eliminated as resources like Pfam become more widely used.

4.8 Discussion

Pfam has grown over 300% while I was managing the database. During that time we have had an additional four people work with the database at different times, all of whom both had to familarise themselves with the tools and had the chance of potentially breaking the database by issuing inappropriate commands. The database management system I wrote both scaled well to the increase of data and also protected users from corrupting the data. We have not lost track of a single family in Pfam, and when there was some major corruption or mistake in the database (thankfully rare) we were able to recover the data easily by direct manipulation of the RCS files. Scalability and robustness are two key features of a database, and I achieved them in this tailored database management system.

The database system relies on RCS files rather than a relational DBMS or a object based DBMS, which would have been a more standard choice. One reason why these solutions is not as appropriate is that some of the main query mechanisms and views of the database are through specialised software to compare HMMs to proteins or view multiple alignments. If we had implemented the database around a more traditional system there would have been additional software to write to view our data. In addition RCS provides for free a full history for each family, giving the security of a reasonably easy way of backtracking changes.

The middleware layer insulates the access of the database by other programs from knowing too much about how the database is actually stored on disk. This allows the database management system to change without breaking scripts that interact with it, and makes writing scripts that interact with the database simpler. Evaluating how useful the middleware layer has been is difficult, as we do not have an alternative. In some cases the middleware layer has saved time and other cases it has prevented quick development as new features have to be implemented in the middleware before they can be actually used.

Pfam has become a member database of the *Interpro* consortium [2]. Interpro is a pooling of the documentation resources of Pfam, Prints and Prosite along with the maintenance of a consistent set of matches of these databases along with other motif/domain databases on SWISSPROT + SPTREMBL. This pooling of the documentation effort is important as documentation is one of the most time consuming parts of these databases, mainly because it is almost impossible to automate. The Interpro project also encourages the different databases to play to their strengths rather than competing to provide all aspects of motif databases.

Pfam provides a way to maintain a large number of multiple alignments robustly: an important part of that strategy is the profile HMM. These profile HMMs are also useful in their own right to be used by other analysis tools. The GeneWise algorithms described in this thesis (Chapter 3) can work with profile-HMM libraries such as those generated by Pfam, as well as alignments to protein sequences. The next chapter details how, using GeneWise, Pfam can be applied to eukaryotic genomes at a large scale.

The curation of a database such as Pfam is a satisfying task. I have been able to keep an interest in a number of protein families such as the RRM and EGF domains over the last 3 years, and I have maintained these families with considerable attention. I believe that the underlying technology and basis of Pfam can be applied across all protein sequences, which would place Pfam in a central role to provide consistent, accurate and automatic identification for all protein families.

[2] The interpro web pages are at http://www.ebi.ac.uk/interpro/

Chapter 5

Genome

5.1 Introduction

Pfam represents a large portion of protein families, covering between 35-50% of proteins in a genome [7]. However, applying Pfam to eukaryotic genomes is dependent on accurate gene prediction to form good protein sequences. GeneWise (Chapter 3) eliminates the need for good gene predictions as it can compare a protein profile HMM directly to the genomic sequence. By combining GeneWise and Pfam it should be possible to find and classify around half the genes in a genome in an entirely automated manner.

Such a comparison has two benefits. Firstly it has the potential of finding genes which other programs have missed, or, when the gene has already been predicted, improving their gene prediction. The accuracy of GeneWise (section 3.9) should provide high quality predictions. Secondly it allows the investigation of the domain content of genomes without worrying about accurate gene prediction (nor for that matter accurate sequencing as GeneWise is tolerant to sequencing errors).

The main complication to applying GeneWise on a genome wide scale is the CPU expense of the method. It takes about a second to compare one protein hidden Markov model against 1,000 bases of genomic sequence. A genome of 100 MBase (the size of *C.elegans*) against 2,000 protein profile HMMs in Pfam will take a total of around 2,500 days of CPU time. Even with large compute farms this is a number of months, and not practical with current resources.

In this chapter I show how I overcame the problem of CPU limitations by using a heuristic speed up. Then gene predictions by Pfam and GeneWise of the *C.elegans* genome (100 Mbases) are compared to the curated annotations on that genome. This provides an estimate of the number of new genes which were missed by the *C.elegans* annotation project and an estimate of the number of domains missed by GeneWise due to more sensitive protein comparisons. Finally I performed comparisons to a portion of the human genome, the finished Chromosome 22 sequence (33.4 Mbases), to assess how useful this analysis is against the human genome.

5.2 Halfwise

The prohibitive computational cost of comparing all 2000 profile HMMs to genomic DNA was a problem I overcame by employing a pre-filter on which profile-HMMs are used in the GeneWise comparison. This pre-filter selects only a small number of the HMMs to do a full GeneWise search with. The combined action of the pre-filter and GeneWise is encapsulated in a single Perl script, called *halfwise*. By selecting only a small fraction of the potential profile HMMs to be searched against each genomic sequence fragment, I could cut the number of profile HMMs down to between 10 to 20 HMMs from 2000, meaning the total running time of GeneWise would be about 1% of the exhaustive search, and, as long as the pre-filter was quick enough, the total time reduced by up to two orders of magnitude.

The pre-filter I used was to search the DNA sequence with BLASTX against a protein database which had been designed to capture nearly all the signal present in the Pfam profile HMMs. This database was constructed by taking the Pfam full alignments and making them non redundant at the 75% identity level, removing close homologs. To ensure that this database did capture the vast majority of the Pfam information, I constructed this database with varying levels of identity and then searched a random selection of 2 proteins from each Pfam family against the database (4,000 proteins in all), scoring whether these families hit or missed any of the Pfam families present. The results are shown in table 5.1.

75% identity seemed to be the lowest cutoff which still provides effective coverage over Pfam with almost no loss (0.1% on the test sample).

% id	Size of DB	Number of missing assignments
90	129987	0
75	96837	4
50	54858	122

Table 5.1: A table showing how the effectiveness of lowering the cutoff for non redundancy effects the number of family hits retrievable by the BLAST method

5.3 Worm Genome

The first genome sequence to be examined was the worm genome. To compare the entire worm genome against Pfam, each sequencing unit, generally a cosmid or a YAC fragment, was run through halfwise. This does mean that there will be problems with genes which span more than one sequencing unit, but as domains are generally small compared to genes, this is not such a great problem. All 100 megabases of the *C.elegans* genome took around 2 and half weeks of off-peak computer time at the Sanger Centre.

Halfwise produced 12,426 predictions of protein domains across 907 different families, on average around thirteen predictions per domain, though this distribution is very skewed. The predictions had 30,190 exons, on average two and a half exons per domain. 4,991 predictions were of only one exon, with the remaining 7,434 having introns.

Domain	Number	Comment
Collagen repeats	707	20 copies of the GXY motif in the HMM
GLTT repeats	406	12 copies of GLTT repeats. Specific to *C.elegans*
Protein kinase	401	The common serine/theronine/tyrosine kinase
EGF like domains	330	Extracellular domain. See also Laminin EGF
Zinc finger C4	300	DNA binding domain
F-box associated domain	264	Expanded domain in worm
F-box domain	219	Ubiquitination receptor domain
MATH domain	196	Signal processing domain
Worm chemoreceptor (7tm5)	194	Worm specific 7tm subfamily
WD40 repeats	190	Beta propeller repeat. Around 7 copies per domain
Transposase	152	Found in mobile genetic elements
LDL receptor A domain	150	Extracellular domain
Zinc finger C2H2 domain	149	DNA binding domain, often multiple domains to a protein
Ankyrin repeats	144	Repeats found in structual proteins
Worm chemoreceptor (7tm4)	131	Worm specific 7tm subfamily

Ig super family domains	131	Ubiquitous metozoan domain
Protamine	129	Sperm histone protein
Ion channel	124	Includes voltage gated channels
Fibronectin type III domains	123	Extracellular domain
RNA recognition motif	123	Involved in many aspects of RNA metabolism
Trypsin inhibitor	120	Protease inhibitors common in signal transduction
Reverse Transcriptase	111	Mobile genetic element domain
Major Sperm Protein domain	110	Found in structual proteins, including sperm tails
Cytochrome p450	107	Involved in metabolism. Important medical implications
DUF40	106	Domain of unknown function, found mainly in *C.elegans*
7tm_1	104	Classical 7tm receptors
laminin EGF	101	sub type of EGF domains
BTB domain	92	Protein interaction domain
EB domain	91	Unknown function, expanded in *C. elegans.*
ABC transporters	91	Involved in metabolite transport
Homeobox	84	DNA binding domain
Lectin C-type	83	C-type lectin domain
Trypanosome mucins	82	Mucin like coat proteins
Spectrin	80	Spectrin repeats, found in structual proteins

Table 5.2: A table showing Pfam families with more than 80 domain hits in *C.elegans* via halfwise analysis. The table shows the raw counts for each Pfam HMM. This can be confusing for some families: for example, the Collagen HMM represents a single repeat which is found many times in a collagen "domain". The numbers here are therefore an overestimate of the number of collagen domains occurring in *C.elegans*. For other cases, such as the 7 transmembrane family, Pfam uses a number of HMMs to cover one domain, meaning that those cases should be summed. See the main text for more details.

Table 5.2 shows the domains with 80 or more hits in the *C.elegans* genome. Interpreting this table requires some knowledge about the different Pfam families. Some Pfam families represent a repeated motif which occurs many times sequentially. These families generally do not conform to the usual idea of a protein "domain", in the sense of a single, independently folding unit, but instead form non-globular extended protein structures. Collagen is a clear example of this, and the GLTT repeat is also likely to be another such non-globular repeat. The numbers therefore in this table are overestimates: ideally one would like to quote number of continuous stretches of these repeats, but as GeneWise cannot distinguish between gaps between HMMs matches due to protein sequence and gaps due to intergenic regions, this is hard to deduce accurately. Other families are unfairly represented in table 5.2 as they are split over a number of Pfam families. There are two examples of this in the table. The 7 transmembrane family is split over 3 different Pfam families (7tm_1, 7tm_4 and 7tm_5). The 7tm_4 and 7tm_5 families were created specifically to model the divergent chemoreceptors found in *C.elegans* which are clearly 7 transmembrane receptors, but cannot be found with the conventional 7tm_1 HMM. The total number considering all three HMMs is 429, making them the second largest single family in *C.elegans* when one ignores the structural repeats of collagen and GLTT repeats. The largest family is the EGF like domains which are split into the EGF domain and the laminin EGF domain in Pfam, making a grand total of 431 domains.

It is annoying that the numbers in 5.2 cannot be used to give precise answers to questions such as "what is the largest worm family" and "which families have had the most expansion". However it is hard to represent the complexity of the biological domains to provide an automatic answer to such queries. As any conclusion one draws is qualitative in any case, the lack of automatic reliable numbers for each domain is not such an issue.

The figures in table 5.2 broadly agree with other genome wide studies of *C.elegans* protein content [33, 7]. In particular the expansion of the Zinc Finger C4 domain [24] and the 7 transmembrane domain [71] proteins have both been previously noted. It is interesting to wonder whether selective family expansion is a general feature to eukaryotic evolution, or whether the worm is unique in its indulgence of specific family expansions. Only the increase in the number of complete or mostly complete eukaryotic genomes will give us this answer; tools such as Pfam and GeneWise will greatly help obtaining these answers when the data is available.

5.4 Comparison to curated worm genes

There is an active database group which maintains curated gene structures on the worm genome. This database contains a set of gene structures which usually were the result of an *ab initio* gene prediction program, genefinder, followed by manual examination and curation by a skilled annotator. In addition the program est2genome [60] was used to match cDNA and EST sequences back to the genomic DNA. These

	Introns	Confirmed introns
Curated Genes	43200	12696 (26%)
Halfwise Genes	8491	1560 (18%)
Intersection	7325	1547 (21%)

Table 5.3: A table showing the number of different introns (both confirmed and not confirmed) in three different datasets: Curated genes in *C.elegans*, Halfwise predictions and the intersection of the two datasets. The curated genes and confirmed introns are only from the Cambridge portion of the database, as this information is not available from the St Louis portion of the data

matches produced a number of "confirmed intron" features, which are introns which have an EST or cDNA crossing the splice boundaries.

Comparison of the worm gene dataset with the halfwise predictions provides a useful feedback to both the worm annotators about a number of potential mispredictions and myself for the accuracy of GeneWise. There are three possible explanations for discrepancies between the halfwise predictions and the curated gene set. Firstly halfwise could have mispredicted the gene. Secondly the annotators could have mispredicted the gene. Thirdly there could be another genomic feature, such as a pseudogene or transposon which explains the halfwise prediction but does not indicate the presence of another real gene. As we expect nearly all the confirmed introns to be correct, marking where confirmed introns disagree with halfwise predictions will provides a good measure of halfwise's accuracy.

As table 5.3 shows, the vast majority of halfwise predictions are consistent with the curated genes, with 86% of halfwise introns being identically placed in the curated set and the halfwise predictions. The numbers of those which are confirmed by EST is shown in the second column. It is interesting to note that a higher percentage of introns are confirmed in the curated genes (26%) than the intersection of the curated genes and halfwise (21%). This indicates that the halfwise predictions are occurring in areas of the genes which generally have less EST confirmation. As the EST sequences are derived from 5' and 3' sequence reads on attempted full length cDNA clones, this is not surprising, as halfwise predictions are more likely to be in the central coding portion and ESTs at the ends.

	Number	Percentage
Consistent	15916	89%
Modifications	445	2.5%
Pseudogenes	375	2%
Repeats	78	0.5%
Novel	1095	6%
Total	17909	

Table 5.4: A table showing the distribution of exons predicted by halfwise between annotated exons, different exons from annotations, but inside an annotated gene (Modifications), exons inside pseudogenes, exons inside the repeats and finally exons not overlapping with any of the other genomic features.

5.4.1 Indication of annotation errors

The halfwise analysis produced a number of gene predictions in regions where there were no curated genes at all. In some cases these overlapped with known pseudogenes: as GeneWise is only matching a domain, and is also tolerant to sequencing error, detecting domains in pseudogenes is a reasonably common occurrence.

Table 5.4 shows the distribution of halfwise exons between annotated genes, annotated pseudogenes and other genomic regions. Exons inside annotated genes are split into two classes - identical to annotated exons (89%) and those contained by annotated exons but implying a different gene structure (2.5%). It is interesting to note that looking at the halfwise statistics on the exon level gives a more favourable view of GeneWise, rather than comparisons at an intron level, which is presented in the next section. This is because there are a large number of GeneWise predictions of single domains inside large exons (for example, a set of Zinc Finger repeats), which improve the annotated exon count. The other 8.5% of exons lie outside of annotated exons. The predominant number here (6% of the 8.5%) do not lie in pseudogenes or repeats, indicating possible new genes, or extensions of existing genes.

5.4.2 GeneWise accuracy in the worm

The figures above do not allow us to assess how many of the halfwise predictions which are not consistent with the curated gene predictions are due to annotation error or halfwise misprediction. Table 5.5 gives an indication of this.

	Total confirmed introns	Consistent (within 1%)	Inconsistent
Halfwise	1883	1560 (83%)	323 (17%)

Table 5.5: A table showing the accuracy of GeneWise when used in genomic scans against the worm. The total confirmed introns are confirmed introns which overlap in some manner with the halfwise gene predictions. The consistent column provides the number of those which are also predicted by halfwise. The final column indicates the inconsistent introns.

The 323 confirmed introns which disagree with halfwise are mainly errors by halfwise. There are a number of cases which are due to other effects, for example a gene nested in the intron of another gene, so there are two overlapping correct introns. Even allowing for a generous 10% of the 323 introns as being due to such effects, the results are not close to the sort of accuracy figures quoted in section 3.9. When examined visually, many of the cases were due to short introns, both in the halfwise predictions and the worm genes. The worm has a large number of short introns of around 50 base pairs and unfortunately it is hard for GeneWise to distinguish between short introns and protein insertions relative to the profile-HMM. The parameterisation which provides effective prediction of short introns also allows the erroneous prediction of an intron to avoid making a long protein insertion.

Assessment of the accuracy of intron placement by GeneWise is fair indication of its effectiveness for finding annotation errors in a curated database. However, the accuracy of GeneWise at a base pair or exon level is more relevant to its effectiveness for gene identification purposes, where the aim is to find exons with a high confidence to design primers. The results in the previous section indicate that GeneWise's exon accuracy is far better than its intron accuracy; however it is less easy to classify exons as "confirmed" in a large scale automatic manner.

5.4.3 Comparison to protein Pfam analysis

The worm database holds the results of a search of the curated worm proteins sequences against Pfam. Comparisons of the Pfam matches on the protein set against the GeneWise matches provides an indication about the loss of sensitivity to finding domains by using Halfwise on genomic DNA compared to the protein analysis.

This analysis is complicated by several factors. Firstly, as presented above there are a number of halfwise predictions which do not overlap with known genes. As we know where these lie, it is easy enough to account for these. The second problem is that the protein analysis produces domains in the peptide coordinates. It is difficult to compare these matches with the halfwise matches on the genomic sequence in a true domain by domain fashion.

The protein database has 14,710 Pfam domains, which is not that larger than the 12,426 predictions from halfwise analysis. Of those predictions we know that only 304 do not overlap in any way with predicted genes, and the number of exons in disagreement with curated genes are small 5.4 enough for us to ignore. This gives a loss of sensitivity of around 20%, which I consider to be quite good. One technical reason for why there is this loss of sensitivity is that GeneWise can only predict genes when the log-likelihood ratio against an genomic DNA sequence model is above zero. As some Pfam families have cutoffs of negative log likelihood scores, which is possible using HMMER software, these domains are impossible for GeneWise to predict.

5.5 Human Chromosome 22

Recently Chromosome 22 was finished, with only 10 small gaps interrupting the otherwise continuous 33.4 megabases of human DNA. This contains the largest single contig of DNA sequence known (23Mb) and the largest amount of contiguous human DNA. To appreciate the scale of the human genome, these 33.4 MB represent approximately 1% of the entire genome.

Halfwise analysis was performed against the individual clones sequences that make up the contiguous sequence, as with *C.elegans*. In total 565 domains were predicted from 196 different Pfam families with 2,468 exons. The exon numbers are broadly in agreement with the worm figures, with somewhat more exons per domain (around 4.5 compared to worm's 2.5) and but the concentration average number of domains per family is considerably lower (2.5 domains per family compared to 13 in the worm).

There are no real surprises in the domain composition of human: as expected, there is a number of extracellular mosaic domains are found, such as the EGF-like domains and the Ig superfamily. The large number of zinc finger C2H2 domains is also expected. There are a number of gene clusters on chromosome 22, such as

Domain	Number	Comment
Ig super family domains	53	Ubiquitous metozoan domain
Zinc finger C2H2	36	DNA binding domain
EGF like domains	23	Extracellular modular domain
Protein kinase domain	16	The common serine/theronine/tyrosine kinase
Kelch domain	11	Beta propeller repeat. Around 7 copies per domain
Mitochondrial Carrier protein	10	Solute transport
Crystallin	10	Lens protein
SH3 domains	9	Signalling domain
Cytochrome p450	9	Involved in metabolism.
G-Protein effector (RhoGAP)	9	Signalling protein
Cadherin	8	Cell adhesion domains
PI3_PI4_kinase	7	Signalling domain
Myosin head	6	Structual protein
Clathrin repeats	6	Vesicle transport
Sushi (SCR)	6	Extracellular domain
Filament	6	Structual domain
Trypsin	6	Peptidase domain
7 transmembrane	6	Classical 7tm proteins
PH domain	5	Signalling protein
Zinc Finger C3HC4 (Ring)	5	Protein protein interactions
SH2	5	Signalling modular domain
Bromodomain	5	Acetyl-lysine (chromatin) binding domain
Actin	5	Structual domain
Glutathione S-transferases	5	
BTB	5	Protein interaction domain
protamine_P1	5	Sperm histone

Table 5.6: A table showing Pfam families on Chromosome 22 with more than 5 domains via halfwise analysis.

	Total Exons	Consistent	Modifications	Pseudogene	Novel
Halfwise	1213	371 (30%)	122 (10%)	144 (12%)	576 (47%)

Table 5.7: A table showing the distribution of exons predicted by halfwise between annotated exons, exons overlapping an annotations, but inside an annotated gene (Modifications), exons inside pseudogenes and novel exons. The reference dataset were the "GD" tagged gene structures from 22ace.

the immunoglobulin λ cluster, a glutathione S transferase cluster and a β crystallin cluster, which explain the high number of these domains.

5.5.1 Comparison to curated genes

As with the worm, chromosome 22 has an active database curation group, and these results can be compared to their predictions. This comparison is hampered by a number of practical issues. Firstly it is harder for technical reasons to get consistent coordinate systems of both the curated genes and the halfwise predictions, as the clone assembly of chromosome 22 is still somewhat in flux. I was only able to compare a little under half of the detected exons (1213 of 2468). Secondly I was unable to get a set of confirmed intron features in the coordinate system I needed. Finally the curators have been using halfwise predictions to guide their experiment design somewhat, and so the comparison is not entirely unbiased.

The comparison of halfwise predictions to curated genes is shown in table 5.7. The most striking figure in this list are the number of entirely novel exons (576, making 47% of the exons used in this comparison). Examination of some of these exons show them to be in or around genes which are being actively curated: however, the correct data structures have not been entered to allow them to be automatically compared to the halfwise predictions. This inability to perform sensible comparisons is annoying but commonplace in this field. For exons which do overlap with annotated genes, around a quarter are at disagreement with the annotations. As for the novel domains, a large number of these disagreements are due to misrepresentations of the genes in the database, so it is hard to really assess the effectiveness of the method.

The chromosome 22 annotation also provided PFAM analysis on the protein genes: a total of 240 predicted proteins have PFAM hits from 164 Pfam families.

These figures indicate that the halfwise analysis may have found more protein genes in chromosome 22. Because of the difficultly in interpreting the gene locations, the precise mismatch cannot be indicated. For example, in the λ locus, all the Ig domains comprising the V regions were not annotated as peptides, but the halfwise analysis predicts them.

5.6 Coding density of Human vs *C.elegans*

Using the two comparisons of these large streches of genomic data with halfwise provides a way to compare coding density in the two genomes. There are a considerable number of potential caveats to this analysis. Is Pfam biased towards domains in one of the genomes? Do the genomes have different overall domain composition in terms of the number of domains per gene? Are there more pseudogenes in one genome? Assuming that none of these biases are significant, this analysis suggests that there is one Pfam domain per 8 kilobases in worm compared to one Pfam domain every 59 kilobases in human, about seven times the coding density. This number is about what is expected. Extrapolating to the whole human genome this would suggest around 90,000 genes, which is consistent with other estimates.

5.7 Discussion

The two comparisons presented in this chapter show that by using Halfwise, Pfam can be compared directly to genomic sequence using the GeneWise algorithm. As GeneWise provides accurate gene prediction and is robust towards errors, this means that Halfwise can provide automatic annotation of genomic sequence. One area for improvement is the detection of small introns in inveterbate sequences such as the worm. It is likely that to be able to detect these, new approaches in parameterisation are required which try to provide the best gene predictions and not the highest likelihood of the parse. The "Class HMM" and conditional maximum likelihood methods of Anders Krogh provide a good framework to derive this parameterisation [51].

The domain composition of the different organisms by the Halfwise analysis is as expected from the domain composition found for their proteomes. In general, there are a number of common domains in each organism, some which seem to be specific to an organism (such as the F-Box in worm) and some which are ubiquitous in eukaryotes, such as protein kinases. It is likely that as more metazoan genomes are analysed, trends in how and why domains are expanded in specific lineages will become clearer - currently, with only two metazoan genomes with large, unbiased genome coverage (human and worm) available it is hard to draw conclusions about domain evolution in metozoans.

The systematic halfwise analysis drew attention to a number of possible annotation errors or oversights in the two genomes. Although the accuracy of the GeneWise predictions in the worm is not enough to confidently expect the halfwise prediction to be correct, possible missannotations can be quite easily spotted, as most of the errors by halfwise are small intron predictions. This analysis provides the annotators with a systematic way of finding curation oversights. As the human genome does not suffer from this small intron prediction problem, the accuracy in human is suspected to be far higher. The same approach of using Pfam and GeneWise is being used to annotate the Drosophila genome and genomes such as Aspergillus.

The combination of quality Pfam profile-HMMs with the GeneWise algorithm in this halfwise analysis provides a useful tool for genome analysis. As the number of sequenced genomes increase, providing more automated tools to provide a better first pass annotation of genomes is crucial. This work shows that such tools are possible to develop, use and apply on a large scale.

Chapter 6

Conclusion

The stream of DNA sequence data emerging from the sequencing projects has caused a revolution in some aspects of biology. One major challenge is interpreting the data on a large scale to connect it with existing hypothesis driven biological research. This thesis presents a number of tools to make that possible.

The focus of the thesis has been the GeneWise algorithm to compare protein information directly to genomic DNA sequence, with the gene prediction occurring at the same time as the protein comparison. To write this algorithm I took the standpoint that both problems have already been well solved: the task was to combine them together so that the intermedaite predicted gene sequence was implicit rather than being required up front. This was done by working out how to merge in general models and applying the solution to this specific case. This solution relies on the established methods being well described as probabilistic finite state machines. I view the ability to conceive of algorithms such as GeneWise as a vindication for using formal probabilistic models to describe protein and DNA sequences: without the previous work on these models it would have been difficult to design this algorithm from scratch.

GeneWise is accurate and can be used on a large scale. There are still many challenges in really using an algorithm such as GeneWise effectively: one clear point is that the maximum likelihood path is unlikely to be the path one wants to report. However, it is with algorithms such as GeneWise that accurate gene prediction can be achieved.

GeneWise's effectiveness is greatly increased when it is used with Pfam. Pfam provides a set of profile hidden Markov models which cover around two thirds of known proteins. I have helped the Pfam database considerably by writing a tailor made database management system and supporting software for Pfam. As well as designing the software I have been an active curator on the database and involved in many aspects of its day to day running, such as the Web Site.

Bioinformatics is focused on providing useful computational techniques for biological data. One aspect of bioinformatics which often gets overlooked is that for a software solution to really have a wide impact it must be able to work in a large scale manner robustly, and must be maintainable often by people who did not write the original program. Chapters 2 and 4 presented some of my work in this area of software engineering in bioinformatics. The language Dynamite freed me from worrying about the implementation of dynamic programming solutions to PFSMs. The Pfam database provided the source of profile-HMMs for the GeneWise analysis. I have been involved in other work in the area of software engineering in bioinformatics here: it is hard to point to specific results from this work and harder still to write about it from a research perspective. However I am a committed promoter of stronger software engineering techniques in the field and I believe this aspect of tool development is crucial to the success of bioinformatics as a whole.

Many of the aspects of this work are being drawn together in my new work as part of the Ensembl project. Ensembl is a project to develop a stable software system to provide automatic analysis of metozoan genomes, starting with the human genome in 2000. I have learned a considerable amount from my implementation of the Pfam database management system and applied the knowledge to this new work. In addition, algorithms such as GeneWise will be invaluable for the automatic accurate prediction of genes. We can also expect to have to write a number of new algorithms, for example for effective genomic to genomic comparisons. In cases which require dynamic programming, Dynamite will greatly shorten our time to produce solutions.

The amount of data which we will expect to process over the next ten years is vast. I believe that deriving information from this data is only achievable by taking principled theoretical approaches and applying them to real life data in a robust engineering environment.

Bibliography

[1] P. Agarwal and D. J. States. Comparative accuracy of methods for protein sequence similarity search. *Bioinformatics*, 14:40–47, 1998.

[2] S. F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[4] T. L. Bailey and M. Gribskov. The megaprior heuristic for discovering protein sequence patterns. In States et al. [80], pages 15–24.

[5] G. J. Barton and M. J. Sternberg. Flexible protein sequence patterns. a sensitive method to detect weak structural similarities. *Journal of Molecular Biology*, 212:389–402, 1990.

[6] G. J. Barton and M. J. E. Sternberg. A strategy for the rapid multiple alignment of protein sequences. *Journal of Molecular Biology*, 198:327–337, 1987.

[7] A. Bateman, E. Birney, R. Durbin, S. R. Eddy, K. L. Howe, and E. L. L. Sonnhammer. The pfam protein families database. *Nucleic Acids Research*, 28:263–266, 2000.

[8] S. K. Bhattacharya, S. Ramchandani, N. Cervoni, and M. Szyf. A mammalian protein with specific demethylase activity for mCpG DNA. *Nature*, 397:579–583, 1999.

[9] E. Birney and R. Durbin. Dynamite: a flexible code generating language for dynamic programming methods used in sequence comparison. In Gaasterland et al. [34], pages 56–64.

[10] E Birney, S. Kumar, and A. R. Krainer. Analysis of the RNA-recognition motif and RS and RGG domains: conservation in metazoan pre-mRNA splicing factors. *Nucleic Acids Research*, pages 5803–5816, 1993.

[11] M. J. Bishop and E. A. Thompson. Maximum likelihood alignment of DNA sequences. *Journal of Molecular Biology*, 190:159–165, 1986.

[12] M. Borodovsky and J. McIninch. GENMARK: parallel gene recognition for both DNA strands. *Computers and Chemistry*, 17(2):123–133, 1993.

[13] J. U. Bowie, R. Luthy, and D. Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253:164–170, 1991.

[14] S. Brunak, J. Engelbrecht, and S. Knudsen. Prediction of human mRNA donor and acceptor sites from the DNA sequence. *Journal of Molecular Biology*, 220(1):49–65, 1991.

[15] R. Bruskiewich. Personal communication.

[16] P. Bucher and K. Hofmann. A sequence similarity search algorithm based on a probabilistic interpretation of an alignment scoring system. In States et al. [80], pages 44–51.

[17] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

[18] C. B. Burge, R. A. Padgett, and P. A. Sharp. Evolutionary fates and origins of u12-type introns. *Molecular Cell*, 2:773–785, 1998.

[19] M. Burset and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, 34:353–367, 1996.

[20] K. C. Burtis. The regulation of sex determination and sexually dimorphic differentiation in drosophila. *Current Opinion in Cell Biology*, 5:1006–1014, 1993.

[21] J. F. Caceres, T. Misteli, G. R. Screaton, D. L. Spector, and A. R. Krainer. Role of the modular domains of sr proteins in subnuclear localization and alternative splicing specificity. *Journal of Computational Biology*, pages 225–238, 1997.

[22] G. A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.

[23] G. A. Churchill and B. Lazareva. Bayesian restoration of a hidden markov chain with applications to dna sequencing. *Journal of Computational Biology*, pages 261–277, 1999.

[24] N. D. Clarke and J. M. Berg. Zinc fingers in caenorhabditis elegans: finding families and probing pathways. *Science*, pages 2018–2022, 1998.

[25] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematics*, 1:269–271, 1959.

[26] J. Ding, M. K. Hayashi, Y. Zhang, L. Manche, A. R. Krainer, and R. M. Xu. Crystal structure of the two-RRM domain of hnRNP A1 (UP1) complexed with single-stranded telomeric DNA. *Genes and Development*, pages 1102–1115, 1999.

[27] S. Dong and D. B. Searls. Gene structure prediction by linguistic methods. *Genomics*, 23:540–551, 1994.

[28] S. R. Eddy. Hidden Markov models. *Current Opinion in Structural Biology*, 6:361–365, 1996.

[29] S. R. Eddy. profile-hidden Markov models. *Bioinformatics*, 14:755–763, 1998.

[30] S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22:2079–2088, 1994.

[31] S. R. Eddy, G. J. Mitchison, and R. Durbin. Maximum discrimination hidden Markov models of sequence consensus. *Journal of Computational Biology*, pages 9–23, 1995.

[32] S. R. Eddy, G. J. Mitchison, and R. Durbin. Maximum discrimination hidden Markov models of sequence consensus. *Journal of Computational Biology*, 2(1):9–23, 1995.

[33] The C. elegans sequencing consortium. Genome sequence of the nematode C. elegans: A platform for investigating biology. *Science*, pages 2012–2018, 1998.

[34] T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia, editors. *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, 1997. AAAI Press.

[35] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences of the USA*, 93:9061–9066, 1996.

[36] N. Goldman, J. L. Thorne, and D. T. Jones. Using evolutionary trees in protein secondary structure prediction and other comparative sequence analyses. *Journal of Molecular Biology*, 263:196–208, 1996.

[37] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[38] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the USA*, 84:4355–4358, 1987.

[39] J. A. Grice, R. Hughey, and D. Speck. Reduced space sequence alignment. *Computer Applications in the Biosciences*, 13:45–53, 1997.

[40] W. N. Grundy, T. L. Bailey, C. P. Elkan, and M. E. Baker. Meta-meme: motif-based hidden markov models of protein families. *Computer Applications in the Biosciences*, pages 397–406, 1997.

[41] R. Guigo and P. Agarwal. manuscript in preparation.

[42] N. Handa, O. Nureki, K. Kurimoto, I. Kim, H. Sakamoto, Y. Shimura, Y. Muto Y, and S. Yokoyama. Structural basis for recognition of the *tra* mRNA precursor by the sex-lethal protein. *Nature*, 398:579–585, 1999.

[43] K. Hofmann, P. Bucher, L. Falquet, and A. Bairoch. The prosite database, its status in 1999. *Nucleic Acids Research*, 27:215–219, 1999.

[44] I. Holmes and R. Durbin. Dynamic programming alignment accuracy. *Journal of Computational Biology*, 5(3):493–504, 1998.

[45] D. S. Horowitz and A. R. Krainer. Mechanisms for selecting 5' splice sites in mammalian pre-mrna splicing. *Trends in Genetics*, 10:100–106, 1994.

[46] T. J. P. Hubbard. RMS/coverage graphs: A qualitative method for comparing three-dimensional protein structure predictions. *Proteins*, 3:15–21, 1999.

[47] D. T. Jones, W. R. Taylor, and J. M. Thornton. A mutation data matrix for transmembrane proteins. *FEBBs Letters*, 339:269–275, 1994.

[48] D. H. Kil and F. B. Shin. *Pattern recognition and prediction with applications to signal characterization.* AIP Press, 1996.

[49] A. M. Krecic and M. S. Swanson. hnrnp complexes: composition, structure, and function. *Current Opinion in Cell Biology*, 11:363–371, 1998.

[50] A. Krogh. Hidden Markov models for labeled sequences. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, pages 140–144, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[51] A. Krogh. Two methods for improving performance of a HMM and their application for gene finding. In Gaasterland et al. [34], pages 179–186.

[52] A. Krogh, I. S. Mian IS, and D. Haussler. A hidden markov model that finds genes in e. coli dna. *Nucleic Acids Research*, pages 4768–4778, 1994.

[53] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden Markov model for the recognition of human genes in DNA. In States et al. [80], pages 134–142.

[54] F. Lefebvre. An optimized parsing algorithm well suited to RNA folding. In Rawlings et al. [68], pages 222–230.

[55] P. Lio, J. L. Thorne, N. Goldman, and D. T. Jones. Passml: combining evolutionary inference and protein secondary structure prediction. *Bioinformatics*, 14:726–733, 1999.

[56] H. X. Liu, M. Zhang, and A. R. Krainer. Identification of functional exonic splicing enhancer motifs recognized by individual sr proteins. *Genes and Development*, 12:1998–2012, 1998.

[57] R. Luthy, A. D. McLachlan, and D. Eisenberg. Secondary structure-based profiles: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities. *Proteins*, 10(3):229–239, 1991.

[58] A. Mayeda and A. R. Krainer. Mammalian in vitro splicing assays. *Methods in Molecular Biology*, 118:315–321, 1999.

[59] R. Mott. Maximum likelihood estimation of the statistical distribution of Smith–Waterman local sequence similarity scores. *Bulletin of Mathematical Biology*, 54:59–75, 1992.

[60] R. Mott. EST_GENOME: a program to align spliced dna sequences to unspliced genomic dna. *Computer Applications in the Biosciences*, pages 477–478, 1997.

[61] A. G. Murzin and A. Bateman. Distant homology recognition using structual classification of proteins. *Proteins*, 1:105–112, 1997.

[62] E. W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

[63] K. Nagai, C. Oubridge, T. H. Jessen, J. Li, and P. R. Evans. Crystal structure of the RNA-binding domain of the U1 small nuclear ribonucleoprotein A. *Nature*, 348:515–520, 1990.

[64] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[65] C. Oubridge, N. Ito, P. R. Evans PR, C. H. Teo, and K. Nagai. Crystal structure at 1.92 å resolution of the RNA-binding domain of the U1A spliceosomal protein complexed with an RNA hairpin. *Nature*, 372:432–438, 1994.

[66] J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia. Sequence comparisons using multiple sequences detect three times

as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284:1201–1210, 1996.

[67] C. P. Ponting and R. B. Russell. Swaposins: circular permutations within genes encoding saposin homologues. *Trends in Biochemistry*, 20:179–180, 1995.

[68] C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors. *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, 1995. AAAI Press.

[69] S. K. Riis and A. Krogh. Hidden neural networks: a framework for HMM/NN hybrids. In *Proceedings of ICASSP '97*, pages 3233–3236, New York, 1997. IEEE.

[70] Eleanor Rivas and Sean Eddy. A dynamic programming algorithm for rna structure prediction including pseudoknots. *Journal of Molecular Biology*, 285:2053–2068, 1999.

[71] H. Robertson. Two large families of chemoreceptor genes in the nematodes caenorhabditis elegans and caenorhabditis briggsae reveal extensive gene duplication, diversification, movement, and intron loss. *Genome Research*, pages 449–463, 1998.

[72] Y. Sakakibara, M. Brown, R. Hughey, I. Saira Mian, Kimmen Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120, 1994.

[73] D. Scherly, W. Boelens, N. A. Dathan, W. J. van Venrooij, and I. W. Mattaj. Major determinants of the specificity of interaction between small nuclear ribonucleoproteins u1a and u2b" and their cognate rnas. *Nature*, 345:502–506, 1990.

[74] J. Schultz, R. R. Copley, T. Doerks, C. P. Ponting, and P. Bork. Smart: a web-based tool for the study of genetically mobile domains. *Nucleic Acids Research*, pages 231–234, 2000.

[75] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler. Dirichlet mixtures: a method for improved detection of weak

but significant protein sequence homology. *Computer Applications in the Biosciences*, 12(4):327–345, 1996.

[76] D. Slonim, L. Kruglyak, L. Stein, and E. Lander. Building human genome maps with radiation hybrids. *Journal of Computational Biology*, 4:487–504, 1997.

[77] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[78] E. L. L. Sonnhammer, S. R. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins*, 28:405–420, 1997.

[79] R. Staden. Measurements of the effects that coding for a protein has on a DNA sequence and their use for finding genes. *Nucleic Acids Research*, 12:551–567, 1984.

[80] D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. F. Smith, editors. *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, 1996. AAAI Press.

[81] R. Tacke and J. L. Manley. Determinants of sr protein specificity. *Current Opinion in Cell Biology*, 11:358–362, 1998.

[82] S. A. Teichmann, J. Park, and C. Chothia. Structural assignments to the mycoplasma genitalium proteins show extensive gene duplications and domain rearrangements. *Proceedings of the National Academy of Sciences of the USA*, 95:14658–14663, 1997.

[83] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *Computer Applications in the Biosciences*, 10:19–29, 1994.

[84] J. L. Thorne, N. Goldman, and D. T. Jones. Combining protein evolution and secondary structure. *Molecular Biology and Evolution*, 13:666–673, 1996.

[85] E. C. Uberbacher and R. J. Mural. Locating protein-coding regions in human dna sequences by a multiple sensor-neural network approach. *Proceedings of the National Academy of Sciences of the USA*, 88:11261–11265, 1991.

[86] A. A. Salamov V. V. Solovyev. The Gene-Finder computer tools for analysis of human and model organisms genome sequences. In Gaasterland et al. [34], pages 294–302.

[87] A. A. Salamov V. V. Solovyev and C. B. Lawrence. Identification of human gene structure using linear discriminant functions and dynamic programming. In Rawlings et al. [68], pages 367–375.

[88] J. Zhu, J. Liu, and C. Lawrence. Bayesian adaptive alignment and inference. In Gaasterland et al. [34], pages 358–368.

[89] J. Zhu, J. S. Liu, and C. E. Lawrence. Bayesian adaptive sequence alignment algorithms. *Bioinformatics*, 14:25–39, 1998.