MATERIALS AND METHODS

2.1 Introduction

In this chapter I explain the different strategies used in learning regulatory signals from DNA. Learning signals or motifs from nucleotide sequences has been quite a difficult task and various methods have been adopted so far. Prominent among them is the Hidden Markov Model (HMM) widely used in speech recognition, sequence alignment and gene prediction. A Hidden Markov Model is a directed graph of states connected by transition paths and throws emission and transition probabilities. Walking through these states and probabilities, HMM models the features in the DNA sequence. However such methods were found to be helpful only in cases where features in similar sequences were aligned with each other.

This requirement makes it difficult to learn transcription termination signals as some motifs downstream of the cleavage site, responsible for polymerase pause and release, are found over a wide range of distances from cleavage site (Dye and Proudfoot, 2001). Hence, here I used another method based on the sparse Bayesian principle that can accommodate the distance variation. The method is a probabilistic generalized linear model which scans for motifs that describe the given set of sequences and learns them by constructing a model. This model can be later used to classify sequences with and without transcription termination signals. Derivation of this model is based on the conditional probability of Bayes theorem given below –

$$P(model \mid data) = \frac{P(data \mid model)P(model)}{P(data)} \qquad (1)$$

where, *data* represents a DNA or cDNA sequence. *P(model|data)* is the posterior probability that gives the probability of a sequence derived from the *model*. The posterior probability depends on the probability of the *data* given the *model* and probabilities of the *model* and *data*.

As estimating the real probabilities of the *model* and *data* are difficult, various approaches have been adopted. One such approach depends on how best the Bayesian model can fit the sequence compared with the chosen null model. Learning the Bayesian models that best fit the given set of sequences with the regulatory motifs is possible with different types of trainers, like Relevance and Support Vector Machines. Relevance Vector Machine (RVM, Tipping, 2001a, b) is a Bayesian treatment of a Generalized Linear Model (GLM) of identical functional form to the Support Vector Machine (SVM, Mackay, 2003; Scholkopf *et al.*, 1999; Vapnik, 1995). However, RVMs has the advantage of emitting a probabilistic output unlike SVMs and using fewer kernel functions to classify the data. In this project, I used an implementation of RVM called *Eponine* (Down and Hubbard, 2004) to learn gene regulatory signals.

Initially I used Eponine to identify transcription termination motifs and then extended it to learn translation start, translation stop and splice sites. During this process I tweaked the default parameters of the trainer to suit the regulatory signals to be learnt. For example, the Gaussian distribution employed to accommodate the positional distribution of motifs in the sequences in learning the termination model was changed to a Delta distribution for splice site models as the splice signals show less positional variation in their occurrence.

The features predicted by these sequence models are then linked together using a dynamic programming based gene component assembler called GAZE. GAZE combines the features and predicts a gene structure in the sequence consistent with a supplied gene structure model (Howe *et al.*, 2002).

After investigating the use of sequence models for detecting transcription termination sites, I also tried secondary structure prediction algorithms with the objective of finding any stem-loop structures that might influence transcription termination. I used two basic algorithms developed by *Nussinov* (Nussinov, 1978) and *Zuker* (Zuker and Stiegler, 1981) for this purpose.

The *Nussinov algorithm* is very simplistic and is based on *base-pair maximisation* metrics. Whereas the *Zuker algorithm*, along with base pair metrics, uses a *free energy minimisation* technique based on experimentally determined energy parameters. The minimal free energy

criterion helps the selection of the best possible structures out of the ensemble of folds predicted.

None of these analyses would have been possible without the excellent databases in the public domain. Here, I have used human chromosome 22 and 20 data widely as these were the most accurately annotated chromosomes available at that time. The recently published annotations on chromosome 22 with experimental support formed an excellent source to derive training and test datasets (Collins *et al.*, 2003). Likewise, manually curated high quality annotation for chromosome 20 was extracted from the *VEGA* project (Ashurst, 2002). This project is an attempt to co-ordinate curated annotation process for the finished vertebrate genomes. Likewise, *ENSEMBL* is another excellent database that contains genome sequence data for organisms, automatically annotates it and serves the annotated sequence through the internet (Birney *et al.*, 2004). Various tools in ENSEMBL along with the supporting evidence for annotation derived from different sources helped me in this project. At other times I have used the *RefSeq* database (Pruitt and Maglott, 2001) and the *RIKEN* mouse cDNA collection (Kiyosawa *et al.*, 2003) as well .

In the remainder of this chapter, I explain the details of these algorithms and databases used in this project and conclude the chapter by briefly describing the two open source projects, *Bioperl* and *Biojava* that I used extensively from formatting sequences to building models.

2.2 Hidden markov models

Several methods have been attempted to model sequence signals around regulatory regions. They range from simple sequence composition bias to complex probabilistic machine learning methods like, Neural Networks (NN) and Hidden Markov Models (HMM).

HMM (Durbin *et al.*, 1998) is one of the common modelling systems employed to learn biological signals from DNA or protein sequences and forms the basis for many gene prediction tools. The use of HMMs involves two components – model architecture and its parameterization.

Figure 11 shows a schematic representation of a model architecture. This comprises a set of states, which might be a match state (circles labelled M), insert states (diamonds labelled

I) or delete states (squares labelled D). The states are connected by arrows that represent possible transitions between the states. A DNA sequence can be generated by moving through the model following the arrows. For instance, starting with the state M_0, which generates a nucleotide (AGCT), the next move might be to any of M_1, I_0 or D_2. M_1 or I_0 would generate a second nucleotide but D_2 would not. From these states, the model continues to the next state connected by arrows thus generating a state path and emitting a DNA sequence. Self transition (looping) is allowed for insert states and they are shown as arrows linking to themselves.

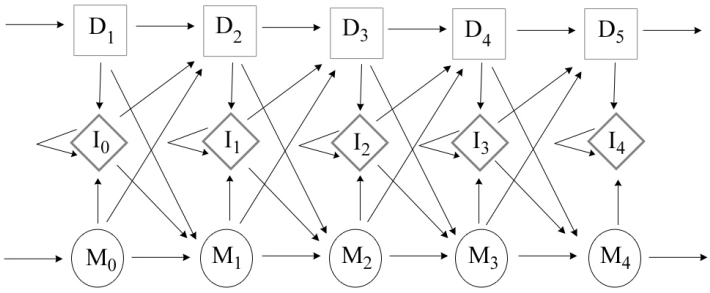

Figure 11. Schematic diagram showing a section of HMM architecture

After designing the architecture of HMM, transition and emission probabilities have to be assigned between and within states respectively. The probability parameters can be easily calculated by counting the number of times each particular transition and emission is used in the set of training sequences when all the state paths are known. However, in cases where the paths are unknown, an iterative method like, Baum-Welch algorithm, is used.

The name ‘Hidden Markov Model’ is used because the sequences are generated by a Markov process, which is defined as a process in which the probability of a particular state depends on the state immediately preceding it in a sequence. Since the state path of the model that generates the sequence is not observed the term ‘hidden’ is used.

I attempted to use HMMs to learn transcription termination signals responsible for RNA polymerase pause and release from the sequences of the end of the gene. These attempts suggest that, HMMs are not a good choice of machine learning technique for use on this problem because of two reasons. Firstly, the sequence motifs at the 3’-end of the gene responsible for termination appear only to be loosely defined, without a strong consensus

and thus are difficult to model with a simple HMM architecture. Secondly, the locations of these termination motifs are present at greatly varying positions from the cleavage sites and HMMs have difficulty in modelling such criteria. Although other methods have been developed to model motifs separately with some flexibility on positions as in Meta-MEME (Grundy *et al.*, 1997), the complex architecture needed for such models needs to be built by hand or heuristic methods. This limits the range of architectures that can be explored. So here I have used the Eponine modelling system to learn transcription termination signals positioned at variable points in the DNA sequences. This system allows model architectures to be learnt from the dataset unlike most HMMs.

2.3 Eponine

Eponine is a supervised machine learning approach that can be applied to the training of a wide range of model types and embodies the principle of selecting the simplest possible model to explain the observed data. In this section I briefly explain the Eponine and its implementation. For a detailed description of the tool refer (Down, 2003).

The Eponine package applies Bayesian theory and is able to learn complex models comprised of one or more weighted constraints. Most models consist only of a simple type of constraint called DNA matrices. These matrices are short, ungapped sequence motifs, which contain a series of column distributions over the DNA alphabet. Parameterizations of the model are learnt using an RVM based trainer which takes a positive dataset with the interesting feature and a negative set without the feature. The trainer starts with an initial set of working matrices and iteratively selects only those matrices that can classify the positive dataset from the negative dataset. This tool comes in many flavours and the one I used here is called *Eponine Anchored Sequence* (EAS) method. In this method, the ‘weighted’ matrix (or constraint) is anchored from a particular ‘anchor point’ and is compounded by a probability distribution that describes the distance relative to the anchor. Constraints with a positional distribution are called *Positioned Constraints* (PC) (Figure 12a). Thus PCs consist of –

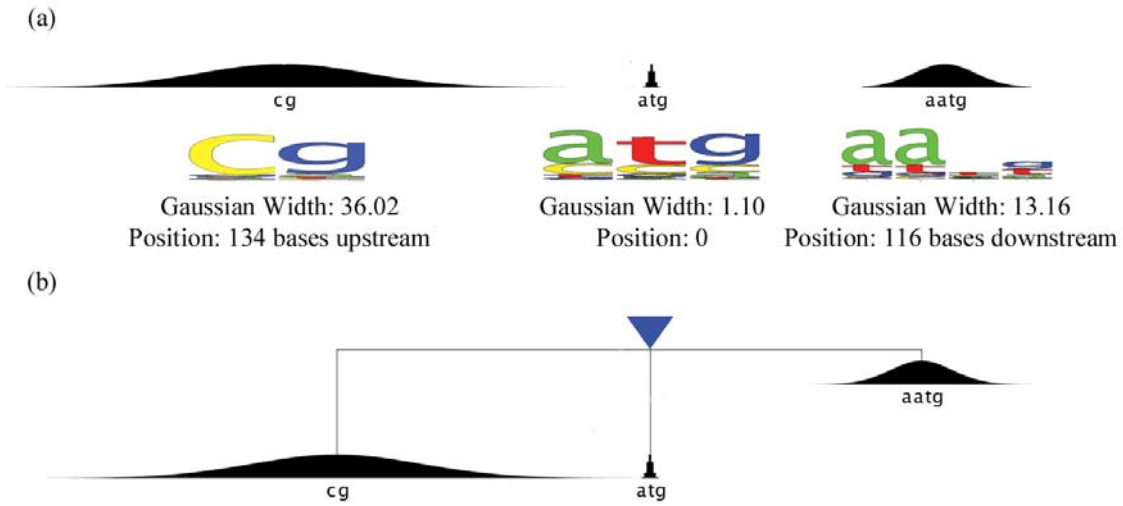

Figure 12. An example of Eponine model. (a) Position constraints along with Gaussian width and position. The nucleotide distribution in the weight matrices are represented as sequence logos. (b) Eponine model constructed from these constraints

- A preferred sequence motif, defined as a position-weight matrix.
- A probability distribution describing the localisation of the motif relative to the anchor point, as described from the integer offsets observed. A Gaussian distribution is used for this purpose as it is simple and less prone to 'overfitting' issues.

New PCs are constructed through training by the following algorithm –

- Pick a sequence given for training.
- Pick a point relative to the anchor point of the sequence.
- Take a sequence motif of 3 to 6 bases at the point and construct a weight matrix.
- Add a Gaussian distribution to the weight matrix of random width centred at the position of the sequence motif found.

After creating the novel PCs from the given data, a range of sampling strategies given below are used in further training to select the PCs that model the training data.

- Select an existing PC and adjust the emission spectrum of one column in the weight matrix by sampling from a Dirichlet distribution (Mackay, 2003) centred on current values.

- Add an extra column to the existing weight matrix till the threshold is reached.
- Remove a column from the start or end of the weight matrix till the threshold is reached.
- Adjust the width parameter of the Gaussian distribution.
- Adjust the centre position for a Gaussian distribution

The score for a PC for a given sequence, *x* is –

$$\phi(x) = \frac{1}{|W|} \log \sum_{i=-\infty}^{\infty} P(i) W(x,i) \qquad (2)$$

where, *P* is a positional probability and *W(x ,i)* is a DNA weight matrix probability for offset *i* relative to anchor point of *x*.

These PCs are then linked together to form an EAS model (Figure 12b) in the form of a *Generalised Linear Model* (GLM, McCullagh and Nelder, 1983), commonly used for classification and regression problems. A Generalised linear function ($\eta(x)$) for variable *x* (such as DNA sequence) is represented as -

$$\eta(x) = \sum_{m=1}^{M} \beta_m \phi_m(x) + k \qquad (3)$$

where, ϕ is a set of *M* basis functions defining the variable *x* (for example, a set of motifs, PCs) and β is a vector of weights (for example, relative importance given to motifs).

Finding an appropriate set of basis function to define the features of the dataset and finding a vector of weights for the given set of basis functions are the two issues to construct an EAS model.

The first problem can be tackled using sparse learning methods like Support Vector Machine (Scholkopf *et al.,* 1999; Vapnik, 1995). Sparsity is a desirable feature as they produce simple models and tend to make useful generalisation of the data. While SVMs have helped to solve biological problems, they are mainly used for numerical data. Nevertheless, deriving such functions is complex and problematic and poses a serious

problem in extending to biological data. Moreover SVMs allow training of GLMs only with limited functions that explain the dataset. So to tackle this, another sparse learning method called RVM was introduced (Tipping, 2001a, b). RVM is a Bayesian approach that can train a GLM with any collection of basis functions and thus opens new possibility of solving biological problems.

In a binary classification problem, where each datum x_n has a label t_n (either 1 or 0, meaning positive or negative sequence respectively), the probability that a dataset is correctly labelled given a classifier EAS model $\pi(x)$ can be given as –

$$P(t \mid x, \beta) = \prod_{n=1}^{N} \pi(x_n)^{t_n} (1 - \pi(x_n))^{1-t_n} \tag{4}$$

where, β is a set of weights.

Now the second problem can be tackled using the Bayes theorem explained before and $P(t \mid x, \beta)$ by inferring likely values of weights given some labelled data.

$$P(\beta \mid x, t) = \frac{P(\beta) P(t \mid x, \beta)}{P(x)} \tag{5}$$

The probability distribution $P(\beta)$ is our prior belief in the values of weights, β. The basic prior is an independent Gaussian distribution, N, over the weight of each basis function and can be derived by inferring the values of inverse of Gaussian, α. As the α values are inferred it is necessary to provide an additional hyperprior value and in this case a non-informative a very broad gamma distribution is used.

$$P(\beta) = \sum_{m=1}^{M} N(\beta_m \mid 0, \alpha_m^{-1}) \tag{6}$$

When a basis function providing additional information to the model gets a non-zero value, the amount of information learnt about the labelled dataset increases and thus the probability of the model given the data. If the basis function provides no information either because of redundancy or irrelevancy, no weight is added that will lead to a significant increase in the

likelihood of the function. At this juncture, by setting the α_m parameter to a large value will set the $P(\beta_m)$ to zero and thereby the posterior probability of the model is maximized. Thus a higher α makes the basis function irrelevant and removed from the model and thus simple models are derived resulting in generalisation.

Incorporating as little prior knowledge about the dataset would be an ideal way of training a model. However this will end up exploring a large amount of features of the data for basis functions leading to a computationally expensive process. So a subset of basis function called, working set, is initialized from large sets of candidate basis functions. As described above, when the trainer is run, it calculates the α values for these basis functions and those that get a higher value are removed from the set. Once the size of the subset drops below a certain limit, new functions are added from the pool and the α and β values are initialized and set for training. This is continued until the basis functions from the pool get exhausted. The trainer stops training when there is no significant difference between priors and weights between cycles and converges to an optimal solution.

2.4 Modifying Eponine parameters

2.4.1 Distribution

In splice site and transcription termination models the positional distribution model used to capture the offsets of the motifs; relative to the anchor point in a PC was extended.

In the transcription termination models, the position of the downstream sequence motif from the anchor point (in this case, cleavage site) was found to be variable both form recent experimental results and various training runs. So to accommodate the large variation in the offset values the allowed Gaussian distribution width was modified significantly from the default parameters used for other models. The beauty of the trainer is that despite being allowed to use a broader distribution, it could still learnt both a broad distribution for downstream motifs and a tight Gaussian for the poly(A) and auxiliary signals.

Similarly, in the case of splice site models, various trails lead to the conclusion that the model is simpler if a Delta rather than a Gaussian distribution is used to capture the offset values of the PC relative to the anchor point, so this function was implemented and the

modelling system configured to automatically select between Gaussian and Delta during training. Also, the model training was supplemented by providing simple weight matrices as a sample set of basis functions while training.

2.4.2 Position weight matrix

In a Weight Matrix (WM), each column represents the probability distribution of the nucleotides at a particular position in the sequence. A weight matrix can be treated as a probabilistic model *M* of fixed length sequence with no gaps. Then the probability of a sequence *x* fitting this model can be given as –

$$P(x \mid M) = \prod_{i=1}^{L} e_i(x_i) \qquad (7)$$

where, *L* is the length of the matrix and $e_i(x_i)$ represents probability of observing base x_i at position *i*. This model is considered as a zero order model as each position in the motif is assumed to be independent of all others. The probability is estimated as log odds score by comparing with the probability of observing *x* under a random model, *q*. The log-odds score is calculated using this formula –

$$S = \sum_{i=1}^{L} \log \frac{e_i(x_i)}{q_i(x_i)} \qquad (8)$$

Position weight matrix can be viewed as trivial HMM where a series of states are separated by transitions with probability of 1. This means, after observing an emission probability over the ACGT alphabets of each state (column in the matrix) the machine moves to the next state in a fixed manner. This simple WM can be wrapped as HMM by adding a few additional states to emit a variable number of flanking sequences of each side of the motif. This simple case can then be blown up to complex HMM by adding states to deal with insertions and deletions. With this model architecture, the maximum likelihood estimate of the parameters that explain the set of datasets can be found using trainers based on the Baum-Welch algorithm (Durbin *et al.*, 1998). However in my case, instead of using these WM as HMM independently they form a set of working basis functions for the RVM to classify positive sequences from negative. I used this strategy for splice site training, as the default parameters in Eponine were not suited to derive a convergent model. Adding

external WM as a set of basis functions helped me to derive sparse splice site models. This strategy is commonly used in gene prediction programs as well. Position weight matrices were also used in HMMs before with allowance for small insertions and deletions to the expected consensus motif. Pfam protein models are built with this strategy (Bateman *et al.*, 2004).

2.5 Nussinov algorithm

Single stranded RNA molecules tend to form higher order structures which are recognised by the proteins regulating various functions of the cell. The structures are mainly based on base pairing and hairpins are the most common structures found in RNA. The base pairings are conserved due to functional constraints on these RNA molecules. Secondary structures in RNA have various features and are represented in Figure 13. A stem is a double stranded (paired) region whereas a hairpin loop is where the RNA folds back on itself. An internal loop is where a short unpaired region exists between two stems. If the internal loop is asymmetrical and only one strand forms a loop, while the other continues directly from one stem to the other, it is referred to as a bulge. In a multi-branched loop, several stems come together. A pseudoknot is a long range interaction, where a loop pairs with another region.

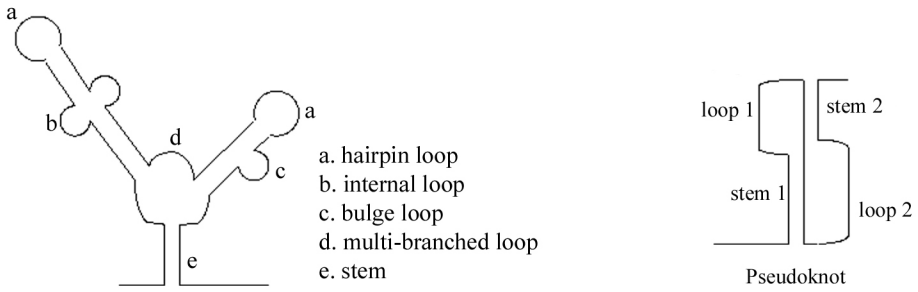

Figure 13. RNA secondary structure features.

Predicting RNA secondary structures from a single sequence is a formidable task as a simple sequence of 200 bases long has the potential to form 10^{50} possible base-paired structures (Durbin *et al.*, 1998). So there is a need to identify the correct structure from false and score them appropriately.

The simplest approach to predict secondary structures is to find the configuration with the greatest number of paired bases as defined by Nussinov (Nussinov, 1978). Testing and

scoring each possible structure is numerically impossible and therefore a dynamic programming can be used to find an optimal solution. In the Nussinov algorithm this is done by extending a sub-optimal structure in four possible ways as shown in Figure 14.

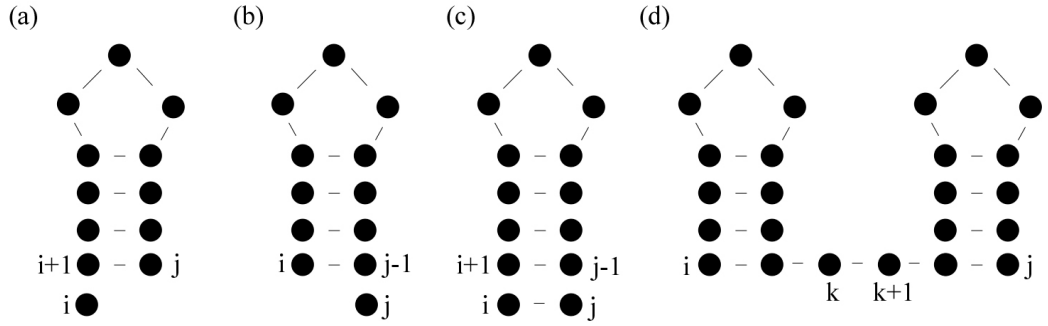

Figure 14. Four possible ways of extending a sub-optimal structure using Nussinov algorithm. (a) i unpaired (b) j unpaired (c) i, j pair (d) bifurcation.

(a) Add an unpaired base *i* to the best structure for the subsequence *i+1, j*

(b) Add an unpaired base *j* to the best structure for the subsequence *i, j-1*

(c) Add paired bases *i-j* to the best structure for the subsequence *i+1, j-1*

(d) Combine two optimal substructures *i, k* and *k+1, j*

A recursive equation for this extension of sub-optimal structure is represented as below –

$$\gamma(i, j) = \max\begin{cases} \gamma(i+1, j) \\ \gamma(i, j+1) \\ \gamma(i+1, j-1) + \delta(i, j) \\ \max_{i<k<j}\left[\gamma(i,k) + \gamma(k+1, j)\right] \end{cases} \quad (9)$$

where, $\gamma(i, j)$ is the score for the maximum number of base pairs that can be formed for sub-sequence $x_i, ..., x_j$ and $\delta(i, j)$ is the score of a base pair x_i and x_j. If x_i and x_j are complimentary, $\delta(i, j) = 1$ else $\delta(i, j) = 0$.

Although the model is simple, it requires several improvements. Firstly, the algorithm allows for hairpin loops of any length. In reality, RNA is not that flexible and a minimum of about 3 nucleotides is needed to form a hairpin. Secondly, in the scoring matrix, bases that

lie on the diagonal correspond to the hairpin loops. Hence while traceback; any base-pairing solution in proximity to the diagonal should be prevented.

A further possibility of improving the Nussinov algorithm is to use *Stochastic Context Free Grammar* (SCFG) to generate a probabilistic model. The original algorithm is changed slightly to allow various probabilities in scoring and regarded as an adapted *CYK algorithm*. Details of this algorithm can be found in *Biological sequence analysis* by Durbin *et al*.

I implemented this algorithm and used it for identifying secondary structures that are responsible for transcription termination. The implementation also formed a part of the Eponine trainer for sampling secondary structure constraints in the stem-loop model explained in chapter 3.

2.6 Zuker algorithm

An improvement over Nussinov algorithm was later developed by using free energy parameters apart from base pair metrics. Energy parameters are included to score base-pair stacking, single dangling nucleotides, terminal mismatches and the lengths of hairpin loops, bulge loops, interior loops and multi-branched loops. This was aided with results from wet-lab experiments leading to different algorithms.

The first algorithm based on energy minimization using nearest neighbour energy parameters was attempted by Tinoco *et. al.*, and Delisi *et. al.* In this algorithm, free energies assigned to base pair stacks and loops and are summed to calculate the overall free energy difference of folding. Later new concepts like dynamic programming methods were incorporated and modified by many people. The popular among them is Zuker's *mfold* ('m' stands for 'multiple') program. The algorithm predicts a minimum free energy, ΔG, as well as minimum free energies for foldings that contain any particular base pair. The success of the program depends on the accuracy of the energy parameter for base pairs and recent versions use the free energy data from Mathews *et al.*, 1999 with the folding temperature of 37˚C and ionic conditions $[Na^+] = 1M$ and $[Mg^{++}] = 0M$ (Zuker, 2003).

The secondary structure is a list of base pairs, denoted by *i:j* for a pairing between the i^{th} and j^{th} nucleotides, r_i and r_j, where $i < j$ by convention. Generally only Watson-Crick base

pairings and G:U wobble pair are treated as base pair rules. However exceptions exist. RNA has A-form helices and two helices are said to form a pseudoknot if base pairs *i:j* from one and *i':j'* from the other satisfy $i < i' < j < j'$ criterion. Pseudoknots are often excluded in the definition of secondary structures as current algorithms have difficulty in identifying them (Zuker, 2000).

Free energy minimization programs generally analyse a large ensemble of structures (called *suboptimal structures*) at different stages. To reduce this range, auxiliary information might be useful and *mfold* program employs base-pair metrics.

The base pair metrics defines RNA molecule as a collection of base pairs that occurs in its three dimensional structure. If R is represented as an RNA sequence then S is a set of ordered pairs, written as *i:j* ($1 \leq i < j \leq n$) satisfying these conditions -

1. $j - i > 3$
2. If *i:j* and *i':j'* are 2 base pairs, then either
   (a) $i = i'$ and $j = j'$ or
   (b) $i < j < i' < j'$ or
   (c) $i < i' < j' < j$ (This condition excludes pseudoknots).

The optimal structure will have the lowest free energy. Each of the various loops and stacked pairs will contribute a certain amount of energy to the secondary structure configuration. The energy of each base pair might be represented as $e(r_i, r_j)$ and the energy of the whole structure as $E(S)$ is then given by –

$$E(S) = \sum_{i,j \in S} e(r_i, r_j) \qquad (10)$$

Reasonable values of e at 37˚ are -3,-2 and -1 kcal/mole for GC, AU and GU pairs respectively. However to capture the destabilizing effects of various loops or the nearest neighbour interactions in helices and loops a more sophisticated algorithm is required.

So to achieve the minimum energy $E(i, j)$ for nucleotides, i and j, the following recurrence relation is used –

$$E(i,j) = \min \begin{cases} E(i+1, j) \cdots\cdots\cdots\cdots\cdots\cdots\cdots (Ia) \\ E(i, j-1) \cdots\cdots\cdots\cdots\cdots\cdots\cdots (Ib) \\ e(r_i, r_j) + E(i+1, j-1) \cdots\cdots (II) \\ \min_{k=i+1}^{j-1} (E(i,k) + E(k+1, j)) \cdots (III) \end{cases} \quad (11)$$

In this relation, there are totally $\phi(n^2)W(\cdot)$ matrices and each of them takes $\phi(n)$ time to calculate. Hence the running time of the algorithm is $\phi(n^3)$ and the memory requirement to store the W matrices is $\phi(n^2)$.

If *(i, i')* and *(j, j')* are two base pairs in the optimal pairing then,

1. $i < i' < j < j'$, i.e. the i pair precedes the j pair
2. $i < j < j' < i'$, i.e. the i pair includes the j pair

The first case will be handled by the condition III in the recurrence relation whereas the second case by conditions Ia, Ib and II.

The algorithm assumes constant energy for multi-branch loops and ignores single base stacking. If these issues are to be considered then some auxiliary information is added, at which point the algorithm gets complicated.

With the free energy determined for each base pair, a traceback algorithm (Zuker, 2003) is used to find the minimum free energy for the folding.

To predict RNA secondary structures in this project, I used the *Vienna RNAfold* package where Zuker algorithm is implemented.

2.7 Biojava and Bioperl

The open source bioinformatics toolkits Bioperl (http://www.bioperl.org) and Biojava (http://www.biojava.org) provided multiple functionalities that made my analysis much easier.

Bioperl (Stajich *et al.,* 2002) is the oldest and most downloaded distribution. It is a toolkit of PERL modules useful in building bioinformatics solutions in PERL language. The modules have a wide array of functionality, including codes for handling and indexing most popular bio-specific database and flat-file formats; for auto-generating bio-related graphics for web pages, classes and methods and for describing and manipulating biological sequences, annotations, trees, alignments and maps.

It is built in an object-oriented manner so that many modules depend on each other to achieve a task. I have extensively used the 'Bio::Seq:IO' and 'Bio::Seq' modules for accomplishing different tasks in this project.

Likewise, Biojava is another toolkit developed using Java language for analysing and presenting biological sequence data (for overview, Mangalam, 2002; Pocock, 2003; Pocock *et al.*, 2000). The toolkit has around 40 packages covering simple sequence manipulation to complex machine learning modules. The Eponine implementation is built using the Biojava package and hence in several cases I used relevant modules from Biojava to train and analyse the gene regulatory signals.

2.8 Databases

2.8.1 ENSEMBL

Various genome projects release DNA sequences into the public domain from throughout the world, making the subsequent task of assembling and annotating it difficult. ENSEMBL (http://www.ensembl.org, Birney *et al.,* 2004) is a joint project between EMBL-EBI and The Wellcome Trust Sanger Institute to develop a system which automatically tracks all the assemblies of a genome and annotates them by finding genes and other features of interest to biologists and medical researchers. This is done by taking sequences from the public domain and storing them in a large database. Automatic annotation using *pmatch* (Ohrt, 2004), *exonerate* (Slater) and *GENEWISE* (Birney and Durbin, 1997) to build genes from protein and mRNA evidence detects most genes and the results are published over the web based interfaces. A separate automatic prediction using EST evidence is carried out, although, the resulting gene structures are less reliable. Any match to the candidate genes in the public databases forms the 'supporting evidence' suggesting the annotations are

accurate. All analysed data are stored in a relational database, which makes it easy to access. To facilitate the analysis process and access the results, ENSEMBL has created a set of PERL modules to connect to this database and query it. I have used these modules to access the data required for this project. ENSEMBL also has excellent web based interfaces and a sequence viewer (ContigView) that allowed me to add my own annotation of the region using the DAS protocol (Dowell *et al.*, 2001). This helped me to view my predictions along with other annotations available in the public domains.

2.8.2 VEGA

The Vertebrate Genome Annotation (VEGA) database (Ashurst, 2002) is a central repository for manual annotation of several finished vertebrate genome sequence. As the data is manually curated the quality of annotation is high. Curation is done on a clone by clone basis using a combination of similarity searches against DNA and protein databases as well as a series of *ab initio* gene prediction programs like GENSCAN (Burge and Karlin, 1997) and FGENESH (Salamov and Solovyev, 2000). Comparative genome analyses are also used for the annotation purposes. Thus the genomic features are added to the sequences based on supporting evidences.

Based on the evidence available, each annotated gene has been classified into the following categories –

(a) *Known* – Identical to known human cDNAs or protein sequences with an entry in LocusLink (Pruitt and Maglott, 2001) or GDB (Harger *et al.*, 2000).
(b) *Novel CDS* – Containing an open reading frame determined based on spliced ESTs and/or similarity to known genes/proteins.
(c) *Novel transcript* – Similar to novel CDS, however with an ambiguous ORF.
(d) *Putative* - Based on spliced human ESTs but without an ORF.
(e) *Pseudogene* – Similar to known proteins but with in-frame stop codons and/or frame shifts disrupting the open reading frame.

I used ‘Known’, ‘Novel CDS’ and ‘Novel transcript’ annotations from this database to extract sequences and features from human chromosome 20 and 13 for various analyses.

I also used annotation for human chromosome 22. Chromosome 22 is also available from VEGA, however as a result of it being the first human chromosome to be sequenced and annotated (Dunham *et al.*, 1999) the annotation has been extensively refined. The third generation gene annotation on chromosome 22 published in 2003 (Collins *et al.*, 2003) is one of the high quality data available for human genome sequences. For annotations, Expressed Sequence Tags (EST), comparative sequence analysis and wet-lab experimental verifications were used. Availability of this high quality annotation helped me to derive datasets for various aspects of the project.

2.8.3 RefSeq

The Reference Sequence (RefSeq) project (Pruitt and Maglott, 2001) run by the National Center for Biotechnology Information (NCBI) provides a collection of non-redundant DNA, RNA and protein sequences along with available information for those sequences. Non-redundancy is ensured by clustering identical or related sequences and representing one sequence out of each cluster. Based on the information available for a particular sequence, RefSeq records are available in four categories –

(i) *Genome annotation* – This category includes contigs, modelled mRNAs and corresponding modelled proteins.
(ii) *Predicted* – Predicted records represent genes of unknown function that are supported by full length mRNA, EST or homologous sequences.
(iii) *Provisional* – Records with known or inferred function not subjected to review.
(iv) *Reviewed* – Records with known function that are manually curated.

Reviewed RefSeqs are richly annotated with publications, gene description, UTR sequences, transcript variants and cDNA sequence removed of any vector or linker contaminating sequences. Hence in this project I used only reviewed records for analyses.

2.9 Other programs

I used several programs available in the public domain to compare with the performance of the models I created using Eponine. Describing all of them is beyond the scope of this chapter and so I will limit myself in briefly explaining them when and where required.

However here, I will give a few details about the *ERPIN* poly(A) prediction program used to compare with my transcription termination model and the GAZE method used to predict genes with Eponine model features.

2.9.1 ERPIN

ERPIN (*E*asy *R*NA *P*rofile *I*dentificatio*N*) is an RNA motif search program developed by Daniel Gautheret and Andre Lambert (Gautheret and Lambert, 2001). ERPIN reads a sequence alignment and secondary structure, and automatically infers a statistical 'secondary structure profile' (SSP). A dynamic programming algorithm is used to scan any target sequence with this SSP to find matches and score them. SSP profiles are constructed using two weight matrices – one for single strand regions in the given sequence and another for helical regions.

Helix profiles are 16-row matrices with a lod-score for each possible base-pair, while single strand profiles are generally five-row matrices with lod-scores for the four bases and the gap character. For a helix of size *n*, the profile has 16 rows and *n* columns in the matrix. For a single-strand of size *n*, the profile has five rows and *n* columns. The lod-score for a base at position *i* is given as –

$$S_i = \log\left(\frac{O_i}{E_i}\right) \qquad (12)$$

where, O_i and E_i are the observed and expected frequencies respectively for the base at position *i*. ERPIN treats gaps as another base rather than issuing penalties as done in sequence alignment.

Likewise, a lod-score for each base-pair at position *i* and *i+1* (consecutive bases) is given as –

$$S_{i,i+1} = \log\left(\frac{O_{i,i+1}}{E_i \times E_{i+1}}\right) \qquad (13)$$

where, $O_{i,i+1}$ is the observed frequency for the base-pair at position i and $i+1$ and E_i, E_{i+1} are the expected frequencies of individual bases.

The helix profiles capture both Watson-Crick base pairs and non-canonical base pairs, however, gap character is not allowed in helical regions.

In this project I used an ERPIN model, trained to predict poly(A) signals with the following command line –

```
erpin polya.epn <database file> 2,3 -umask 2 -umask 2 3 -cutoff 70% 74% -
unifstat –smp
```

I then compared the predictions of ERPIN with those of Eponine transcription termination model.

2.9.2 GAZE

GAZE (Howe *et al.,* 2002) is a gene prediction tool that assembles evidences of gene components or features into complete gene structures. The gene features and the model structures are supplied by the user making it completely configurable.

As described earlier, almost all the gene prediction methods first do an extensive search for signal and content information's on the sequence to identify gene components. However, they differ in the subsequent mechanism of integration of this information's to predict the gene structure. This unified two step process adapted by various programs introduces an inherent rigidity in extending them to incorporate new knowledge about gene structures. GAZE tackles this by separating the two steps and allowing to build a customised version of the *ab initio* gene prediction system with user defined features. In that way, GAZE is not tied to any specific signal or content sensors as well. Another key feature of GAZE is that it does not work directly with genomic DNA sequence. Instead it predicts gene structure from an input file with signal and content information marked in GFF format (WTSI). The configurations for integrating the features to form the gene are defined in another input file in XML format. Thus GAZE is a generic system that uses dynamic programming to obtain the highest scoring gene structure based on external features and configurations.

The algorithm also has a run time effectively linear with the length of the sequence without compromising accuracy.

The gene structures are scored by taking a list of features ordered by their sequence position and the rules defined in the configuration file. For example, for a sequence of 1000 bp long with the features – transcription start site @ 100 bp, donor site @ 250 bp, acceptor site @ 500 bp and Poly(A) signal @ 750 bp and a configuration allowing a gene to be formed with or without introns can lead to 2 gene structures. One, a single-exon gene without taking donor and acceptor sites and another with an intron defined with donor and acceptor site. A score of each of these gene structures are assigned and the highest scoring gene structure is defined to be the most probable given the features and configuration. Mathematically this is represented in the following equation -

$$E(\phi) = \sum_{i=0}^{n} \mathrm{Re}\, g_{t(\phi_i)\to t(\phi_{i+1})}(l(\phi_i), l(\phi_{i+1})) + g(\phi_{i+1}) \qquad (14)$$

where, $t(\phi_i)$ defines the type of feature, ϕ_i and $l(\phi_i)$ is the location of the feature, ϕ_i, in the sequence and $g(\phi_i)$ is the respective score of the feature. $\mathrm{Re}\, g_{t(\phi_i)\to t(\phi_{i+1})}$ represents the region score for interval (ϕ_i, ϕ_{i+1}) bordered on the left and right with the types of the features, referred as source (*src)* and target (*tgt)* features.

The ‘target’ feature and its potential origins (‘source’ features) define the rules of the gene structure model. In the above example, the target feature, Poly(A) signal can be preceded by an acceptor site or transcription start site and thus allowing for 2 gene structures to be built. However, an acceptable gene structure cannot be formed by allowing Poly(A) signal preceded by the donor site and thus defining a set of rules how structures can be built. Additional constraints, given below, can also be added to these rules to define more stringency.

(a) *Distance constraint* specifying the length of the segment defined by source and target features.

(b) *Phase constraint* specifying the source and target features that should occur - 0, 1 and 2 bases apart.
(c) *Interruption constraint* specifying an illegal occurrence of a feature between source and target.
(d) *DNA constraint* specifying an illegal occurrence of a DNA sequence between source and target.

A length penalty function and segment qualifier defined by these constraints add to the final score of the gene structure. The highest scoring gene structure with these rules and constraints is obtained by using dynamic programming. For more details about the scoring and algorithmic issues, please refer to this thesis (Howe, 2003).

Taking advantage of the user configurable GAZE system, in this project, I used Eponine model features to predict gene structures with the rules and constraints defined in Appendix C. Two configurations with and without translation features are used in predicting Eponine based gene prediction. I employed *phase constraint* in gene configuration with the translation feature but no *distance constraint* in both the configurations thus kept no restrictions on the maximum length of exons and introns.

2.10 Concluding remarks

In this chapter I have given an overview of all the strategies, tools and databases used to find gene regulatory signals in this project. In the following chapters I will explain in detail how the package was employed to derive transcription termination, translation start and stop and splice site sequence models. Also apart from the sequence model from Eponine, in chapter 3, I have explained the results of Nussinov and Zuker algorithms used to search for stem-loop structures in the 3’ end of the genes. I implemented Nussinov algorithm using PERL modules for this purpose. RNAfold implementation of Zuker algorithm in Vienna package was used to find stem-loop structures based on free energy metrics.