Gene Prediction using a configurable system for the integration of data by Dynamic Programming

Thesis by

Kevin Howe

submitted for the degree of

Doctor of Philosophy

University of Cambridge

St. John's College

and

The Wellcome Trust Sanger Institute

Wellcome Trust Genome Campus

Hinxton

Cambridge

(Submitted on February 20, 2003)

Summary

A new approach to the computational identification of protein-coding gene structures in genomic DNA sequence is described. It overcomes rigidities inherent in most existing gene prediction methods, for example those based on Hidden Markov Models (HMMs), by supporting a flexible computational model of how sequence signal signals fit together into complete gene structures.

The primary result of the work is a gene prediction tool for the assembly of evidence for individual gene components (features) into complete gene structures. The system is completely configurable in that both the features themselves, and the model of gene structure against which candidate assemblies are validated and scored, are external to the system and supplied by the user. The gene prediction process is therefore tied neither to any specific techniques for the recognition of sequence signals, nor any specific underlying model of gene structure.

The methodology is implemented in a piece of software called "GAZE" which uses a dynamic programming algorithm to obtain (i) the highest scoring gene structure consistent with the user-supplied features and gene-structure model, and (ii) posterior probabilities that each feature is part of a gene. The algorithm includes a novel pruning strategy, ensuring that it has a run-time effectively linear in the length of the sequence without compromising accuracy. The effectiveness of the approach is explored by applying it to the prediction of gene structures in sequences of the nematode worm *C. elegans*.

GAZE allows the integration of gene prediction data from multiple, arbitrary sources. It is important for the accuracy of the system that the various pieces of evidence are weighted appropriately with respect to each other. A novel strategy for the automatic determination of optimal values for these weights is described. The method uses numerical analysis and dynamic programming to maximise a probabilistic accuracy function with respect to the weights. Its effectiveness is demonstrated in the context of the development a gene prediction system for vertebrate sequences using GAZE.

Contents

Preface **ix**

Introduction **x**

1 Methods for the computational identification of gene structures **1**

1.1 Identifying the elements of gene structure 2

1.1.1 The recognition of gene structural elements 3

1.1.2 The recognition of gene regions 5

1.2 Identifying complete gene structures 7

1.2.1 Gene fragment assembly methods 7

1.2.2 Hidden Markov models 9

1.3 Using similarity to other sequences 11

1.3.1 Expressed sequences 11

1.3.2 The sequences of other genomes 12

1.4 Assessing gene prediction accuracy 14

1.4.1 Gene prediction accuracy metrics 15

1.5 Other issues 17

2 GAZE **20**

2.1 Introduction 20

2.2 From features and segments to gene structures 22

2.3 Elements of a GAZE configuration 25

2.3.1 Defining the validity of candidate gene structures 25
2.3.2 Defining the scoring of valid gene structures 29
2.4 Prediction with a GAZE gene structure model 31
2.4.1 The GAZE scoring function 31
2.4.2 Obtaining the highest scoring valid gene structure 33
2.5 A probability distribution over gene structures 34
2.5.1 Gene Structure probabilities 35
2.5.2 Feature and Region posterior probabilities 37
2.5.3 Stochastic traceback 38
2.6 Practical considerations 39
2.6.1 Maintaining numerical stability 39
2.6.2 Working within practical limits of space and time 41
2.6.3 A novel pruning strategy 46
2.7 Relationship to other similar systems 52
2.7.1 Other gene prediction toolkits 52
2.7.2 HMM methods 54

3 Using GAZE for gene finding in *Caenorhabditis elegans* 56
3.1 Introduction 56
3.2 Gene prediction materials for *C.elegans* 57
3.2.1 WormBase and The WormSeq dataset 57
3.2.2 A source of gene prediction data: GENEFINDER 60
3.3 Definition of a GAZE configuration in three steps 62
3.3.1 A single, single-exon gene 63
3.3.2 Extension to spliced structures 64
3.3.3 Extending to multiple genes on both strands 66
3.4 Applying the model to *C.elegans* sequences 68
3.4.1 Predicting genes in WormSeq 68
3.4.2 Using feature-selection to refine the predictions 68
3.4.3 Adjusting the score to refine the predictions 70

3.4.4 A comparison with GENEFINDER 72
3.5 Towards a *C.elegans*-specific model of gene structure 73
3.5.1 Splicing mechanisms in *C.elegans* 74
3.5.2 *Trans*-splicing confuses gene prediction programs 75
3.5.3 A GAZE model accounting for *trans*-splicing 75
3.6 Integrating similarity information 79
3.6.1 ESTs and gene prediction 80
3.6.2 A GAZE model for the use of EST alignments 82
3.7 A closer look at the accuracy of GAZE 87
3.7.1 Gene-level accuracy 88
3.7.2 Accuracy at base-pair and exon-level 89
3.7.3 Accuracy by exon-type 89
3.7.4 Genome scale accuracy 91
3.8 Examining the probabilistic aspects of GAZE 93
3.8.1 The reliability of GAZE predictions 93
3.8.2 Feature probabilities can aid manual curation 95
3.8.3 Feature probabilities could be used to identify alternative splicing events 97

4 A method for estimating optimal parameters for a GAZE model 100
4.1 Introduction 100
4.2 Evidence weighting in GAZE 101
4.2.1 Optimally parsing a sequence according to weighted evidence 101
4.2.2 Accommodating weights in the GAZE scoring function 102
4.3 Two approaches to obtaining an optimal set of weights 105
4.3.1 Maximum Likelihood 105
4.3.2 Maximal Feature Discrimination 106
4.4 Optimising the objective functions by gradient descent 108
4.4.1 A conjugate gradient descent method 109
4.5 Calculating the gradient by dynamic programming 110

4.5.1 The derivative of the ML function . . . 110
4.5.2 The derivative of the MFD function . . . 112
4.5.3 Computing the weighted average of the derivatives . . . 113
4.6 Implementation issues . . . 115
4.6.1 Numerical stability . . . 115
4.6.2 Parameter tying . . . 117
4.6.3 Time and memory usage . . . 117
4.7 A comparison with other methods . . . 119
4.7.1 Other methods based on weighted evidence . . . 119
4.7.2 Hidden Markov model methods . . . 122
4.8 Optimising evidence weights for GAZE_EST . . . 129
4.8.1 Choice of parameters and optimisation method . . . 129
4.8.2 Accuracy of the trained model . . . 130

5 Application of GAZE training to the development of a vertebrate gene finder 133

5.1 Introduction . . . 133
5.2 Materials for gene prediction in vertebrate sequences . . . 134
5.2.1 Datasets for training and testing . . . 134
5.2.2 Properties of the gene sets . . . 136
5.2.3 A source of gene prediction features: GENEID . . . 139
5.3 A GAZE configuration for human gene finding . . . 141
5.3.1 A GAZE configuration based on GENEID . . . 141
5.3.2 Accuracy of the model . . . 143
5.4 Optimising the parameters of the model . . . 143
5.4.1 Defining the parameters of the model . . . 144
5.4.2 Accuracy of the trained model . . . 145
5.5 Investigating three ways to improve accuracy . . . 151
5.5.1 Incorporating promoter prediction data . . . 152
5.5.2 Using exon length distributions . . . 158

5.5.3 Introducing C+G%-dependent model parameters 163
5.5.4 Combining all three types of evidence 168

6 Conclusions **170**

Bibliography **174**

A Some example GAZE configurations **187**
A.1 GAZE_std 187
A.2 GAZE_EST 195
A.3 GAZE_GeneID 208

List of Tables

3.1 Gene level accuracy of standard model 70
3.2 Gene level accuracy of *trans*-splice model 78
3.3 Gene-level accuracy of the EST model 85
3.4 Comparative gene level accuracy of all configurations 88
3.5 Base-pair and exon-level accuracy 89
3.6 Accuracy by exon type 90
3.7 Accuracy in genome-scale assessment 92

4.1 Evidence weights determined by training 131
4.2 Accuracy of GAZE_EST after training 131

5.1 Properties of training and test datasets 136
5.2 Gene level accuracy of standard model 143
5.3 Comparative accuracy of ML and MFD training 146
5.4 Promoter predictions at different thresholds 153
5.5 Accuracy of the 5' UTR model 156
5.6 Accuracy of the exon-length model 161
5.7 C+G content of test and training sets 164
5.8 Accuracy of C+G%-specific model 166
5.9 Accuracy of model combining all evidence 168

List of Figures

1.1 Components of a eukaryotic protein-coding gene 2

2.1 Prediction of genes by GAZE 24
2.2 Problems with feature ordering 28

3.1 A simple GAZE configuration 63
3.2 Extensions to the configuration to model spliced genes 65
3.3 Extensions to the configuration to model multiple genes 67
3.4 Pictorial representation of a GAZE configuration 69
3.5 Increased occurrence of wrong genes 71
3.6 *Trans*-splicing in *C.elegans* 74
3.7 Confusion of standard methods by *trans*-splicing 76
3.8 A GAZE configuration modelling *trans*-splicing 77
3.9 Pictorial representation of GAZE configuration for using ESTs . . . 86
3.10 Posterior feature probabilities for predicted genes 94
3.11 Posterior probabilities for all candidate features 96
3.12 Detecting alternative splicing with GAZE 98

5.1 A GAZE configuration for vertebrate gene finding 142
5.2 Accuracy in training set after each line minimisation 149
5.3 Accuracy in test set after each line minimisation 150
5.4 A GAZE configuration modelling the 5' UTR 155

Preface

Too many people have helped me during my time at the Sanger Centre to name individually. I feel it appropriate however to give some people a special mention.

I first came to the Sanger in October 1998 to work on the Pfam database, under the guidance of Alex Bateman and Ewan Birney. Being the young and impressionable newcomer to bioinformatics that I was, Alex and Ewan have to take a degree of credit/blame for the way I now approach problems in this field. Although my involvement with Pfam has diminished in recent years due to other commitments, Alex in particular has continued to to take an active interest in my scientific development, and for that I thank him.

This thesis represents the result of these other commitments, work which I began in October 1999. During this time, my primary source of guidance has been my supervisor, Richard Durbin. I thank him for ideas, direction, encouragement and not least for tolerating my (what must be sometimes infuriating) indecisiveness.

Declaration of originality

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

Introduction

The working draft of the human genome is now nearly two years old [112], with announcement of the finished article expected later this year. The near-completion of this effort has seen a redirection of resources, resulting in an acceleration in the genome-sequencing of other organisms studied in experimental biology, such as mouse and zebrafish. According to the National Centre for Biotechnology Information, nearly 900 genomes are either finished or currently being sequenced[1]. The fact that such large scale sequencing is possible represents an incredible achievement, both in technology/engineering, and sheer organisation. However, genomes only become useful resource for science through biological interpretation, i.e. *annotation* of the role of the different parts of the sequence in cellular processes. Without annotation, genome sequencing is, to paraphrase Ernest Rutherford, nothing more than stamp collecting.

The specific problem addressed by this thesis is that of the annotation of the *gene structures* in a genome. Annotation of a genome in terms of its constituent genes and their intron-exon structure allows us to infer a set of proteins for an organism. Furthermore, the genomic context of the genes can provide insight into the regulatory mechanisms that determine where and under what conditions the corresponding proteins are expressed, as well as being a useful resource for experimental biology.

Gene structure annotation of genomic sequence is still most accurately performed by trained experts, combining the results of a number of computational and experimental analyses with biological knowledge and heuristics. This is naturally a slow

[1]http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/allorg.html, 3rd February 2003

process, and the huge volume of sequence data being generated places an unrealistic demand on the number of experts required to perform this skilled activity. In addition, for the annotation of a large vertebrate genome to be completed in any reasonable amount of time, it is necessary to divide the sequence amongst up to a hundred annotators. This can have the undesirable result that different sections of a genome can be annotated with different standards and procedures. Reliable, completely automated methods for gene structure annotation would therefore firstly cope with the rate at which genome sequence data is being generated and secondly provide gene structure annotation that is *consistent*.

This thesis describes a new approach to the automated prediction of gene structures in genomic DNA sequences. Despite progressive improvement in the accuracy of computational methods in the last fifteen years, they remain imperfect. The problem therefore still attracts considerable research interest both into the biological processes of transcription, RNA processing and translation that determine the gene structure of a genome, and into methods for the recognition of the sequence signals involved in these processes. The integration of new knowledge and methods into complete gene prediction systems is however often inhibited by rigidities of design, such as a fixed assumed underlying model of the compositional and structural properties of genes.

The primary motivation for my research has been to accelerate the integration of new and possibly disparate knowledge and techniques into the gene prediction process. To this end, I have developed a structured framework for the assembly of gene prediction evidence from multiple, arbitrary sources into complete gene structure predictions. Careful design and certain assumptions allow the system to make probabilistic statements about its predictions, and this in turn facilitates a principled approach to the problem of determining an optimal weighting strategy for the various types of evidence employed.

The organisation of the dissertation is as follows. Chapter 1 discusses some of the issues and techniques of computational gene-structure prediction. The aim is to

provide an introduction and broad survey, as many of the issues that are directly relevant to the work presented in the remainder of the thesis are expanded upon where appropriate.

After this short review, the dissertation can be viewed as comprising of two parts. The first part (chapters 2 and 3) describes a framework for the integration of arbitrary gene prediction data, and its application to the development of a gene finder for *C. elegans* sequences; the second part (chapters 4 and 5) describes a new approach to probabilistic parameter estimation and its application to the performance-tuning of a gene prediction system for vertebrate sequences.

Chapter 2 describes the details of the framework, implemented in a program called "GAZE". I briefly explain the elements of the system, with focus on the *configuration file* that controls the assembly of the external evidence into complete gene predictions. I then go on to describe the dynamic programming algorithms used by GAZE for the calculation of the optimal gene structure and posterior probabilities for parts of gene structures, including a novel search-space pruning strategy. To end, I contrast GAZE with other, similar approaches to computational gene prediction.

Chapter 3 describes the application of GAZE to gene prediction in *C. elegans* sequences. I outline the stepwise development of an initial configuration, and explore the effects of extending the model in two ways, first to account for a worm-specific peculiarity of gene structure, and second to make use of sequence similarity information. I also examine the probabilistic aspects of the system and explore some of their potential applications.

Chapter 4 addresses the problem of identifying an optimal weighting for the scores attached to the different types of evidence employed in an integrated gene prediction system. Two methods for estimating optimal weights for the elements of a GAZE configuration are described. The first is based on a classical maximum likelihood approach; the second is a novel method which I call *Maximal Feature Discrimination* (MFD). I contrast these with other similar techniques, particularly those used for Hidden Markov Models.

Chapter 5 describes the application of Maximal Feature Discrimination to the training of a simple GAZE model for gene finding in vertebrate sequences, and compares the results with those obtained using the classical maximum likelihood method. I extend the simple model with each of three types of additional evidence and demonstrate the effectiveness with which MFD is able to determine weights for the new model elements.

Finally chapter 6 concludes the dissertation by briefly summarising the important aspects of the work, and suggests possible areas for further research.

Chapter 1

Methods for the computational identification of gene structures

This short review outlines some of the current approaches to the computational identification of gene structures in genomic DNA. It focuses on the very specific problem of the prediction of the complete structures of protein-coding genes in the sequences of eukaryotic organisms. Gene prediction in prokaryotes is traditionally viewed as less challenging due to high gene density and lack of introns (although genes with overlapping coding regions are far more common in prokaryotes than in plants and animals). The prediction of non-protein-coding genes is on the other hand considered by most to be a harder problem and is currently attracting considerable research interest in its own right (see section 1.5).

I aim here to summarise the main issues rather than attempting to be comprehensive; there are many existing reviews on this subject, notably by Fickett [39], Claverie [26], Guigo [51], Burge and Karlin [22], Stormo [108], and recently by Zhang [122] and Mathe *et.al.* [77]. Many of the issues summarised here are discussed in further detail in later chapters of this thesis.

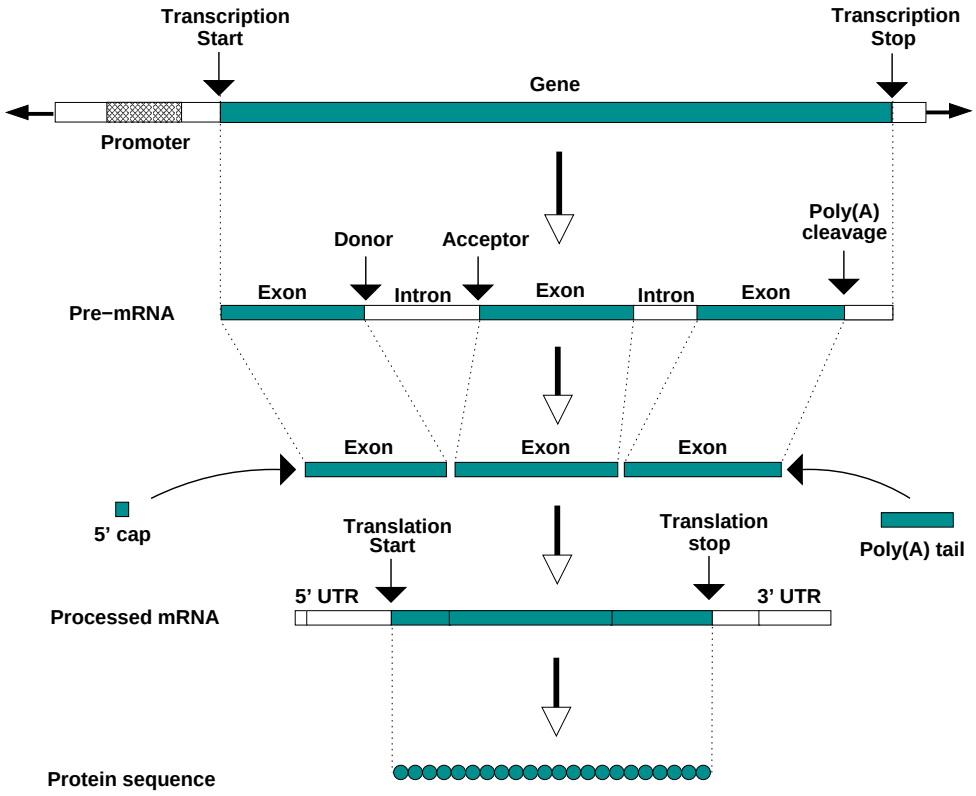


Figure 1.1: The primary components of a typical eukaryotic protein-coding gene, showing its transcription, the processing of the transcribed RNA, and the translation of the processed RNA.

1.1 Identifying the elements of gene structure

Figure 1.1 depicts the classic view of how a eukaryotic protein-coding gene specifies its protein sequence [115] [1]. With this model, the cellular processes of transcription, RNA processing and translation can be viewed abstractly as a series of segmentations: firstly of the genomic DNA sequence into gene and non-gene regions, each gene giving rise to one or more primary transcripts; secondly of the transcript sequence into exonic and intronic regions, the exons forming a processed transcript; and thirdly of the processed transcript into translated and untranslated regions, the translated region giving rise to a protein. The problem of gene prediction in this context can therefore be described as identifying the precise regions of the genomic

DNA that correspond with translated regions in processed RNAs, allowing at least in theory the inference of the proteome from the genome.

At present, our understanding of the mechanisms of transcription, RNA processing and translation is insufficient to be able to model them precisely *in silico*. Until our knowledge reaches a level where a deterministic computational model is realistic, we must supplement what we know with a variety of statistical methods.

1.1.1 The recognition of gene structural elements

The localised sequence signals that form the basis for the cellular recognition of the boundaries between functional regions of gene structure are most intuitively expressed as consensus sequences. The fact that these signals are neither unique to genes nor completely conserved across all genes makes their computational detection non-trivial.

Of the three processes of transcription, RNA processing and translation, it is the sequence signals of the first that are the most poorly understood. In attempting to define the site of transcription initiation, the AT-rich TATA-box signal located around 30 base-pairs upstream forms the basis for the many of detection techniques [40], but only around 70% of human promoters (for example) have this characteristic [19]. The association of promoters with CpG islands (short regions where suppression of methylation leads to a higher-than-average proportion of the dinucleotide CG [8]) provides another useful signal, but one that can give only low resolution mapping of the transcription start site. The site of transcription termination is even less well characterised, and most methods instead focus on trying to identify the upstream site of the transcript cleavage that occurs prior to polyadenylation, the poly(A) site. In many organisms, this is characterised by the presence of the hexanucleotide AATAAA around 30 nucleotides upstream, but in humans the signal is absent from around half of all genes [26], and in *C. elegans* there are as many as 23 one or two base variants documented [14].

Although the sites of *translation* initiation and termination are difficult to iden-

tify in isolation, they do have elements that are completely conserved: the translated regions of all eukaryotic genes begin in the processed mRNA with the triplet corresponding to the start codon, almost always ATG (with a slightly less well conserved context upstream of the ATG known as the Kozak sequence [65]), and end with the first in-frame occurrence of a triplet corresponding to one of the three possible stop codons, TAA, TGA, and TAG. This information however is insufficient to be able to to identify these sites in the genome with any accuracy at all. The degeneracy of the translation initiation site can be explained in part by the fact that the ribosome need "search" only a relatively small stretch of processed mRNA for a place to bind rather than a far longer stretch of genomic DNA. We still do not understand enough about translation however to identify the initiation site even in a processed mRNA with certainty.

As with translation, searching the genomic DNA for the signals important in RNA processing provides a more difficult problem than that encountered by the spliceosome, which is faced only with an orders-of-magnitude smaller RNA molecule. Even so, our understanding of the signals involved in the splicing process, although far from comprehensive, provides the most useful information for the identification of gene structures directly in the genomic DNA. The Donor and Acceptor splice sites have the consensuses AG—**GT**RAGT and YYYYYYYYYYNC**AG**—G respectively [122], with the GT and AG defining the beginning and end of virtually all introns. In vertebrates, the acceptor signal is often extended upstream into the intron to capture the completely conserved A residue that occurs at the branch-point [20].

Although these signals of transcription, translation and RNA-splicing are quite different, the same kinds of techniques are used to identify features of all of them. Most correspond to variations on the idea of a generalised consensus sequence for the signal, using a probability distribution over the possible residues occurring at each position. The widely used Weight Matrix method (WMM) [105] treats each position in the consensus as independent. Where it is believed that there are linear dependencies between the positions, the Weight Array method (WAM) is often used,

where the probability of residue s appearing in position i depends not only on the position (as with the WMM), but also on the residues appearing at positions $i-1, i-2, \ldots, i-k$ (where k is chosen according to the amount of training data available). Where the dependencies between columns are known but non-adjacent, Maximal Dependence Decomposition (MDD) [21] can be used, and multi-layered artificial networks can model complex dependencies between positions even when the dependencies themselves are poorly understood [17]. Finally, profile Hidden Markov Models are able to model insertions and deletions with respect to the generalised consensus [36].

In practice, the degeneracy of the short sequence-motifs characteristic of these gene features makes these so called *signal sensor* methods most useful when used in combination with *content sensor* methods for assessing the likelihood of the longer more extensive regions between the features, such as intron and exons (see below). However, there are many examples of where these methods are employed directly, for example, in the detection of transcription start sites [34], polyadenylation sites [111], translation start sites [55] and splice sites [86].

1.1.2 The recognition of gene regions

A protein coding region in the genomic DNA consists of a string of triplets, each of which corresponds to a codon that will be translated into an amino acid. Biases in the usage of amino acids in proteins, in the usage of codons for a single amino acid, and the absence of stop codons provide the basis for the majority of methods for differentiating coding regions from non-coding regions.

Inspired by the survey of coding statistics conducted by Fickett and Tung [41], most techniques for assessing the *coding potential* of a sequence are based on the relative frequency of occurrence of frame-specific hexamers (runs of 6 nucleotides) in coding regions compared with non-coding regions (usually introns). These differences can be expressed probabilistically as a kth order *Markov* model [36] which is a stochastic model that assumes that the probability of a base at a given position

depends only upon the previous k bases. The simple weight matrices above are therefore related to 0th-order Markov models, with a different probability distribution for each column, whereas the weight array matrices are related to kth order Markov models. Using hexamer frequencies therefore corresponds to a 5th order Markov model. In practice, protein-coding regions are modelled by using a separate 5th order Markov model for each of the three positions in a codon, as in the GENMARK program [15]. More recently, Interpolated Markov models (IMMs) have been utilised for eukaryotic gene finding [97], in which the probability of a base at a given position is a weighted average of the probabilities according to several Markov models of different orders (e.g. between 0 and 8).

Because these content sensing methods are poor at identifying the precise boundaries of the coding regions, they are most often used in combination with the signal sensing methods described earlier. A pair of detected signals of appropriate type defines a candidate gene region; for example, a translation start site followed by a donor splice site defines a candidate initial protein coding region of an exon, the likelihood of which can be judged by using a content sensor such as the Markov models outlined above. This approach is appropriate for detecting all types of gene region. For example, a candidate intron can be defined by upstream donor and downstream acceptor splice sites, and an exon by a transcription start site / acceptor splice site upstream followed by a polyadenylation site / donor splice site downstream.

In order to be able to discriminate between true and false candidate regions defined in this way, a *score* for the region is commonly computed by combining the information from many sources, typically (i) the signal scores of the defining boundary elements; (ii) content scores for the intervening sequence; (iii) a score obtained from a probability distribution over the possible lengths of the region. If the scores correspond to (log) probabilities and we assume independence, they can be combined most naturally by (addition) multiplication. More generally, it is necessary for the score components to be weighted and combined appropriately [109]. In the pioneering GRAIL system for the detection of protein-coding exons, the scores

reported by several sensors were combined with an artificial neural network [113]. An alternative approach results from viewing each of the n region discriminators as the axis of a high dimensional space, in which case candidate regions can be described by a point in this space. Techniques such as linear and quadratic discriminant analysis can be used to find the surface in this space that optimally separates true and false examples. Of the signals discussed earlier, it is the donor and acceptor splice sites that are perhaps the best characterised, so these techniques have both been employed most successfully for the identification of protein-coding *internal* exons [103] [120]. More recently quadratic discriminant analysis has been used for the recognition of complete initial [29] and terminal [110] exons.

1.2 Identifying complete gene structures

The prediction of complete gene structures is seemingly a more difficult problem than that of the prediction of localised gene features and regions. However, the accuracy of identification of such features and regions can be improved by considering not only their local properties, but also their relationship to other more distant gene features. At the very simplest level, the protein-coding regions of exons almost never overlap in eukaryotes, successive protein-coding regions on the same strand in the same gene must be frame compatible, and the length of the coding regions of the complete gene must be a multiple of three. Such constraints are the basis for most methods for the identification of complete gene structures. The widely used techniques can be broadly divided into two categories: gene fragment assembly, and Hidden Markov Models.

1.2.1 Gene fragment assembly methods

The majority of early programs for the prediction of complete gene structures conceptually broke the problem into two distinct sub-problems: (a) identify a set of candidate gene fragments (e.g. coding exons) using the methods such as those de-

scribed in the previous section; (b) of the many possible complete gene structures assemblies implied by these candidate gene fragments, eliminate *illegal* assemblies, i.e. those that do not adhere to the constraints of gene structure. Then, of those remaining, identify the assembly that is optimal according to some scoring function.

The constraints that determine the legality of candidate assemblies are conveniently expressed as *rules* that dictate which pairs of gene features or regions are allowed to appear next to each other in a legal gene structure. More formally, the rules can be expressed in the form of a grammar [33]. Early programs assumed that the input sequence contained exactly one complete gene on the forward strand of the sequence. Generalisation of the assumed gene structure constraints has given the majority of the more recent programs the ability to predict multiple genes, genes on both strands, and partial genes at the ends of the sequence. It is still the case however that most of these assumed gene models make no allowances for specific peculiarities of certain gene structures, such as the presence of introns that occur completely within the non-coding region of an mRNA (so-called UTR introns), and the presence of genes within an intron of another gene [57].

Disregarding assemblies that do not adhere to the constraints of an assumed gene structure model reduces the number of possible assemblies, but typically many will remain. The first programs worked by iterating though all legal gene structures explicitly [42] [47], but as the number of possible legal assemblies typically grows exponentially with the number of gene fragments [49] [116], this approach was only practicable for small sequences. Subsequent research addressed this problem by using heuristics to reduce the search-space, an example being the original version of the GENEID program which used series of hierarchical rules to filter out unlikely exons before the assembly stage [89]. However, such methods are not guaranteed to identify the optimal assembly. The application of *Dynamic Programming* [6] for the efficient identification of the optimal assembly therefore represented a significant advance. Dynamic Programming is a generic name for a family of recursive optimisation techniques that work by the progressive construction of a solution from

simpler sub-solutions. The technique allows the exploration of a search space that grows exponentially with the number of gene fragments in time that grows only polynomially ($O(n^2)$ or even $O(n)$). For this reason, dynamic programming algorithms are employed in most of the popular gene fragment-assembly based programs [101] [118] [84].

1.2.2 Hidden Markov models

Hidden Markov models (HMMs) provide a convenient framework for the representation of the signal and content displayed by gene features and regions and the constraints of gene structure, in one unified probabilistic model. An HMM can be thought of a probabilistic finite state automaton for generating a sequence from left to right according to an underlying *hidden* state path. At each stage the model (i) emits a single residue according to an *emission* probability distribution over residues that is dependent upon the current state, and (ii) moves to a new state (possibly the same state) according to a *transition* probability distribution over states that is dependent upon the current state.

HMMs can naturally model sequences that are partitioned into regions of different types, for example genes. At the very simplest level, we might have a state for 'exon' with two transitions, one into an 'intron' state, and another looping back into the 'exon' state. These two states can have different emission distributions, and can thus represent inherent compositional differences in introns and exons. Each path through the HMM therefore corresponds to a gene structure, and inference of the underlying state path (given the sequence) corresponds to a prediction of gene structure.

The transition and emission probabilities of an HMM amount to a joint probability distribution over pairs of state-paths and sequences. Given a query sequence, established, efficient (linear-time) techniques can then be used to obtain (i) the most likely state-path given the sequence (using the *Viterbi* dynamic programming algorithm [114]); (ii) the full probability of the sequence considering all state paths

(using the forward or backward algorithm [90]); and (iii) the posterior probability that residue i was generated by state k (using the forward-backward algorithm [90]). One of the key advantages of HMMs is that given a *training* set of sequences with known gene structure, maximum likelihood parameters for the model (i.e. emission and transition probabilities) can be obtained by counting.

HMMs as described generate a region of a particular function class by successively looping to the same state, jumping to a different state at the end of the region. This imposes a geometric distribution on the duration of the states, which is an unrealistic model for most of the functional regions in genes, particularly protein-coding exons [20] [112]. One of two enhancements to the standard HMM architecture is therefore usually employed to represent the non-geometric nature of the length of protein-coding exons.

The first is to model the sequence as a list of classes, with each residue being *labelled* as belonging to a particular class, and being generated by one of a *group* of states labelled as belonging to the same class. Such a formalism is known as a *Class*-HMM [67]. By chaining more than one state together with the label "exon" and then obtaining the most likely *labelling* by summing over all paths with the same labelling, a length distribution resembling that displayed by protein-coding exons is observed. This is the approach taken by HMMGENE [68], which uses a novel hybrid of the Viterbi and forward algorithms to identify the most likely labelling.

The second enhancement often employed is to allow the emission of an entire sequence region from each state, the length of which is chosen according to an explicit length probability distribution for the state. Such models are known as *semi*-Hidden Markov models [21] or *Generalised* HMMs (GHMMs) [72]. In theory, the worst-case run-time of the Viterbi, forward and backward algorithms when applied to GHMMs grows quadratically in the length of the sequence, making their computation prohibitively expensive for true genomic sequence fragments which may be hundreds of kilobases long. In practice, this problem is often addressed by pre-processing the sequence for a list of candidate gene features, alleviating the need to consider every

base of the sequence during the computationally intensive dynamic programming algorithms [72]. This does not address the problem of the quadratic growth of the run-time however. In practice, the run-time does not grow truly quadratically, because the length of coding regions is limited naturally by the fact that they cannot extend past an in-frame stop codon. The GENSCAN program [21] restricts the semi-Markov property to these protein-coding regions only, with non-coding regions being generated by standard self-looping transitions. Although imposing a geometric distribution on the length of such regions, the assumption allows the run-time of the program to grow effectively linearly with sequence length.

1.3 Using similarity to other sequences

1.3.1 Expressed sequences

Despite progressive advances in the *ab initio* gene prediction methods discussed above, it is still the case that the most reliable way to identify the gene structure of a piece of genomic DNA is by aligning it to a corresponding expressed sequence, either cDNA (complementary DNA, a DNA copy of an expressed mRNA) or homologous protein.

A cDNA can be aligned to genomic sequence with a standard pairwise comparison tool like BLASTN [2]. However, in most cases this will not identify the intron-exon boundaries precisely, since similarities often by chance extend partially into the introns flanking an exon. A variety of "spliced" alignment programs address this problem by incorporating knowledge of the donor and acceptor splice site consensuses, scoring long gaps that start with a GT and end with an AG in the genome more favourably [79] [44] [61]. These methods can be applied equally well to full-length cDNA and the more abundant partial cDNAs such as Expressed Sequence Tags (ESTs), although the use of the latter will typically reveal only a portion of the gene structure.

Correspondingly with cDNA methods, the alignment of a homologous protein

to the genomic sequence using a tool such as BLASTX [2] is useful for discovering regions that are likely to be protein-coding, but will usually not reveal the intron-exon boundaries precisely. Spliced alignment programs also exist for the alignment of a protein to genomic DNA. PROCRUSTES [48] for example first obtains a list of candidate internal exons that are delimited by the AG and GT acceptor and donor splice consensuses, and then obtains the assembly of the these exons for which the implied translation is most consistent with the given protein. GENEWISE [10] on the other hand aligns a profile Hidden Markov model constructed from the given protein sequence (or multiple sequence alignment) directly to the genomic DNA, implicitly considering all possible gene predictions. Both of these methods are extremely accurate when the protein is a close homolog to that encoded in the genomic sequence [53].

Techniques that integrate methods of *ab initio* and similarity based gene prediction approaches have the theoretical advantage of maximising accuracy where a similar sequence exists, without compromising the ability to predict novel gene structures where one does not. One common way to do this is to use the alignments resulting from a database search to modify the scores of the appropriate gene structural elements used in an *ab initio* program, e.g. using BLASTX matches to boost the scores of protein-coding exons [71] [69] [119]. As one would hope, these approaches can outperform traditional *ab initio* methods, although the margin of this improvement is limited by the quality of the sequences in the public databases used for prediction. The high-throughput, error-prone nature of ESTs in particular makes their alignment to the genome and subsequent use by an integrated gene prediction method non-trivial [91].

1.3.2 The sequences of other genomes

There have recently been published a number of gene prediction techniques that make use of the genome sequences of more than one organism. These methods are based on the hypothesis that the coding regions of the genomes of a group of

organisms that share a common ancestor should be under greater selection, and therefore be more conserved, than the non-coding regions. Such a comparative approach to gene prediction is appealing for many reasons, not least of which is the ability to discover truly novel genes that share neither the sequence feature characteristics of known genes, nor any observable similarity to expressed sequences in the public databases. In addition, it provides the possibility of identifying conserved non-coding regions that are also under selective pressure, for example those involved in gene regulation.

The most obvious way to look for conserved coding regions between genomes is to use a similarity search tool such as TBLASTX [2] which performs a heuristic-based pair-wise alignment of each possible translation of one sequence to each possible translation of the other. This will typically return a list of small, localised candidate exon-pairs. Programs such as WABA [62] and GLASS [4] on the other hand perform the global alignment of longer syntenic genomic regions, classifying parts of the alignment as likely coding or non-coding.

Although neither these local nor global alignment methods provide direct predictions of gene structures, they form the first stage of many programs that do. CEM [3] for example identifies the optimal assembly of candidate exon pairs reported by TBLASTX into complete pairs of gene structures. ROSETTA [4] on the other hand uses the longer alignments of GLASS and identifies the most likely *parse* of the alignment into intronic, exonic and intergenic regions. Both of these programs simultaneously predict gene structures in both sequences. TWINSCAN [64] is a generalisation of GENSCAN that uses BLASTN alignments of a query sequence to an "Informant" genome to predict gene structures in the query sequence only. The BLASTN alignments are used to construct a "conservation" sequence mirroring the query that classifies each position as "match", "mismatch" or "unaligned", and the predicted gene structure is that which maximises the joint probability of the query sequence and the conservation sequence.

Finally, it is worth mentioning two recent methods based on *pair* Hidden Markov

models [36]. DOUBLESCAN [78] models exon fusion, splitting, insertion and deletion in one sequence with respect to the other. SLAM [82] on the other hand assumes that the number of exons in the homologous gene structures is the same, but also models conserved non-coding regions explicitly, giving it the potential to identity homologous regulatory regions. Both simultaneously predict gene structures in both sequences. The advantage of this approach is that it is unnecessary for a genomic alignment (or set of alignments) to be produced in advance; gene prediction and alignment are performed simultaneously.

The accuracy of comparative gene prediction methods, although representing an advance over traditional single-sequence *ab initio* methods, has so far fallen short of what one might expect to be achievable by such an approach. One reason for this is that when considering only a pair of genomes (as current methods do), conservation will occur in many non-coding regions (about 50% of the conserved regions between human and mouse are non-coding [122]). The use of more than two genomes would be expected to improve performance, and many of the methods discussed above generalise naturally to multiple genomes. However, the increase in computational complexity that results from adding more genomes unfortunately makes this approach impracticle at present. Future research in this area might therefore be towards the use of heuristics to reduce the explosion of the search-space that results from adding more genomes, perhaps using ideas from classic multiple sequence alignment.

1.4 Assessing gene prediction accuracy

In the field of gene prediction, a new method is generally not considered an advancement unless it is shown to be as least as accurate as existing methods. In this section, I briefly discuss some of the issues involved in assessing the "accuracy" of a gene prediction technique.

Surveys of available techniques and their accuracies appear periodically (see for example refs. [23], [85], [94], and http://predict.sanger.ac.uk/th/brca2) although of course progressive improvement in the techniques can mean the specific figures

become dated. The survey of Burset and Guigo [23] however remains one of the most cited articles in the field because it laid down a standard for the way in which gene prediction accuracy should be assessed.

The biggest problem in providing a fair comparison between the accuracy of different gene prediction techniques is the choice of a test-set. The set of 570 vertebrate sequences constructed by Burset and Guigo has gained widespread use as a benchmark for the assessment of vertebrate gene finders. Its usefulness is limited in two ways however. Firstly, since each sequence contains exactly one complete gene on the forward strand, it provides no way to judge the accuracy of programs that can predict multiple gene on both strands of a sequence. Secondly, the utility of the set diminishes with time, since many researchers will use some (or all) of these sequences in the development and parameterisation of their methods.

For comparisons between gene prediction techniques to be fair, researchers must have a clear and unified idea of how accuracy is to be measured. To this end, Burset and Guigo proposed a series of metrics, each summarising a separate aspect of accuracy as a single floating point number. A number of these metrics are made use of in the remainder of this dissertation, and are outlined next.

1.4.1 Gene prediction accuracy metrics

Accuracy is classically presented at three levels: at the level of individual nucleotides or base-pairs; at the level of whole exons; and at the level of complete gene structures.

Base-level accuracy

Accuracy at the base-pair level is described in terms of sensitivity (Sn), which is the proportion of nucleotides annotated as coding that are predicted as coding; and specificity (Sp), the proportion of nucleotides predicted as coding that are annotated as coding. More formally, we can describe these quantities in terms of (i) the number of nucleotides predicted as coding that are actually coding (true positives, TP); (ii) the number of nucleotides predicted as coding that are actually non-coding

(false positives, FP); (iii) the number of nucleotides predicted as non-coding that are actually non-coding (true negatives, TN); and (iv) the number of nucleotides predicted as non-coding that are actually coding (false negatives, FN). Sensitivity and specificity are then calculated as:

$$Sn = \frac{TP}{TP + FN}$$

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

Each of these measures in isolation gives a poor indication of global accuracy, since a method can be highly sensitive by predicting every base as coding, or highly specific by predicting every base as non-coding. Burset and Guigo therefore defined the Correlation Coefficient (CC) that represents aspects of both sensitivity and specificity and acts as a measure of global accuracy at the nucleotide level:

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

Exon and gene-level accuracy

At the exon level, sensitivity and specificity are slightly less obviously defined because we must be clear what it means for an annotated exon to be predicted correctly (and for a predicted exon to be correct). The standard laid down by Burset and Guigo states that a predicted exon is correct if it matches precisely the boundaries of a single, annotated exon. The same criterion is used to judge whether an annotated exon has been predicted correctly. Sensitivity is therefore defined as the proportion of annotated exons that have been predicted correctly, and specificity the proportion of predicted exons that are correct. The Correlation Coefficient is less useful as a summary of these two quantities in this case, so the average of sensitivity and specificity is often quoted as a global measure of accuracy at the exon level.

Requiring that exon boundaries match precisely does not allow for the fact that many predicted and annotated exons will overlap inexactly. The proportion of annotated exons that are completely missed by the prediction (missing exons, ME) and

the proportion of predicted exons that share no overlap with any annotated exon (wrong exons, WE) are therefore also often reported.

Although Burset and Guigo did not define any measures for assessing the accuracy of prediction of whole genes in their article, the exon-level measures are naturally applied in the same way, defining a predicted gene to be correct if it precisely matches the intron-exon structure of an annotated gene, and an annotated gene to be correctly predicted if its intron-exon structure is precisely matched by that of a predicted gene. Missing genes (MG) and Wrong genes (WG) are defined in a similar way to exons.

Reese and colleagues defined two additional gene-level measures to express the accuracy with which a method delineates a sequence containing several genes into its constituent genic regions [91]. In their scheme, an annotated gene is considered *Split* if it is overlapped by more than one predicted gene. Likewise a predicted gene is considered *Joined* if it overlaps more than one annotated gene. Split genes (SG) is defined as the number of predicted genes having some overlap to an annotated gene (i.e. total predicted genes minus number of wrong genes), divided by the number of annotated genes having some overlap to at least one predicted gene (i.e. total annotated genes minus number of missed genes). Joined genes (JG) is the number of annotated genes having some overlap to a predicted gene (i.e. total annotated genes minus number of missed genes) divided by the number of predicted genes having some overlap with an annotated gene (i.e. total predicted genes minus number of wrong genes). These measures are naturally only useful for assessing the accuracy of programs that are capable of predicting several genes in a sequence.

1.5 Other issues

One issue that complicates gene prediction somewhat is that the structural and compositional features of genes can vary according to the particular species. A brief comparison of the genes of a simple animal (the nematode worm, *C. elegans*) and a warm-blooded mammal (human) highlights some of these differences. The propor-

tion of the genome that is protein coding is far higher in the worm than in human: around 25% in the worm compared with under 5% in the human genome. The gene density is also uniformly higher, with the worm having only half as many genes as human, but compressed into a thirtieth of the genomic space. Worm genes also characteristically have longer exons and shorter introns than human genes [112]. These factors give a higher signal-to-noise ratio in *C.elegans* sequences, facilitating gene prediction. On other hand, worm introns lack a detectable branch-site consensus [13], reducing the sequence signal available for the detection of acceptor splice sites. Also many *C.elegans* genes have atypical structural organisation, being transcribed as part of a multi-gene transcript called an operon [12], or *trans*-spliced [66] (see chapter 3 for more details).

These organism-specific properties are usually addressed by determining separate parameter sets for each organism (although the degree to which this approach can reflect structural as well as compositional differences is limited). The properties of genes can also differ markedly within a single species as well as between species. The human genome, and the genomes of other warm-blooded vertebrates tend to exhibit long-range variations in certain base compositional properties (e.g. C+G content) which has been explained in terms of "isochores" [7]. It has been shown that certain structural properties of human genes, for example intron-length and intergenic distance, vary according to C+G content of the background genomic DNA [37] [20]. This makes it difficult to arrive at a set of parameters that work well uniformly across the whole genome. Many programs therefore have distinct sets of parameters for different C+G% strata, and this has been shown to improve performance [102] [21].

One of the peculiarities of RNA processing not addressed by the majority of current techniques is that of alternative splicing, where two (or more) transcripts from the same gene are spliced in different ways, often giving rise to distinct proteins. It has been estimated that over half of human genes give rise to products that are alternatively spliced [112]. The most reliable way to identify such events is by

manual inspection of a collection of cDNA or EST alignments and making gene structures consistent with this set by hand. There have been recent attempts at the automation of this process, including a promising method used in the ENSEMBL automatic annotation system [60], which identifies the minimal set of gene structures that "explains" a collection of EST alignments to a region of the genome [E. Eyras, pers. comm.]. Our understanding of the signals involved in alternative splicing is still insufficient for their *ab initio* computational prediction directly [73], but it has been shown that *sub-optimal* exons (i.e. those with significant probabilities that are not part of the single most-likely gene structure) are sometimes involved in alternative splice forms of the gene transcript [20].

To end, it is worth mentioning that many transcribed RNAs are not translated into proteins but assume some other role in the cell. These include for example ribosomal RNAs (rRNAs), transfer RNAs (tRNAs), and small nuclear RNAs (snRNAs). There is considerable interest in computational methods for the identification of genomic regions that give rise to these non-coding RNAs, or ncRNAs [38]. In the absence of the normal signals present in protein-coding genes, the majority of methods work on the assumption of a consensus secondary structure for ncRNAs of a particular type. Mathematical representations of the consensus called *Stochastic Context Free Grammars* (SCFGs) [36] provide a basis for the modelling of base-paired stems that occur in RNA secondary structure, and can be aligned to a genomic sequence providing simultaneous prediction of a ncRNA and its secondary structure. This technique has been applied extremely successfully to the prediction of transfer RNAs in the human genome [75]. More recently, *pair*-SCFGs have been used to represent the long range compensating mutations that are observed in the alignment of homologous ncRNAs, allowing them to be discriminated from conserved coding regions [93].

Chapter 2

GAZE

2.1 Introduction

Many of the gene identification techniques discussed in the previous chapter have two things in common: (i) signal and content measures are used to detect components and regions belonging to genes; (ii) these are assembled into a complete gene structure prediction for the sequence that is optimal with respect to some discrimination measure, often by dynamic programming. For the first of these steps, choices must be made as to the nature of the specific signal and content measures used, for example whether to use simple weight matrices or weight array models for splice site detection or whether to use pentamer or hexamer frequencies for coding-region scoring. For the second of these steps, a choice must be made as to the *model* of gene structure over which the assembly is to be performed, i.e. how the gene components relate to each other and fit together into complete genes. It is usually the case that choices in both of these steps are made to produce a system that has the the highest possible accuracy.

As we find out more about the biological processes of transcription, RNA processing and translation, we might want to adjust or extend these gene prediction methods to reflect our increased understanding. For example, promoter identification methods, although advancing, are still not sufficiently accurate be able to

identify the 5' ends of genes with any confidence [40]. But if an improved method becomes available, then there would be much to gain from incorporating it into a gene prediction system.

Furthermore, programs that make use of similarity information (for example alignments of cDNAs to the genomic sequence) give more accurate results than *ab initio* methods (see chapter 1) where such evidence exists. If *ab initio* methods can be extended to make use of similarity evidence where it exists, we might hope to improve gene prediction accuracy without compromising the ability to predict novel genes where it does not.

The incorporation of new information into many existing gene prediction systems may not be straightforward due to rigidities inherent in their implementation. At best, knowledge of the underlying software is required, and even given this, it is often necessary to produce a custom version of the software that is designed to work *only* when the new data is present. This is true for at least four of the gene prediction systems discussed earlier [69] [92] [96] [119].

My work has focused on the development of a framework for gene prediction that decouples the assembly of signal and content data into gene structure predictions from the generation of this data itself. I have implemented a program called **GAZE** for the assembly of **features** (corresponding to signal sensors) and **segments** (corresponding to content sensors) that is tied neither to any specific signal or content detection techniques nor any assumed model of gene structure. Both of these elements are external to the system. The goal has been to provide a method for the rapid and seamless integration of new/improved methods and data into the gene prediction process.

The main novelty of GAZE is that it does not work directly with genomic DNA sequence. It instead predicts gene structures from input files containing the results of various signal and content sensors with associated scores, typically log probability ratios. These files are assumed to be in the General Feature Format [GFF; http://www.sanger.ac.uk/Software/GFF], a format which has rapidly become

a widely used standard for the exchange of gene prediction information. The assembly of this information is directed by a *configuration* file (in the eXtensible Mark-up language, XML [http://www.w3.org/XML]), which affords the user control over the validation and scoring of candidate gene structures.

This chapter describes the details of the GAZE system. I start by outlining the main approach taken, explaining how gene structures might be inferred from lists of features and segments. Next, I describe the details of the GAZE configuration file, and how it affords control over both the *validation* and *scoring* of candidate gene structures. Two technical sections follow, detailing firstly the GAZE scoring function and secondly how it is used to obtain a probability distribution over gene structures and why this is useful. I then discuss some of the innovations implemented to improve the efficiency of the algorithms, including a novel search-space pruning technique. To end, I relate the GAZE approach to other gene prediction programs and methods. The following chapters show examples of the use of GAZE for implementing gene finders, and further theory and technical issues relating to estimating parameters for GAZE.

2.2 From features and segments to gene structures

The primary input to GAZE is a file containing the results of arbitrary signal and content sensors. Each comes with a *position* on the sequence (i.e. a start and end) and a *score*. From this file, collections of *features* (from the signal sensors) and *segments* (from the content sensors) are constructed.

The GAZE approach is that a gene structure can be described by an ordered subset of specific features taken from the given collection. For example, for a sequence of 1400 nucleotides, the following describes a structure with two genes, consisting of two exons and a single exon respectively (unless otherwise stated, I will use the term "exon" to mean the protein-coding part only; this definition is inconsistent with the classical definition used by molecular biologists, but is both convenient and consistent with other literature on gene finding):

Feature	Start	End
BEGIN	1	1
start	201	203
donor	305	306
acceptor	900	901
stop	1040	1042
start	1101	1103
stop	1218	1220
END	1400	1400

whereas a list describing a structure for which the protein-coding part is found on a single exon on the reverse strand might be:

Feature	Start	End
BEGIN	1	1
stop_rev	1151	1153
start_rev	1208	1210
END	1400	1400

Gene structures that consist of no genes can also be described by an effectively empty list which includes only the features marking the beginning and end of the sequence:

Feature	Start	End
BEGIN	1	1
END	1400	1400

Given a candidate set of features, GAZE predicts genes by obtaining the ordered subset (list) of features that according to its model is most likely to correspond to the correct gene structure. It does this by assigning a score to each list and and defining the most likely gene structure to be the list with the highest score. As explained in more detail in section 2.4, the score assigned to each candidate structure is a

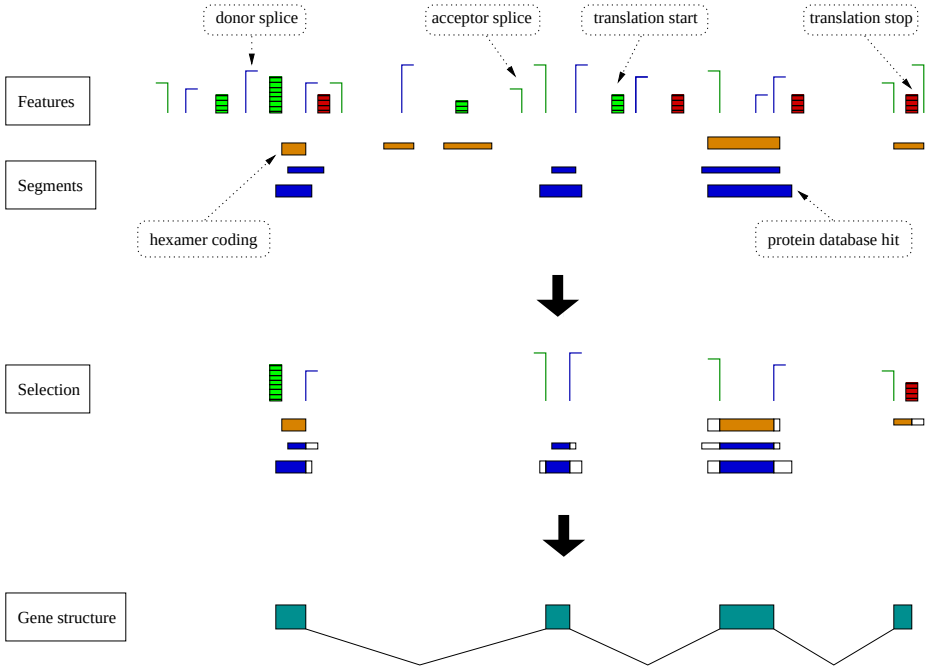


Figure 2.1: How GAZE predicts genes. The input is a list of ordered features and segments, drawn here in different sizes to reflect their scores. A candidate gene structure corresponds to a selection of these features, and is assigned a composite score based on (i) the scores of the *features* themselves and (ii) the scores of specific *segments* that lie in the appropriate regions between adjacent pairs of the features (here, both types of segment represent regions with a high likelihood of being protein-coding, so therefore contribute only towards the scores for protein coding exons). The feature selection with the highest composite score is output as the most likely gene structure.

function of the given scores of the features which make it up, and the segments lying in the regions defined by pairs of adjacent features in the structure (see figure 2.1).

Not all lists of features correspond to sensible gene structures. For example, any list which contains a donor splice site immediately followed by a stop codon cannot possibly be correct, because we know that in real gene structures, a donor splice site is necessarily followed by an acceptor splice site. Furthermore not all segments should contribute towards the score for all regions. For example, a segment corresponding to a match to a protein database should not contribute towards the score for a region between a donor splice and an acceptor splice; that is, evidence for

protein-coding regions should not be used to support candidate non-coding introns. For many gene prediction systems, such rules and constraints upon gene structure are encoded into the logic of the program itself. In GAZE however, the *gene structure model* is external to the system and supplied by the user in a *configuration* file.

2.3 Elements of a GAZE configuration

Specific examples of the GAZE configuration language are shown in the next chapter, but here I describe the elements of the most important aspect of a configuration, namely the gene structure model. The model has two main purposes: firstly to define which lists of features are *valid* gene structures, and secondly to define how valid structures are to be scored, with reference to both the segments and a set of *length penalty* functions.

2.3.1 Defining the validity of candidate gene structures

The model is initially constructed by giving a set of rules for each type of *target* feature, defining which types of *source* feature can immediately precede them in a valid structure. In the first gene structure above, a "stop" target feature can be immediately preceded upstream by "start" or "acceptor" source features, and the model would therefore need to to contain rules for $start \rightarrow stop$ and $acceptor \rightarrow stop$ (as well as others) to allow this gene structure to be recognised as valid.

The $source \rightarrow target$ rules themselves can be qualified with constraints that candidate (source, target) pairs of features should satisfy. There are four types of constraint:

Distance constraints, indicating that there should be no more than a maximum and no fewer than a minimum number of bases between the source and target.

Phase constraints, indicating that the source and target should occur 0, 1, or 2 nucleotides (modulo 3) apart.

Interruption constraints, indicating that a (source, target) pair is invalid if the region defined by the pair is interrupted by the occurrence of the specified feature at the specified distance (modulo 3) from either the source or target. These constraints are used to invalidate potential coding exons that are interrupted by an in-frame stop codon.

DNA constraints, indicating that a (source,target) pair is invalid if the DNA located at the source and/or target has a specified sequence. These constraints are used to invalidate gene structures that would give rise to in-frame stop codons across exon-exon boundaries in the spliced messenger RNA.

The space of valid gene structures can be further refined by denoting specific features in the input set as **selected** or **de-selected**. Any candidate gene structure that does not include *all* of the features flagged as selected is considered invalid. Likewise, any candidate gene structure that includes *any* of the features flagged as de-selected is also considered invalid.

Technical caveats

For practical purposes, I define the "distance" between a pair features as the length, in nucleotides, of the region between them. For example, the region between bases 567 and 890 has length $890 - 567 + 1 = 324$ nucleotides. Conceptually then, the distance between a source feature and a target feature is the location of the target minus the location of the source (plus one). A technical problem with this is that firstly features often do not have single nucleotide positions, and secondly, some features are considered part of the region they delimit, and some not. Furthermore, this can depend on whether a feature is acting as a source or a target (marking respectively the left or right boundary of the region). For example, a feature representing a candidate translation start site not only covers three nucleotides (ATG), and therefore does not have a position on the sequence that be described in a single number, but needs to be treated differently depending on whether it is a source or

a target. When acting as a source, it marks the start of a candidate protein-coding exon, and should itself be considered part of this region, whereas when it acts as a target, it marks the end of a non-protein-coding region, and should not be itself considered part of the region.

To address this problem, GAZE requires the definition, for each feature type, of a "source offset" and a "target offset". The source offset is an integer number that is *added* to the *start* position of the feature when it is treated as a source, and the target offset is an integer *subtracted* from the *end* location of the feature when it acts as a target. The translation start feature above will commonly be defined as having a start offset of 0 and an end offset of 3. So, for an instance of this feature-type occurring at 567-569 (say), the region beginning with the feature when it acts as a source starts at $567 + 0$, whereas the region ending with the feature when it acts as a target will end at $569 - 3 = 566$, i.e. one nucleotide before the given start of the feature, which has the desired effect.

The imprecise notion of feature location needs to be addressed when sorting features by their position on the sequence. The concept of a total order over all candidate features is important not only for the dynamic programming recursions described in section 2.4, but also because such an ordering is assumed when defining the space of valid gene structures; interruption constraints are violated when a designated feature C occurs between feature A and B, i.e. when the relative order that the features occur in the list is A, C, B.

Because features have both a start and an end point in sequence co-ordinates, the obvious approach is to construct the total order by sorting the features first by start point, and then by end point, and finally (in the case of 2 features having the same start and end location) by feature type. The first part of figure 2.2 shows why this sorting strategy can give incorrect results.

The problem is caused by features that overlap, in particular features involved in interruption constraints (in most uses of GAZE, stop codons). If an acceptor splice site occurs in the middle of a candidate stop codon, then the protein-coding region

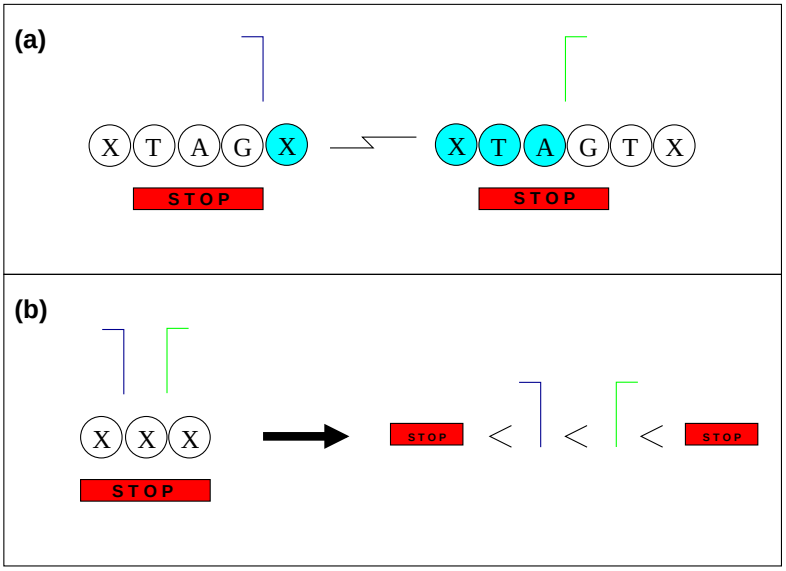


Figure 2.2: The difficulty in defining a total order over the positions of features in the sequence. (a) The candidate exon (indicated by the light-blue residues) is valid as neither candidate stop codon (red boxes) occurs completely within it. However, when the features are ordered naturally by their start points, the second stop occurs before the splice donor at the 3' end of the exon (green hook), giving it the appearance of invalidating the region. Similar problems arise when the features are ordered by their end-point. (b) In this case, we would like to engineer an ordering strategy whereby the stop codon occurs upstream of the candidate donor splice site (blue hook) and downstream of the candidate splice acceptor, with the two splices in their correct orientation. No such ordering exists. Although this specific situation is implausible (the candidate internal exon is 1 nucleotide long) such problems need to be considered because they become more likely as the range and diversity of the gene prediction data being used increases.

upstream of the splice acceptor is not invalidated by it. We would therefore like to place the stop codon *before* the splice acceptor in the list. If the same situation occurs with a stop codon and a splice donor at the 3' end of en exon, we would like to place the stop *after* the splice donor, for the same reason.

The aim therefore is to define a sorting strategy that takes this information into account when deciding upon the relative order of a pair of features. Unfortunately, it is also possible to imagine a situation where such a strategy might break down. The second part of figure 2.2 shows that a circularity in the ordering function arises when candidate splice acceptor, splice donor and stop codon appear in close proximity.

Such a circularity can lead to non-determinism in feature sorting, whereby the final ordering depends not only upon the condition dictating the relative order of a pair of features, but also on the original order of the features. For this reason, it was decided to abandon the idea of trying to determine definitive total order over a list of features. Features are therefore sorted in a natural, deterministic way, first by start-point, then by end-point, and then by type. The problem of the interruption constraints is dealt with in the dynamic programming algorithms themselves. Specifically, an apparent violation of an interruption constraint is checked to see if the interrupting feature really does lie within the region of interest. The primary disadvantage of this is that it makes the code more complicated and therefore more difficult to maintain.

2.3.2 Defining the scoring of valid gene structures

The overall score of a gene structure is the sum of of scores for the individual *features* (as given in the GFF file) and for the *regions* between each adjacent pair of features in the structure (this is defined more precisely in section 2.4). The region scores can be tailored precisely for each (source,target) pair, by specifying in the $source \rightarrow target$ rule two elements: the name of a **length penalty** function, and a list of **Segment Qualifiers**.

Length Penalty Function

Length penalty functions reflect the fact that it is often more likely for a given source and target to appear at certain distances apart than others. Each $source \rightarrow target$ rule can be qualified with the name of a length penalty function, mapping distances to a floating point number that will be subtracted from the score for the region between the source and target (see section 2.4). The functions themselves are defined by simply listing (distance, penalty) pairs, with linear interpolation used to derive penalties for distances not given. For distances greater than the largest given, the final two given points are extrapolated. It is straightforward to define penalty functions that are eventually uniform by making the penalties for the last two given

distances equal.

Segment Qualifiers

Segment qualifiers control which segments contribute to the score for the region between the source and target, and under what conditions. Since the region between an acceptor splice site and a donor splice site (for example) defines a candidate protein-coding region, both a *likely_coding* segment (indicative of a region of high protein-coding potential by some statistical measure), and a *protein_match* segment (corresponding to a region of strong similarity to an entry in a database of protein sequences) lying in this region provide evidence that the region *is* protein-coding, and can therefore be used to increase the score of the region.

Each $source \rightarrow target$ rule can contain several segment qualifiers. A separate score is calculated for each qualifier, and these are added to get a total segment score for the region (see section 2.4). The qualifiers themselves can contain constraints to restrict which segments should be considered *relevant* in the calculation of the score for that qualifier, namely:

Type constraints, indicating that only segments of the designated type should be considered. This is a compulsory constraint.

Phase constraints, indicating that that the starts of relevant segments should occur at 0, 1, or 2 nucleotides (modulo 3) away from the source or target. This gives the facility to consider only segments that are in-frame with respect to the source/target.

match constraints, indicating that the start and/or end of the segment must lie at the same position as the source and/or target. This gives the facility to consider only segments that fit the region precisely, allowing the use of the output of of programs that identify potential exact intron, exons, or other gene regions.

completeness constraints, indicating (when specified) that the segment must lie completely within the region to be considered relevant.

2.4 Prediction with a GAZE gene structure model

GAZE predicts genes by choosing from a large set of candidate features, the ordered subset (list) that (a) is consistent with the gene structure model (i.e. does not violate any constraints), and (b) has a score at least as high as all other consistent gene structures.

2.4.1 The GAZE scoring function

Given firstly a list $\phi = \phi_1, \phi_2, \ldots, \phi_n$, of features ordered by sequence position defining a *valid* gene structure according to a GAZE model, their types $t(\phi_i)$, their locations[1] on the sequence $l(\phi_i)$, and their given scores $g(\phi_i)$, then the score of ϕ, $E(\phi)$ is calculated as:

$$E(\phi) = \sum_{i=0}^{n} Reg_{t(\phi_i)\rightarrow t(\phi_{i+1})}(l(\phi_i), l(\phi_{i+1})) + g(\phi_{i+1}) \tag{2.1}$$

The features ϕ_0 ("BEGIN")and ϕ_{n+1} ("END") are not supplied by the user but are present in all gene structures and act to mark the beginning and end (respectively) of the sequence. Their "given" scores are always 0. $Reg_{src\rightarrow tgt}(x, y)$ is the region score for the interval [x,y], where the interval is bordered on the left and right by features of type *src* and *tgt* respectively:

$$Reg_{src\rightarrow tgt}(x, y) = Seg_{src\rightarrow tgt}(x, y) - Len_{src\rightarrow tgt}(y - x + 1) \tag{2.2}$$

$Len_{src\rightarrow tgt}(x)$ is the distance penalty function specified for the rule $src \rightarrow tgt$. Each function maps a distance (in base pairs) to a penalty score. If no penalty

[1] location is a function of the start co-ordinate and start offset when the feature acts as a source, and end co-ordinate and end offset when the feature acts as a target, as explained in section 2.3.1.

function is specified for the $src \rightarrow tgt$ rule, a default, mapping all distances to zero, is assumed.

$Seg_{src \rightarrow tgt}(x, y)$, the segment score, is sum of separate scores for each segment qualifier in the $src \rightarrow tgt$ rule. If $Q_{src \rightarrow tgt}$ is the list of segment qualifiers appearing in the $src \rightarrow tgt$ rule, and ψ^q is the relevant subset of segments for segment qualifier q, i.e. those that satisfy the type, phase, match and completeness constraints defined in q, then

$$Seg_{src \rightarrow tgt}(x, y) = \sum_{q \in Q_{src \rightarrow tgt}} Seg^Q(\psi^q, x, y) \tag{2.3}$$

If the given scores of each of the segments in a list ψ are denoted by $g(\phi_i)$, then $Seg^Q(\psi, x, y)$ itself is calculated in one of two ways:

$$Seg^Q(\psi, x, y) = \max_{\psi_i \in \psi} \frac{|\psi_i \cap x \ldots y|}{|\psi_i|} g(\psi_i) \tag{2.4}$$

$$Seg^Q(\psi, x, y) = \sum_{r=x}^{y} \max_{\psi_i \in \psi} \frac{|\psi_i \cap r \ldots r|}{|\psi_i|} g(\psi_i) \tag{2.5}$$

where $|\psi_i|$ is the length of segment ψ_i and $|\psi_i \cap x \ldots y|$ is the number of bases of overlap between segment ψ_i and the region $x \ldots y$. The first function corresponds to a "maximum single" approach, using the single segment with the highest score (after it has been scaled according to the proportion of segment lying in the region being considered). The second function corresponds to a "projected per-base" approach, summing the scores for the individual bases lying in the region, each of which is calculated by taking the maximal per-base score of all segments covering that base.

It is up to the user to decide which of these two functions should be used to score segments of each type. The default is the second, but some segments, by their construction, may give better results if the first function is used to score them.

2.4.2 Obtaining the highest scoring valid gene structure

A straightforward way for GAZE to obtain the highest scoring model-consistent gene structure would be to enumerate all gene structures, calculate the score for each one (a simple linear combination of terms, as shown above), and retain the structure with the highest score. This direct approach is not practicable since the number of possible gene structures grows exponentially with sequence length [116]. However, dynamic programming can be used to explore the search-space efficiently, by constructing partial solutions in a left-to-right manner, at each stage in effect discarding partial gene structures that cannot possible be prefixes of the optimal structure.

The GAZE algorithm for obtaining the highest scoring gene structure is in many ways similar to others described for the same problem ([89], [49], [109], [72], [116]). It relies upon an ordering of all candidate features by their position on the sequence. Taking now $\phi_1, \phi_2, \ldots, \phi_n$ to be a complete set of candidate features ordered by position on the sequence (with again ϕ_0 being "BEGIN" and ϕ_{n+1} being "END"), the maximal-scoring valid gene structure is obtained by the following dynamic-programming recurrence:

$$v(0) = 0 \tag{2.6}$$

$$v(i) = \max_{j<i}[v(j) + Reg_{t(\phi_j)\rightarrow t(\phi_i)}(l(\phi_j), l(\phi_i)) + g(\phi_i)] \tag{2.7}$$

The score of the optimal gene structure is $v(n+1)$, and the gene structure itself can be obtained with a *traceback* procedure. This involves maintaining a separate vector d, where $d(i)$ contains the index j that was found to be the maximum. The maximal scoring structure itself can be obtained by successively pushing features onto a stack, starting with ϕ_{n+1} (the "END" feature) and continuing with $\phi_{d(n+1)}$, $\phi_{d(d(n+1))}$ and so on until ϕ_0 (the "BEGIN" feature) has been pushed. The gene structure itself is then obtained by successive popping of features from the stack

until it is empty.

This algorithm is almost identical to the Viterbi algorithm for finding the sequence of states through a Hidden Markov Model that maximises the joint probability of the state path and the sequence. It is also very similar to Dijkstra's algorithm [32] for obtaining the shortest path through a directed weighted graph, generalised to account for negative edge-weights by Bellman [6].

The procedure above will find the gene structure with the highest score, regardless of whether it is consistent with the model or not. However, the constraints that define model consistency are all defined at the $source \rightarrow target$ level, as explained earlier. Since the dynamic procedure above works at this level too, it is straightforward to check that all constraints have been satisfied. When looping back over the sources for a given target, sources that give rise to a violation of a constraint are not considered valid sources for this target. If there are no valid sources for a given target, the target itself is invalidated, and not itself considered as a valid source for any subsequent target.

2.5 A probability distribution over gene structures

No matter how well the scoring function represents the characteristics of gene structure, it is often the case that the optimal (i.e. highest scoring) structure is not the correct one. It is therefore useful to know the relationship of the optimal gene structure to other candidate gene structures. I have adopted a probabilistic approach in assigning a *posterior probability* to firstly each input feature, and secondly each potential region (formed by candidate pairs of adjacent features). One can then ask for the features and/or regions with posterior probability greater some threshold, regardless of whether those features/regions are part of the optimal structure or not ("sub-optimal").

To calculate posterior probabilities, I first define a probability distribution over gene structures. If the given feature and segment scores are log-probabilities, then the probability of a gene structure can be calculated simply as an exponentiation of

the score. Some care is needed in the model to ensure that the whole DNA sequence is accounted for in every gene structure and that the sum of the probabilities of all gene structures sums to 1.

GAZE takes the somewhat more pragmatic stance that it is often impossible (or at the very least, extremely difficult) to formulate the scores as log-probabilities. Indeed, the scores presented by most signal-recognition programs are usually log probability *ratios* with respect to some background or "null" model. For this reason, GAZE imposes no restrictions upon the given feature scores except that they should generally increase monotonically with the degree of confidence in the correctness of the feature, i.e. that large scores are good, and small scores (or large negative scores) are bad. This means that the score for a complete gene structure can no longer be assumed to be a log-probability. A more general approach is therefore necessary.

2.5.1 Gene Structure probabilities

By treating gene structure scores $E(\phi)$ as "energy" values, we can use the Boltzmann distribution, ubiquitous in statistical physics, to define a probability distribution over all possible gene structures Φ:

$$Z = \sum_{\phi \in \Phi} \mathrm{e}^{E(\phi)} \tag{2.8}$$

$$\mathrm{P}(\phi) = \frac{\mathrm{e}^{E(\phi)}}{Z} \tag{2.9}$$

The Z of this fraction is known in statistical physics as the "partition function", and acts as a normalisation factor, making all gene structure sum to 1, satisfying the conditions for a discrete variable probability distribution.

In this formalism, the gene structure scores are interpreted as logarithms. This assumption is implicit in the design of the scoring function in that the scores of individual gene components are *added* to obtain the total score. The assumption of *natural*-logs specifically is not limiting since a log score with respect to a base k

can be transformed into a log score with respect to base e by multiplication by a constant.

The partition function can be computed with the following dynamic programming recurrence, similar to the forward algorithm for Hidden Markov Models [90], and almost identical to the score-maximisation algorithm presented earlier (the differences being the exponentiation step, and the replacement of maximisations with sums):

$$f(0) = 1 \tag{2.10}$$

$$f(i) = \sum_{j<i} f(j) \mathrm{e}^{Reg_{t(\phi_j)\to t(\phi_i)}(l(\phi_j),l(\phi_i))+g(\phi_i)} \tag{2.11}$$

Each $f(i)$ denotes the sum of exponentiated scores of all of the "upstream" partial gene structures ending at feature ϕ_i. The sum of exponentiated scores of all upstream-partial gene structures ending at the "END" feature, i.e. all complete gene structures, is contained in $f(n+1)$. The probability of a gene structure ϕ can therefore be computed as:

$$\mathrm{P}(\phi) = \frac{\mathrm{e}^{E(\phi)}}{f(n+1)} \tag{2.12}$$

This approach is essentially due to Stormo and Haussler [109], the difference there being that gene structure scores were treated as the log of the joint probability of ϕ and the sequence S, and posterior probabilities are presented as explicitly conditional upon the sequence S, i.e. $\mathrm{P}(\phi|S)$. In the GAZE framework, the sequence itself is implicit. Considering GAZE posterior probabilities as conditional upon S however leads to some interesting correspondences to other methods, particularly those involving Hidden Markov models. This is discussed briefly at the end of this chapter, and in more detail in chapter 4.

2.5.2 Feature and Region posterior probabilities

Having defined a probability distribution over gene structures, it is now possible to define posterior probabilities for features and regions. The posterior probability of a feature ϕ_i, $\mathrm{P}(\phi_i)$, is the sum of the probabilities of all model-consistent gene structures that contain the feature ϕ_i. Likewise, the posterior probability of a region $\phi_i \rightarrow \phi_j$, $\mathrm{P}(\phi_i, \phi_j)$, is the sum of the probabilities of all model-consistent gene structures that include ϕ_i and ϕ_j as adjacent pairs of features.

Informally, a feature posterior probability can be interpreted as a measure of belief in the correctness of the feature, conditional upon the surrounding gene structure landscape. More informally still, it can be interpreted as an indicator of how well it can be accommodated in a "good" gene structure.

It is straightforward to calculate the sum of the probabilities of all gene structures consistent with a feature by running the forward algorithm while using the feature selection mechanism (section 2.3) to force the inclusion of the feature. This will give a new partition function value, corresponding to the sum over all gene structures that include the feature. Dividing this by the unrestricted partition function gives the desired posterior probability. However, this strategy requires a separate execution of the forward algorithm for each feature, and the computational expense of the algorithm makes the strategy infeasible.

To make the computation more efficient, we can compute "backward" analogues of the forward variables, $b(i)$, which store the sums of the exponentiated scores of all "downstream" partial gene structures that *start* with feature ϕ_i:

$$b(n+1) = 1 \tag{2.13}$$

$$b(i) = \sum_{k>i} b(k) \mathrm{e}^{Reg_{t(\phi_i)\rightarrow t(\phi_k)}(l(\phi_i), l(\phi_k)) + g(\phi_k)} \tag{2.14}$$

It can be shown that multiplying $f(i)$ by $b(j)$ $(i \leq j)$ corresponds to summing the exponentiated scores of all possible pairings of a partial upstream gene structure

ending at feature ϕ_i with a partial downstream structure *beginning* at feature ϕ_j [109]. Hence $f(i)b(i)$ is the sum of the exponentiated scores of all gene structures that include feature ϕ_i, and the posterior probability of ϕ_i, $\mathrm{P}(\phi_i)$ is:

$$\mathrm{P}(\phi_i) = \frac{f(i)b(i)}{f(n+1)} \tag{2.15}$$

Posterior *region* probabilities can be defined in a similar way:

$$\mathrm{P}(\phi_i, \phi_j) = \frac{f(i)\mathrm{e}^{Reg_{t(\phi_i)\rightarrow t(\phi_j)}(l(\phi_i),l(\phi_j))+g(\phi_j)}b(j)}{f(n+1)} \tag{2.16}$$

A straightforward generalisation results in the posterior probability of a *partial* gene structure $\phi_i, \phi_j, \ldots \phi_x$, i.e. covering an internal sub-region of the original sequence:

$$\mathrm{P}(\phi_i, \phi_j, \ldots, \phi_x) = \frac{f(i)\mathrm{e}^{E(\phi_i,\phi_j,\ldots,\phi_x)}b(x)}{f(n+1)} \tag{2.17}$$

If ϕ_i and ϕ_x, the boundary features of the sub-region, are chosen to be the start and stop of a single, individual gene, then the above is a posterior probability for that gene. However, one of the unfortunate consequences of the general approach adopted by GAZE is that it has no knowledge of which feature-types define the boundaries of individual genes. For this reason, individual gene probabilities are not reported by GAZE.

2.5.3 Stochastic traceback

A probability distribution over gene structures offers the possibility of a *stochastic* traceback procedure. In the standard traceback procedure, we choose, for a target feature ϕ_i, the source feature ϕ_j that gives rise to the highest scoring partial gene structure ending at ϕ_i. With stochastic traceback, we instead make use of a Boltzmann probability distribution over all model-consistent sources for a given target. This follows simply from the definition of the forward variables presented earlier:

$$\mathrm{P}(k|i) = \frac{\mathrm{e}^{Reg_{t(\phi_k)\rightarrow t(\phi_i)}(l(\phi_k),l(\phi_i))+g(\phi_i)} f(k)}{\sum_{j<i} \mathrm{e}^{Reg_{t(\phi_j)\rightarrow t(\phi_i)}(l(\phi_j),l(\phi_i))+g(\phi_i)} f(j)} \tag{2.18}$$

Instead of choosing source that gives rise to the maximum partial-gene-structure score, we instead sample a source stochastically, with the relative probabilities of each source being computed with the above equation.

Stochastic traceback is an alternative method of identifying confident parts of a gene structure; parts of a gene structure that are conserved over many samples can be construed as more reliable (similar to the bootstrapping technique used in phylogenetics). Although I have provided an implementation of it in GAZE, my work has concentrated on the use of the posterior feature and region probabilities.

2.6 Practical considerations

2.6.1 Maintaining numerical stability

Implementing dynamic programming recursions in the obvious way can often lead to numerical underflows and overflows that even the most sophisticated modern floating point processor are unable to deal with gracefully. In the standard HMM formalism for example, each (state, residue) pair is assigned a probability, calculated as the product of probabilities for the state (conditional upon the previous state) and for the residue (conditional on the state). The joint probability assigned to a complete sequence of the order of a 10^6 bases, for even a simple HMM with few states, will therefore be of the order of $0.5^{1000000}$ (assuming an average probability per state-residue pair of 0.5, which is generously high). This gives an underflow error on my desktop calculator, and even though floating point units in modern processors would be expected to handle higher degrees of precision, it is not difficult to imagine a set of transition and emission probabilities for a given HMM architecture that will lead to underflow even on the most sophisticated processors.

The standard technique used in the field of HMMs is to work in log-space. Rather than multiplying probabilities, we add logarithms of probabilities. For example,

assuming that we use base 2 logarithms, $\prod_1^{1000000} 0.5$ becomes $\sum_1^{1000000} \log_2 0.5 = -1000000$. Classically, replacing many multiplications with additions would also lead to a performance improvement on some older computers. Modern floating-point unit technology makes this less true nowadays, but even on modern processors, addition should be no slower than multiplication.

For the dynamic programming performed in GAZE, the log transformation is not required in the maximum-based computation used to find the highest scoring gene structure consistent with the model (the Viterbi algorithm analogue). This is because the design of the scoring function, and its additive nature, places a log-based interpretation on the feature and region scores anyway. However, the log-transformation is required for the sum-based computations. For the forward algorithm, we define a new vector $F(i) = \ln f(i)$, and the recursions are defined in terms of $F(i)$ directly:

$$F(0) = 0 \tag{2.19}$$

$$F(i) = \ln \sum_{j<i} \mathrm{e}^{F(j)+Reg_{t(\phi_j)\rightarrow t(\phi_i)}(l(\phi_j),l(\phi_i))+g(\phi_i)} \tag{2.20}$$

The posterior probability of a gene structure is now calculated as:

$$\mathrm{P}(\phi) = \mathrm{e}^{E(\phi)-F(n+1)} \tag{2.21}$$

and the posterior feature probabilities calculated thus:

$$\mathrm{P}(\phi_i) = \mathrm{e}^{F(i)+B(i)-F(n+1)} \tag{2.22}$$

It is necessary to perform one last trick to avoid overflow when performing the exponentiations in 2.20. We can use the following observation:

$$\ln \sum_{x=a}^{b} \mathrm{e}^{x} \quad = \quad \ln \left(\mathrm{e}^{k} \sum_{x=a}^{b} \frac{\mathrm{e}^{x}}{\mathrm{e}^{k}} \right) \tag{2.23}$$

$$= \quad k + \ln \sum_{x=a}^{b} \mathrm{e}^{x-k} \tag{2.24}$$

Any k can be used, but by storing the exponents in the summation, and choosing k to be the maximum of these exponents, we ensure that all exponentiations are 0 or less, eliminating the possibility of overflow.

2.6.2 Working within practical limits of space and time

Complexity of naive implementation

As noted earlier, the dynamic programming recursions for identifying the highest scoring gene structure (2.7) and for calculating the "partition function" over all gene structures, (2.11, 2.14) are essentially the same. The run-time and memory usage of the algorithms depends of course on the specific problem, but we can use complexity theory to reason about the growth in the requirement of these resources with respect to the size of the input. For GAZE, the algorithms proceed over input features, but since the number of features for a given DNA sequence would be expected to grow linearly with the length of the sequence, it makes no difference whether we define the problem size in terms of sequence length or in terms of the number of features. It is therefore convenient to talk about a problem size of n, which can be interpreted both as the length of the sequence region being considered, or the number of features attached to that sequence region.

Because the dynamic programming recursions are one-dimensional, the recursion variables can be stored as a vector rather than a matrix (as is the case with classical sequence alignment dynamic programming). The storage requirements are therefore $O(n)$.

Examination of 2.7, 2.11 and 2.14 shows that any algorithm must essentially examine all feature pairs in the list ϕ and perform a region score calculation. Since there are $\frac{n}{2}(n+1)$ pairs, this implies $O(n^2)$ region score calculations. Each region score calculation involves a length penalty component (which can be calculated in constant time by table look-up), and a segment score component. Assuming that

both 2.4 and 2.5 are implemented the way suggested by their definition, then in the worst case the segment calculation is linear in the number of segments. Although the actual number of respective features and segments for a given sequence may be, and often are, quite different, we would expect them to both scale in the same way with respect to the length of the sequence, i.e. linearly. This means that the segment calculation can be described as $O(n)$ in complexity, giving the algorithms a run-time complexity of $O(n^3)$ overall, making it apparently prohibitively expensive for large sequences.

In the remainder of this section, I outline two methods employed in the implementation of GAZE to improve both the theoretical worst-case and practical average-case run-time and storage complexity. In the next subsection, I describe a novel search-space pruning strategy employed in GAZE, which is the biggest contributing factor to its efficiency.

Segment pre-processing

The segment calculation for a candidate region, as defined by 2.3, 2.4 and 2.5, consists of a separate calculation for each segment qualifier listed in the rule that applies to the feature-pair defining the region. Each of these calculations in turn requires an ordination of the list of segments, for each firstly checking that it meets constraints defined in the qualifier, and secondly scaling the score according to how much of lies in the region of interest. Partitioned storage of the segments, primarily by type, but also by reading frame, allows the consideration of a much smaller list of candidate segments for a given segment qualifier, but the number of segments that need to be examined is still $O(n)$.

Since segments falling outside the considered region do not contribute to the score, sorting the segments by position along the sequence is the natural starting point towards reducing the number of segments that need to be examined. By sorting the segments by start position on the sequence, the segment with the *leftmost* start lying completely to the *right* of the end of the considered region can be identified

by a simple binary search. All segments to the right will lie completely outside the region. It is tempting to assume that the segment list can now be processed from this point to the left, stopping when a segment that has an end-position that is strictly to the left of the start of the considered region. In general segments may overlap and in the extreme the segment with the smallest start position may have the largest end position. In that case it will always be necessary to traverse leftwards from the point in the list identified by the binary search, right back to the start of the list. However, the rarity of this situation can be exploited by *indexing* the segments. In essence, we calculate and store an additional piece of information for each segment: the maximal right-position of all segments to the left:

$$I(\psi_i) = \max_{j<i} \psi_j.end \tag{2.25}$$

$$= \max(I(\phi_{i-1}), \psi_{i-1}.end) \tag{2.26}$$

The second equality gives rise to a simple linear-time dynamic programming algorithm to calculate the segment indices, and this only needs to be performed once for each segment list, before the main score/probability dynamic programming.

It is interesting to note that although partitioned storage and indexing of the segments improve the worst-case time spent performing the segment score calculations, they do not change the theoretical worst-case *complexity*. At the extreme, the when the region being considered encompasses the whole sequence the computation is still O(n). This is a classic example of where worst-case complexity is a misleading indicator of the expected increase in run-time with problem size.

As a final note on this subject, it must be said that it *is* straightforward to implement the segment calculation in such a way as to make it constant time. This technique is employed in several gene prediction programs, but not GAZE. The technique involves keeping cumulative per-residue arrays for each specific set of constraints referred to in the segment qualifiers of a model. If ψ^q is the list of segments that match a specific set of segment qualifier constraints q then the cumulative array for this constraint set, C_{ψ_q} can be defined as:

$$C_{\psi^q}(i) = C_{\psi^q}(i-1) + Seg^Q(\psi^q, i, i) \tag{2.27}$$

where $Seg^Q(\psi, x, y)$ is as defined in 2.5. Like the segment indexing presented earlier, the $C_{\psi^q}(i)$ arrays can in theory be calculated in linear time in advance of the main dynamic programming. Then, the segment calculation performed during the Viterbi, forward and backward algorithms becomes:

$$Seg^Q(\psi^q, x, y) = C_{\psi^q}(y) - C_{\psi^q}(x-1) \tag{2.28}$$

There are a number of reasons why this technique is not implemented in GAZE. Firstly, it requires much more memory. It is true that this will only become as problem when very large sequences are being analysed (of the order of Megabases long), but for small sequences, the run-time reduction afforded by the technique becomes negligible.

Secondly, and most importantly, it is only applicable for a specific kind of segment scoring, namely the "projected per-base" approach defined in 2.5. In addition, the technique requires the segments to be partitioned in advance into lists matching each segment qualifier referred to in the model. However, for some constraints (namely "completeness" and "exactness"), it is not known until the dynamic programming stage which segments will satisfy them. The cumulative arrays technique is therefore only available to segment qualifiers with no completeness or exactness constraints, scored by equation 2.4. Given its limited applicability, it was decided insufficiently worthwhile to implement.

Split-and-merge for whole genome analysis

Although the space requirements of GAZE are linear in the length of the sequence, the memory of a standard desktop computer is not likely to be sufficient to handle feature-sets from the complete genomes of eukaryotic organisms. To analyse whole genomes, it is necessary to design a method that is constant in its memory usage, regardless of the size of the input feature-set. Such an aim is not fanciful, especially

considering that the dynamic programming search-space pruning method (presented below) makes the algorithm effectively local. One possible strategy might therefore be to discard elements of the V, F and B vectors, as well as the features corresponding to these elements, when they are not needed any more. This idea is similar in essence to linear-space sequence alignment methods [80]. However, although it is possible to obtain the score of the optimal gene structure in this way, the gene structure itself is more difficult to obtain; the standard trace-back procedure is no longer possible.

A natural constant-memory method for obtaining both the highest scoring gene structure and posterior feature and region probabilities involves *off-lining*, where parts of the dynamic programming structures, and the features the themselves, are written to disk, discarded from memory, and read back in when required. I have implemented such a method, but approached the problem from a slightly different angle. The technique is based upon a split-and-merge strategy, which conceptually involves splitting the input into several manageable chunks, running GAZE separately on each one, and finally merging the results together. This has been implemented in a Perl script called GENOME_GAZE.

The first stage of GENOME_GAZE is the split, but GAZE itself provides an option to make this stage trivial. Specifically, it can be told to consider a specified subsequence window of the given arbitrary sized query sequence / feature-set; the input DNA sequence and features sets do not therefore have to be physically split at all. The "split" phase of GENOME_GAZE therefore involves running GAZE on windows $w_1 \cdots w_k$ to produce output files $o_1 \cdots o_k$, where the window size is chosen according to the available computational resources (the bigger machines available, the bigger the window size can be), and k is chosen to be large enough to cover the whole input sequence region with a specified overlap between w_i and w_{i+1}. The overlap allows the second, "merge" phase of GENOME_GAZE to be performed.

An overlap is necessary between subsequence windows w_i and w_{i+1} because there may be cases where a predicted gene structure straddles the boundary between

two windows. The final output gene prediction is formed generally by pushing the features in the output files onto a list in sequence order from o_1 to o_k. However, for the first feature ϕ_x in o_i that lies in a region that is also covered by the start of w_{i+1}, o_{i+1} is searched for the occurrence of that feature. If it is found, the rest of the features in o_i are ignored, and the pushing of features continues from that point in o_{i+1} at which ϕ_x was located. If it is not found, ϕ_x is pushed onto the list, and o_{i+1} is searched for the occurrence of ϕ_{x+1} in o_i. This continues until the appropriate cross-over point from o_i to o_{i+1} is identified.

When the output files o_i consist not of feature-lists representing predicted gene structures, but the complete input feature set ordered by sequence position, with posterior probabilities attached to each feature, a slightly simpler strategy suffices, where the midpoint of the overlapping region between o_i and o_{i+1} is chosen as the cross-over point between the two regions.

Split-and-merge offers a natural parallelisation strategy, because each window w_i can be analysed independently of other windows. Only the final stage of forming the consensus gene structure for all windows relies upon their order in the sequence, and therefore cannot be performed until all windows have been processed first by GAZE. This is one of the key advantages of post-processing approach via GENOME_GAZE over a split and merge algorithm in GAZE itself.

2.6.3 A novel pruning strategy

Earlier it was described how the region-score calculation is made less computationally expensive in order to reduce the overall run-time of the algorithm. An complementary approach is to reduce the number of region calculations that need to be performed, which is essentially $O(n^2)$ in the number of features. To this end, two pruning strategies have been implemented, one exact and one heuristic. Both rely upon the examination of the search-space in a directed manner.

In the calculation of the F vector outlined earlier (and likewise for the V and B vectors), the elements need to be computed in a directed way, starting with $F(0)$

and ending with $F(n+1)$. The calculation of $F(i)$ relies upon the values $F(j)$, $j < i$, which corresponds to examining the source features ϕ_j for a given target feature ϕ_i. These sources *need not* be visited in any directed way, but doing so provides opportunities for pruning. In particular, sources for a given target are examined firstly in order of type, and secondly (for sources of a given type) in order of proximity to the target, i.e. $\phi_{i-1}, \phi_{i-2} \ldots \phi_0$.

A pruning strategy based upon model constraints

The first method makes use of the fact that certain constraints specified in the model can be used to prune away partial gene structures that cannot possibly be model-consistent. In particular, when scanning back through the sources for a given target ϕ_k, violation of an interruption or maximum distance constraint by the region $\phi_i \rightarrow \phi_k$ defined by a specific source ϕ_i means that all subsequent regions $\phi_j \rightarrow \phi_k$, $j < i$ will violate the same constraint. Therefore sources ϕ_j $(j < i)$ need not be considered for target ϕ_k.

Most models used in practice will not contain interruption or maximum distance constraints in many rules. In fact, for the models explained in chapter 3, only rules defining protein-coding regions make use of interruption constraints, to disallow in-frame stop codons. A more general pruning strategy is therefore necessary.

A pruning strategy based upon Dominance

The main idea of the strategy is based upon this aggressive assumption: when accumulating information for gene structures ending at a particular target feature ϕ_k, if the contribution made by source ϕ_j is insignificant when compared with that made by another source ϕ_i, then we can *ignore* ϕ_j as a potential source for all subsequent targets of the same type as ϕ_k. I formalise a general approach based upon this idea, and then consider cases where the main assumption might not hold, refining the strategy at each step. The method is presented in terms of the computation of the forward score $F(i)$, but the method applies equally to the backward ($B(i)$) and

Viterbi ($V(i)$) calculations.

To formalise the notion, I introduce the concept of *dominance*. For a given target feature ϕ_k, and two valid source features *of the same type*, ϕ_j and ϕ_i, where $j < i$, ϕ_i *dominates* ϕ_j if the contribution made to forward score for ϕ_k ($F(k)$) by the component involving ϕ_j is insignificant (given the limits of machine precision) compared to the contribution made by the component involving ϕ_i. More precisely:

$$\text{Dom}(\phi_i, \phi_j, \phi_k) \qquad \text{if} \qquad \text{Rdiff}(\phi_i, \phi_j, \phi_k) > X \tag{2.29}$$

The right-hand-side can be read as the relative difference between the contribution made to the forward score of ϕ_k by respectively ϕ_j and ϕ_i, and is defined thus:[2]

$$\begin{aligned} \text{Rdiff}(\phi_i, \phi_j, \phi_k) &= Reg_{t(\phi_i)\rightarrow t(\phi_k)}(l(\phi_i), l(\phi_k)) + F(i) \\ &\quad - Reg_{t(\phi_j)\rightarrow t(\phi_k)}(l(\phi_j), l(\phi_k)) - F(j) \end{aligned} \tag{2.30}$$

It is important to note that because the F vectors are computed in log-space, a *difference* of X between the ϕ_i and ϕ_j components means that the former is e^X times greater than the latter in probability space. A small value for X (20-30) is therefore sufficient for ϕ_i to dominate ϕ_j.

The pruning strategy relies on the fact that, at least under certain conditions, the dominance is *time-invariant*. That is, if ϕ_i dominates *all* sources of the same type ϕ_j (for $j < i$) with respect to a target ϕ_k, then it will dominate the same sources for subsequent downstream targets ϕ_q of the same type as ϕ_k. When considering potential sources for a ϕ_q, we need not therefore search back further than ϕ_i. The feature ϕ_i in this case is an *omnipotent* source of its type with respect to targets of the same type as ϕ_k.

[2] the contribution to the forward score also contains the given score of ϕ_k, but since this is the same for sources i and j, it cancels. In the calculation of the *backward* score however, the relative difference includes terms for the given scores of targets ϕ_i and ϕ_j.

In the implementation, a matrix $Onmi(src, tgt)$ is maintained which stores for each $src \rightarrow tgt$ rule, the index of the current omnipotent source of type src for targets of type tgt. As the dynamic programming progresses, the dominance condition is continually checked, and the $Omni$ matrix updated. The desired effect is that at any time in the algorithm k, $Omni(src, t(\phi_k))$ will not be much less than k for all source feature types src. This means that only a constant number of sources need to be examined for each target, rather than $O(n)$ sources.

The assumption of time invariance underlying this pruning technique is important for its soundness. In order to reason about the conditions under which time invariance holds, it is convenient to re-define the assumption in terms of the relative difference between the contribution made to the forward score of ϕ_k by respectively ϕ_j and ϕ_i (equation 2.30). The assumption is therefore as follows: this quantity will remain constant or increase when measured with respect to subsequent downstream targets ϕ_q of the same type as ϕ_k, preserving the domination condition. However, this assumption is *not* valid in the following circumstances:

1. Feature ϕ_q is in a different reading frame to ϕ_k, the relevant rule includes a phase constraint, and the region $\phi_i \rightarrow \phi_q$ violates this constraint; in this case, the omnipotent feature is not even a valid source for ϕ_q.

2. The region $\phi_j \rightarrow \phi_k$ violates a DNA constraint which is not violated for the region $\phi_j \rightarrow \phi_q$.

3. The region $\phi_i \rightarrow \phi_q$ violates a DNA constraint, even though $\phi_i \rightarrow \phi_k$ did not.

4. Feature ϕ_j is located at the same position as the upstream end of an "exact_match" segment; feature ϕ_k lies upstream from the location of the downstream end of this segment, which therefore does not contribute towards the score for region $\phi_j \rightarrow \phi_k$. Feature ϕ_q however lies at the same location as the downstream end of the segment, which *does* contribute to the score for region $\phi_j \rightarrow \phi_q$.

5. The $src \rightarrow tgt$ rule includes a length penalty function, and difference in the penalties for $\phi_j \rightarrow \phi_k$ and $\phi_i \rightarrow \phi_k$ is *greater* than the difference between penalties for $\phi_j \rightarrow \phi_q$ and $\phi_i \rightarrow \phi_q$.

In all these cases, it is wrong to consider ϕ_i to be an omnipotent source of that type with respect to features of type ϕ_k. It is therefore necessary to revise the ideas of dominance and omnipotence, addressing each of the problems in turn.

The first problem can easily be addressed by adding an extra "absolute-reading-frame" dimension to the *Omni* matrix. This allows us to represent a different set of omnipotent sources not only for each target type, but for each target-type in each absolute reading frame.

The second and third problems can be dealt with by revising the domination criterion with a $DNA\_constraint\_condition$ which states that ϕ_i does *not* dominate ϕ_j with respect to ϕ_k if (a) $\phi_j \rightarrow \phi_k$ is violated by a DNA constraint, or (b) ϕ_i has the potential to violate a DNA constraint for a future region $\phi_i \rightarrow \phi_q$.

The fourth problem is addressed by adding a $Exact\_Segment\_condition$ to the domination criterion which states that ϕ_i does *not* dominate ϕ_j with respect to ϕ_k, if (a) the relevant rule contains a segment qualifier with an "exact match" condition and (b) a segment of the appropriate type begins at the same location ϕ_j but extends *beyond* feature ϕ_k.

The final problem is very likely to occur, because many useful length penalty functions used in practice will have this behaviour. My approach is to remove the length-penalty component from the domination condition. Equation 2.30 then becomes:

$$\begin{aligned} \text{Rdiff}(\phi_i, \phi_j, \phi_k) &= Seg_{t(\phi_i)\rightarrow t(\phi_k)}(l(\phi_i), l(\phi_k)) + F(i) \\ &\quad - Seg_{t(\phi_j)\rightarrow t(\phi_k)}(l(\phi_j), l(\phi_k)) - F(j) \end{aligned} \tag{2.31}$$

In doing this we must now consider the possibility that ϕ_i dominates ϕ_j by the above criterion, but *only* when the length penalty component is ignored, i.e. if

we have inadvertently extended the criterion in trying to restrict it. If the length penalty for $\phi_j \rightarrow \phi_k$ is greater than that for $\phi_i \rightarrow \phi_k$, then ϕ_i will still dominate ϕ_j even if the penalties are included in the comparison. Although many length penalty functions used in practice would be expected to be monotonically increasing with distance, one of the strengths of GAZE is that it allows the definition of arbitrary length penalty functions. However, it is likely that this facility will be most useful in the early, small-distance portion of the functions. All functions used in practice will be *eventually* monotonically increasing (or at least monotonically constant); a length function for which this is not the case implies a kind of region that, in the limit, become *more* likely the longer it is. So, assuming that each function has a distance d at which it becomes monotonic, the relative difference *including* the length penalty (2.30) is at least as big as the relative difference *ignoring* it (2.31) if the distance from ϕ_i to ϕ_k is bigger than d. If this is the case, then $\phi_j \ldots \phi_k$ will also be in the monotonic part of the function, and furthermore so will the regions $\phi_j \ldots \phi_q$ and $\phi_i \ldots \phi_q$, for all subsequent ϕ_q of the same type as ϕ_k.

It is straightforward to derive the monotonic point for length penalty function p, Mpoint(p):

$$\text{Mpoint}(p) = \min_{x=0}^{\infty} s.t. \forall yz \left[(x < y < z) \rightarrow p(x) \leq p(y) \leq p(z)\right] \tag{2.32}$$

Given these considerations, I formulate a revised, and final, domination condition:

$$\text{Dom}(\phi_i, \phi_j, \phi_k) \quad \text{if} \quad \begin{array}{l} \text{Rdiff}(\phi_i, \phi_j, \phi_k) > X \quad \text{and} \\ Exact\_Segment\_Condition \quad \text{and} \\ DNA\_Constraint\_Condition \quad \text{and} \\ l(\phi_k) - l(\phi_i) \geq \text{Mpoint}(Len_{t(\phi_i) \rightarrow t(\phi_k)}) \end{array} \tag{2.33}$$

The domination/omnipotence pruning strategy means that in practice, it is only ideally necessary to consider a constant number of sources for each target. This means that that effective number of pairwise feature comparisons (and therefore

region-score calculations) necessary is now effectively $O(n)$. When considering also the segment indexing strategy explained earlier (which makes the region-score calculation effectively $O(\log n)$), the run-time of complexity of GAZE can be described as pseudo log-linear. My experience shows log-linearity to be an upper bound, at least on my own library of GAZE models (see chapters 3 and 5); many models display linear growth in run-time with sequence length.

Linear-time dynamic programming algorithms for gene prediction have been published before. In particular, there is at least one example of a linear-time algorithm over an external model of gene structure ([52], and section 2.7). However, the flexibility of GAZE in allowing user-defined length-penalty functions and segments makes it difficult to design a linear time algorithm for obtaining both the highest scoring gene structure, and posterior feature and region probabilities.

2.7 Relationship to other similar systems

2.7.1 Other gene prediction toolkits

Although the signal and content detection techniques used by most existing integrated gene prediction systems are hard-coded in the software, their specific parameters are usually abstracted into an external file which is read at run-time. This allows the development of distinct parameter sets for different organisms for example. In this way, the majority of systems are configurable. Some systems however, like GAZE, take this idea further in attempt to provide a "toolkit" for the development of gene prediction methods.

Dong and Searls [33] for example represent the rules of gene structure as formal grammars [58]. They have constructed a toolkit for the graphical definition and computational parsing of certain kinds of grammars, based upon the Prolog programming language. They have used their toolkit to produce the GENLANG gene prediction program.

The work of Gelfand, Roytberg and co-workers is comparable to GAZE in the

that it forms the basis of a research tool for the investigation of gene prediction methods. Their technique, called *Vector Dynamic Programming*, identifies the *set* of gene structures that is guaranteed to contain the "optimal" structure with respect *all* scoring functions that adhere to certain mathematical properties [95]. The fact that this set is usually orders of magnitude smaller than the complete set of all possible gene structures allows for the rapid investigation of several different scoring functions. They have used their tool to design the scoring function that is used in the GREAT program [46].

The GENAMIC algorithm [52] lying at the heart of the GENEID system ([89], [84]) has many elements in common with GAZE. In particular, it accepts as input a list of scored candidate exons in GFF, and also a model for how the different types of exon fit together into complete gene structures. It then identifies the highest scoring exon assembly consistent with the model rules. Although similar in these respects, there are notable differences between GAZE and GENAMIC. Firstly, GAZE works with the signal and content data *before* it has been pre-processed to produce a set of candidate exons with pre-assigned frames and scores. This gives greater flexibility in the way in which external evidence is incorporated. Secondly, the model constructs offered by GENAMIC are more restricted than those offered by GAZE. In particular, GENAMIC allows the specification of a minimal and maximal distance between exons, but not arbitrary length penalty functions. Thirdly, GENAMIC does not compute posterior probabilities. These last two differences in particular however mean that the dynamic programming recursions of GENAMIC are less general and therefore more amenable to optimisation; the highest scoring gene structure is identified by means of a algorithm for which the run-time grows strictly linearly with the number of candidate exons, making it extremely fast.

Some of the motivations for the DYNAMITE system [9] were similar to those for GAZE. It is based upon the observation that many differing applications in bioinformatics have at their heart quite similar dynamic programming algorithms. Furthermore, implementation of these algorithms can be time consuming and error-

prone. DYNAMITE provides a simple language for the specification of such dynamic programming recursions, allowing large and complex models to be defined in an intuitive way. A compiler then generates code (in C) for the specified recursions that can be linked into a stand-alone application. DYNAMITE differs from GAZE primarily in the way that it is designed for *sequence alignment*, rather than feature selection. It is therefore particularly suitable for development of sequence-similarity based gene prediction applications; a large part of code in the GENEWISE program [10] for example was generated by the DYNAMITE compiler.

2.7.2 HMM methods

In outlining some of the common methods for the prediction of gene complete gene structures in chapter 1, I drew a distinction between gene fragment assembly techniques and Hidden Markov models. I have presented GAZE from the angle of gene fragment assembly, but it can also be viewed as a kind of Generalised Hidden Markov model. To see the correspondence, it is instructive to look at other systems based explicitly on GHMMs. GENSCAN [21] and GENIE [72] are examples of such.

Both GENIE and GENSCAN work in practice by first scanning the sequence for candidate state transitions. These are used as anchor points for a dynamic programming procedure to identify the state path with highest probability. In this way, they can be viewed feature-based methods (like GAZE), rather than classical single-base-at-a-time HMMs (for example HMMGENE [68]).

The advantage that GHMMs have over standard Hidden Markov models is that they allow the lengths of the regions to be modelled by arbitrary, non-geometric probability distributions. This is particularly useful because the lengths of protein-coding exons in particular are not geometrically distributed (see chapter 1). This aspect of GHMMs is reflected in GAZE by the length penalty component of the scoring function. For reasons of efficiency, GENSCAN in particular restricts the use of fully defined length probability distributions to alternating states (in practice those corresponding to protein-coding regions). The search-space pruning in GAZE means

that it is not limited in this way. It is not clear whether this is the case for GENIE.

In GENSCAN, the dynamic programming is performed over an assumed, fixed GHMM architecture. Like GAZE, GENIE is not subject to the same restriction. Signal and content sensors are treated as external modules which results in a "plug-and-play" architecture. Unlike GAZE however, it is necessary for GENIE to make specific assumptions about the scores reported by the components, namely that they are probabilities. The treatment of feature scores by GAZE as arbitrary "energies" makes it strictly more general.

Since the emission probabilities of a GHMM can be reflected by feature and segment scores, and the length probability distributions by length penalty functions, only the transition probabilities of a GHMM do not have a direct analogue in GAZE. They can however be represented by adjusting the appropriate length penalty function. Since it is possible to define a distinct penalty function for every pair of feature types, this is fully general. The disadvantage of such an approach is that the resulting models lack the intuitive appeal of a finite state automaton.

One of the key aspects of HMMs is that they are fully probabilistic, defining a joint probability distribution over gene structures *and* sequences. Careful model construction and scoring of features and segments can allow GAZE gene structure scores to correspond directly to (log) joint probabilities, although this will often not be possible when using data from external sources. Taking this idea to the extreme, the representation of a standard base-at-a-time HMM is also possible in GAZE. This could be done by having a feature type for each state, and a specific feature of each type for each position of the sequence, with emission probabilities represented by the feature scores and transition probabilities by length penalties[3]. This of course would take away one of the advantages of GAZE, namely the representation of a large number of DNA bases by a relatively small number of sequence features.

[3]Interruption constraints would need to be used to ensure that only the previous base is considered at each stage of the dynamic programming.

Chapter 3

Using GAZE for gene finding in *Caenorhabditis elegans*

3.1 Introduction

This chapter documents my work in applying the GAZE system to the prediction of gene structures in the genome sequence of the nematode worm *Caenorhabditis elegans*. Much work in the past has focused on gene prediction in the sequences of vertebrates (particularly human), and a later chapter shows how GAZE can be applied effectively to vertebrate genomes. However, GAZE was originally envisioned as a curation tool for *C.elegans* sequence annotators, and this work on its application to the worm genome is rooted in the origins of the project.

Although gene prediction in *C.elegans* sequences is considered by most researchers to be easier than in vertebrate sequences, certain complications make it a non-trivial problem (see section 1.5). Indeed, the author of one of the most widely used and accurate gene prediction programs [21] has admitted difficulty in arriving at a set of parameters that work well on worm sequences [http://genes.mit.edu/-Limitations.html].

Below, I outline the steps involved in the definition of a configuration from first principles, starting with only candidate gene features generated from simple signal

and content sensors. I then go on to successively refine this model, at each stage explaining both specific steps taken and exploring the resulting impact on prediction accuracy. In particular, I make use of the flexibility of GAZE to firstly take account of a post-transcriptional modification process that is peculiar to *C.elegans* and similar animals, namely *trans*-splicing; and secondly to improve accuracy by the incorporation of similarity information, specifically alignments of Expressed Sequence Tags.

3.2 Gene prediction materials for *C.elegans*

3.2.1 WormBase and The WormSeq dataset

To evaluate the accuracy of the various gene-structure models presented in this chapter, both in comparison with each other and with other gene prediction programs, it is necessary to construct a test-set. The principle source of data for the test-set I have constructed was WormBase[1] [107], a database containing the complete, annotated genome sequence of *C. elegans*.

WormBase provides as part of its annotation a complete set of gene structures for the *C. elegans* genome. These gene structures represent manually-inspected predictions of the *coding* regions (or *CDS*) of the genes, based on GENEFINDER predictions (see below) and other available supporting evidence for the structure. Such evidence most frequently comes in the form of *spliced alignment* of expressed sequence (cDNAs or ESTs) back to the genomic sequence by a program such as EST_GENOME [79] or BLAT [61], giving the intron-exon junctions of the structure (see chapter 1). As more cDNAs and ESTs are sequenced and deposited in nucleotide databases, they are aligned to the genome and the set of curated gene structures revised and updated to be consistent with the alignments. At any one time then, the set of curated gene structures in WormBase represents a current "best-guess" based on the available supporting evidence.

[1] Specifically WS52, September 2001

The test-set I have built represents an attempt to identify the subset of curated gene structures that have sound and complete supporting evidence for their validity. An initial set was constructed by taking the curated structures supported by the alignment of at least one "external" cDNA to the genome (i.e. those deposited in the EMBL database by a group not working directly on the *C.elegans* genome sequencing project). Since these alignments (produced by EST_GENOME, [79]) were only present (at the time of construction) for the half of the worm genome that is maintained at the Sanger Institute ("WormBase_Sanger"), the set contained only Sanger Institute genes. However, the restriction to external cDNAs provides a degree of independent verification to the structures.

The initial set of gene structures was then subjected to a set of filtering steps, removing the following entries:

1. Those for which the set of supporting cDNAs did not contain at least one entry annotated as having "complete CDS" in its EMBL entry. It is possible for a structure to be confirmed by two separate partial cDNAs aligning to different parts of the structure, but I took the conservative approach of removing such entries.

2. Those structures that overlap with at least one other curated gene structure. This removes those genes that are known to be alternatively spliced, and also those situated within the introns of other genes.

To check the consistency of the gene structures with respect to the cDNAs supporting them, I performed a Smith-Waterman [100] local alignment of each cDNA to the CDS of the corresponding gene structure, using the program DNAL (E. Birney, unpublished). Those entries where the gene CDS did not align precisely with the EMBL-annotated CDS of the cDNA were presented to the Sanger Institute *C.elegans* curation group for examination, which resulted in the editing of some of these gene structures. The final set consisted of 325 gene structures (157 situated on the forward strand of the genomic sequence and 168 on the reverse). The average

number of exons per gene in the set is 6.9 (compared to 6.3 for all curated gene structures in WormBase), with 16 single-exon genes.

Traditionally, assessments of gene prediction programs have been performed against sequences that contain a single gene for which the structure has been confirmed [23] [94]. However, when gene prediction programs are used in a production environment on large, unannotated fragments of genomic DNA, they cannot know in advance how many genes, if any, the sequence contains. Modern programs are therefore capable of predicting many genes in a query sequence. To assess the multiple-gene-prediction capabilities of such programs, test sequences containing several genes are necessary. Large genomic fragments for which the entire exon-intron structure for all genes has been confirmed are extremely hard to come by however. There is also the problem of having confidence that we know about all of the genes on a test sequence, and that the regions annotated as intergenic really do contain no genes.

Having identified a set of 325 cDNA-confirmed genes, it would be ideal if they were located together in the worm genome. Unsurprisingly, this was not the case, and the genes are spread across 9 genomic contigs. In an attempt to recreate the conditions faced by gene-prediction programs in practical use, I constructed an artificial genomic contig containing the confirmed genes. Unlike the dataset made by Guigo and fellow workers [53], where the intergenic regions were generated "randomly", I took the approach of embedding the gene sequences in real genomic DNA. Specifically, the DNA underlying each confirmed gene structure was extracted, along with half of the intergenic DNA to the nearest other *curated* gene (confirmed or unconfirmed) in each upstream and downstream direction. These sequences were then concatenated in the order in which the were situated on the original genomic contigs to make a contiguous artificial genomic sequence of 2,079,582 base pairs. This sequence is referred to as *WormSeq*. The proportion of WormSeq that is protein-coding is 0.24, which is representative of estimates for the genome as a whole based upon all curated structures in WormBase.

The WormSeq sequence, and its annotated gene structure, forms the basis for much of the analysis presented in this chapter. It can be obtained from http://www.sanger.ac.uk/Software/analysis/GAZE/wormseq.

3.2.2 A source of gene prediction data: GENEFINDER

GAZE is not an integrated gene prediction program; it requires a set of features, segments, and length penalty functions. For the analyses presented in this chapter, the GENEFINDER program (P. Green, unpublished) was the effective source for these data. Although unpublished, GENEFINDER is widely regarded as one of the most accurate gene prediction programs for the worm. The GAZE scoring function explained in chapter 2 is similar to the one used in GENEFINDER in that the score for a complete gene structure is comprised of a sum of the scores of the features that define the structure, along with scores for the regions between these features. Also, GENEFINDER includes length penalty files and frequency tables for various gene features, the details of which are described below. Permission for use of these files was kindly granted by the author [P. Green, pers.comm].

The ACeDB package [www.acedb.org] contains a module adapted from the original GENEFINDER code, GF_FEATURES, that takes as input a set of GENEFINDER frequency tables and a query DNA sequence and produces predictions of features corresponding to the given tables, in GFF. The GF_FEATURES program was used together with the tables from the 980506 distribution of GENEFINDER to produce features and segments in the manner described below for all of the analyses in this chapter, unless stated otherwise.

Signal sensors in GENEFINDER

The GENEFINDER tables are used to construct weight matrices of log-likelihood ratios of nucleotide b in position i for true sites compared to randomised DNA. Specifically, if for a feature of interest *True* and *Rand* are respectively the tables for the true sites and randomised DNA, then the log likelihood-ratio for nucleotide b in position

i is calculated as

$$llr_i(b) = ll_i^{True}(b) - ll_i^{Rand}(b)$$

where the log likelihood calculated from a table T, ll_i^T is:

$$\begin{aligned} ll_i^T(b) &= \log(1 + \frac{1}{T_i(b)})T_i(b) \\ &- \log(1 + \frac{1}{\sum_j T_i(j)}) \sum_j T_i(j) \\ &+ \log(\frac{1 + T_i(b)}{1 + \sum_j T_i(j)}) \end{aligned}$$

If the width of the table for a particular type of site is n, then a score $g(S)$ for a candidate site $S = s_1 s_2 \ldots s_n$ can be calculated as:

$$g(S) = \sum_{i=1}^{n} llr_i(s_i)$$

GENEFINDER provides tables for the following gene features: translation start sites representing positions -9 through +11 (where 0 is the position of the A in the completely conserved ATG, which is enforced); both the donor and acceptor splice sites, representing 6 and 25 nucleotides of the corresponding exon and intron respectively, with enforcement of the GT-AG intron rule; and finally, translation termination sites representing 13 nucleotides of the upstream coding region and 92 nucleotides of the downstream untranslated region, with enforcement of the (TAG|TAA|TGA) rule. The table for each feature is accompanied by a table of corresponding dimensions populated by counts from "random" DNA. The GF_FEATURES programs constructs llr matrices from the given tables, scores windows of the query sequence using the matrices, and outputs predictions of each feature scoring above the default threshold (usually 0.0, but -2.0 for acceptor splice sites).

Content sensors in GENEFINDER

As well as predicting features using frequency tables, GF_FEATURES can also be used as a content sensor, in particular in the detection of protein-coding regions. It cal-

culates log-likelihood ratios for each n-mer based on the frequency of occurrences in protein-coding regions compared with randomised DNA. It then scans the sequence in each of the six reading frames, at each position storing the sum of scores over all non-overlapping n-mers up to that point. The cumulative score array thus obtained for the whole query sequence for a reading frame is then used to obtain a set of maximal scoring coding segments for that frame, and those segments scoring above 1.0 are output in GFF as predicted coding regions.

The n-mer tables in GENEFINDER 980506 for the detection of coding-regions are in-frame 3-mers, i.e. codons.

Length Penalty functions in GENEFINDER

GENEFINDER 980506 includes length-penalty tables for introns, and initial, internal and terminal exons. These are defined as (length, penalty) pairs, so could be used largely as found, except where otherwise stated. Single exon genes and intergenic regions are subject to a constant, length-independent penalty.

3.3 Definition of a GAZE configuration in three steps

This section outlines the steps involved in the development of a simple GAZE model for drawing together the signal, content and length penalty information provided by GENEFINDER into predictions of complete gene structures. The rules presented here form the core for all of the models that I have subsequently developed; indeed the principal way in which I envision GAZE being used in practice is for models to be developed by taking existing models and tweaking them for specific situations. I see this simple model as forming a base for practically all future models that one might develop.

```
<?xml version="1.0" encoding="US-ASCII"?>

<gaze>
    <declarations>
        <feature id="start" st_off="0" en_off="3"/>
        <feature id="stop" st_off="3" en_off="0"/>
        <segment id="coding_seg" scoring="standard_max"/>
        <lengthfunction id="single_exon_pen"/>
    </declarations>

    <gff2gaze>
        <gffline feature="atg" source="Genefinder" strand="+">
            <feat id="start"/>
        </gffline>
        <gffline feature="stop" source="Genefinder" strand="+">
            <feat id="stop"/>
        </gffline>
        <gffline feature="coding_seg" source="Genefinder" strand="+">
            <seg id="coding_seg"/>
        </gffline>
    </gff2gaze>

    <dna2gaze>
        <dnafeat pattern="atg">
            <feat id="start"/>
        </dnafeat>
        <dnafeat pattern="taa">
            <feat id="stop"/>
        </dnafeat>
        <dnafeat pattern="tag">
            <feat id="stop"/>
        </dnafeat>
        <dnafeat pattern="tga">
            <feat id="stop"/>
        </dnafeat>
    </dna2gaze>
```

```
    <lengthfunctions>
        <lengthfunc id="sngl_ex_pen">
            <point x="0" y="4"/>
            <point x="1" y="4"/>
        </lengthfunc>
    </lengthfunctions>

    <model>
        <target id="END">
            <source id="BEGIN">
                <output feature="no genes"/>
            </source>

            <source id="stop">
              <output feature="intergenic"/>
            </source>
        </target>

        <target id="start">
            <source id="BEGIN">
                <output feature="intergenic"/>
            </source>
        </target>

        <target id="stop">
            <killfeat id="stop" target_phase="0"/>
            <useseg id="coding_seg" target_phase="0"/>

            <source id="start" mindis="6" len_fun="sngl_ex_pen" phase="0">
                <output feature="CDS" strand="+" frame="0"/>
            </source>
        </target>
    </model>

</gaze>
```

Figure 3.1: A complete GAZE-XML configuration file for the prediction of a single, single-exon gene on the forward strand of a DNA sequence

3.3.1 A single, single-exon gene

Figure 3.1 shows a GAZE configuration for the prediction of a single, single-exon gene on the forward strand. Although simple, this configuration makes use of the majority of features of GAZE.

The XML configurations file can be viewed quite simply as comprising five sections. The *declarations* sections declares the features, segments and length penalty functions that GAZE is going to work with, along with some of the core properties that are common to all elements of each type. Of particular note here is the *scoring* attribute given for the "coding_reg" segments, which dictates in this case that segments of this type should be scored according to the "maximal single" scheme, given by equation 2.4.

The *gff2gaze* section dictates how the input GFF files are used to obtain lists of features and segments. In particular here, the *gffline* tag is used, together with the *source*, *feature* and *strand* attributes to specify which GFF lines are relevant,

and which features to create when lines matching those criteria are observed. The *dna2gaze* section also allows for the creation of features from simple sequence motifs observed in the input DNA sequence.

The *model* section contains the *source* → *target* rules. Those involving the "stop" target are of particular interest here because they make use of the majority of features available in GAZE.

Firstly, a segment qualifier (denoted by the *useseg* tag) that is *global* to the target is used to denote the fact that "coding_reg" segments that are in-phase with respect to the target (via the *target_phase* attribute) are considered relevant for *all* legal sources for this target.

Secondly, an interruption constraint (denoted by the *killfeat* tag) is used in a similarly global way to invalidate the region between the target and *any* legal upstream source when interrupted by a "stop" feature that is in-phase with the target.

Thirdly, the *start* → *stop* rule specifically contains minimum-distance and phase constraints, as well as denoting that the "sngl_ex_pen" length penalty function should be used. The length function itself, taken from GENEFINDER, is defined at the top of the second column. Note that this particular length penalty is actually length independent, achieved by giving consecutive distances the same penalty.

Finally, the output qualifiers define information for how the regions in the final gene structure should be presented to the user. The output format of GAZE is GFF, hence output GFF tags can be attached to each rule.

3.3.2 Extension to spliced structures

Although this simple model is satisfactory for explaining some of the features of GAZE, it does not have much worth in practice because coding portions of the majority of *C. elegans* genes are interrupted by introns. Figure 3.2 show how the simple model in the configuration above can be easily extended to model spliced gene structures.

The first point of note is the way that intron *phases* are dealt with. Introns can

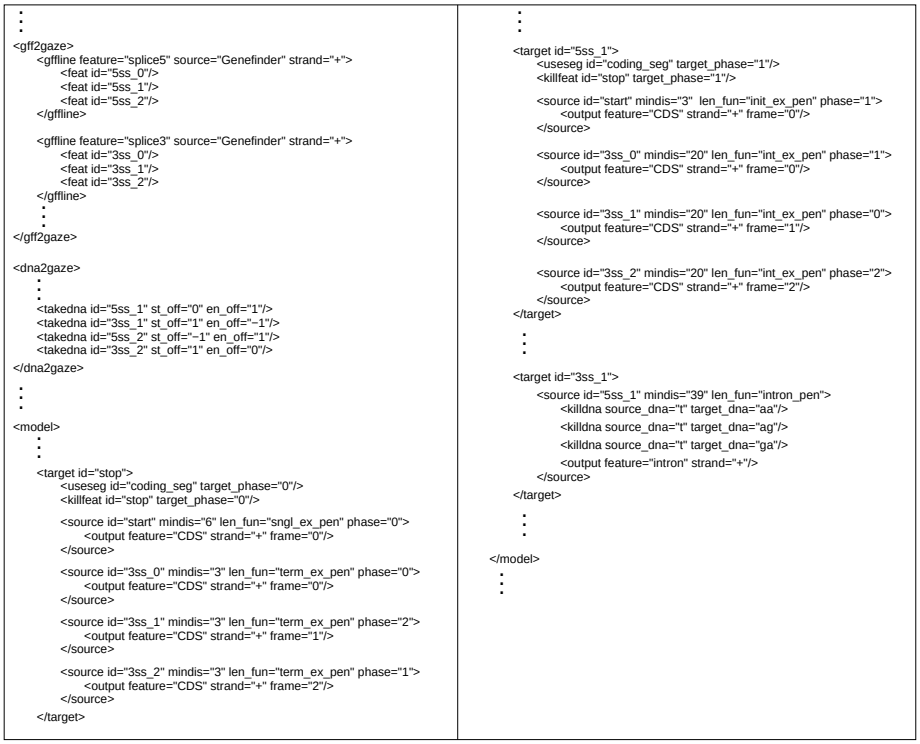


Figure 3.2: The main elements of a GAZE configuration for the prediction of single, possibly-spliced gene structures of the forward strand of a DNA sequence

interrupt the coding region of a gene either between two codons (a phase 0 intron) or in the middle of a codon, either between the first and second codon positions (phase 1 intron) or between the second and third positions (phase 2 intron). The total length of the spliced coding region must be a multiple of 3 in order for it to be successfully translated, which means that is important for programs to keep track of intron phases in order to produce sensible predictions.

The technique used here to account for intron phases is to consider the phases of the donor and acceptor splice site features that define the intron region. As shown in figure 3.2, for each given predicted splice site, three features are made, one for each of the possible codon positions that the splice site may occur at. For example, "5ss_1" denotes a donor splice site occurring between the 1st and 2nd codon position. The

rules are constructed in such as way as to ensure that any partial candidate gene structure ending with a "5ss_1" ends with the first base of an incomplete codon; this is achieved firstly by the use of phase constraints, and secondly by dictating that phases must be conserved across introns; notice from the figure that a phase 1 acceptor splice site ("3ss_1") can only be legally preceded by a donor splice site of the same phase ("5ss_1").

In the simple model presented earlier, candidate coding exons with in-frame stop-codons were disallowed by using an interruption constraint. Splicing also introduces the possibility that stop-codons occur at the junction formed by the concatenation of two exons. The figure shows how DNA constraints are used to disallow such structures. Firstly, in the *dna2gaze* section, *takedna* directives are given for certain features, which are instructions for GAZE to keep a record of the DNA sequence occurring at the location of these these feature, specifically between the given start and end position of the features, each adjusted by the offsets defined in the directive ("st_off" and "en_off"). The rule for the "3ss_1" target contains the DNA constraints themselves (*killdna*), an example of which states that the region between a "5ss_1" and "3ss_1" is illegal if the DNA at the "5ss_1" is 'T' and the DNA at the "3ss_1" is 'AA.'

This mechanism has a limitation: it is technically possible for a stop-codon to be formed from three successive exons, where the second exon consists of a single base-pair. Internal exons this small, even if biologically possible, would be extremely rare, and in fact are disallowed in the all of my models by use of a minimum distance constraint. As shown in the figure, the minimum I use, taken from GENEFINDER, is 20 base pairs.

3.3.3 Extending to multiple genes on both strands

Figure 3.3 gives a flavour of extensions to the model to allow for the possibility of more than one gene on the query sequence. The "start" feature can now be immediately preceded by a "stop" source marking the end of another gene, with

Figure 3.3: A fragment of a GAZE configuration showing the elements involved in modelling multiple genes on both strands of a DNA sequence.

a new length-penalty function for the intergenic region implied by a pair of such features.

Some gene prediction programs model reverse strand genes by predicting separately on each of the given sequence and its reverse complement, and then merging the predictions back together. This can cause problems if the gene prediction signal is strong on both strands at the same location, where it is not obvious what the gene prediction should be in this region. Other programs, for example GENSCAN, incorporate a single integrated model for the prediction of genes on both strands. They do this by firstly choosing (arbitrarily) a strand as the reference strand, and secondly treating opposite strand genes as comprising of the same features but occurring in reverse order. Hence from the point of view of the reference strand, opposite-strand genes begin with a stop-codon and end with a start-codon. This approach is easily implemented in GAZE, demonstrated by the rules for the "start" target in figure 3.3, which include the "start_rev" source, marking the start of a gene on the opposite

strand. The full model, which I refer to as GAZE_std, is represented pictorially in figure 3.4.

3.4 Applying the model to *C.elegans* sequences

The accuracy of the model presented above in comparison to other available programs is examined in detail in a subsequent section, where it is shown how various refinements affect the performance. This section uses the application of the model to the WormSeq test sequence to demonstrate some aspects of the functionality of GAZE.

3.4.1 Predicting genes in WormSeq

Table 3.1 shows the accuracy of GAZE_std at the whole gene level. For the purposes of specificity, a prediction is considered correct only if the complete gene structure matches precisely the structure of the corresponding cDNA-confirmed structure; likewise, for the purposes of sensitivity, a confirmed gene structure is considered to be correctly predicted only if it is matched precisely by a corresponding GAZE-predicted gene. The stringency of this measure is shown in the table; only about a third of WormSeq genes are predicted correctly by GAZE_std and only a third of GAZE_std predictions are correct. Upon visual inspection of the predictions in comparison to the correct gene structures in ACeDB, it is apparent that many of them have the correct or nearly-correct intron-exon structure, but err in the location of their start and/or end. It is widely observed that the ends of genes are more difficult to identify correctly than their internal intron-exon structure (see chapter 1), and observation of the accuracy of GAZE_std on WormSeq supports this.

3.4.2 Using feature-selection to refine the predictions

If the starts and ends of genes are the most difficult features to identify, it is interesting to ask how the accuracy of GAZE_std changes when it is told where the

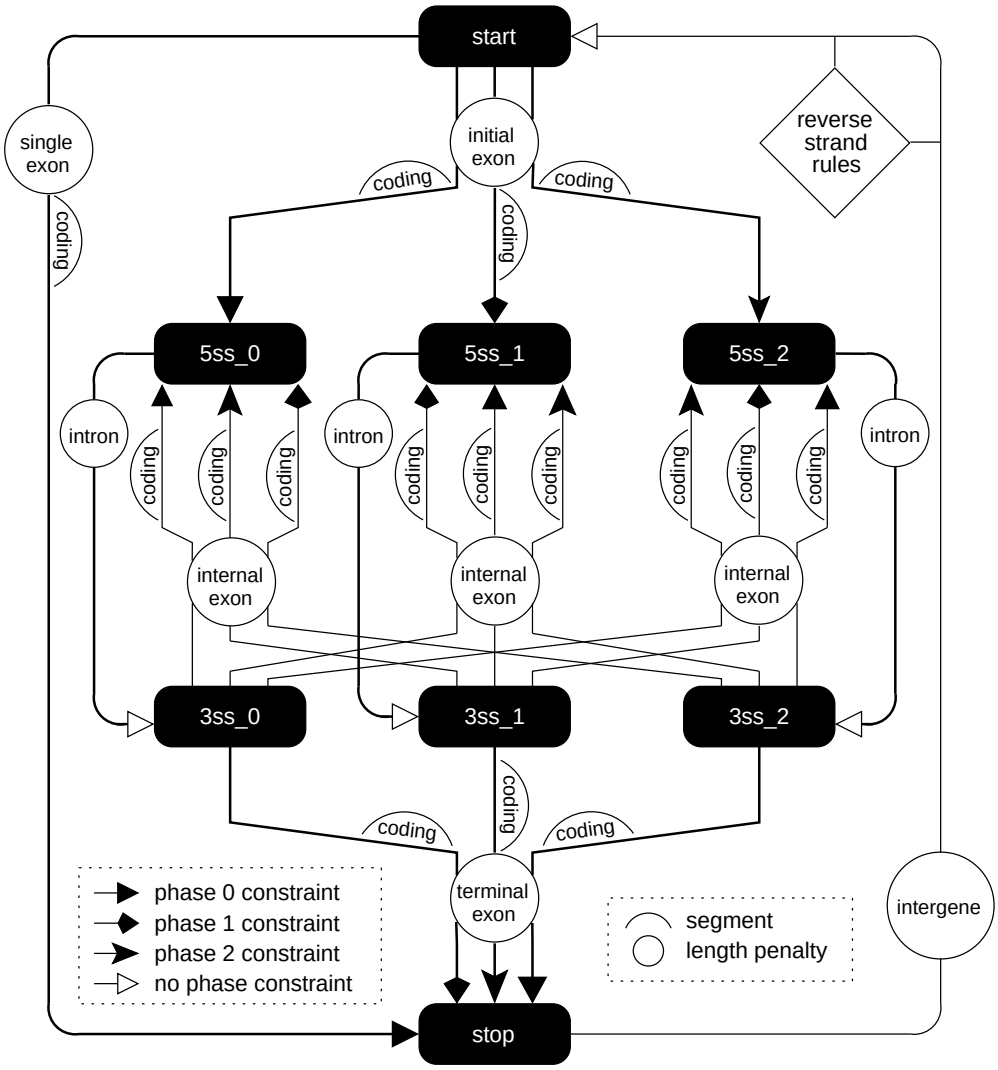


Figure 3.4: A pictorial representation of a GAZE-XML model for multiple genes on both strands. The features are represented by filled boxes, and *source* → *target* rules by different types of arrows, each corresponding to a phase constraint. The labelled circles give the name of the length-penalty function used for each rule, which are themselves defined elsewhere in the configuration file; the labelled humps indicate the segments that contribute to the score for the region implied by the rule. The rules for reverse-strand target features are not shown in their entirety for reasons of clarity, but are reverse complementations of the forward-strand rules. Also omitted are the distance, interruption and DNA constraints, as well as the BEGIN and END features, which mark the ends of the sequence being searched for genes and act as source and target (respectively) to all other features; this accounts for the possibility of a gene structures which extend beyond the end(s) of the sequence.

	Sn	Sp	Av	MG	WG	SG	JG
GAZE_std	0.41	0.24	0.33	0.003	0.368	1.14	1.03
GAZE_std+	0.67	0.36	0.52	0.000	0.387	1.13	1.00
GAZE_std++	0.71	0.71	0.71	0.000	0.000	1.00	1.00
GAZE_std_gf	0.35	0.35	0.35	0.012	0.076	1.03	1.08
GENEFINDER	0.50	0.44	0.47	0.012	0.104	1.07	1.04

Table 3.1: Gene-level accuracy of GAZE_std plus variants on WormSeq. Sensitivity (Sn), Specificity (Sp), Average (Av), Missing genes (MG), Wrong genes (WG), Split genes (SG) and Joined genes (JG) are the measures described in section 1.4.1

starts and stops of the genes in WormSeq really are. It is straightforward to answer this question in the context of GAZE. I made a GFF file of the confirmed starts and stops of the WormSeq genes, and used the feature selection mechanism of GAZE to force the inclusion of these features in the prediction. The results, referred to as GAZE_std+, are shown in the second row of table 3.1. Several things are notable. Firstly there is a big jump in gene-level sensitivity; 26% more WormSeq genes are identified precisely correctly. Secondly, the figure of 1.00 for Joined genes indicates that no predicted gene extends over the region covered by two or more WormSeq genes. This is expected, because the feature-selection forces the correct splitting of such genes. Thirdly, there is a noticeable increase in 'wrong' genes. It is counter-intuitive that supplying the system with gene starts and ends should lead to more predictions that do not overlap any confirmed gene structure. However, figure 3.5 shows how additional wrong genes can arise from such an approach.

3.4.3 Adjusting the score to refine the predictions

For the purposes of demonstration only, I made a slight modification to the model, increasing the length-independent penalty for intergenic regions from 4.0 to 100.0. This penalty is incurred whenever a gene is introduced; in making it large, the prediction of genes that do not contain any user-selected features is effectively dis-

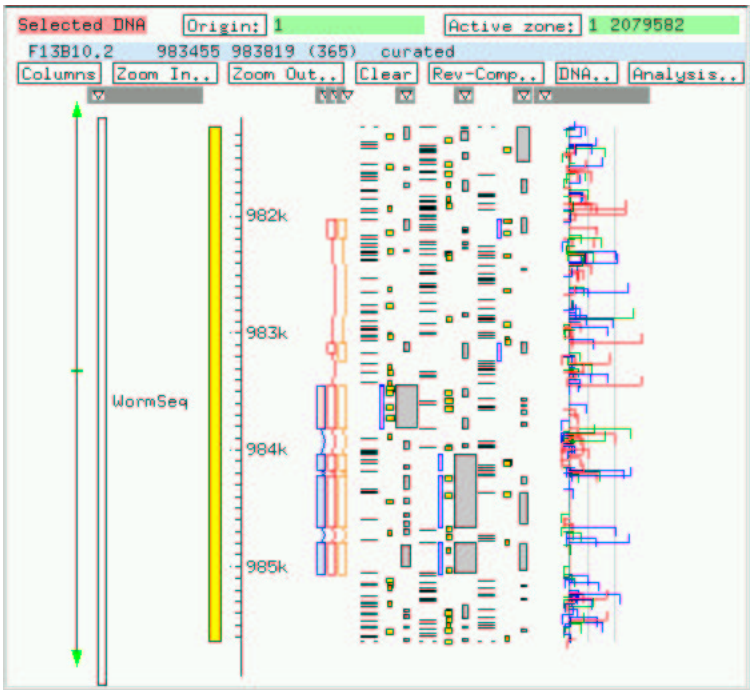


Figure 3.5: How specifying gene start/end information can lead to an increase in the number of Wrong Genes (WG). The GAZE_std prediction (in red) extends into the region 5' of the cDNA-confirmed structure (in blue), but is not classed as "wrong" because it overlaps with the correct structure. Although the GAZE_std+ model (orange) correctly identifies the structure of the gene, the pseudo-signal in the 5' region is strong enough to lead to the prediction of a separate gene which does not overlap any correct gene structure.

allowed. The results of this refinement appear as GAZE_std + + in table 3.1. The MG, WG, SG and JG lines show that the intended increase in specificity has been achieved, and exactly one gene is predicted for each cDNA-confirmed structure.

It will never be the case in practice that the starts and ends of all the gene structures will be known *a priori* for an otherwise unannotated large stretch of genomic DNA. It may be that the starts and ends for *some* of the genes will be known, but in that case, the trick of increasing the length penalty will lead to the missing of gene structures for which they are not known. For this reason, using GAZE in a way such as this is artificial and was largely for demonstration only.

However, some insights can be gleaned from the results. In particular, they show that GAZE_std is 71% accurate in identifying the complete internal intron-exon structure of the genes in this test-set. Furthermore, the feature-selection mechanism has proved and will prove useful for the manual curation of gene structures; it gives the ability to anchor parts of the structure and identify the most likely total structure that is consistent with the anchored points. In this way, it provides an elegant means for curators to make use of incomplete evidence.

3.4.4 A comparison with GENEFINDER

Since GAZE_std integrates the signal, content and length-penalty information from the GENEFINDER program, it is natural to ask how the accuracy of the two compare. The bottom row in table 3.1 shows the gene-level accuracy of GENEFINDER on WormSeq. It is immediately noticeable that GENEFINDER is significantly more accurate than GAZE_std at the precise identification of complete gene structures. It is also more specific, with only 38 wrong genes, compared with a figure of 210 for GAZE_std. The difference in Split genes, 1.07 compared to 1.14 for GAZE_std, is also striking. GAZE_std and GENEFINDER use the same gene prediction signal, content and length-penalty information, so where is this difference coming from?

Inspection of the GENEFINDER source-code reveals some of the answers. Firstly, GENEFINDER subjects candidate exons to a further penalty just before they are assembled into gene structures, and this penalty is different for internal exons ($\log(0.8)$), initial exons ($\log(0.2) + \log(0.5)$) and other non-internal exons ($\log(0.2)$). Since the penalties for non-internal exons are larger, this has the effect of discouraging the splitting of genes. Secondly, GENEFINDER removes all genes scoring less than 7.0, effectively reducing the number of wrong genes.

These subtleties prove a test of the flexibility of GAZE. It turns out to be straightforward to incorporate the additional exon penalties, by simply adding these terms to the penalties for all distances in the appropriate length-penalty functions. The removal of low scoring genes cannot be performed in GAZE itself, but is achieved

with a simple Perl post-processing filter.

The resulting model, the results of which are referred to as GAZE_std_gf in table 3.1 represents an attempt to duplicate the output of the GENEFINDER program using GAZE. The table shows GAZE_std_gf to be outperformed by GENEFINDER at the whole-gene level, whereas examination of the comparative accuracy at the base-pair and exon level (see table 3.5) reveals no notable difference. This discrepancy in gene-level accuracy is due not to any differences in the signal and content data used by the two systems, nor differences in the length penalty functions, nor any hidden post-processing, but to the fact that GENEFINDER assembles its exons over a model of gene structure that is designed specifically for prediction in *C.elegans* sequences. In the next section, I show how the GAZE framework allows worm-specific model features to be quickly and effectively introduced into the mode of gene structure without any change to the GAZE source-code itself.

3.5 Towards a *C.elegans*-specific model of gene structure

The GAZE_std_gf configuration is specific to *C. elegans* in that it reads features and segments from the GENEFINDER program, which have been detected using worm-specific signal and content models[2], and also length-penalty functions that have been designed from observation of the distances between such components in real worm genes. However, the *model* of gene structure itself over which gene assembly takes place is specific only to eukaryotes in that it can predict spliced gene structures. There is nothing in the model itself that suits it to the prediction of gene structure in *C.elegans* specifically. The GENEFINDER program on the other hand takes account an unusual splicing mechanism that takes place only in the cells of nematode worms and some other primitive eukaryotes, and it is this that gives it greater accuracy.

[2]More accurately, general models that have been parameterised by observation of confirmed gene features in *C. elegans* sequences.

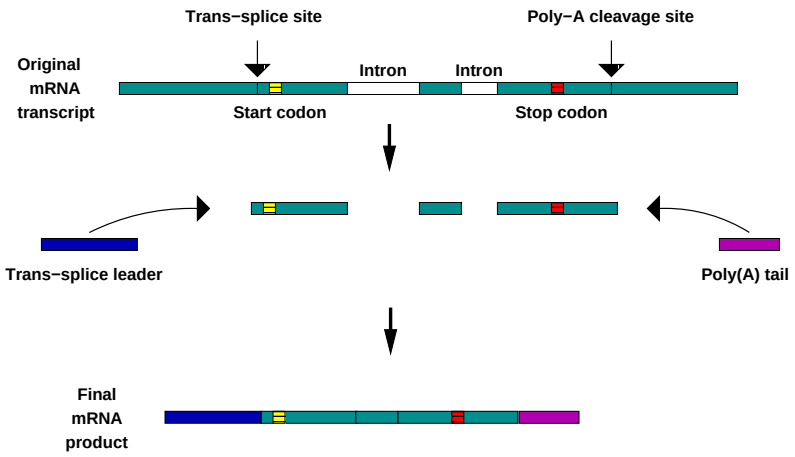


Figure 3.6: Schematic representation of *trans*-splicing in *C.elegans*

3.5.1 Splicing mechanisms in *C.elegans*

In nematode worms such as *C.elegans* and *C.briggsae*, as well as some other primitive eukaryotes such as trypanosomes, a splicing mechanism exists that is unlike the conventional intron-removal mechanism (known as *cis*-splicing). In *trans*-splicing ([66]; review in [13]), the pre-mRNA transcript is cleaved at a site upstream of the translation start site, and the resulting protein-coding fragment is appended to a 21-23 base-pair sequence called the *trans*-splice leader, which itself has been transcribed from elsewhere in the genome. The process is summarised in figure 3.6.

The biochemical process of *trans*-splicing is closely related to that of *cis*-splicing. In fact, it has been shown that the signal for *trans*-splicing to occur is simply the presence of a sequence at the 5' end of the pre-mRNA that looks like an intron but has no functional upstream donor splice site [27]. The *trans*-splice site itself forms the 3' end of this *outron* sequence, and has the same consensus as the splice acceptor involved in intron-removal.

The splice-leader RNAs themselves are always one of two distinct sequences: SL2 leaders are appended to all but the first gene in an operon, and have slight variation

in their sequences. The more common SL1 leaders are appended to all other *trans*-spliced gene products, and are all identical in sequence. For the identification of gene structures in worm genomic DNA, the splice-leader sequences cannot be used as a signal for detection, because they do not appear in the genomic sequence in proximity to the gene they are spliced to.

3.5.2 *Trans*-splicing confuses gene prediction programs

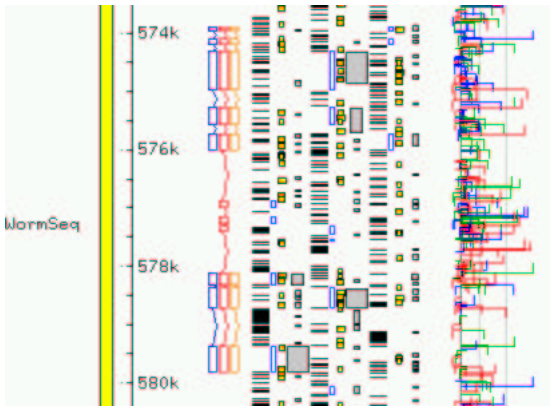
Because the recognition site for *trans*-splicing reaction has the same consensus as an acceptor splice site, gene prediction programs that have not accounted for *trans*-splicing can be confused into mistaking the initial exon of a gene for an internal exon and erroneously extending the prediction upstream. The relatively high gene-density in the worm genome, especially in operons where the genes are typically as close as a thousand bases apart, compounds this problem, often causing a program to mistake two adjacent genes for a single gene. This is apparent in the Joined Genes figure for GAZE_std_gf in table 3.1, which is relatively high compared to the *trans*-splicing aware GENEFINDER. Figure 3.7 illustrates the problem.

3.5.3 A GAZE model accounting for *trans*-splicing

In terms of the scoring function, the problem arises because high-scoring acceptor splice site predictions that are in fact *trans*-splice acceptors can only be included in the gene structure (and thus contribute towards the score) if the 5' end of the prediction is compromised in some way, e.g. by the addition of a low-scoring initial exon upstream. The idea then is to provide a way for the trans-splice acceptor to contribute towards the overall score without having to make this compromise.

Figure 3.8 shows, in spirit, the nature of the changes that are necessary to accommodate *trans*-spliced genes. The first thing to note is that it is not necessary to generate *a priori* predictions of trans-splice acceptor sites; the sequence signal is practically indistinguishable from that displayed by conventional *cis*-splice sites. It is therefore sufficient to direct GAZE to make a candidate *trans*-splice acceptor

(a)



(b)

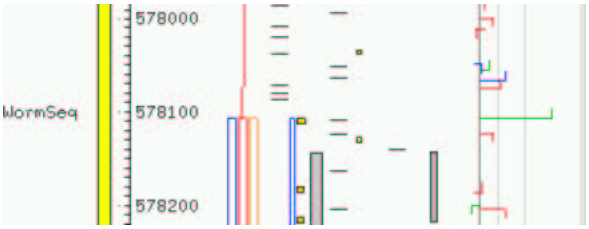


Figure 3.7: (a) Two cDNA-supported gene structures in WormSeq, with WormBase identifiers F11A5.10 and W06D12.3 (blue), and structures predicted by GAZE_std (red) and GAZE_trans (orange). The GAZE_std model, which does not account for *trans*-splicing, has been confused into mistaking the *trans*-splice site for an acceptor splice site, extending the gene-prediction 5', and in this case amalgamating it with the upstream gene; (b) an enlargement of the 5' end of the downstream gene, showing the *trans*-splice site (green hook). The *trans*-splice aware model, GAZE_trans, splits the structures correctly (orange)

feature ("trans_splice") from each predicted *cis*-acceptor encountered in the GFF file.

Secondly, minor modifications are necessary to the gene structure rules to accommodate the new feature; a "start" target feature, representing a start-codon candidate on the forward strand, can now be preceded by the stop-codon of a previous gene as before, or for a *trans*-spliced gene, the *trans*-splice acceptor itself.

The distance from the translation start site to the upstream *trans*-splice acceptor in *trans*-spliced genes is usually small. Data in [13] compiled from a sample of 83 genes experimentally confirmed to be *trans*-spliced, showed that in 43% the *trans*-

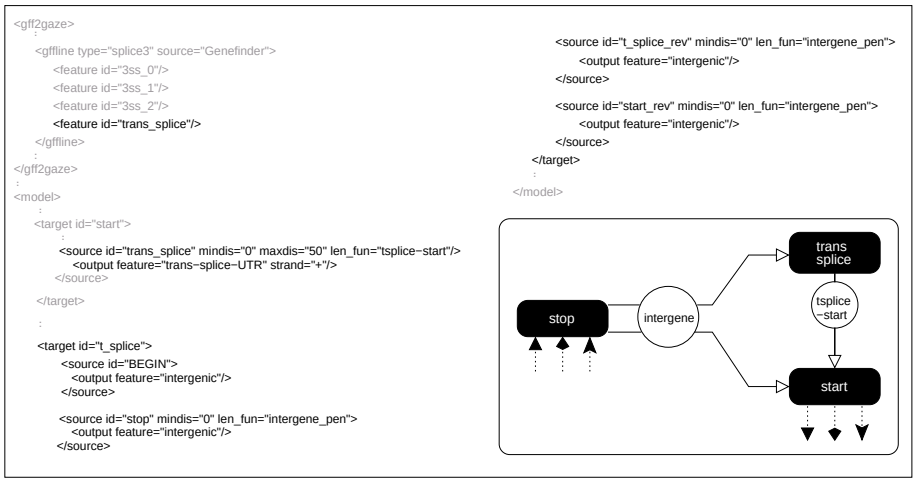


Figure 3.8: Distillation of the changes required to the forward-strand part of GAZE_std to allow for the possibility of *trans*-sliced genes. The inset shows pictorially how the new "trans_splice" feature fits into the model of gene structure

splice site is within 5 base-pairs of translation initiation, 65% fall within 10 base-pairs, 82% fall within 15 base-pairs, and in only 5% is the distance greater than 30 base-pairs.

The decrease in the likelihood of a *trans*-splice candidate with distance can be modelled naturally with a length penalty function. Again, I looked to GENEFINDER, but this time it was necessary to inspect the source-code for the details of the function used. GENEFINDER approaches the problem by considering all initial exons as beginning not with the translation start candidate, but 50 base-pairs upstream. It adjusts the score of a candidate initial exon beginning at base-pair i by adding a log-probability for the extra 50 base-pairs calculated in the following way:

$$Adj(i) = \max \left[\begin{array}{c} \log(1 - P_{tr}), \\ \log P_{op} P_{tr} + \max_{j=i-50}^{i-1} 3ss(j) + (i - j + 1) \log(1 - P_{op}) \end{array} \right]$$

$3ss(j)$ is the log probability-ratio of an acceptor splice site at position j, or $-\infty$ if the score under the splice acceptor model at j does not exceed the cutoff. The values P_{tr}, P_{op} can be given as parameters to GENEFINDER. P_{tr} represents the prior probability that a gene is *trans*-spliced; P_{op} and $1 - P_{op}$ can be thought of as

	Sn	Sp	Av	MG	WG	SG	JG
GAZE_trans	0.47	0.42	0.44	0.012	0.093	1.07	1.04
GAZE_std_gf	0.35	0.35	0.35	0.012	0.076	1.03	1.09
GAZE_std	0.41	0.24	0.33	0.003	0.368	1.14	1.03
GENEFINDER	0.50	0.44	0.47	0.012	0.104	1.07	1.04

Table 3.2: Comparative Gene-level accuracy of GAZE_trans on WormSeq. Accuracy measures are explained in section 1.4.1

the probability of remaining in and leaving (respectively) the *trans*-splice "state" (viewing the 50 residues as being generated by a Hidden Markov model). The default values for P_{tr} and P_{op} are 0.5 and 0.1 respectively.

It is straightforward to derive a GAZE length-penalty table for the distance between a translation start site and an upstream candidate *trans*-splice site using this function. The only caveat is that the length-penalty table for initial exons already includes the term incurred by non-trans-spliced genes ($\log(1 - P_{tr})$), so this term is *subtracted* from the $tsplice \rightarrow start$ function in order to avoid incurring it twice.

The resulting variant of GAZE_std_gf (with additions to allow for the possibility of *trans*-spliced genes on both forward and reverse strands) is referred to as GAZE_trans. Table 3.2 shows gene-level accuracy for the GAZE_trans model in comparison to the standard models.

The number of genes for which the complete structure has been identified correctly is noticeably higher for GAZE_trans compared with GAZE_std_gf, and is now on-par with that achieved by GENEFINDER. What is also noticeable is the sharp decrease in joined genes, which is exactly what was intended (see figure 3.7). However, this increase seems to have come at a cost with respect to Split genes. In introducing a model innovation that is specifically intended to split gene predictions, we run the risk of introducing erroneous splits, and this is what has happened here. In this case though, the extra splits have apparently occurred in predictions that

were also incorrectly predicted by GAZE_std_gf, hence the net increase in gene-level sensitivity. A closer look reveals that 42 structures which are correctly predicted by GAZE_trans were incorrectly predicted by GAZE_std_gf, and only 7 structures correctly predicted by GAZE_std_gf are incorrectly predicted by GAZE_trans.

The GAZE_trans model is, to all intents and purposes, a GAZE implementation of the 980506 version of GENEFINDER. This begs the question of why their accuracies, although comparable, are not identical. The reason for this is the way that GAZE makes use of the GENEFINDER *coding_seg* maximal scoring segments of high protein-coding potential. Inspection of the GENEFINDER source-code reveals that it does not use segments calculated in advance for the whole query sequence (as explained in section 3.2) at all; instead, it obtains a maximal coding score for each candidate *exon*, using the same cumulative-array approach. This difference in exon scoring schemes is only observable when a GAZE "coding_seg" extends beyond either the 5' or 3' end of the candidate exon. In that case, as implied by equation 2.4, the score is scaled by the proportion of the segment that lies in the region. This incorrectly assumes that the score for segment is distributed evenly along its length. For that reason, the GENEFINDER method for computing coding scores is more accurate than the approximate method used by GAZE. However, as shown by the results so far, it seems to have little impact on the overall accuracy; in fact, at the exon level (as shown in table 3.5), GAZE_trans, although marginally less sensitive than GENEFINDER, is slightly more specific.

3.6 Integrating similarity information

As explained in chapter 1, the effective use of similarity information can improve gene prediction accuracy. In this section, I outline the changes required to the GAZE_trans configuration to make use of the similarity information in the form of EST alignments.

3.6.1 ESTs and gene prediction

An Expressed Sequence Tag (EST) is conceptually a sequence read of a cDNA copy of an expressed cellular mRNA. ESTs differ from the full-length cDNAs deposited in the nucleotide databases (such as those used to build the WormSeq dataset) in that: (1) they are often of lower quality, and (2) they represent the sequencing of only a subsequence of the original cDNA, typically around 300-500 base-pairs, read from either the 5' or 3' end of the transcript.

In much the same way as the EMBL full-length cDNAs were used to confirm the genes in WormSeq, the alignment of ESTs to genomic sequence can provide evidence for gene structures. Since ESTs represent only a draft-quality read of a subsequence of a transcript, any single EST will normally only provide evidence for *part* of a gene structure.

The utility of EST data

The alignment of ESTs back to the genomic sequence can be extremely useful for gene prediction in several ways. Firstly, they act as an aid to novel gene discovery, highlighting regions of genomic sequence where genes were not thought to exist before. Secondly, they can provide evidence for the spliced structure of the gene, at least in cases where the EST extends across an exon-exon boundary in the corresponding spliced mRNA. Thirdly, ESTs can help elucidate the structures of genes that give rise to several alternatively spliced transcripts. Finally, they are useful for identifying the extremities of genes. Ideally, the alignment of a 5' EST to the genome identifies the 5' end of a gene, and likewise with 3' ESTs. Many ESTs even exist as pairs, corresponding to 5' and 3' reads of the *same* cDNA, and a pair of such alignments ideally identifies both the 5' and 3' end of a gene, if not the complete internal intron-exon structure.

Problems with EST data

If the data were perfectly reliable, each EST would provide perfect, unquestionable evidence for either the 5' or 3' end of a gene, and in addition perhaps part of the intron-exon structure. Unfortunately, ESTs are naturally error-prone, due to their high-throughput, single-read nature. This can lead to errors when they are aligned. In addition, there are other problems associated with EST data:

Sample bias The pool of available ESTs is unavoidably biased towards genes that are highly and ubiquitously expressed. EST databases are therefore not generally a useful resource for either the discovery or the elucidation of the structures of genes expressed at at low levels or under very specific conditions.

Pseudo poly-A sites The EST sequencing process begins by the reverse transcription of the mRNA into a double-stranded complementary DNA (cDNA), primed from the poly-A tail at the 3' end of the transcript. If the transcript by chance contains a string of A residues somewhere other than the 3' end, then reverse transcription could begin from this place, resulting in a cDNA for which part of the 3' end of the gene is missing. When aligned back to the genome, the 3' EST match ends somewhere upstream of the true 3' end of the gene, sometimes in the protein-coding portion.

5' end incompleteness It is often the case that the 5' end of the cDNA is missing, either because the mRNA it was synthesised from had been partially digested at the 5' end, or because the reverse-transcription reaction did not carry through to completion. When a 5' EST is aligned back to the genome therefore, the match can begin somewhere downstream of the true 5' end of the gene, often in the protein-coding portion.

Ambiguous matching The low-fidelity of EST sequences means that mismatches must be allowed when aligning them to genomic sequence. A common implication of this is that the EST will align to multiple places in the genomic

sequence, and it is a not always obvious to identify the "correct" alignment.

Annotation errors It is not uncommon for an EST sequence to be deposited in a nucleotide database with the incorrect assignment of orientation, e.g. a 5' EST being annotated as a 3' EST. This can cause confusion when inferring information from an EST alignment such as the strand to which it matches.

The source of *C.elegans* EST data

The WS52 release of WormBase contained around 100,000 *C.elegans* EST sequences. As part of the Sanger Institute worm sequence curation process, each of these ESTs is isolated to a small number of localised regions in the genome by BLASTN [2], and then accurately aligned using the spliced-local-alignment program EST_GENOME [79]. It is important to note that some ESTs are aligned to several places in the genome using this procedure, and others not at all. For those that are aligned, the result is a set of "exons" each of which can be described by a 5-tuple: (EST-identifier, EST-start, EST-end, genome-start, genome-end).

For WormSeq, these exons were re-mapped into (EST-identifier, EST-start, EST-end, WormSeq-start, WormSeq-end) 5-tuples, discarding those falling outside the regions extracted from the genome to form artificial sequence, and truncating those with partial overlap to these regions where necessary. As a result, 261 of the 325 gene loci in WormSeq have at least one EST match.

3.6.2 A GAZE model for the use of EST alignments

The natural approach is to use the EST match exons as segments providing evidence for protein-coding regions. These segments will be expected overlap with the untranslated regions of genes however, so such a method would lead to the over-prediction of the coding portions of the genes. I therefore extend the model of gene structure to include the untranslated regions at the ends of genes, incorporating features for the beginning and end of *transcription*: "transcript_start" and "transcript_stop".

With this modification, EST match segments can now be used as evidence for protein-coding regions and untranslated regions. However, such a model is not exploiting the power of EST alignments in their identification of intron-exon boundaries and gene extents. I therefore devised a general EST pre-processing strategy that would make the gene structure information encoded in the alignments more readily available to be used by GAZE. The pre-processing is performed according to the following schedule:

- From the pool of EST 'exons', construct a set of EST 'transcripts', lists of the exons from a single EST ordered by their location on the genome. Transcripts with an average match identity to the genome of less than 95% are discarded at this stage.
- For each transcript, generate:
  - "EST_match" segments for the exons;
  - "EST_intron" segments for the regions between transcript exons that are adjacent in the EST but separated in the genome sequence;
  - if EST is a 5' read, a "transcript_start" feature for the beginning of the transcript;
  - if EST is 3' read, a "transcript_stop" feature for the end of the transcript.
- For those cDNAs for which there is exactly one 5' and one 3' transcript, generate an "EST_span" segment for the region between the start of the 5' transcript and the end of the 3' transcript.

Although the matches produced by the EST_GENOME program score each exon according to its percentage identity to the genome, the model was found to be ineffective when these scores were used directly (data not shown). In addition, derived features and segments do not have an associated score. I therefore used the following scoring scheme for the EST-derived features and segments:

EST_match These were given the score $\frac{(identity-95).length}{100}$. The rationale for this is to ascribe more confidence to long, contiguous exons with high identity than to short exons with high identity. The latter are an artifact of the EST alignment process, where single base-pair deletions in the EST with respect to the genome cause two shorter high-identity exons to be created. Also, since it is expected that the number of "EST_match" segments falling in a given region will be highly variable (from 0 to hundreds), the "projected per-base" segment scoring approach (equation 2.5) is most appropriate.

EST_intron These were scored according the average identity of the flanking exons. A scaling factor of 0.05 was applied to bring the order of the scores into line with other segments being used. Since these segments are expected to match candidate intron regions precisely, their use is qualified with a match constraint[3].

EST_span These segments correspond to regions containing exactly one complete gene. However, due to the problems with EST data explained earlier it will often be the case that an "EST_span" segment covers only part of a gene. The segments are therefore interpreted as regions that should contain *no more* than one gene. The way that this is realised in GAZE is to use the segments as supporting evidence for intergenic regions, but to give them very high negative scores, in this case -10000. This penalises the prediction of intergenic regions where EST_spans lie, effectively preventing gene splitting. This is pertinent as the Split Genes figure for GAZE_trans was quite high (see table 3.2).

transcript_start and **transcript_stop** features were assigned the neutral log probability ratio of 0.

Figure 3.9 shows pictorially a GAZE configuration for incorporating these EST-derived features and segments, over a model of gene structure that now accounts for

[3]Recall from chapter 2 that match constraint stipulates that the a segment should only contribute towards the score if its extent matches the region being considered precisely.

	Sn	Sp	Av	MG	WG	SG	JG
GAZE_EST	0.58	0.53	0.56	0.009	0.088	1.02	1.03
GAZE_trans	0.47	0.42	0.44	0.012	0.093	1.07	1.04
GENEFINDER	0.50	0.44	0.47	0.012	0.104	1.07	1.04

Table 3.3: Comparative gene-level accuracy of GAZE_EST in comparison to GAZE_trans and GENEFINDER. Accuracy measures are defined in section 1.4.1.

the untranslated regions at the ends of genes (as well as *trans*-splicing). This model is referred to as GAZE_EST.

Table 3.3 shows the gene-level accuracy of the GAZE_EST model in comparison to the GAZE_trans model and GENEFINDER. GAZE_EST displays an improvement over all models presented so far, as well as GENEFINDER.

In describing the GAZE_trans model, I noted that some incorrect splitting of gene structures was inevitable but showed that the *net* gain in accuracy achieved by this innovation was significant. In introducing EST evidence, there is no obvious reason why the accuracy of some predictions might become worse. However the net gain of 37 additional correct structures identified by GAZE_EST over GAZE_trans consists of 39 that GAZE_EST correctly identifies whilst GAZE_trans did not, and 2 that GAZE_trans correctly identified whilst GAZE_EST does not. There are also 2 additional examples of a GAZE_EST prediction having fewer correct exons than the corresponding GAZE_trans predicted structure. Closer examination of these 4 examples reveals a variety of reasons for the decrease in accuracy:

Introns in UTR. In two of the cases, the EST evidence supports the presence of an intron in the 5' untranslated region of the gene. Since the GAZE_EST model only allows for introns in the coding portion of the gene, the prediction of this coding portion has been extended in the 5' to accommodate the intron.

Overlapping transcription units. In one case, the final coding exons of a gene on the forward strand overlapped with the 3' UTR of a gene on the reverse strand,

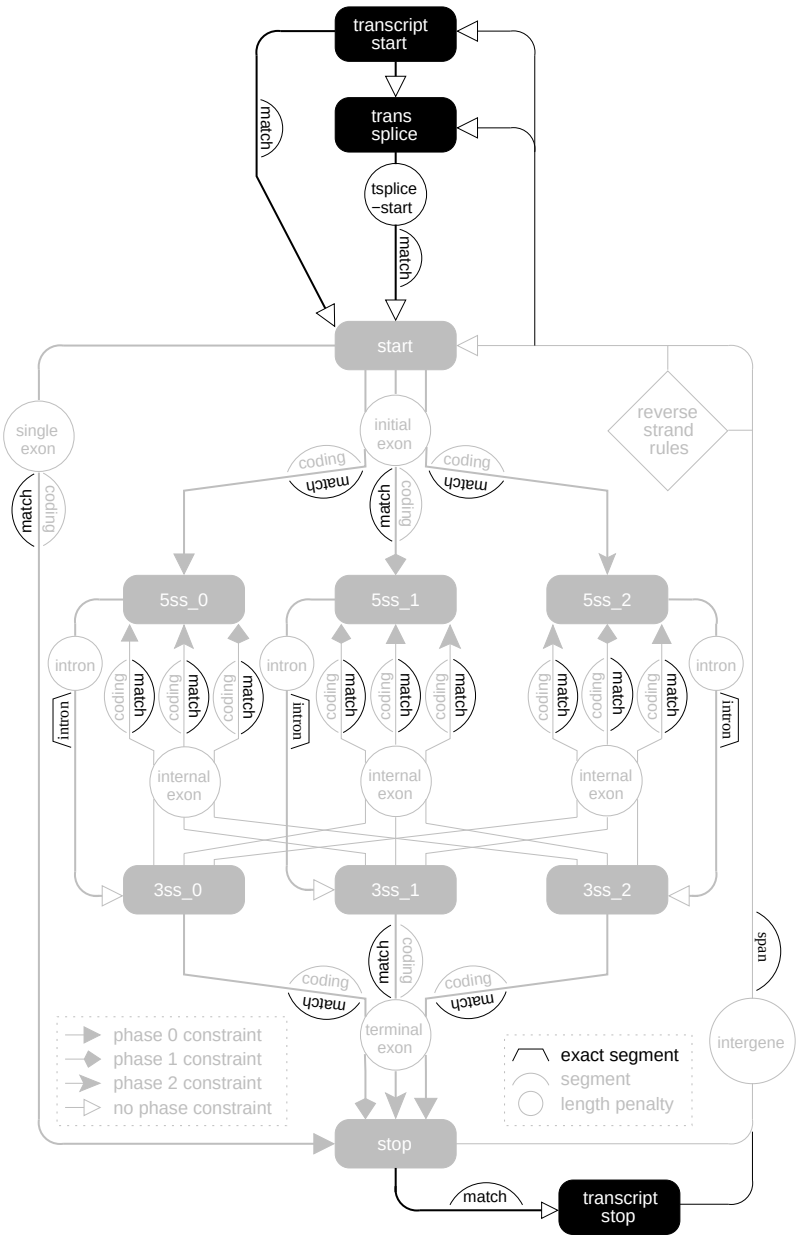


Figure 3.9: The GAZE_EST model, allowing for *trans*-spliced genes and untranslated regions. It is a simple extension of the GAZE_std model (figure 3.4), which is shown in pale-shade for reference. The "match", "intron" and "span" segments shown are the "EST_match", "EST_intron" and "EST_spans" segments referred to in the text.

as supported by an EST alignment. GAZE, at least as I have presented it so far, classifies each base in the sequence as belonging to exactly one function class. The region here was classified as a 3' UTR on the reverse strand, causing the final coding exon of the forward strand gene to be missed.

Incorrect alignment In the final case, a cluster of EST alignments in the intron of the gene caused GAZE_EST to split the structure incorrectly. This region also had a partial match to a nucleotide database cDNA that matched with a higher score elsewhere in the genome, suggesting the possibility of a pseudogene.

Such problems were not confined to these four examples; in other cases though, the prediction was unaffected. In the design of the scoring scheme I was attempting to achieving a balance between the gains to be had by treating the EST exons as strong evidence for genic regions, and the losses incurred by treating them as 'the truth.' The fact that only a small number of prediction were affected by EST-confusion supports the validity of the scheme.

3.7 A closer look at the accuracy of GAZE

This section, provided for completeness, examines various aspects of the accuracy of the GAZE models presented earlier. Thus far, results have been presented at the gene level only for purposes of illustration.

The performance of the GAZE models has so far been assessed in comparison to the GENEFINDER program which, indirectly at least, works with the same signal, content and length-penalty models. As the basis for a more objective evaluation, I therefore also include in the following results for FGENESH [96], an HMM-based gene prediction program that works in a similar manner to the more widely used GENSCAN [21]. Unlike the latter, however, FGENESH comes with a parameter file for prediction in *C.elegans* sequences specifically, as well as a "-nematode" command-line directive.

	Sn	Sp	Av	MG	WG	SG	JG
GAZE_EST	0.59	0.53	0.57	0.009	0.088	1.02	1.03
GAZE_trans	0.47	0.42	0.44	0.012	0.093	1.07	1.04
GAZE_std_gf	0.35	0.35	0.35	0.012	0.076	1.03	1.09
GAZE_std	0.41	0.24	0.33	0.003	0.368	1.14	1.03
GENEFINDER	0.50	0.44	0.47	0.012	0.104	1.07	1.04
FGENESH	0.51	0.42	0.47	0.006	0.144	1.08	1.03
FGENESH_NO-W	0.18	0.16	0.17	0.012	0.125	1.07	1.09

Table 3.4: Comparative gene-level accuracy of various programs on WormSeq. Accuracy measures are explained in section 1.4.1

3.7.1 Gene-level accuracy

Table 3.4 collates the gene-level accuracy results for all GAZE models presented earlier (with the exception of GAZE_std+ and GAZE_std + +, which were for illustrative purposes only), as well as GENEFINDER and FGENESH.

The table shows that the accuracy of FGENESH is comparable with that obtained by GENEFINDER and GAZE_trans, although the former is slightly more sensitive and less specific. Reassuringly, the GAZE model making use of EST evidence performs best of all.

The fact that the GENEFINDER and GAZE_trans take account of *trans*-splicing begs the question of whether the comparable accuracy of FGENESH is obtained by it too incorporating a worm-specific model of gene structure. Since FGENESH is known to perform well on the sequences of a variety of organisms, it is natural to assume that it is the parameter files that give it organism specificity. Examination of an FGENESH parameter file reveals elements for signal, content and length distribution models, but nothing for the model of gene structure itself. It is informative therefore to run the program against WormSeq with the *C.elegans* parameter file but without specifying the "-nematode" command-line option (referred to in table 3.4 as FGENESH_NO-W). A marked decrease in accuracy is observed, suggesting that the option

	Base accuracy			Exon accuracy				
	Sn	Sp	CC	Sn	Sp	Av	ME	WE
GAZE_EST	0.99	0.93	0.94	0.90	0.84	0.87	0.02	0.09
GAZE_trans	0.98	0.91	0.93	0.86	0.80	0.83	0.03	0.11
GAZE_std_gf	0.98	0.90	0.92	0.84	0.77	0.80	0.04	0.12
GAZE_std	0.99	0.84	0.88	0.85	0.67	0.76	0.03	0.24
GENEFINDER	0.98	0.90	0.92	0.87	0.78	0.83	0.03	0.13
FGENESH	0.98	0.91	0.92	0.88	0.80	0.84	0.03	0.13

Table 3.5: Comparative base-pair-level and exon-level accuracy of GAZE_std on WormSeq. The accuracy measures are explained in section 1.4.1

activates a *C.elegans*-specific strategy, either a model of gene structure (as here), or something else such as a different evidence weighting scheme. Without access to the source-code, it is impossible to tell what exactly this is.

3.7.2 Accuracy at base-pair and exon-level

Table 3.5 shows the accuracy of the all models at the base-pair and exon-level.

The results are largely consistent with those at the gene-level, but some interesting elements are evident. Firstly, all programs show strikingly similar accuracy at the base-pair level, making it in this context at least not very useful as an accuracy measure. Secondly, although FGENESH was more sensitive and less specific than GENEFINDER at the gene-level, at the exon level it is both slightly more sensitive and specific. This suggests that the incorrect genes predicted by FGENESH contain small numbers of exons.

3.7.3 Accuracy by exon-type

Table 3.6 shows exon-level accuracy for each of initial, internal and terminal exons, as well as single-exon genes (termed "single").

The greater accuracy of all programs in the identification of internal exons sup-

		Sn	Sp	Av	ME	WE
Initial (309)	GAZE_EST	0.79	0.74	0.77	0.06	0.16
	GAZE_trans	0.72	0.67	0.70	0.11	0.19
	GAZE_std_gf	0.57	0.56	0.57	0.13	0.30
	GAZE_std	0.64	0.43	0.54	0.10	0.46
	GENEFINDER	0.72	0.66	0.69	0.11	0.22
	FGENESH	0.75	0.61	0.68	0.11	0.24
Internal (1620)	GAZE_EST	0.92	0.86	0.89	0.01	0.06
	GAZE_trans	0.89	0.83	0.86	0.01	0.08
	GAZE_std_gf	0.90	0.80	0.85	0.01	0.09
	GAZE_std	0.90	0.80	0.85	0.01	0.11
	GENEFINDER	0.92	0.82	0.87	0.01	0.10
	FGENESH	0.92	0.87	0.90	0.01	0.07
Terminal (309)	GAZE_EST	0.85	0.80	0.83	0.04	0.16
	GAZE_trans	0.81	0.74	0.78	0.06	0.18
	GAZE_std_gf	0.78	0.78	0.78	0.06	0.15
	GAZE_std	0.82	0.55	0.69	0.04	0.37
	GENEFINDER	0.80	0.72	0.76	0.07	0.21
	FGENESH	0.84	0.68	0.71	0.06	0.26
Single (16)	GAZE_EST	0.94	0.73	0.84	0.00	0.22
	GAZE_trans	0.94	0.59	0.77	0.00	0.26
	GAZE_std_gf	0.94	0.71	0.83	0.00	0.24
	GAZE_std	0.88	0.13	0.51	0.00	0.85
	GENEFINDER	1.00	0.63	0.81	0.00	0.29
	FGENESH	0.63	0.77	0.70	0.00	0.23

Table 3.6: Exon level accuracy on WormSeq, by exon type. The number exons of each type in WormSeq is shown in parentheses. Accuracy measures are explained in section 1.4.1

ports the observation that the ends of genes are more difficult to identify than the internal exon-intron structure. This is where EST evidence helps, and the table shows that the win in overall accuracy of GAZE_EST over all other programs is due largely to its better identification of initial and terminal exons; indeed, FGENESH has the slight edge for internal exons. The other notable aspect of these results is the difference between GENEFINDER and FGENESH in their identification of single-exon genes. GENEFINDER identifies all of them correctly at the expense of predicting many exons as single when in fact they belong as part of a multi-exon structure. FGENESH is apparently more conservative in predicting single-exon genes.

3.7.4 Genome scale accuracy

The precise identification of complete gene structures can depend on the genomic context of the genes; that is, their relationships to each other with respect to distance and orientation. It might be argued that extracting the genes from their genomic context, as was done in the construction of WormSeq, provides an artificial problem for gene prediction programs.

In construction of the WormSeq dataset, I have tried to provide as far as possible a realistic context for the cDNA-confirmed genes, by not inserting them into a randomly generated intergenic landscape as has been done by others [53], but by extracting the surrounding intergenic DNA with the genes. The technique of taking half of the region to the next curated gene in each upstream and downstream direction was an attempt to ensure that the distances between the genes was also realistic. However, it remains the case that certain aspects of gene organisation, such as operon structure, are disrupted by extracting the genes from their genomic context.

To address such concerns, I applied all of the models (as well as GENEFINDER and FGENESH) to the WormBase_Sanger DNA sequence, from which the WormSeq genes were extracted. This DNA amounted to 48,722,743 nucleotides, arranged in 9 contiguous sequences ranging in length from around 1 million to 12 million

	Base accuracy		Exon accuracy			Gene accuracy			
	Sn	Pred	Sn	Pred	ME	Sn	Pred	MG	SG
GAZE_EST	0.99	12.61M	0.90	60559	0.02	0.57	9075	0.012	1.02
GAZE_trans	0.98	12.76M	0.86	60860	0.03	0.80	9393	0.009	1.07
GAZE_std_gf	0.98	12.81M	0.85	61487	0.03	0.39	8645	0.009	1.03
GAZE_std	0.99	14.05M	0.86	71545	0.02	0.44	14685	0.003	1.14
GENEFINDER	0.98	13.00M	0.88	63560	0.03	0.51	9584	0.009	1.07
FGENESH	0.98	12.91M	0.88	62668	0.03	0.50	10707	0.004	1.07

Table 3.7: Comparative accuracy on WormBase_Sanger. Accuracy measures are defined in 1.4.1. The number of predicted residues, exons and genes (Pred) are quoted in lieu of specificity measures, which are not defined for a genome scale analysis

nucleotides. On a standard Compaq DS10 workstation, GAZE, GENEFINDER and FGENESH are happy dealing with gene-prediction data from sequences of a million base pairs or more, but for 12 million bases, all programs require memory resources that are beyond the limits of such a machine. For GAZE, the GENOME_GAZE script introduced in chapter 2 makes the analysis of such large sequences straightforward, without any requirement to split the DNA into several files. No such luxury existed for GENEFINDER and FGENESH, so it was therefore necessary to split the DNA into 1.1 Megabase chunks, with 0.1 Megabase overlap between each chunk. The predictions in the overlapping regions for these two programs were resolved by inspection. The results of applying all programs to what amounts to half of the *C.elegans* genome are depicted in table 3.7.

It is interesting that GENEFINDER seems to have a slight edge over FGENESH here, both in sensitivity and specificity. The reverse was true for WormSeq. Overall though, these results are consistent with those presented for WormSeq, suggesting that the test sequence is indeed a good approximation of a real genomic contig.

3.8 Examining the probabilistic aspects of GAZE

One of the novelties of GAZE with respect to other "exon assembly" based gene prediction systems is the ability to interpret the predictions in a probabilistic manner. As explained in chapter 2, the definition of a probability distribution over all possible gene structures (given a gene structure model), allows the calculation of *posterior* probabilities for individual gene components. In this section, I show how this facility can be used to reason about the reliability of predictions made by GAZE. I also discuss how posterior probabilities can act as an aid for manual curation of gene structures, in particular beginning to address the difficult problem of identifying alternatively spliced genes in *C.elegans*.

GAZE can be instructed to report posterior probabilities for (a) the *features* comprising the highest-scoring gene structure; (b) all candidate features; (c) the *regions* defined by adjacent features in the highest-scoring gene structure; and (d) all candidate regions of specified types. In the analysis presented here, I make use of the first two of these facilities, largely because predicted and candidate features can be designated as correct (with respect to the cDNA-confirmed gene structure) or incorrect. Predicted and candidate *regions* also carry the possibility of being *partially* correct, which adds an unnecessary complication to the analysis.

3.8.1 The reliability of GAZE predictions

GAZE reports a posterior probability for each feature that it identifies as belonging to the optimal gene structure. Since these are intended to be interpreted as degrees of belief in the correctness of the features, it is worth investigating how well they perform as indicators of reliability. For example, does a reported posterior probability of 0.5 for a feature really mean that we can be "50 percent sure" that the feature is correct? Figure 3.10 shows the posterior probabilities for the features comprising GAZE-predicted genes. Plotted for a variety of probability intervals are the number of features belonging to GAZE-predicted genes with a posterior in that interval, and the proportion of them that are correct.

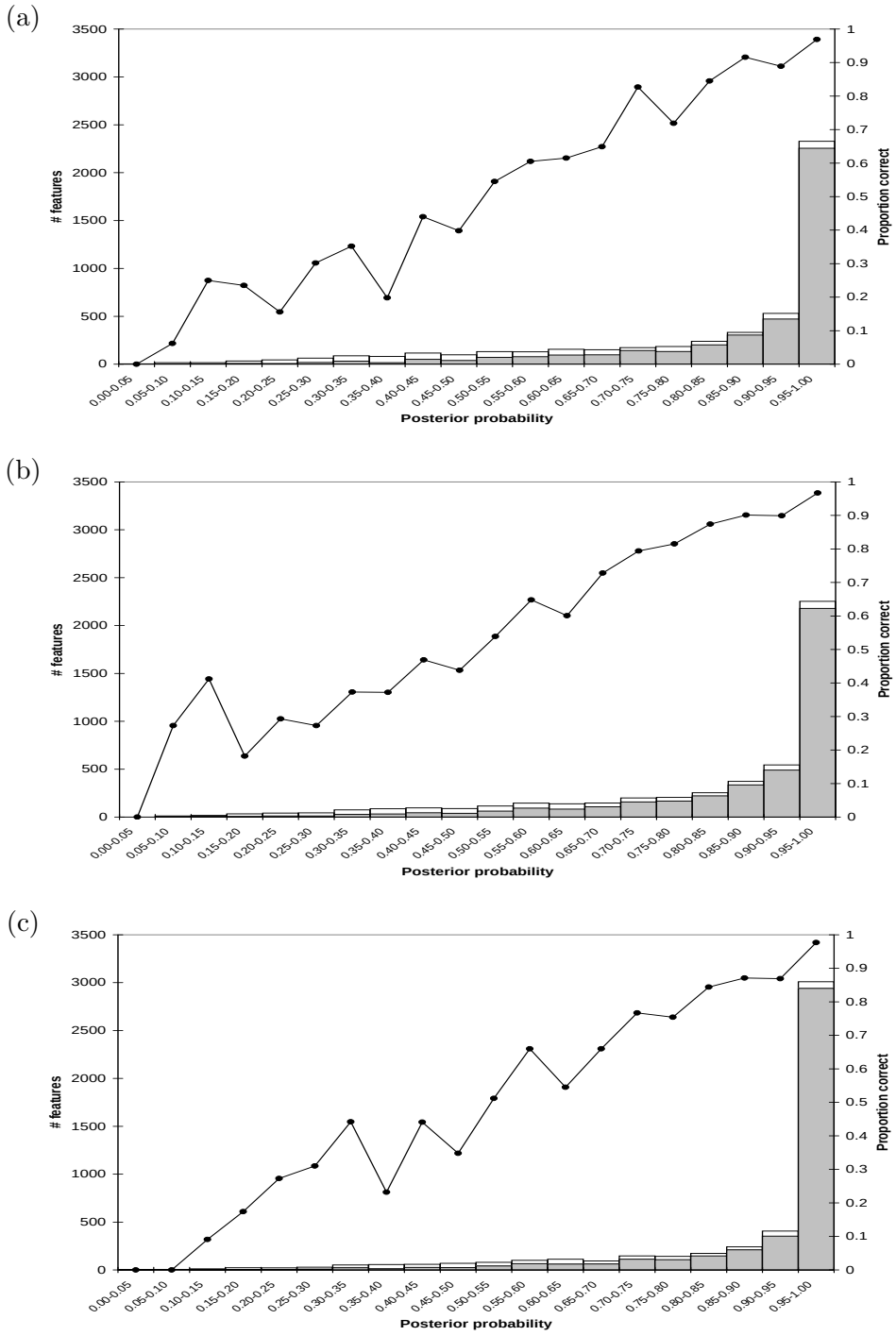


Figure 3.10: Posterior feature probabilities for GAZE models. Shown for features part of the coding-portions of predicted gene structures in each case are the number of features with a posterior probability in each interval (bars), the number of those predicted features that were actually correct (shaded portions of bars), and this number as a proportion of the predicted features in the interval (line). (a) GAZE_std_gf (4916 features); (b) GAZE_trans (4866 features); (c) the GAZE_EST (4832 features).

Two points are evident from the plots. Firstly, as the sophistication of the model increases, so does the number of features with high posterior probability. This implies that improving the model not only increases the accuracy of GAZE predictions, but also generally improves the confidence that it assigns to predictions. This particularly makes sense with the GAZE_EST model, where the EST evidence would be expected to add weight to many features belonging to gene structures predicted by the less sophisticated models. Secondly, the lines plotting the proportion of predicted features that are correct in each posterior probability range are close to the ideal, 1:1 line. This shows that the posterior probabilities are accurate indicators of reliability.

3.8.2 Feature probabilities can aid manual curation

The posterior probabilities reported by GAZE can also provide an aid for the *manual* curation of gene structures, which involves selecting from large numbers of candidate gene features, those that imply gene structures that are most consistent with the evidence. Figure 3.11 shows posterior probabilities for all of the candidate features presented to the GAZE models. Again, the proportion of features within each interval that are correct is consistent with the posterior probability computed by GAZE, suggesting that they are good indicators of reliability.

A striking feature of figure 3.11 is the number of features with low posterior probability. Because features with low or zero posterior probabilities do not fit into sensible gene structures, such features can be ignored by a human annotator, reducing the number of possible assemblies and therefore the likelihood of mistakes. Of the 872482 candidates for features that comprise the coding part of WormSeq genes (i.e. the starts, stops and splice sites), 587021 (67 percent) have zero probability (to 4 decimal places) according to GAZE_std_gf model, 576517 (66 percent) according to GAZE_trans, and 632067 (72 percent) according to GAZE_EST.

It is necessary to assess the likelihood that, by discarding features with zero probability, we discard features that are in fact correct. The only example of this

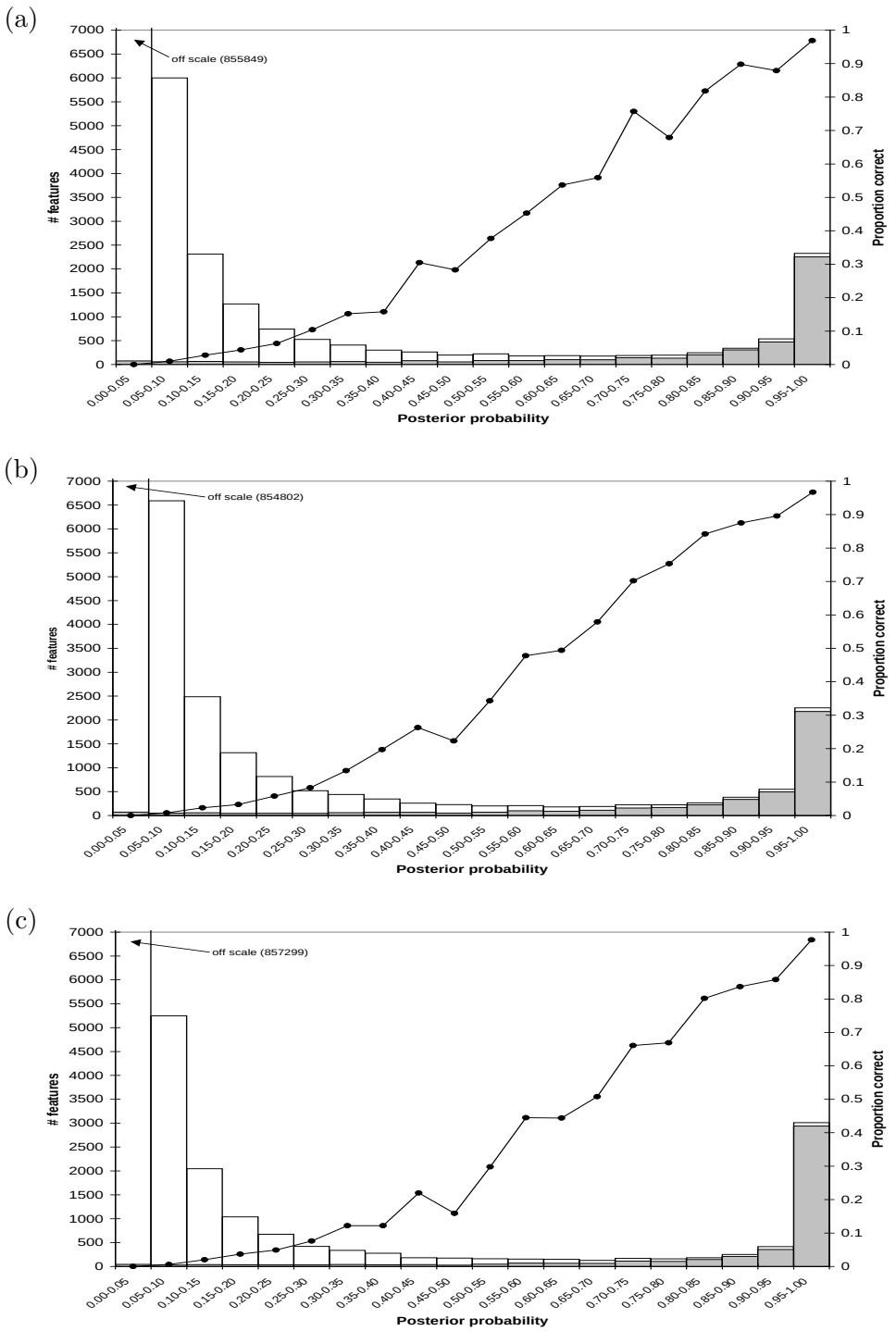


Figure 3.11: Posterior feature probabilities for the three GAZE models, for all candidate features; (a) GAZE_std_gf; (b) GAZE_trans; (c) GAZE_EST

occurring in WormSeq is the F09E8.3 gene, for which the high-scoring splice acceptor at the 5' end of the third exon is given zero probability by all three GAZE models. The problem here is that the donor splice at the 3' end of the exon is not detected by GENEFINDER at the default cutoff (it scores marginally below). In this case, it therefore becomes impossible to incorporate the acceptor splice feature into any gene structure with measurable probability. When the missing donor supplied as an external feature (a trivial task with GAZE), not only does the acceptor feature now have a very high posterior probability under all models (0.9993), but the low-scoring donor itself has a posterior probability of 1.0, underlining its vital importance in the gene structure. This particular example demonstrates well the utility to be gained from the posterior probabilities and the care that should be taken when interpreting them.

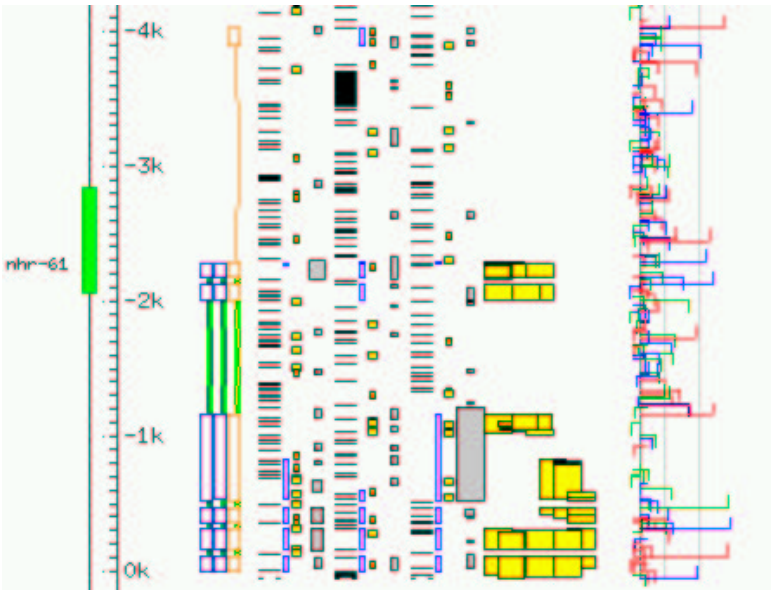
3.8.3 Feature probabilities could be used to identify alternative splicing events

Like most existing gene prediction programs, GAZE does not explicitly address the problem of identifying all of the variant gene structures for genes that are alternatively spliced, and this is still very much an open problem (see section 1.5). Burge [20] gave an example of how the sub-optimal exons reported by the GENSCAN program can sometimes be correct in alternative splice forms of the gene. GAZE offers the possibility of identifying not only sub-optimal exons, but also introns and other types of region, as well as features themselves. I show here how the *feature* posterior probabilities can begin to be used in the prediction of alternatively spliced genes.

Figure 3.12 depicts the gene structure of the nhr-61 locus in *C.elegans*, with its two alternatively spliced isoforms. The structure of this gene is has been confirmed by the alignment of full-length cDNAs for each isoform to the genome. As shown by the figure, the initial exon of neither isoform is identified precisely by any of the GAZE models presented (the one predicted by GAZE_EST is shown in the figure).

Upon examination of the posterior probabilities, we find first that of the fea-

(a)



(b)

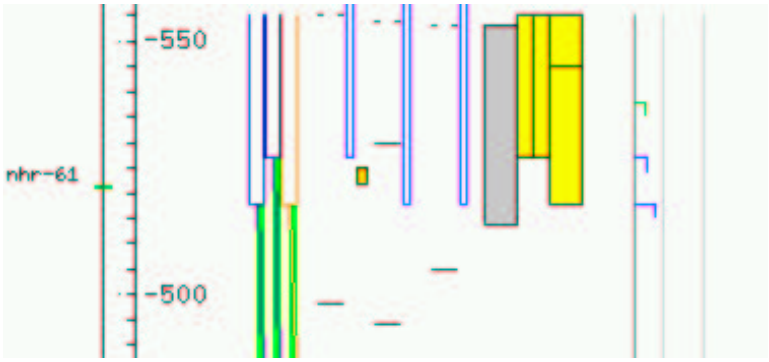


Figure 3.12: The *C.elegans* nhr-61 gene locus (WormBase WS52 identifier W01D2.2), with its two alternatively spliced isoforms (blue), and the GAZE_EST predicted structure (orange). (a) GAZE_EST fails to correctly identify the initial exon of either isoform; (b) An enlargement of the 3' end of the third exon of the correct gene structures, showing alternative splice donors, both supported by alignments of ESTs to the genomic sequence by EST_GENOME (yellow). Although only one of the two alternative donor features belongs to the correct gene structure, the posterior feature probabilities reported by GAZE provide evidence for both, as explained in the text.

tures predicted by GAZE that are not part of either correct gene structure, none of them have strikingly high probability; 0.432, 0.431 and 0.705 for the incorrect start, donor and acceptor. The probability reported for the *correct* start (not predicted by GAZE) is 0.269, not insignificant, suggesting that it is a viable alternative. Secondly, of the GAZE-predicted features part of both correct gene structures, all of them have probability of 0.999 or greater. Taken together, these two observations support what the earlier graphs showed, that the posterior probabilities are good indicators of reliability; GAZE has reported a higher degree of confidence about the parts of its prediction that turn out to be correct. Where the two isoforms differ, in their choice of donor splice site at the 3' end of the third exon, GAZE is less confident; 0.751 for the donor that is part of the GAZE prediction. Upon inspection of the posterior probabilities for all candidate features in this region, it becomes apparent that the "missing" probability is found in the alternative donor (0.249).

Although further work is required to bring together these ideas and observations into an automated system, this example (another was presented in [59]) demonstrates well how the GAZE feature posterior probabilities can point towards firstly elements of predicted gene structures that may not be correct, and secondly elements not part of predicted gene structures that might be correct, either in the single correct structure for the gene, or in one of the several possible structures of alternatively spliced genes.

Chapter 4

A method for estimating optimal parameters for a GAZE model

4.1 Introduction

A key property of the GAZE system is the ability to include signal and content information of arbitrary types from multiple sources. This was evident in the GAZE_EST model of the last chapter, where separate segment types corresponding respectively to regions of high coding potential and EST database hits both contributed to the score for candidate coding exons. For such an approach to work effectively, the evidence of each type must be *weighted* appropriately. For the GAZE_EST model, this problem was addressed in a rather *ad hoc* manner, by applying a scaling factor to the scores of the "EST_match" segments to make them approximately of the same order as the segments of high coding potential. This scheme worked well in this case, but in general it will be difficult to to obtain optimal evidence weightings in this way. Far more desirable would be a way to automatically derive optimal weights for the segments used in a model.

This chapter describes my work on an automated method for obtaining optimal weights for the scores of the various pieces of evidence referred to in a GAZE model. The idea is to obtain the set of weights for which the prediction accuracy of the model is as high as possible.

I start by modifying the GAZE scoring function to accommodate weights for each type of evidence. I then go on to describe the design and implementation of a method for obtaining the optimal set of weights for a GAZE model with respect to a set of training sequences with known gene structures. The technique is inspired by the work of Stormo and Haussler [109] and Krogh [68] (amongst others) and therefore contrast it with other kinds of parameter estimation in the field of gene prediction. The chapter ends with an example application, namely training the evidence weights for the GAZE_EST gene prediction system for *C. elegans* sequences (see chapter 3). A more detailed examination of the performance of the method is presented in the following chapter, whereby it is applied to the implementation of a GAZE gene prediction system for vertebrate sequences.

4.2 Evidence weighting in GAZE

4.2.1 Optimally parsing a sequence according to weighted evidence

In 1994, Stormo and Haussler published a method for *parsing* a sequence into regions of "intron" and "exon" using multiple types of weighted evidence for each of introns and exons [109]. In their method, exons and introns are scored according to the following scheme:

$$T_E(i,j) = \sum_{\mu \in M} w_\mu C_\mu(i,j)$$

$$T_I(i,j) = \sum_{\kappa \in K} w_\kappa C_\kappa(i,j)$$

M is a collection of types of *evidence* for exons; $C_\mu(i,j)$ is the raw "score" for evidence-type μ over the region $i \ldots j$ of the sequence; w_μ is a real number *weight*

for evidence-type μ. K, $T_\kappa(i,j)$ and w_κ are similarly defined for introns. The score of a *parse* of a query sequence into exons and introns is simply obtained by addition of the appropriate values from the T matrices. For example, for a sequence 1000 base pairs long, the parse (1-50,intron):(51-75,exon):(76-600, intron):(601-950, exon):(951-1000, intron) would be scored as:

$$T_I(1,50) + T_E(51,75) + T_I(76,600) + T_E(601,950) + T_I(951,1000)$$

The authors show how the highest scoring parse can be obtained by dynamic programming. They then go onto define a probability distribution over all possible parses, and present a procedure for obtaining the set of weights that maximises the probabilities of the correct parses of a set of training sequences. The method described in the following sections has at its core a generalisation of this procedure to make it applicable to GAZE, where the model of gene structure is not known in advance. I start though by describing how evidence weighting is accommodated in the GAZE scoring function.

4.2.2 Accommodating weights in the GAZE scoring function

A weight is attached to every piece of evidence that contributes to the score; that is, each feature, segment and length penalty function. Each weight is a floating point number by which the score of the associated element is scaled, and features (and segments) of the same type are scaled by the same weight. Intuitively therefore, the weights can be viewed as a way of ascribing a relative importance firstly to each element of the model (e.g. splice site versus translation start sites) and secondly to different types of evidence supporting the same region type (e.g. coding segments versus database matches for potential protein coding exons). I refer to the complete set of weights for a given model as $\mathbf{w}$.

In order to accommodate the weights, I start by reformulating the GAZE scoring (section 2.4). Given again an ordered list $\phi = \phi_1, \phi_2, \ldots, \phi_n$ of features defining a valid gene structure according to a GAZE model (with ϕ_0 and ϕ_{n+1} denoting

respectively the special features "BEGIN" and "END" marking the beginning and the end of the sequence; see chapter 2), then the weighted score of ϕ, $E(\phi, \mathbf{w})$ is calculated as:

$$E(\phi, \mathbf{w}) = \sum_{i=0}^{n} T(\phi_i, \phi_{i+1}, \mathbf{w}) \tag{4.1}$$

$$\begin{aligned} T(\phi_i, \phi_j, \mathbf{w}) &= Seg_{t(\phi_i) \to t(\phi_j)}(l(\phi_i), l(\phi_j), \mathbf{w}) \\ &+ Len_{t(\phi_i) \to t(\phi_j)}(l(\phi_j) - l(\phi_i) + 1, \mathbf{w}) \\ &+ Loc(\phi_j, \mathbf{w}) \end{aligned} \tag{4.2}$$

The function $T(\phi_i, \phi_j, \mathbf{w})$ is calculated as a composite score for the region $\phi_i \to \phi_j$, comprising components for segment, length penalty and local feature scores as before (see equations 2.1, 2.2). The difference is that weights for the various types of evidence are included in T.

I construct a mapping W from the types of the features, segments and length penalty functions to the elements of $\mathbf{w}$ such that each of the former is associated with exactly one of the latter. For example, $W(\mathbf{w}, t(\phi_i))$ is the element of $\mathbf{w}$ that is the weight for feature ϕ_i and all other features of the same type. Given this mapping, the feature, segment and length-penalty components of the score are scaled in the following way:

Feature score weighting

The weighted local score for a feature ϕ_i is calculated as the given score scaled by the weight :

$$Loc(\phi_i, \mathbf{w}) = g(\phi_i) W(\mathbf{w}, t(\phi_i)) \tag{4.3}$$

Segment score weighting

Recall from equation 2.3 that the segment score for a region is a sum of scores for each segment qualifier given in the model rule for the region. If again ψ^q is the relevant subset of segments for segment qualifier q, and $t(q)$ is the type of those segments[1], then the weighted segment score is calculated as:

$$Seg_{src \to tgt}(x, y, \mathbf{w}) = \sum_{q \in Q_{src \to tgt}} Seg^Q(\psi^q, x, y) W(\mathbf{w}, t(q)) \tag{4.4}$$

The additive nature of both specific segment scoring strategies (equations 2.4 and 2.5) means that this weighting can be applied in practice directly to the given scores for each segment in advance (according to type), as was the case for the features above.

Length penalty weighting

Finally, the weight for a length penalty function is a simple scaling factor applied to each penalty value:

$$Len_{src \to tgt}(x, \mathbf{w}) = Len_{src \to tgt}(x) W(\mathbf{w}, t(Len_{src \to tgt})) \tag{4.5}$$

where $t(Len_{src \to tgt})$ is an identifier for the length penalty function used in the rule $src \to tgt$. This approach assumes that the *shape* of the function has already been determined by some sort of inversion of a frequency-of-occurrence histogram. An alternative would be to model each function as a series of constants, each subject to a separate scaling factor. This would potentially provide a powerful method for the simultaneous estimation of the shape and weight of each penalty function. Doing this however would greatly increase the number of free variables of the system, and I will not consider it further here.

[1] all relevant segments for a Segment Qualifier must have the same type due to the compulsory type constraint

In practice, specific values for the weights are defined in the configuration file, via a "mul" attribute attached to the the declaration of each element of the gene structure model. When GAZE is used in prediction mode, the given values are used to scale the scores of the appropriate model element in the way described above before the dynamic programming is performed. When used in parameter-estimation mode, the given values define the starting value of the function to be optimised.

4.3 Two approaches to obtaining an optimal set of weights

The aim of the method is to obtain the set weights $\mathbf{w}^{opt}$ that maximises the gene prediction accuracy in a set of training sequences for which the gene structures are known. The natural approach would be to design a function of the weights that represents the gene prediction accuracy in the training sequences, and then maximise this function with respect to the weights. A problem with this method is that the standard measures of accuracy are not continuous functions of the weights. This is because "accuracy", as it has been defined so far, depends only upon the highest scoring gene structure, and a small change in the weights can lead to a large change in the optimal gene structure. By using the posterior probabilities of chapter 2 however, it is possible to construct an accuracy function that is continuous in the weights. I have implemented two accuracy functions based on these posterior probabilities. The degree to which they give rise to an effective set of weights is investigated in the following chapter. The remainder of this chapter focuses on the design, implementation and optimisation of the functions themselves.

4.3.1 Maximum Likelihood

This method involves identifying the set of parameters that maximises the log-probability of the correct (i.e. annotated/confirmed) gene structures in a set of training sequences. Stormo and Haussler [109] showed how to do this in the context of a gene prediction system based on weighted evidence, and their method is directly

applicable to the GAZE framework. Recall equation 2.9 which defines a probability distribution over all possible valid gene structures (given a model). If there are K training sequences, and the *correct* gene structure in training sequence k is ϕ^{k*}, then the ML approach is to maximise the following as a function of $\mathbf{w}$:

$$\alpha^{\mathrm{ML}}(\mathbf{w}) = \sum_{k=1}^{K} \ln \mathrm{P}(\phi^{k*}|\mathbf{w}) \tag{4.6}$$

The posterior probability of the correct gene structure is thus made to be as high as possible.

4.3.2 Maximal Feature Discrimination

By maximising the posterior probability of the correct gene structure, we at the same time minimise the summed probabilities of all *incorrect* structures. Intuitively, this appears desirable, but the drawback is that all of the incorrect gene structures are considered equally "incorrect". This is clearly not a good representation of gene prediction accuracy; some candidate structures, although not completely correct, will be closer to the correct structure than others. By maximising the probability of the correct structure only, we may (and often do) increase the probability of individual incorrect structures so that they score better than the correct structure, and the highest scoring structure is even "less correct" than before.

The other main problem of the ML approach as presented above is that it relies heavily on the knowledge of a single complete correct gene structure. We know that many sequences have more than one structure that can be described as correct, due to alternative splicing. It will also generally not be the case that we have accurate data about all the feature types present in a model. In the previous chapter, I showed how a generic model of gene structure could be extended to incorporate *trans*-splice site candidates (via acceptor splice site predictions) and transcription initiation and termination candidates (via EST information). The predictions made by the extended model include regions defined by the additional features, such as untranslated regions (where evidence was present). The assessment of the accuracy

of these predictions was not completely straightforward, because the "correct" gene structures with which I was comparing included only the protein-coding part of each gene. This is true of the majority of benchmark sets used to train and assess the accuracy of gene prediction programs. I therefore took the approach of considering only the protein-coding part of each predicted structure in the assessment, and this worked adequately. Such a method will not work for the maximum-likelihood training function above however; supplying a "correct" gene structure consisting of a proper subset of the feature-types referred to in a GAZE model will cause the weights for the other feature types to tend to negative infinity. We therefore need a more general way of dealing with correct gene structures for which only the regions involving specific feature types are known.

To address both of these problems, I have implemented a new objective function of $\mathbf{w}$ that maximises the posterior probabilities of the *features* that are part of the correct structure, while minimising the same for those that are not. In addition, I provide a way of ignoring features of particular types in the optimisation. For consistency with above, I work in log probabilities. If $\phi_1^k \ldots \phi_{n(k)}^k$ this time is the list of candidate features for training sequence k (where the special features ϕ_0^k and $\phi_{n(k)+1}^k$ mark the beginning and end of the sequence as usual), then the function is:

$$\alpha^{\mathrm{MFD}}(\mathbf{w}) = \sum_{k=1}^{K} \sum_{i=0}^{n(k)+1} r(\phi_i^k)[c(\phi_i^k) \ln \mathrm{P}(\phi_i^k|\mathbf{w}) + (1 - c(\phi_i^k)) \ln(1 - \mathrm{P}(\phi_i^k|\mathbf{w}))] \quad (4.7)$$

$$\begin{aligned} c(\phi_i^k) &= 1 \quad \text{if} \quad \phi_i^k \in \phi^{k*} \\ &= 0 \quad \text{otherwise} \end{aligned} \quad (4.8)$$

$$\begin{aligned} r(\phi_i^k) &= 1 \quad \text{if} \quad t(\phi_i^k) \notin Ignore \\ &= 0 \quad \text{otherwise} \end{aligned} \quad (4.9)$$

The ideal result of maximising this function is a set of weights for which *all* gene structures use as many correct features and as few incorrect features as possible. We might expect, by optimising this function, to perform better on feature-based measures of gene prediction accuracy. I refer to this method as Maximal Feature Discrimination (MFD).

The problem of knowing only portions of the correct gene structure that involve specific feature types is also addressed by this function: in its computation, we simply ignore the features that are not of one of these specific types. We therefore maximise the probabilities of the features which we know to be correct and minimise the probabilities of those that are known to be incorrect; the probabilities of the other, ignored features are unconstrained. It is important to note that this is *not* a general strategy for making use of partial correct gene structures; if a feature type is considered relevant for the computation of the above function, then all of the features of that type present in the given "correct" gene structure are considered correct, and, importantly, all other candidate features of that type are considered incorrect. Hence the method is for dealing with feature *types* that are absent from the the correct gene structures, not individual features.

4.4 Optimising the objective functions by gradient descent

There are many published methods for maximising a multi-dimensional function such as those described above (see [88] for a detailed review). I have chosen to use a gradient-based method. Each evaluation of the functions above requires dynamic programming (see chapter 2), and a gradient-based method should give rise to fewer function evaluations than a method that does not employ such information, particularly when the number of dimensions is high.

I first describe the maximisation scheme that was used in terms of an anonymous multi-dimensional objective function of a vector of parameters, $\alpha(\mathbf{w})$. I go on to show

in the next section how the derivatives of the two objective functions described in the previous section can be obtained, thus making the method directly applicable to GAZE.

4.4.1 A conjugate gradient descent method

To be consistent with other literature on function optimisation, I present this as a minimisation algorithm. Maximisation of a function is equivalent to minimisation of the negative of the function.

We wish to find the values for the elements of $\mathbf{w}$ for which the function $\alpha(\mathbf{w})$ is at its minimum. The method of *steepest descent* begins with an arbitrary point $\mathbf{w}^0$, and then repeatedly (a) computes a vector of partial derivatives $\nabla\mathbf{w}^t$ at the current point $\mathbf{w}^t$ and then (b) minimises along the line $\mathbf{w}^t + \eta\nabla\mathbf{w}^t$ to obtain a new current point $\mathbf{w}^{t+1}$:

$$\mathbf{w}^{t+1} = \min_{\eta} \alpha(\mathbf{w}^t + \eta\nabla\mathbf{w}^t)$$

The procedure is terminated when some stopping condition is satisfied, for example when the change in function value is small enough.

The "line minimisation" step is the most complicated part of the procedure and involves two basic steps: (a) identifying a range for η within which the minimum must lie (i.e. *bracketing* the minimum) and (b) successively narrowing this region until it is small enough to define the minimum to within a tolerable error based on the precision of the machine (i.e. a *section* search). For both of these steps, I have implemented multi-dimensional versions of the algorithms described in the book by Press *et. al.* [88]. In particular, the section search method was originally described by Brent [16].

As explained in [88], the problem with the simple steepest descent method is that for many common functions, it will perform many small steps in descending a long narrow valley-like function. This is because consecutive line minimisations are necessarily in orthogonal directions, so the process zig-zags from side to side.

I have therefore implemented a variation of the steepest descent method proposed by Fletcher and Reeves [43] and later modified by Polak and Ribiere [87]. The idea of their method is to minimise not in the direction of the gradient at the new point, but in a direction that does not interfere with any of the previous directions travelled. The concept of non-interfering directions is formalised by Press *et. al.* [88] as *conjugacy*. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be *conjugate* (i.e. non-interfering) if the following condition holds:

$$\mathbf{u}.\mathbf{A}.\mathbf{v} = 0$$

where $\mathbf{A}$ is the matrix of second partial derivatives. The authors show that by scaling and subtracting the last direction travelled $\mathbf{u}$ from the gradient at the new point, a new direction conjugate to $\mathbf{u}$, $\mathbf{v}$ can be obtained without explicit calculation of this matrix. The resulting method belongs to a sub-family of similar methods known in the literature as *conjugate gradient descent*.

The gradient descent method as I have described it has the desirable property that the process is divided into "chunks" (line minimisations), between which progress can be monitored and assessed if necessary.

4.5 Calculating the gradient by dynamic programming

To use conjugate gradient descent, or any gradient-based method, it is necessary to be able to obtain, for any function point, the partial first-derivatives with respect to each variable. Furthermore, it is necessary to do this efficiently, as the reason for using gradient information in the first place is to keep the time taken to reach the minimum as small as possible. In this section, I show how the derivatives of both of the functions proposed in section 4.3 can be efficiently computed.

4.5.1 The derivative of the ML function

The partial derivative of the Maximum Likelihood function (equation 4.6) with respect to single variable w in $\mathbf{w}$ is obtained by expressing the log probability of the

correct gene structure in terms of its constituent components. Since the function comprises a sum over the training sequences, I simplify the notation by assuming a single training sequence with correct gene structure denoted by ϕ^*.

$$\begin{aligned}\frac{\partial\,\alpha^{\mathrm{ML}}(\mathbf{w})}{\partial\,w} &= \frac{\partial\,\ln P(\phi^*|\mathbf{w})}{\partial\,w} \\ &= \frac{\partial}{\partial\,w}\ln\frac{\mathrm{e}^{E(\phi^*)}}{Z} \\ &= \frac{\partial\,E(\phi^*)}{\partial\,w} - \frac{Z'}{Z}\end{aligned} \tag{4.10}$$

where Z is the partition function (given by equation 2.8), and Z' is its derivative. The derivative of the score of the correct gene structure can be computed simply as the sum of the derivatives of the components. Given again a list $\phi_0, \phi_1, \ldots, \phi_{n+1}$ defining the correct gene structure, we have:

$$\frac{\partial\,E(\phi,\mathbf{w})}{\partial\,w} = \sum_{i=0}^{n}\frac{\partial}{\partial\,w}T(\phi_i,\phi_{i+1},\mathbf{w}) \tag{4.11}$$

$$\begin{aligned}\frac{\partial}{\partial\,w}T(\phi_i,\phi_j,\mathbf{w}) &= \frac{\partial}{\partial\,w}Seg_{t(\phi_i)\to t(\phi_j)}(l(\phi_i),l(\phi_j),\mathbf{w}) \\ &+ \frac{\partial}{\partial\,w}Len_{t(\phi_i)\to t(\phi_j)}(l(\phi_j)-l(\phi_i)+1,\mathbf{w}) \\ &+ \frac{\partial}{\partial\,w}Loc(\phi_j,\mathbf{w})\end{aligned} \tag{4.12}$$

The feature, segment and length-penalty components are each themselves linear sums of terms involving at most one variable, so it is straightforward to obtain their derivatives. In particular, the derivative with respect to parameter w will be zero for all components of the score that are not relevant for that weight, and equal to the sum of raw (unweighted) scores for the component otherwise.

If Φ is the space of all gene structures, the derivative of the partition function can be expressed as follows:

$$Z' = \sum_{\phi\in\Phi}\frac{\partial\,E(\phi)}{\partial\,w}\mathrm{e}^{E(\phi)} \tag{4.13}$$

It can thus be thought of as an "weighted average" of the derivatives of all gene structures. A different but related term occurs in the gradient of the Maximal Feature Discrimination objective function. I will show how these quantities can be calculated in section 4.5.3, after deriving the gradient of the MFD function.

4.5.2 The derivative of the MFD function

For the Maximal Feature Discrimination function (given by equation 4.7) I again simplify notation by assuming a single training sequence with a complete list of candidate features $\phi_0 \ldots \phi_{n+1}$.

$$\begin{aligned}
\frac{\partial\, \alpha^{\mathrm{MFD}}(\mathbf{w})}{\partial\, w} &= \frac{\partial}{\partial\, w} \sum_{i=0}^{n+1} r(\phi_i)[c(\phi_i) \ln P(\phi_i|\mathbf{w}) + (1 - c(\phi_i)) \ln(1 - P(\phi_i|\mathbf{w}))] \\
&= \sum_{i=0}^{n+1} r(\phi_i) \left[\frac{c(\phi_i)}{P(\phi_i|\mathbf{w})} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} - \frac{1 - c(\phi_i)}{1 - P(\phi_i|\mathbf{w})} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} \right] \\
&= \sum_{i=0}^{n+1} r(\phi_i) \left[\frac{c(\phi_i) - P(\phi_i|\mathbf{w})}{P(\phi_i|\mathbf{w})(1 - P(\phi_i|\mathbf{w}))} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} \right] \qquad (4.14)
\end{aligned}$$

In order to obtain the derivatives of the posterior feature probabilities, I first define Z_i to be the sum of exponentiated scores of all gene structures that include feature ϕ_i:

$$Z_i = \sum_{\phi \in \Phi : \phi_i \in \phi} \mathrm{e}^{E(\phi)} \qquad (4.15)$$

The term Z_i can be thought of as an "i-restricted" partition function, and dividing it by the unrestricted partition function gives the posterior probability for feature ϕ_i (see equation 2.15). The derivative of the posterior probability of ϕ_i can now be derived:

$$\begin{aligned}
\frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} &= \frac{\partial}{\partial\, w} \frac{Z_i}{Z} \\
&= Z^{-1} \frac{\partial\, Z_i}{\partial\, w} - Z_i Z^{-2} \frac{\partial\, Z}{\partial\, w} \\
&= P(\phi_i|\mathbf{w}) \left(\frac{Z_i'}{Z_i} - \frac{Z'}{Z} \right) \qquad (4.16)
\end{aligned}$$

The derivative of the i-restricted partition function, Z'_i can be expressed in terms of a "weighted average of derivatives" in the same way as the unrestricted function:

$$Z'_i = \sum_{\phi \in \Phi : \phi_i \in \phi} \frac{\partial E(\phi)}{\partial w} \mathrm{e}^{E(\phi)} \tag{4.17}$$

Expressing the derivatives of the i-restricted and unrestricted partition functions in this way is the first step towards the design of a single method for the computation of both, shown next.

4.5.3 Computing the weighted average of the derivatives

The aforementioned Stormo and Haussler article [109] includes a sketch of a dynamic programming procedure for the efficient computation of a quantity analogous to Z' above, i.e. the gradient of the partition function. A simple extension of their method allows for the simultaneous computation of the gradients of both the i-restricted and unrestricted partition functions.

The method relies firstly upon the fact that a gene structure $\phi_1 \dots \phi_n$ can be decomposed into two partial gene structures $\phi_1 \dots \phi_i$ and $\phi_i \dots \phi_n$, with the score of the complete structure being the sum of the two partial structures[2]. If we denote the set of all partial gene structures *ending* with feature ϕ_i as $\Phi^{\cdot\cdot\phi_i}$, and likewise the set of all partial structure *beginning* with feature ϕ_i as $\Phi^{\phi_i\cdot\cdot}$, then Z'_i can be expressed in terms of all partial structures beginning and ending (respectively) with feature ϕ_i:

$$\begin{aligned} Z'_i &= \frac{\partial}{\partial w} \sum_{\phi^x \in \Phi^{\cdot\cdot\phi_i}} \sum_{\phi^y \in \Phi^{\phi_i\cdot\cdot}} \mathrm{e}^{E(\phi_x)} \mathrm{e}^{E(\phi_y)} \\ &= \sum_{\phi \in \Phi^{\cdot\cdot\phi_i}} \mathrm{e}^{E(\phi)} \sum_{\phi \in \Phi^{\phi_i\cdot\cdot}} \frac{\partial E(\phi)}{\partial w} \mathrm{e}^{E(\phi)} \end{aligned}$$

[2] the asymetry of the scoring function is helpful here; the local score for feature ϕ_i is included in the partial structure for which it forms the end, but not for the structure for which it forms the start

$$
\begin{aligned}
&+ \sum_{\phi \in \Phi^{\phi_i \cdot \cdot}} \mathrm{e}^{E(\phi)} \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \\
= \; & f(i) \sum_{\phi \in \Phi^{\phi_i \cdot \cdot}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} + b(i) \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \qquad (4.18)
\end{aligned}
$$

The final step follows from chapter 2, where the exponentiated scores of partial structures ending and beginning with feature ϕ_i were computed respectively as $f(i)$ and $b(i)$ (equations 2.11, 2.14).

All that remains is to obtain the weighted average derivatives for structures beginning and ending (respectively) at ϕ_i. Let $u^f(i, w)$ be the weighted average derivative of all partial structures ending with feature ϕ_i, with respect to parameter w. Each of these partial structures can be decomposed further into a partial structure ending at ϕ_j $(j < i)$, and the final region $\phi_j \rightarrow \phi_i$. The score of this region can be calculated independently as $T(\phi_j, \phi_i, \mathbf{w})$, which allows us to derive a dynamic programming recurrence for u^f:

$$
\begin{aligned}
u^f(i, w) &= \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \\
&= \sum_{0 \leq j < i} \sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \left(\frac{\partial\, E(\phi)}{\partial\, w} + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} \right) \mathrm{e}^{E(\phi)} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \\
&= \sum_{0 \leq j < i} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \left(\sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} \sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \mathrm{e}^{E(\phi)} \right) \\
&= \sum_{j < i} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \left(u^f(j, w) + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} f(j) \right) \qquad (4.19)
\end{aligned}
$$

The vectors for each parameter u^f can thus be populated by dynamic programming (with $u^f(0, w) = 0$), and the weighted average derivative over all structures, Z', is obtained directly as $u^f(n + 1, w)$. It is straightforward to construct an analogous "backwards" matrix where $u^b(i, w)$ denotes the weighted average of the derivatives of partial structures *beginning* with feature i :

$$
u^b(i, w) = \sum_{i < k \leq n+1} \mathrm{e}^{T(\phi_i, \phi_k, \mathbf{w})} \left(u^b(k, w) + \frac{\partial\, T(\phi_i, \phi_k, \mathbf{w})}{\partial\, w} b(k) \right) \qquad (4.20)
$$

The derivative of the i-restricted partition function then reduces to the following:

$$Z'_i = f(i)u^b(i,w) + b(i)u^f(i,w) \tag{4.21}$$

Drawing all of these elements together, the derivatives of both the Maximum Likelihood and Maximal Feature Discrimination functions can now be written down in terms of elements that are directly computable by the methods presented here and in chapter 2:

$$\frac{\partial\,\alpha^{\mathrm{ML}}(\mathbf{w})}{\partial\,w} = \left(\sum_{\phi_x \to \phi_y \in \phi^*} \frac{\partial\,T(\phi_x,\phi_y,\mathbf{w})}{\partial\,w}\right) - \frac{u^b(0,w)}{b(0)} \tag{4.22}$$

$$\frac{\partial\,\alpha^{\mathrm{MFD}}(\mathbf{w})}{\partial\,w} = \sum_{i=0}^{n+1} r(\phi_i)\left(\frac{c(\phi_i) - P(\phi_i|\mathbf{w})}{1 - P(\phi_i|\mathbf{w})}\right)\left(\frac{u^b(i,w)}{b(i)} + \frac{u^f(i,w)}{f(i)} - \frac{u^b(0,w)}{b(0)}\right) \tag{4.23}$$

4.6 Implementation issues

4.6.1 Numerical stability

As in chapter 2, the calculation of the derivatives of the i-restricted and unrestricted partition functions is performed in log-space to cope with limitations in machine precision:

$$U(0) = -\infty$$

$$U^f(i,w) = \ln \sum_{0 \le j < i} \left(\mathrm{e}^{T(\phi_j,\phi_i,\mathbf{w}) + U^f(j,w)} + \frac{\partial\,T(\phi_j,\phi_i,\mathbf{w})}{\partial\,w}\mathrm{e}^{F(j)}\right)$$

Unlike the log-space computation of the forward and backward variables presented in chapter 2, the sum above is not guaranteed to be positive. This is due to the unexponentiated term in the sum, the derivative of $T(\phi_i,\phi_k,\mathbf{w})$, which is often negative. Additional book-keeping is therefore necessary in which I take the log of *absolute* value of the sum, keeping track of those (i,w) for which $\mathrm{e}^{U^f(i,w)}$ is in reality

negative and reversing the sign of the exponentiation of the $U^f(j, w)$ in the equation above for those cases.

The Maximal Feature Discrimination function and its derivative are problematic to implement in practice due to the fact they are both undefined in the cases of a correct feature having posterior probability of 0 or an incorrect feature having a posterior probability of 1. Such situations are not impossible, and can be approached arbitrarily closely if the feature set is incompatible with the correct gene structure. I therefore took the natural approach of replacing $\log 0$ and $\frac{-x}{0}$ with a large negative numbers. Both the value and the gradient of the function are somewhat inaccurate therefore at extreme values for the weights, but since such extremities are typically only encountered whilst bracketing the minimum, this is unimportant.

It is often the case that functions with complex behaviour such as those being optimised here have many local minima; the local minimum obtained depends greatly on the starting conditions of the procedure, in this case the initial values for the weights. There are two techniques commonly used to address this problem. The first is *simulated annealing* [63], whereby a gradually diminishing level of "noise" is added to the parameter values to give the function a stochastic opportunity to escape from a local minimum. I have implemented the other common technique which is to perform the optimisation several times from different starting points, and choose the set of parameters that give the overall lowest function value at termination. This undirected strategy works well in my case, because there is usually some approximate idea of what the weights should be for a given model. By starting the process from such weights we increase the chance of reaching a global minimum, or at least a set of weights that work well. Re-optimising with different starting conditions and obtaining a solution that is the same or larger is verification that the minimum obtained is likely to be global.

4.6.2 Parameter tying

In many cases, it makes sense that the score for two (or more) different pieces of evidence should have the same weight. This is true for example when different feature or segment types are used for the forward and reverse strand of a sequence. I have therefore included a mechanism for indicating that the weights for two or more model elements should be *tied* together during the optimisation process. Parameter tying is a common technique in the application of Hidden Markov Models to speech recognition. It also has widespread use in the field of neural networks [11], where is known as *weight sharing*.

In the GAZE framework, tying is achieved simply via the function W that maps feature, segment and length penalty types onto weights. For the description earlier, I implied that this mapping was one-to-one. By making it many-to-one, more than one evidence type can map to the same weight. In practice, this means that a weight tied to many pieces of evidence is updated according to a sum of contributions from each.

It is also worth saying here that although the framework presented assumes for flexibility that the scores for *every* type of evidence referred to in a GAZE model is associated with a weight (with some possibly associated with the same weight, as explained above), this need not be the case. The optimisation system has a mechanism to allow the user to specify which types of evidence are to be treated as having variable weight. This has no effect on the equations presented above; the derivative of all functions with respect to non-variable weights is zero.

4.6.3 Time and memory usage

Most of the issues discussed in chapter 2 concerning the time and memory usage of GAZE are not different here and are addressed in the same way; the algorithm for the gradient calculations is ostensibly the same as the forward and backward algorithms, and in fact can be implemented as extensions of both. The run-time is increased by a constant factor of up to 10 however, so it is therefore sensible to only

calculate the gradient when necessary.

The optimisation algorithm can be summarised as a series of gradient calculations, each followed by a minimisation along the line defined by the current point and the new direction (a function of the new gradient and previous gradients). The first of these steps involves a single forward-backward-with-gradient run over the training sequences. The second, line minimisation requires a succession of function evaluations. The section search algorithm by Brent [16] has been extended by Press [88] to make use of derivative information with the aim of reducing the number of evaluations necessary in the line minimisation. However, I found the overhead in calculating a forward-backward-with-gradient for *every* function evaluation as opposed to a straightforward forward-backward run was too significant for an overall benefit to be observed. My implementation therefore does not use gradient information at the line minimisation stage.

The main issue with the gradient calculation itself is memory usage. It is now necessary to store with each feature, as well as the usual properties (e.g. location and score), the u^f and u^b variables, each of which consists of an array of floating point numbers, the size of which is the number of different weights that are to be simultaneously optimised. If we wish to optimise weights for all evidence types for a simple, double-stranded model separately, then this requires an approximate additional 100 bytes per feature (16 feature types, 4 segment types, 5 length penalties), twice as much if we wish to use double-precision floating point arithmetic. Since a 1 Megabase sequence typically produces of the order of half a million features, this gives 50 Megabytes for the gradient variables alone!

The situation is never this bad in practice. Tying the forward and reverse versions of the features and segments together effectively halves the storage required, and I would expect this to be done for all sensible applications. In addition, it would be normal to tie together the weights for related feature types on the same strand, such as the six different splice sites (three donor, three acceptor).

4.7 A comparison with other methods

In this section I contrast the optimisation scheme described above with some of the ways in which parameters are estimated for other gene prediction systems. There are some elements of the scheme that are similar to methods used by exon-assembly predictors to incorporate scores from several distinct signal and content-sensors into a single exon score. It is also true that the procedure is probabilistic in flavour and has much in common with certain techniques used in the estimation of parameters for Hidden Markov models.

I present the comparison therefore in two parts. In the first part, I briefly examine some of the ways in which evidence of several different types is accommodated and weighted in some other gene prediction systems, relating them back to the GAZE strategy. The second part is more theoretical and discusses how the GAZE approach relates to various techniques used in the field of HMM parameter estimation.

4.7.1 Other methods based on weighted evidence

In many gene prediction systems, a variety of signal and content sensors are employed for the recognition of individual exons, which are then assembled into gene structures by dynamic programming. The specific way in which weights are chosen for the exon features varies. In FGENES for example [104], linear discriminant analysis (see chapter 1) is used to find a linear, weighted combination of a series of exon quality measures that provides maximum discrimination between true and false exons [103]. The exon likelihoods reported by the method are then used as the basis for a dynamic programming algorithm which identifies the highest scoring exon assembly. GRAIL [113][117] on the other hand uses a neural network to combine the scores from several signal and content sensors, trained to discriminate between true and false individual exons. The result gives rise to a procedure for identifying a set of candidate exons, which are assembled by a separate, dynamic programming procedure called GAP [118].

The weight optimisation technique presented in this chapter differs from these

methods by the fact that it considers whole gene structures[3]. I focus now on two methods that are similar to GAZE strategy in this respect.

GeneParser

The GAZE method is a fully implemented extension of the unimplemented framework proposed by Stormo and Haussler [109], generalised to account for a gene structure model that is not known in advance. The most comparable previously implemented work is the GENEPARSER system of Snyder and Stormo [101] [102]. The final gene prediction program is identical to the one proposed by Stormo and Haussler (section 4.2.1), but the difference is how optimal weights for various types of evidence are obtained.

Like GRAIL, GENEPARSER uses a neural network to combine the scores from several signal and content sensors. Unlike GRAIL however, the dynamic programming procedure to obtain the optimal parse is included in the neural network training regime. This allows for it to be trained not on individual exons but on complete gene structures. An iterative procedure is used, where the input to the network at each state is the difference (for each type of evidence) between the score of the *optimal* parse and the *correct* parse using the weights obtained in the last stage. A set of weights is thus obtained which maximises the number of training sequences for which the optimal parse is the correct one.

This approach is distinct from the maximum likelihood and maximum feature discrimination methods for GAZE, where the parameters are estimated with respect to the correct gene structure, or individual features belonging to it; the GENEPARSER method focuses directly on the relationship between the correct structure and the *highest scoring* structure. It could be argued that this makes more sense because it relates directly to the way in which the program will be used for prediction after

[3]This is true of both the ML and MFD methods; the latter is a function of feature *posterior* probabilities which, as explained in chapter 2, can be interpreted as a measure of how well the feature can be accommodated into a high-scoring complete gene structure.

training.

EuGene

EUGENE [99] is a program similar to GAZE in the way that it is tailored for integrating the output of external programs predicting gene features. It models the sequence as a directed acyclic graph (DAG) with the nodes organised into rows, or "tracks". Each track corresponds to a region of functional class and comprises a node for each base and edges joining the nodes of adjacent bases. The graph is then decorated with edges between tracks, depending on the results of the external feature-prediction programs. For example, if a program of choice reports donor splice site at position $(x, x+1)$, then an edge is added between node x of the "exon" track and node $x+1$ of the "intron" track. Once this is done, any path from the "source" node to the "sink" through the graph corresponds to a gene structure. Scores are added to the edges of the graph and the predicted gene structure is that corresponding to the "shortest" path through the graph[4].

What makes EUGENE particularly interesting from the perspective of the work presented here is the way in which the scores reported by the external programs are weighted. It is assumed by the authors that the scores c_i reported by a program predicting evidence type i are numbers between zero and one, and to transform such scores into a penalty they use the function $-\log(ac_i^b)$. The constants a and b are obtained by maximising the percentage of correct predictions in a set of single gene *A.thaliana* sequences. The optimisation is performed first by a simple genetic algorithm [50], and then by sampling the local parameter space. The procedure is extremely computationally intensive and is not guaranteed to find either a global or local maximum. The authors justify it however as a method that identifies sets of parameters that work well. A system with a similar architecture is DAGGER [25], which combines evidence using a complex function with two free variables for each type of (in their system fixed) evidence. Optimal values for the variables are found by

[4]The scores are therefore penalties, with a high score corresponding to an unlikely feature.

the downhill simplex function optimisation method [81] which does not make use of gradient information and is therefore be inefficient for multi-dimensional functions. The authors also give very little justification for the form of the objective function except that it works well.

It is interesting to note that although these methods have weaknesses, they go one step beyond the GAZE method in associating *two* parameters with each piece of evidence, namely a multiplication factor and an additive constant (analogous to bias terms in neural networks). This is however achievable in GAZE under certain circumstances by using a fixed, length-independent penalty function to act as the bias term for the transformation of a feature score (see for example section 5.5.1 describing the introduction of transcription start site predictions). Care must be taken to apply this technique in general as there would be indeterminacy between some sort of additive constants, for example, those for "intron" and "exons" regions.

The general approach of associating each model element with two parameters would double the number of free variables in the system, putting a strain on the time an memory usage of the training procedure, as well as making it much more vulnerable to over-fitting. For this this reason, I have not examined the possibility further. It is however a natural extension to this work to extend the gradient calculation to account for additive constants.

4.7.2 Hidden Markov model methods

In order to place a probabilistic interpretation upon the weight-optimisation procedure presented in this chapter, it is instructive to look to the way in which the parameters of methods based on Hidden Markov models are estimated.

The basic set of parameters for a HMM gene predictor are the state emission and transition probabilities. The standard approach to obtaining these parameters is maximum likelihood, whereby the probability of the observations (in this case a set of training sequences) is maximised:

$$\max_{\mathbf{w}} \sum_{k=1}^{K} \log \mathrm{P}(S_k|\mathbf{w})$$

I use $\mathbf{w}$ to denote the complete set of model parameters, to be consistent with the notation earlier. How this function is optimised depends on whether the training sequences come complete with annotated gene structures or not.

Supervised estimation

If each base of the training sequence is labelled according to its role in the gene structure ("intron", "exon", "UTR" etc.), and there is a one-to-one correspondence between the states of the model and these labels then a "correct" state path can be inferred directly from the labelled training sequence. The ML approach therefore is to calculate values for each transition and emission probability based on their frequency of occurrence in the correct paths of the training sequences[5]. This in fact corresponds to maximising the joint probability of the sequence and the correct path ϕ^*:

$$\mathbf{w}^{\mathrm{ML}} = \operatorname*{argmax}_{\mathbf{w}} \sum_{k=1}^{K} \log \mathrm{P}(S_k, \phi^*|\mathbf{w})$$

The same applies to the Generalised HMM architecture used by many of the most successful gene prediction programs, for example GENSCAN [21], GENIE [72], FGENESH [96] and developmental versions of GENEMARK.HMM [76]. In these models, it is often the case each state corresponds to a particular functional class (i.e. a single label). Sub-models can therefore be estimated for each component individually, based on only the corresponding regions from the training sequences. The sub-models are then combined with transition probabilities based again on counts in the paths inferred from the training sequences. The GHMM formalism also allows for arbitrary probability distributions over the durations of each state, although as

[5] With pseudo-counts being added when data available for the estimation of the parameters is sparse.

explained in chapter 1 for practical reasons this is usually only exploited for certain state types. State duration probability distributions can be derived directly from histograms compiled from observed data.

Unsupervised estimation

It is sometimes not possible to infer a correct state path from a training sequence. This could be because the sequence does not have an annotated gene structure, or because there is many-to-one relationship between states and annotation labels, meaning that there is more than one state path that could have given rise to the annotated gene structure. A general approach in this case is therefore to maximise the unconditional probability of the training sequence(s), summing over all paths (or if there is partial information, summing over all paths consistent with this information; see later). There are various iterative techniques for maximising the likelihood in this case, based on the fact that the log probability of a training sequence can be expressed as a sum over all state paths, the partition function:

$$\log \mathrm{P}(S|\mathbf{w}) = \log \sum_{\phi} \mathrm{P}(S, \phi|\mathbf{w})$$

I showed above how the gradient of this function can be calculated in the context of the GAZE scoring function, and for HMMs the result is similar [67]:

$$\frac{\partial}{\partial w} \log \mathrm{P}(S|\mathbf{w}) = \frac{n_k(S)}{w}$$

where $n_k(s)$ is the "expected" number of uses of the kth parameter in generating the training sequences, i.e. the weighted average over all possible paths. This can be calculated by the standard forward-backward algorithm for HMMs [90] [36]. A gradient descent method such as that presented earlier can therefore be used to maximise the likelihood. However, the more efficient *Baum-Welch* procedure [5] is often used. The algorithm works on the principle that by estimating new parameter values from "expected" counts above, the log-likelihood of the training sequences is *necessarily* increased.

An example of the use of the Baum-Welch method can be found in one of the first applications of HMMs to gene prediction [70]. The authors use the algorithm to estimate a profile-based HMM for two distinct types of intergenic regions in the genome of *E. Coli*. Elements of the resulting model were observed that were biologically meaningful, validating the method.

A combined approach: HMMGene

HMMGENE is a gene prediction program based on the notion of a *Class* Hidden Markov model (CHMM), where each state is associated with a label (see chapter 1). This framework allows for the identification of the most probable *labelling* (and hence most likely gene structure) in addition to the most probable state path.

The estimation of the parameters for HMMGENE begins with a set of labelled training sequences, but it is not possible to estimate maximum likelihood parameters from counts directly. Although HMMGENE implements a restricted version of the CHMM framework whereby each state emits a single, constant label, it is still true that a given label could have been generated by more than one state.

The natural approach to this problem is to use a modified version of a Baum-Welch algorithm where, during the forward-backward calculation, only valid path through the model are allowed. In the context of labelled sequences, a valid path is one in which the state labels agree with the sequence labels. This maximises the joint probability of the sequence and the labelling L:

$$\mathbf{w}^{\mathrm{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} \log \mathrm{P}(S, L|\mathbf{w})$$

This is the function that is optimised in the training of the VEIL program, which also uses the CHMM framework [56]. The problem with this technique is that it can often give rise to parameter values that are poor for prediction; although the probability of emitting a series of "Exon" labels for an exonic region of the training sequence is made as high as possible, the parameters so obtained can make the probability of emitting a series of "Intron" or "Intergenic" labels with the sequence even higher.

HMMGENE uses a technique called *Conditional* Maximum Likelihood (CML), to address this problem. The idea is to optimise the probability of the correct *labelling*, given the sequence:

$$\begin{aligned} \mathbf{w}^{\mathrm{CML}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \log \mathrm{P}(S|L,\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\log \mathrm{P}(S,L|\mathbf{w}) - \log \mathrm{P}(S|\mathbf{w})\right] \end{aligned} \tag{4.24}$$

where $\mathrm{P}(S|\mathbf{w})$ is the sum over all state paths and labellings. This can be computed with the standard forward algorithm (i.e. ignoring the labels). Since there is a one-to-one correspondence between labellings and gene structures, maximising it corresponds to maximising the probability of the annotated gene structure.

The effect of CML is that the relative difference between the probabilities of consistent state paths (i.e. those where the labelling of the states agrees with the labelling of the training sequence) and inconsistent state paths (i.e. where the state labelling does not agree with the labelling of the training sequences) is maximised. This ideally results in the probability of invalid paths being very low, meaning that *all* paths through the model (with significant probability) will give rise to the correct gene structure. Since the aim of CML is to arrive at a set of parameters that provides maximum discrimination between consistent and inconsistent state paths (with respect to the labelling of the training sequences), the technique is also known as *discriminative estimation* [36].

Relationship to the GAZE approach

The estimation problem in GAZE is simplified with respect to the above in that the parameters of signal and content recognition sub-models are fixed. The problem is therefore to find a way to ascribe relative importance to each element so that their resulting integration results in gene predictions that are as accurate as possible.

To see how my training method relates to HMM training, it is insightful to consider the GAZE gene structure probabilities as implicitly conditional upon the

underlying sequence (see section 2.5). Making the conditioning explicit, the ML objective function (equation 4.6) can therefore be written as:

$$\begin{aligned}
\mathbf{w}^{\mathrm{ML}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \ln \mathrm{P}(\phi^* | S, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} E(\phi^*, \mathbf{w}) - \ln \sum_{\phi \in \Phi} \mathrm{e}^{E(\phi, \mathbf{w})} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \ln \mathrm{P}(\phi^*, S | \mathbf{w}) - \ln \mathrm{P}(S | \mathbf{w}) \qquad (4.25)
\end{aligned}$$

When using this framework it is assumed that a correct gene structure (in terms of a list of features) can be inferred directly from an annotated sequence. Since this means that there is no distinction between gene structures and labellings, the above might be interpreted as a kind of conditional maximum likelihood. This makes intuitive sense, since like CML, the above minimises the probability of all incorrect gene structures. The principle difference between the above and CML in the CHMM context is that the latter simultaneously estimates the transition and emission probabilities of the sub-models. In the context of GAZE, the sub-models are implicit in the features, hence their internal model parameters are not variable.

The MFD method goes one step beyond CML in attempting to discriminate between individual incorrect gene structures as well as between the single correct and all incorrect structures. The idea is that even in the case where the highest scoring gene structures are not correct, they at least use as many elements (features) of the correct structure as possible. This should intuitively give rise to sets of weights that are more accurate. Whether this turns out to be the case is examined in the following chapter.

To end, it is interesting to consider whether unsupervised learning methods are applicable to the estimation of optimal evidence weights in GAZE. Maximising the unconditional probability of the training sequences (i.e. the partition function) directly by classical methods is not possible because a GAZE score is not a real log-probability; there is no comparison to other sequences. It is likely therefore that increasing the weights will increase the partition function in an unbounded

way. The possibility of an indirect method however remains open. The Baum-Welch method is a special case of *Expectation Maximisation* [31] for maximum-likelihood-style estimation in the case where certain data (in this case the correct gene structures) are not known. Specifically, expectation maximisation is based upon the following calculation:

$$\mathbf{w}^{t+1} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{\phi} \mathrm{P}(\phi|S, \mathbf{w}^t) \log \mathrm{P}(\phi, S|\mathbf{w})$$

It can be shown that under a wide range of conditions, this approach necessarily increases the likelihood of the model, and in many situations, the maximum is calculable directly without iteration. Such a case is that of HMMs and the Baum-Welch algorithm. The straightforward application of the EM method to HMMs arises due to their transparency in the way that once posterior path probabilities corresponding to expected gene structures are calculated by the forward-backward procedure, transition and emission probabilities that increase the objective function are obtained by simply summing the expected usages and normalising.

To apply EM to GAZE, we would have to use the Maximum Likelihood approach, weighting gene structure probabilities by their posterior probability in the previous iteration:

$$\begin{aligned} \mathbf{w}^{t+1} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{\phi} \mathrm{P}(\phi|\mathbf{w}^t)\mathrm{P}(\phi|\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \frac{\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w}^t)+E(\phi|\mathbf{w})}}{[\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w}^t)}][\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w})}]} \end{aligned}$$

It is not obvious how this quantity might be maximised analytically, as is possible with HMMs and the Baum-Welch algorithm. The standard numerical optimisation techniques however are applicable. To use conjugate gradient descent method in particular would the require the computation of the gradient of the above quantity with respect to the model element weights. Such an approach would be a natural extension to this work.

4.8 Optimising evidence weights for GAZE_EST

In the previous chapter, I described the design and refinement of a GAZE model for the prediction of genes in *C. elegans* sequences. In the final stage of this refinement, data derived from EST alignments were used to improve the accuracy of the system. However, the weights for the scores for these data were determined in a rather *ad hoc* way. The Maximal Feature Discrimination training method provides the opportunity to attack this problem in a more principled way. To end the chapter, I illustrate the use of MFD training by applying it to this specific problem. A more rigorous examination of the method in comparison with maximum likelihood is detailed in the following chapter, in the context of the development of a gene prediction system for vertebrate sequences.

4.8.1 Choice of parameters and optimisation method

The types of additional evidence in the GAZE_EST model (with respect to the GAZE_trans model) are described in section 3.6.2. They are four features ("transcript_start" and "transcript_stop", and reverse strand versions of these) and five segments (forward and reverse strand versions of "EST_match" and "EST_intron", as well as the "EST_span" segments which were introduced to prevent gene splitting). Tying the weights of corresponding forward and reverse strand elements together results in five parameters to optimise. However, since the transcription initiation and termination features have zero score (see section 3.6.2), any scaling applied to them will have no effect. In addition, the "EST_span" segments are designed to eliminate from consideration any candidate gene structures that make use of them; since the correct gene structure will not make use of them, any weight attached to their scores will be unbounded. It is appropriate therefore to optimse only two parameters.

For the work in chapter 3, the WormSeq dataset was used to assess the accuracy of various GAZE models. To be able to use this dataset for training as well as testing, it was split in half (adjusting the division point slightly to occur in the intergenic region between two genes). I refer to these two new sequences as *WormSeq-1* and

WormSeq-2. Splitting the dataset in this this way enables a twofold cross-validation, testing the parameters obtained by training on WormSeq-1 on WormSeq-2, and vice versa.

Of the two objective functions described earlier, only Maximal Feature Discrimination is applicable in this instance. The reason for this is that since only coding regions of the confirmed genes in WormSeq are known, the correctness of certain types of candidate feature (specifically *trans*-splice sites, and transcription and initiation sites) is not known during the training. However, MFD allows the declaration of a subset of feature types that should be ignored in the optimisation (see section 4.3.2).

4.8.2 Accuracy of the trained model

Table 4.1 shows the weights obtained from the MFD optimisation, and table 4.2 shows the accuracy resulting from using these weights. The most noticeable difference in the values compared with those chosen by hand in chapter 3 is that the weight for "EST_match" segments has doubled. However, this gives only marginal (if any) improvement in accuracy. It might be argued that this should not be too surprising, since the reason for choosing the original weights is that they seemed to give good accuracy.

One point that was not considered when choosing the weights in the previous chapter is that by introducing additional elements to the model, the weights of the existing model elements may no longer perform well. In this application, "EST_match" and "coding_seg" segments both contribute towards the score for candidate protein coding exons. It is therefore possible that the scores for "coding_seg" segments have to be down-weighted to compensate for the addition of "EST_match" segments. Tables 4.1 and 4.2 also show the result of simultaneously optimising weights for the "EST_match", "EST_intron" and "coding_seg" segments (the entry labelled GAZE_EST$^{\mathrm{MFD}-3}$). The optimal value for the "coding_seg" weight is 0.4 (compared with 1.0 used in chapter 3, and the new set of weights gives significant

	EST_match	EST_intron	coding_seg
MFD-2^1	0.02	0.05	-
MFD-2^2	0.02	0.06	-
MFD-3^1	0.02	0.05	0.42
MFD-3^2	0.02	0.06	0.41
Chapter 3	0.01	0.05	1.00

Table 4.1: Evidence scoring weights determined by Maximal Feature Discrimination. In addition to the weights used in chapter 3 (bottom row), the final weights for the relevant elements are shown for both two-parameter (MFD-2) and three-parameter (MFD-3) training runs described in the text, and for each training sequence WormSeq-1 and WormSeq-2 (indicated by superscript 1 and 2 respectively).

(a)

Exon level	WormSeq-1					WormSeq-2				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_EST$^{\mathrm{MFD-2}}$	0.89	0.83	0.86	0.02	0.09	0.90	0.85	0.87	0.02	0.09
GAZE_EST$^{\mathrm{MFD-3}}$	0.88	0.88	0.88	0.06	0.06	0.88	0.89	0.89	0.06	0.05
GAZE_EST	0.89	0.83	0.86	0.02	0.10	0.90	0.84	0.87	0.02	0.09

(b)

Gene level	WormSeq-1					WormSeq-2				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_EST$^{\mathrm{MFD-2}}$	0.56	0.50	0.53	0.01	0.01	0.62	0.58	0.60	0.01	0.07
GAZE_EST$^{\mathrm{MFD-3}}$	0.59	0.58	0.58	0.04	0.07	0.65	0.64	0.65	0.03	0.04
GAZE_EST	0.54	0.49	0.52	0.01	0.11	0.61	0.58	0.60	0.01	0.07

Table 4.2: Accuracy of the GAZE_EST configuration after training with Maximal Feature Discrimination. For the trained models, the results for WormSeq-1 were obtained by training on WormSeq-2, and vice versa. The first rows are the result of training two parameters (weights for "EST_match" and "EST_intron" segments), and the second rows a further parameter in addition (a weight for "coding_seg" segments). The bottom row represents the accuracy of the weights used in chapter 3 when the model is applied separately to WormSeq-1 and WormSeq-2. The accuracy measures are explained in section 1.4.1.

improvement in accuracy (for example a 5-6% increase at the gene level average).

This simple application shows Maximal Feature Discrimination to be a good method for identifying sets of weights that give rise to accurate gene predictions. The effectiveness of the method is examined in more detail in the next chapter, where amongst other things it is compared to the more traditional maximum likelihood method.

Chapter 5

Application of GAZE training to the development of a vertebrate gene finder

5.1 Introduction

In this chapter, I demonstrate how the parameter estimation methods described in chapter 4 can be used to tune the performance of a gene prediction system created with GAZE. By way of a contrast to chapter 3, the methods are applied to the problem of predicting gene structures in the sequences of higher vertebrates such as human [112] and mouse [28]. Gene prediction is more difficult in vertebrate sequences than in the sequences of primitive animals such as *C.elegans* due primarily to a lower signal-to-noise ratio and also other factors discussed in section 1.5 and briefly in chapter 3.

The chapter follows a similar format to that of chapter 3. After summarising the materials for vertebrate gene finding that are to be used, an initial GAZE configuration is outlined. I then describe how the methods of the previous chapter can be used to optimise the parameters of the model, and the effectiveness of the Maximum

Likelihood and Maximal Feature Discrimination methods is compared. I finally go on to produce three variant models, each incorporating a different, new type of gene prediction evidence, and investigate the effectiveness of Maximal Feature Discrimination in weighting the scores of the new data appropriately.

5.2 Materials for gene prediction in vertebrate sequences

5.2.1 Datasets for training and testing

Programs to predict gene structures in vertebrate genomic DNA have historically been trained and tested on sequences that each contain a single, complete gene structure. In chapter 3, I discussed the disadvantages of using single gene sequences for testing. Large, contiguous genomic sequences which are completely understood in terms of their gene structures are even harder to come by for vertebrate sequences than for *C.elegans*. Even for the human chromosomes that have been declared "finished" at time of writing [35][24][30], the gene structures are under continual review, and it is likely that these sequences contain unannotated genes.

In the work on *C.elegans* gene predictions described earlier, the problem was addressed by constructing an artificial genomic sequence of cDNA-confirmed gene structures. This was done by extracting the genomic DNA underlying each gene structure, along with some upstream and and downstream intergenic sequences. Another approach was taken by Guigo and co-workers [53] in the generation of their SAGS (semi-artificial genomic sequences) dataset. They also started with a set of single-gene sequences and their annotated gene structures, but not having any genomic context for the sequences, they created an artificial one. In essence, the sequences were separated by artificial intergenic regions, the lengths of which were normally distributed, and the content of which were based on a fifth order Markov chain based on a C+G content of 38% [54].

The problem with both of these methods is that they do not take account of the variation in mammalian gene structural properties with C+G content (see chapter

1). For programs like GENSCAN that have distinct sets of parameters for different C+G% strata, a sequence constructed with either method would provide an unfair test. By way of illustration, each of the 42 sequences comprising the Guigo SAGS dataset has a C+G content of less than 40%. GENSCAN therefore uses the same, low-C+G% set of parameters for these sequences, which could be one of the main reasons for its relatively poor performance [53].

For this reason, and also for consistency with other literature assessing the accuracy of gene prediction programs in vertebrate sequences, the results presented in this chapter are based upon two distinct sets of single-gene sequences. It is important to note however that all of the GAZE gene structure models used are sufficiently general to allow for multiple genes on both strands of the input sequence, even allowing partial genes at the ends of the sequence.

Training: the H176 dataset

This set was constructed by Guigo and colleagues [53] for an analysis of the accuracy of various gene prediction programs available at the time. It was made in a similar manner to the Burset and Guigo dataset benchmark set vertebrate gene sequences [23], the principle difference being that this set comprises human gene sequences only, and that entries were extracted from the EMBL nucleotide database version 50 (March 1997). There are 178 sequences in the set, from which I removed 2 sequences which had annotated gene structures that were clearly incorrect (HSADH6, with a gene structure containing seven introns each 25 base-pairs in length; and HSPVALB, gene structure with three 24-base introns). The resulting 176-sequence set is referred to as H176 or *the training set*.

Testing: the HMR195 dataset

This set was constructed by Rogic and colleagues for another survey of the accuracy of gene prediction methods [94], using filtering rules ostensibly the same as those used by Guigo in the construction of H176, the primary differences being (i) mouse and

	Human Genome	H176	HMR195	WormSeq
Number of genes	-	176	195	325
Coding density	-	0.13	0.14	0.24
Single exon genes	-	40	43	16
Exons per (multi-exon) gene	8.8	6.2	6.0	7.2
Mean internal exon length	145	146	138	241
Mean CDS length	1340	970	1021	1542
Mean intron length	3365	672	854	308

Table 5.1: Some properties of the training and test sets of vertebrate sequence in comparison with the WormSeq dataset of chapter 3, and estimates for the human genome published in [112].

rat sequences were retained, as well as human; (ii) the Genbank nucleotide database was used (version 111.0, April 1999), and sequences deposited before August 1997 were discarded, to ensure as far as possible that none of the sequences could have been used to train the gene prediction programs being evaluated. The resulting 195-sequence set is referred to as HMR195 or *the test set*. There are no entries in the test set that are also in the training set. The degree of overlap between this set and the training set is discussed below.

By retaining mouse and rat genes in this set, I am assuming that any properties of the H176 (human only) dataset learned by the training process also generalise to the sequences of these organisms. The architects of the dataset partitioned it into human (103 sequences) and mouse/rat (92) subsets, and examined the accuracy in each of a number of programs that were trained on human sequences only. They found the difference to be insignificant, with the many of the programs having marginally better accuracy in the mouse/rat sequences.

5.2.2 Properties of the gene sets

Some properties of HMR195 in comparison with H176 and the WormSeq dataset of chapter 3 are summarised in table 5.1.

The table shows that two vertebrate datasets have properties that are fairly consistent with each other. They both support the observation that vertebrate genes have longer introns and shorter exons than worm genes, although the larger survey summarised in the first column [112] suggests that both sets are still atypical, most noticeably having smaller introns and fewer exons.

The two vertebrate datasets provide a far easier problem for gene prediction programs than they would usually face in practice; a protein coding density of 13-14 percent is about 5 times higher than in the human genome as a whole (for example). Furthermore, the complexity of the genes, in terms of their numbers of exons and introns, and total CDS length, is lower than that of the WormSeq dataset of *C. elegans* sequences. Also, 23 percent of the vertebrate genes contain coding regions that are confined to a single exon, compared to 5 percent in the *C. elegans* dataset). This is not indicative of any characteristic difference in complexity of vertebrate and worm genes, but due to simple sample bias; short genes with small numbers of exons were easier to sequence genomically before the human genome project, hence their disproportionate frequency as separate entries in the nucleotide databases. The *C. elegans* gene structures in WormSeq were confirmed by full-length cDNAs however, so would be affected less by such database bias.

It is technically necessary to correct for such biases in the dataset before training. When calculating the transition probabilities of the GENSCAN HMM for example, Burge took the "true" proportion of single-exon genes to be one half of that observed in the training set. He comments that although this approach is rather *ad hoc*, it is better than no correction at all, although it is more difficult to correct for other biases in databases. I have chosen to ignore all biases because the primary aim of this chapter is to assess the effectiveness of the training methods, and there is no reason to believe that that any biases present in the training set should not also be present in the test set.

There are no entries that occur in both datasets (ensured by construction; see above), but the approach taken by many would be to remove entries in the test set

that show a significant degree of similarity to an entry in the training set. Burge for example in the training and testing of GENSCAN removed sequences from his training set for which the translation of the annotated gene showed more than 25% identity to the translation of a gene in his intended test set [20]. I take the approach of not attempting to make the training and test sets non-redundant with respect to each other, also taken by others [102] [94]; it is reasonable to expect a gene prediction program to often be presented with a sequence containing a gene which shows some degree of similarity to a member of its training set.

One final point to note is that all analysis presented in this chapter was performed on data derived from the raw sequence, with no masking of repeats. There is no mechanism in GAZE itself to account for the possibility that the input feature list may have been derived from repeat-masked sequence, and will therefore contain regions with no stop-codons that should not be considered as candidate coding exons. Repetitive regions therefore have to be explicitly accounted for in the configuration file. One approach is to make segments for the repeats, giving their scores high negative weights, and making them contribute towards the score for candidate protein-coding regions.

Accounting for repeats in this way would be necessary for the analysis of large, unannotated stretches of genomic DNA, but is less important here. Although the repeat-content of the human genome (for example) has been estimated to be in excess of 50% [112], RepeatMasker [A.F.A. Smit and P. Green, unpublished] identifies 21% and 15% of the training and test sets (respectively) as repetitive[1]. Masking these repeats had no significant impact on the accuracy of the standard gene prediction programs (results not shown).

[1]The -nolow option was used which does not mask out low-complexity regions, as these can sometimes be protein-coding, for example in proteins with coiled-coil regions.

5.2.3 A source of gene prediction features: GENEID

As in chapter 3, I draw upon the work of others for a source of gene prediction data. In this case, I used the most recent version of the GENEID program [84], primarily because it is designed for gene finding in vertebrate sequences (specifically human) and optionally outputs candidate gene features as well as predictions of complete gene structures.

The signal and content sensing models used by GENEID are fixed and described below. The parameters for these models however are external to the system and supplied by the user in a file. The program comes with two parameter files for gene finding in human sequences. The first provides distinct parameter-sets for sequences in three C+G% strata. The second file contains a single set of parameters obtained without considering C+G content. Except where otherwise stated, I have chosen to work with features generated from this latter, homogeneous set of parameters.

Signal sensing models in GENEID

GENEID detects signals by calculating weight matrices from frequency tables compiled from real and pseudo examples of the feature of interest. The signals provided are: (i) translation start sites, representing positions -8 through +5 (where 0 is the position of the A in the conserved ATG); (ii) translation stop sites, representing positions -5 through +3 (where 0 is the position of the first nucleotide of one of the three stop codons); (iii) donor splice sites, representing positions -3 through +5 (where 0 is the position of the G in the conserved GT); and (iv) acceptor splice sites, representing positions -22 through +4 (where 0 is the position of the A in the conserved AG). The first three of these signals are modelled using the weight matrix method. The acceptor splice site model is more sophisticated being a first order weight array model. Predicted features using these models were generated using GENEID in signal output mode with default cutoffs (command line option '-bdal').

Content sensing models in GENEID

Unlike GENEFINDER, GENEID does not explicitly provide prediction of likely protein-coding regions, but the supplied parameter files contain details of the model used to score candidate exons for coding potential. As described in [84], log-likelihood ratios $C^j(b_1b_2b_3b_4b_5b_6)$ for hexamer $b_1b_2b_3b_4b_5b_6$ beginning in codon position j, are calculated according to the relative frequency of the hexamer in a set of real exons compared with introns. A "coding" score for a candidate exon $S_1 \ldots S_n$ with pre-assigned phase j (i.e. the codon position of the first base of the exon is known) can then be calculated by summing the scores for the hexamers along the length of the exon:

$$L_C(S_1 \ldots S_n, j) = \sum_{i=1}^{n-5} C^{(j+i-1)\%3}(S_i \ldots S_{i+5})$$

where % is the modulus operator. Rather than using C^j values given in the GENEID parameter file to obtain a set of segments corresponding to likely coding regions in each frame (as was done in chapter 3), six GFF segments were made for each 6 base-pair region in the training and test sequences, three for each strand, by assuming that the region starts in each of the three codon positions. A coding score for a candidate exon can be calculated by summing the segments lying in the region, and I show later the configuration file directives that make this possible. Although requiring a large amount of storage for the GFF files of these segments (largely due to the un-necessary redundancy in this case of GFF), this technique will allow for the calculation of an exact coding likelihood for a candidate exon. The reason that the GAZE models for *C.elegans* gene prediction were not able to duplicate the performance of GENEFINDER was due to the inexactness of the coding score calculation; see chapter 3.

Length penalty functions in GENEID

The GENEID scoring function, explained in more detail shortly, does not include a length penalty component. However, exon scores are subject to a constant, length-

independent penalty, the value of which is supplied as a parameter. These were used in arriving at initial configuration for human gene finding, as explained next.

5.3 A GAZE configuration for human gene finding

A natural approach towards developing a GAZE configuration for the integration of signal and content sensors from GENEID is to start by implementing a system that integrates the data in a similar manner to that program.

5.3.1 A GAZE configuration based on GENEID

The model of gene structure used by GENEID for human gene finding has the same basic components and connectivity as the GAZE_std model for *C.elegans* (figure 3.4). The are minor differences in the maximum and minimum distance constraints as well as the way that partial exons are disallowed (with incomplete introns at the end of the sequence still permissible).

There are however more significant differences in the way in which gene structures are scored. In GENEID, the score of a candidate gene structure is a sum of scores for each of the exons it comprises:

$$L_G(e_1 e2 \ldots e_n) = L_E(e_1) + L_E(e_2) + \ldots + L_E(e_n)$$

The exon scores themselves are calculated as sums of the scores of (i) the upstream defining feature L_U (translation start site or acceptor splice site), weighted by a constant W_U; (ii) the downstream defining feature L_D (translation stop site or donor splice site), weighted by a constant W_D; (iii) a coding score L_C as as described above, weighted by a constant W_C and (iv) a constant W_E:

$$L_E(e) = W_U L_U(e) + W_C L_C(e) + W_D L_D(e) + W_E$$

Restrictions on the form of the parameters W reduces their number from 4 to 2, namely $W_U = W_D$, and $W_U + W_D + W_C = 1$. Values for the two parameters

were chosen to maximise the correlation coefficient (CC) between annotated and predicted protein-coding nucleotides in a set of training examples [84].

It is straightforward to mimic this scoring function in a GAZE configuration. The additive exon score constant W_E can by achieved by the creation of a length-penalty function for which the penalty is the same for all distances. The other parameters W_U, W_C and W_D can be accommodated as the "evidence weights" of the modified GAZE scoring function of the last chapter. The only slight difficulty is the calculation of the coding likelihood $L_E(e)$ for a candidate exon e. Figure 5.1 shows how segments computed from the hexamer log-likelihoods are incorporated in the model. I refer to the complete configuration as GAZE_GeneID.

```
<declarations>
    ⋮
    <segment id="hex_0" scoring="standard_sum" partial="FALSE"/>
    <segment id="hex_1" scoring="standard_sum" partial="FALSE"/>
    <segment id="hex_2" scoring="standard_sum" partial="FALSE"/>
    ⋮
<declarations>

<gff2gaze>
    ⋮
    <gffline feature="cod_hex" source="GENEID" strand="+" frame="0">
        <seg id="hex_0"/>
    </gffline>

    <gffline feature="cod_hex" source="GENEID" strand="+" frame="1">
        <seg id="hex_1"/>
    </gffline>

    <gffline feature="cod_hex" source="GENEID" strand="+" frame="2">
        <seg id="hex_2"/>
    </gffline>
    ⋮
<gff2gaze>
```

```
⋮
<model>
    ⋮
    <target id="stop">
        <useseg id="hex_0" phase="0" />
        <useseg id="hex_2" phase="1" />
        <useseg id="hex_1" phase="2" />
        <killfeat id="stop" phase="0"/>

        <source id="start" mindis="60" len_fun="sngl_ex_pen" phase="0">
            <output feature="CDS_term" strand="+" frame="0"/>
        </source>

        <source id="3ss_0" mindis="0" len_fun="term_ex_pen" phase="0">
            <output feature="CDS_term" strand="+" frame="0"/>
        </source>

        <source id="3ss_1" mindis="0" len_fun="term_ex_pen" phase="2">
            <output feature="CDS_term" strand="+" frame="1"/>
        </source>

        <source id="3ss_2" mindis="0" len_fun="term_ex_pen" phase="1">
            <output feature="CDS_term" strand="+" frame="2"/>
        </source>
    </target>
    ⋮
</model>
```

Figure 5.1: Fragment of a GAZE-XML configuration derived from GENEID showing how hexamer coding segments contribute towards the score. Each 6-tuple in the query sequence has three segments, one for each possible codon position that the hexamer might begin at. A segment type for each codon position is therefore defined, and the GFF "frame" attribute is used to construct segments of the corresponding types. The segment score for regions ending with for example the "stop" target feature, comprises a sum of separate scores for each segment type, and the "phase" attribute is used to ensure that only the single correct segment at each sequence position (i.e. that starting at the appropriate codon position) contributes towards the score. All reverse-strand elements of the model are omitted for clarity.

	Base	Exon			Gene		
	CC	Av	ME	WE	Av	MG	WG
GAZE_GeneID	0.818	0.642	0.179	0.133	0.104	0.04	0.09
GENEID	0.821	0.644	0.181	0.129	0.112	0.04	0.09

Table 5.2: Accuracy of GAZE_GeneID compared with GENEID on a combination of the H176 and HMR195 datasets. The accuracy measures are explained in section 1.4.1.

5.3.2 Accuracy of the model

Table 5.2 shows the accuracy of GAZE_GeneID compared with GENEID on both datasets of vertebrate sequences. Reassuringly, the results are extremely similar but it is interesting to ask why they are not identical.

There are two reasons for the discrepancy. The first reason is that GENEID does not allow for the fact that under its own model of gene structure, initial and terminal exons can be less than 6 base-pairs in length. Its calculation of a coding score for such small exons is therefore incorrect. The same small exons are possible in GAZE_GeneID, but since no hexamer-derived segment can fit completely within a region smaller than 6 base pairs in length, they are given a segment score of zero.

The second reason is that when forming a pool of candidate exons, GENEID considers only the 5 highest-scoring upstream candidate acceptor splice sites for each candidate donor. This heuristic was probably implemented for reasons of space or time efficiency but means that GENEID is not guaranteed to identify the globally highest-scoring gene structure in the sequence. This does not seem to have any impact on the accuracy however, as the tables above show. Although GENEID missed more exons than GAZE_GeneID, fewer of its exon predictions are wrong.

5.4 Optimising the parameters of the model

As explained above, the GENEID scoring function has effectively four free parameters which are chosen to maximise the correlation coefficient between annotated and

predicted coding nucleotides in a training set. To make this optimisation procedure tractable with straightforward methods, a specific relationship between the parameters is imposed, reducing the effective number of free variables in the function to 2. The GAZE training method described in the last chapter however is specifically designed for the simultaneous optimisation of a function with several free variables. In addition, both the Maximum Likelihood and Maximal Feature Discrimination estimation methods are quite different from a technique based on maximising the correlation coefficient. In this section, I investigate the effectiveness of the two GAZE training methods in obtaining values for the parameters that give rise to accurate gene predictions.

5.4.1 Defining the parameters of the model

As a starting point, I expand the effective two free parameters of the GENEID scoring function to eight in GAZE_GeneID in the following way. Firstly, the scaling factor for the scores of the defining features of a candidate exon ($W_U = W_D$) is replaced by three untied weights in GAZE_GeneID for each of translation start sites, translation stop sites, and splice sites. There are in fact 16 distinct exon-defining features in GAZE_GeneID but sensible tying reduces the number of weights first to six (for example, the scores of donor and acceptor splice sites in each of three phases are all subject to the same weight) and then to three (the model is made strand neutral by tying together the weights for the corresponding forward and reverse strand version of a feature).

Secondly, the scaling factor for the coding score for a candidate exon in GENEID (W_C) is represented by a single weight for the same quantity in GAZE_GeneID.

Thirdly and finally, the additive constant for candidate exons in GENEID (W_E) is replaced by four separate but identical length-independent penalty functions for initial, internal, terminal and single exons (i.e. single exon genes). Each function has its own weight, effectively allowing different additive constants for each exon type.

5.4.2 Accuracy of the trained model

Optimal values for these 8 parameters on the H176 training set of sequences were obtained by both the Maximum Likelihood and Maximal Feature Discrimination methods described in the last chapter. The resulting accuracy in the prediction of gene structures in both the H176 training set and the HMR195 test set are is shown in table 5.3. The table shows that the MFD method seems to perform better than the ML method, at all levels of accuracy. Indeed, at the base-pair and exon levels ML training leads to an increase in specificity and decrease in sensitivity, with little difference in average performance. Both ML and MFD training give significantly improved accuracy at the gene level however, with again MFD having the edge.

It is apparent from the results that both the ML-trained and the MFD-trained models perform slightly worse on the test set than on the training set. This could be because the test set is by chance a more challenging data set for gene prediction, either because it has more "difficult" genes (for example 5.1 shows the test set to have an observably larger mean intron length than the training set), or because the signal and content sensors used by GENEID were derived from examples that share more in common with the training set than the test set. Inspection of the gene level results however reveals GAZE_GeneID to perform better on the test set than on the training set, whilst the reverse is true for the trained versions of the model.

A natural explanation for this is a problem often encountered in machine-learning, namely that the GAZE_GeneID$^{\mathrm{ML}}$ and GAZE_GeneID$^{\mathrm{MFD}}$ have been "over-fitted" and the derived parameters represent specific aspects of the training set that do not generalise to the test set. A related problem is that observed by Henderson *et. al.* in the training of the VEIL gene prediction program [56]. They used the Baum-Welch algorithm [5] to obtain the set of parameters for their Class hidden Markov model that maximised the likelihood of a set of training sequences. Aware of a possible non-correspondence between the likelihood of the model and its accuracy of gene prediction (in terms of the classical measures), they they took a series of snap-shots of the parameter values after each iteration of their optimisation procedure,

(a)

Base level	H176 (training)			HMR195 (test)		
	Sn	Sp	CC	Sn	Sp	CC
GAZE_GeneID	0.86	0.83	0.82	0.81	0.87	0.82
GAZE_GeneID^{ML}	0.83	0.86	0.82	0.76	0.89	0.79
GAZE_GeneID^{MFD}	0.86	0.85	0.83	0.81	0.88	0.82
GENSCAN	0.97	0.86	0.90	0.95	0.86	0.89
FGENESH	0.81	0.79	0.76	0.83	0.82	0.80

(b)

Exon level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_GeneID	0.64	0.67	0.66	0.17	0.14	0.61	0.65	0.63	0.19	0.13
GAZE_GeneID^{ML}	0.63	0.70	0.66	0.21	0.13	0.56	0.68	0.62	0.28	0.14
GAZE_GeneID^{MFD}	0.69	0.70	0.69	0.17	0.16	0.64	0.69	0.66	0.20	0.14
GENSCAN	0.82	0.75	0.79	0.06	0.15	0.77	0.73	0.75	0.08	0.14
FGENESH	0.64	0.60	0.62	0.14	0.19	0.64	0.63	0.63	0.19	0.19

(c)

Gene level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_GeneID	0.09	0.08	0.08	0.03	0.08	0.13	0.12	0.12	0.04	0.10
GAZE_GeneID^{ML}	0.26	0.22	0.24	0.03	0.18	0.23	0.19	0.21	0.05	0.16
GAZE_GeneID^{MFD}	0.29	0.23	0.26	0.02	0.22	0.27	0.22	0.24	0.04	0.17
GENSCAN	0.40	0.35	0.37	0.01	0.13	0.37	0.33	0.35	0.02	0.12
FGENESH	0.30	0.25	0.28	0.04	0.15	0.32	0.29	0.31	0.04	0.13

Table 5.3: Accuracy at (a) base-pair level, (b) exon level and (c) gene level on the H176 and HMR195 datasets of GAZE_GeneID after training with the Maximum Likelihood (ML) and Maximal Feature Discrimination (MFD) methods on H176. Results for GENSCAN and FGENESH are shown for comparison. The accuracy measures are explained in section 1.4.1.

and evaluated the accuracy of prediction in the *training* sequences obtained with each (as measured by the average of exon sensitivity and specificity). The result of this analysis was that even though the likelihood of the model was increased with each iteration, after a certain number of iterations the accuracy of prediction in the training sequences started to decrease. The authors refer to this phenomenon as "over-training" [56].

The situation is similar here in that neither optimisation function is the same as gene prediction accuracy when measured in the standard ways, although of course we would hope for a good correlation. It is therefore insightful to plot how the accuracy of the model on the *test* set (for over-fitting) and *training* set (for over-training) evolves during both optimisation procedures. This is made simple by the conjugate gradient descent method described in the last chapter, whereby the end of each line minimisation provides a natural point at which to "freeze" the process and assess the accuracy.

Figures 5.2 and 5.3 show how the accuracy of prediction in the training sequences, and respectively the test sequences, varies during the conjugate gradient descent algorithm, for both the ML and MFD objective functions. The main conclusion to draw from these plots is that gene level accuracy in both the training and test sets increases steadily during the optimisation. Although at the exon level the increase in accuracy is less smooth, it is certainly not the case that there is a point during the optimisation after which further line minimisations reduce the accuracy. With these datasets and this configuration at least, neither over-fitting nor over-training seems to be a problem.

The pattern of increase of gene level accuracy during the optimisation is not dissimilar to that shown by the objective functions themselves, suggesting that it is this accuracy measure that these functions most closely reflect. At the exon level, noticeable improvement is only observed in the MFD-trained model; ML-training results in only marginal increase in exon accuracy, in both training and test sets.

Figures 5.2 and 5.3 reveal much about the way in which overall performance is

improved by the two methods. Both at the gene-level and exon level, the untrained model with parameters taken from GENEID has similar sensitivity and specificity. As the training progresses, the sensitivity and specificity diverge, producing models that are more sensitive than specific at the gene level, and more specific than sensitive at the exon level. This is true of both ML and MFD training, but the divergence is most dramatic in the exon-level accuracy of the ML-trained model. Another reason to favour the MFD-training therefore is that it seems to produce parameters which achieve a better balance of sensitivity and specificity.

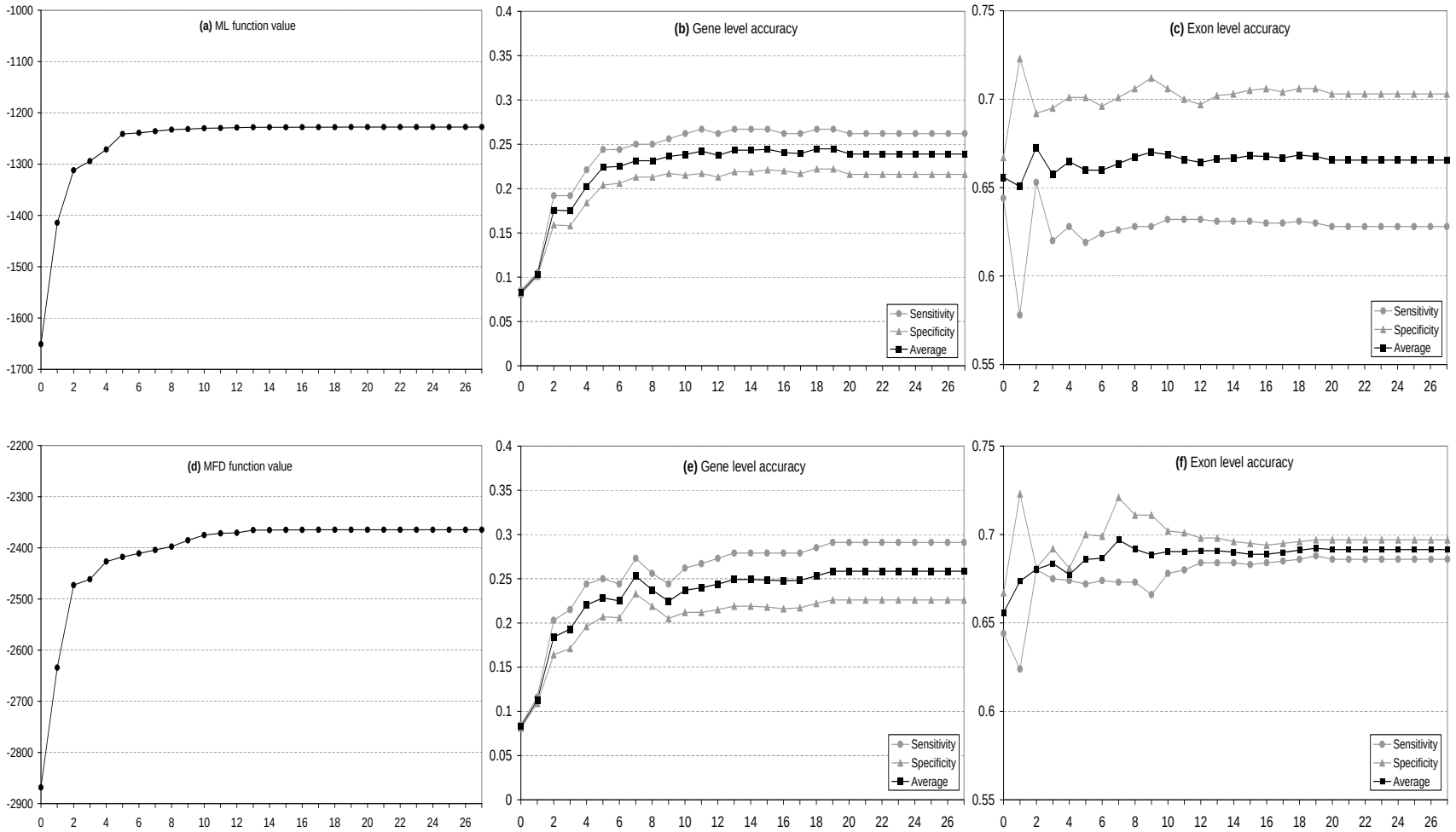


Figure 5.2: Prediction accuracy in the *training* set (H176) after each line minimisation iteration of the conjugate gradient descent algorithm. Plotted are (a) the value of the maximum likelihood function and resulting (b) gene-level and (c) exon-level accuracies. (d,e,f) Similar plots for the maximum feature discrimination training procedure.

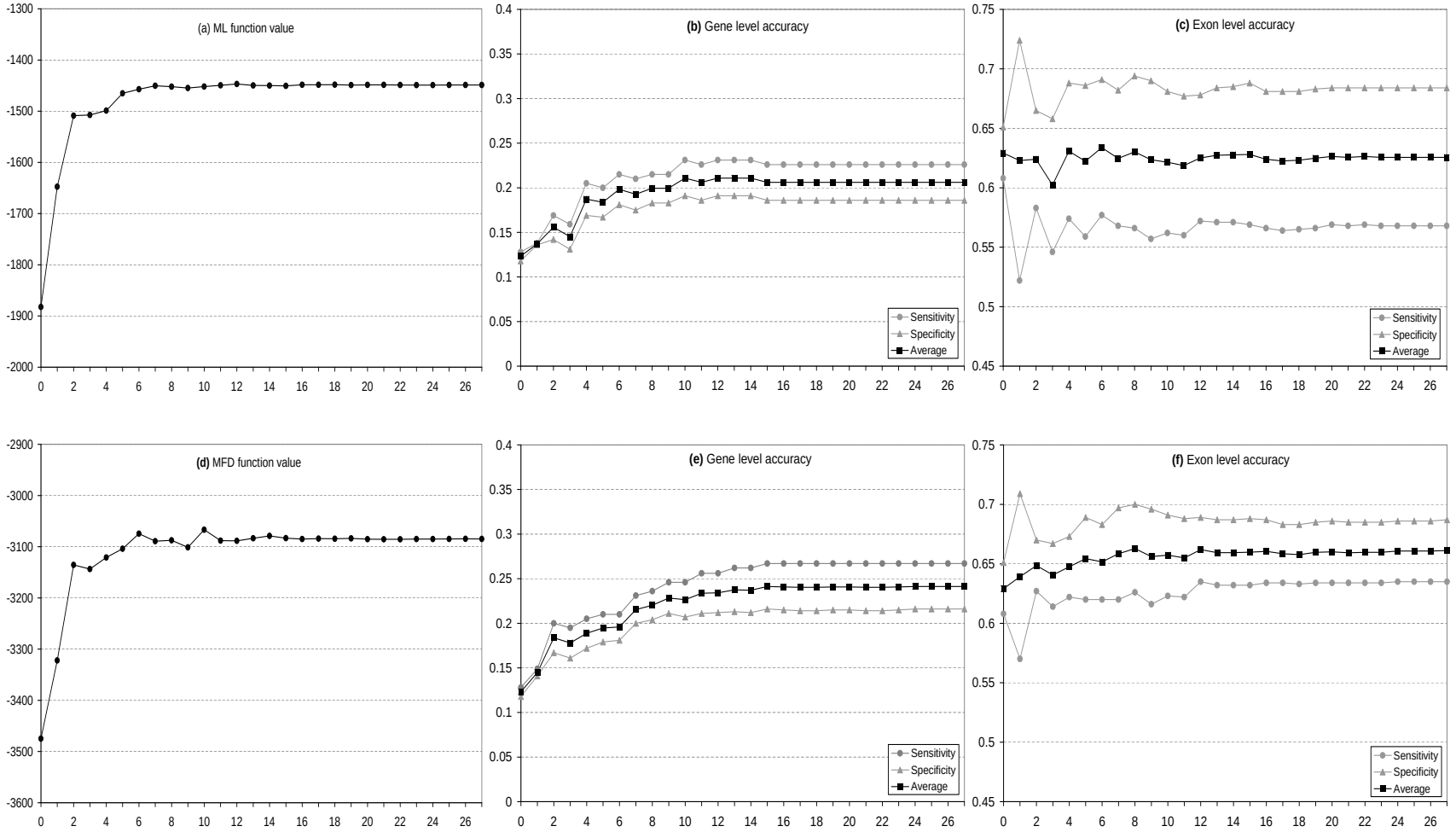


Figure 5.3: Prediction accuracy in the *test* set (HMR195) after each line minimisation iteration of the conjugate gradient descent algorithm applied to the *training* examples (H176). See legend for figure 5.2.

5.5 Investigating three ways to improve accuracy

One of the most striking aspects of the results of the previous section is the degree to which GENSCAN out-performs all GENEID-derived GAZE configurations, untrained, ML-trained and MFD-trained. Given the sophistication of the GENSCAN model of gene structure and the signal and content sensing models that it employs, this is perhaps unsurprising. In particular, the program contains at least three specific innovations that might contribute to its impressive accuracy. Firstly, its model of gene structure includes sub-models for non-protein-coding elements of the transcript, in particular the sites of transcription initiation. This effectively provides an additional source of evidence for genes. Secondly, rather than using standard statistical distributions for the lengths of initial, internal, terminal and single exons (or not having any length distribution at all) the probability of observing each exon length (given the type) was estimated directly from the training set, using a smoothing procedure to avoid over-fitting. Thirdly, to take account of variations in human gene structural properties with C+G content, the model has distinct parameters for each of four C+G% strata. When presented with a query sequence, its C+G% is computed and the appropriate set of parameters used for prediction.

In this section, I examine the effectiveness of the GAZE framework, supplemented by Maximal Feature Discrimination training, in accommodating each of these particular kinds of innovation in turn. I start by showing how promoter prediction data produced by the EPONINE_SCAN program [34] can be put to effective use by a simple refinement to the gene structure model. I then go on to examine the effectiveness with which external region-length distribution data can be incorporated, by supplementing the standard configuration with the smoothed exon-length data derived by Burge [20]. Thirdly, I show how GAZE can be used with datasets stratified by C+G content. In all three cases, the focus of the presentation is the role of the Maximal Feature Discrimination training method and its effectiveness in weighting the scores of the relevant model elements.

5.5.1 Incorporating promoter prediction data

The most well-known and widely used model for eukaryotic promoter recognition (used by GENSCAN and FGENESH for example) was defined by Bucher [19]. It consists of a pair weight matrices for the TATA-box (15 base pairs) and the downstream transcription initiation site (8 base pairs), separated by a distance of 14-20 nucleotides. Each of the distances in this range is assumed to be equally probable, and the DNA itself in the interval is generated according to a "background" intergenic model.

A source or promoter prediction data: EPONINE_SCAN

Rather than re-implementing this model for eukaryotic promoters, I have chosen to make use of a more recent development in the field of promoter detection. The EPONINE_SCAN program for transcription start site detection [34] consists of four weight matrices, representing (1) a CpG island downstream of the start site; (2) a TATAAA motif upstream of the start site (corresponding to the TATA-box mentioned above); (3) a short region of high C+G content just upstream of (2); (4) likewise a short region of high C+G content just downstream of (2). Representing the promoter signal as a series of weight matrices is not new idea, but unlike the simple Bucher model described above, each weight matrix is associated with a probability distribution describing its position relative to the transcription start site. This allows for a more accurate representation of the way in which the distances between different components of the promoter signal vary in real examples. In comparisons with other published promoter detection methods, EPONINE_SCAN performs favourably, being as sensitive as the next best method (PROMOTERINSPECTOR; [98]), but more specific.

Using EPONINE_SCAN with GAZE

EPONINE_SCAN is freely available on the world-wide web (http://www.sanger.ac.uk/-Software/analysis/eponine), and outputs transcription start site predictions with associated scores in GFF, making it ideal to be used with GAZE.

Threshold	0.999	0.99	0.5	0.1
Total predictions	2491	24887	377087	923369
Predictions per sequence	21	117	1030	2489
Seqs with no prediction	252	159	5	0

Table 5.4: Number of predictions reported by EPONINE_SCAN on the H176 and HMR195 datasets at four different score thresholds. In each case, sequences with no prediction were excluded from the calculation of the mean number of predictions per sequence.

The scores reported by the program are intended to be interpreted as the probability of the correctness of the prediction, so range between 0 and 1. Since the scores of the other features used in the system are log probability-ratios, it is likely that the best results would be obtained if the EPONINE_SCAN scores were also in this form. The reported scores x were therefore transformed into log probability-ratio form x' in the following way, inverting the logistic link function used as the output stage of EPONINE_SCAN (T. Down, pers. comm.):

$$x' = \log \frac{x}{1-x}$$

The scoring threshold recommended by the authors to give the highest average of sensitivity and specificity is 0.999. However, the sensitivity reported for this cutoff for a dataset based on the confirmed transcripts in human chromosome 22 is 0.54 [34]. For GAZE to be effective in using these predictions, it is therefore necessary to lower the threshold to increase the sensitivity. Table 5.4 shows the number of predictions made by EPONINE_SCAN on the H176 and HMR195 datasets for a variety of thresholds. Although a threshold of 0.1 leads to a very high false positive rate (only at most one prediction on each sequence can be correct), this should not be a problem for GAZE; both the scores of the predictions of themselves and the surrounding genomic context (particularly in the protein-coding part of the gene) should provide enough information to disregard most (ideally all) of the false positives.

It is interesting to note that even at the recommended threshold of 0.999, the mean number of predictions per sequence when those sequences for which there is no predictions are excluded is 21. This gives an upper bound on the specificity for these sequences (assuming one prediction on each sequence is correct) of 0.05, far lower that the figure of 0.74 previously reported [34]. The authors state that it was necessary to group predictions into clusters to achieve the results reported, with prediction within 1000 nucleotides of each other being considered a single prediction. Again, it is not be necessary to perform this step with GAZE, because the gene structure rules will ensure that only the highest scoring site in a cluster will be included in the prediction of a downstream gene.

Only a slight modification to the standard gene structure model is required to make use of the transcription start site predictions reported by EPONINE_SCAN. The resulting model architecture is a simpler version of that shown pictorially in figure 3.9 for modelling the untranslated regions at the end of *C. elegans* genes, omitting the *trans*-splice and transcription termination features. Figure 5.4 shows how the new EPONINE_SCAN-produced "transcript_start" feature slots into the standard model of gene structure. The resulting model is referred to as GAZE_GeneID$_{\text{tss}}$.

Obtaining optimal weights for EPONINE_SCAN scores

After modifying the model to incorporate the new feature, the next step is to obtain an optimal weight for the converted scores reported by EPONINE_SCAN. In addition to a parameter for the feature itself (with weights for the forward and reverse versions of the feature tied to be the same), I have defined a constant length penalty function for untranslated regions (with associated weight); the reason for this is to maintain consistency with the rest of the elements of the GAZE_GeneID configuration, the coding exons of which are also subject to a weighted, constant penalty (see earlier). There are therefore two free parameters associated with the scores reported by EPONINE_SCAN: a multiplication factor (corresponding to the weight of the "transcript_start" feature score) and an additive constant (corresponding to the

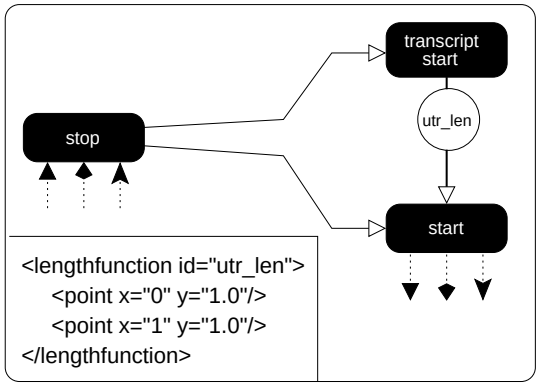


Figure 5.4: Changes to the standard GAZE_GeneID model of gene structure to allow for the possibility of a transcription start site upstream of the start of the protein-coding region. The forward-strand half of the model only is shown for clarity. A length-penalty function for the untranslated region between the transcription start and translation start ("utr_len") is defined to be an arbitrary constant (inset), which will be optimised by training.

weight of the "utr_len" length penalty function).

Considering also the parameters of the basic GAZE_GeneID configuration, the GAZE_GeneID$_{\text{tss}}$ model has 10 parameters to optimise. It is natural to think that only 2 of these 10 parameters need to be optimised because optimal values for the other 8 have already been obtained (GAZE_GeneID$^{\text{MFD}}$ in table 5.3). However it is not necessarily true that these values are optimal with respect to GAZE_GeneID$_{\text{tss}}$ model. The overall score of a gene (which effectively determines whether it appears in the output) will now have two additional components if it includes a putative transcription start. It may be therefore that the other elements of the score (for the features, segments and length penalty functions defining the coding part of the gene) have to be down-weighted to compensate.

Table 5.5 shows the result of optimising the parameters of GAZE_GeneID$_{\text{tss}}$ on the H176 training set, for both the training set itself and the HMR176 test set. In this case, it is interesting to note that only the MFD method is applicable, because it was not known in advance which (if any) of the EPONINE_SCAN-predicted transcription start sites were correct; the ML method requires every candidate feature to be

(a)

Base level	H176 (training)			HMR195 (test)		
	Sn	Sp	CC	Sn	Sp	CC
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-2}$	0.87	0.89	0.86	0.81	0.90	0.83
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-10}$	0.88	0.89	0.86	0.82	0.91	0.84
GAZE_GeneID$^{\text{MFD}}$	0.86	0.85	0.83	0.81	0.88	0.82

(b)

Exon level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-2}$	0.71	0.73	0.72	0.15	0.13	0.64	0.71	0.67	0.21	0.12
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-10}$	0.72	0.73	0.73	0.15	0.13	0.65	0.71	0.68	0.20	0.12
GAZE_GeneID$^{\text{MFD}}$	0.69	0.70	0.69	0.17	0.16	0.64	0.69	0.66	0.20	0.14

(c)

Gene level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-2}$	0.34	0.27	0.30	0.02	0.19	0.28	0.23	0.25	0.05	0.15
GAZE_GeneID$_{\text{tss}}^{\text{MFD}-10}$	0.32	0.27	0.30	0.02	0.18	0.27	0.23	0.25	0.06	0.15
GAZE_GeneID$^{\text{MFD}}$	0.29	0.23	0.26	0.02	0.22	0.27	0.22	0.24	0.04	0.17

Table 5.5: The resulting accuracy of the GAZE model incorporating EPONINE_SCAN predictions (GAZE_GeneID$_{\text{tss}}$) trained using the Maximal Feature Discrimination method in two different ways as explained in the text. The results of using the same method to train the standard model (GAZE_GeneID), described earlier, are repeated here for comparison. (a) base-level accuracy; (b) exon-level accuracy; (c) gene-level accuracy. The accuracy measures are explained in section 1.4.1.

declared "correct" or "incorrect" whilst the MFD method requires this only for a subset of feature types (see previous chapter).

The model was trained in two ways: firstly with the optimal values obtained for GAZE_GeneID$^{\mathrm{MFD}}$ fixed and only the two parameters associated with the new feature subject to optimisation (GAZE_GeneID$^{\mathrm{MFD}-2}_{\mathrm{tss}}$) and secondly with all 10 parameters optimised simultaneously (GAZE_GeneID$^{\mathrm{MFD}-10}_{\mathrm{tss}}$). The table shows that there is little (if any) significant difference between the accuracies of the models resulting from the two optimisations. This can be explained by the fact that the difference between the two optimisations in terms of the final value of the objective function and resulting weights for the model elements was marginal.

Given this result, it is tempting therefore to assume that when introducing a new feature type into a gene structure model, only the weights for the scores of the new features (and related length penalty functions) need to be re-estimated. This would be of practical value to the system as a whole, due to the fact that models with smaller numbers of free parameters can be optimised far more quickly with the conjugate gradient descent algorithm. However, there is no obvious reason why the assumption should hold in general.

We would expect promoter information to provide more accurate identification of the 5' end of the gene, particularly the translation start site. Looking at the results more carefully, this seems only marginally to be the case; the GAZE_GeneID$^{\mathrm{MFD}-2}_{\mathrm{tss}}$ model identified 109 of the translation starts in the HMR195 data set, compared with 105 for the GAZE_GeneID$^{\mathrm{MFD}}$ model. When looking at the results for the H176 (training) set however, it is evident that a degree of over-fitting has occurred, the increase being from 90 to 111 in the training set.

With only a marginal increase in accuracy in the test-set, it is natural to ask whether the transcription start site predictions are being used at all in prediction. The optimal values for the weights of the feature scores and the associated 5' UTR length-independent penalty function (with constant value 1.0) obtained by the MFD method were 5.1 and 24.7 respectively. This means that only EPONINE_SCAN predic-

tions with a converted score before weighting of above 4.8 will contribute positively towards the prediction. For future applications, predictions with score less than this can be filtered out before running GAZE without affecting the highest scoring gene structure.

As well as improving prediction accuracy, the inclusion of promoter prediction information in the model also gives it the ability to predict 5' untranslated regions in the genome (although real UTRs are defined in the processed mRNA). It is difficult to measure the accuracy of these UTR predictions without knowledge of the correct site of transcription initiation for the test sequences. However, some insight can be gained by isolating those predicted genes for which the *translation* start site has been identified correctly. The assumption is that the UTRs implied by the subset of these gene structures that include an EPONINE_SCAN-predicted site of transcription initiation carry a higher than otherwise likelihood of being correct. For the HMR195 (test) dataset: of the 235 genes predicted by the GAZE_GeneID$^{\mathrm{MFD}-2}_{\mathrm{tss}}$ model, 109 include a correctly-identified translation start site, 75 include an EPONINE_SCAN-predicted site of transcription initiation (and therefore UTR), and 43 contain both.

5.5.2 Using exon length distributions

The length penalty component of the GAZE scoring function has been used in a very simplistic way for the models presented so far in this chapter. Although penalty functions for each of initial, internal, terminal and single exons have been defined and weighted independently, the functions themselves are defined to be constant, i.e. the same penalty is applied to exons of a particular type, whatever their length. It is certainly not the case however that exon lengths are uniformly distributed in the genomes of eukaryotes. A plot of the lengths of internal exons (for example) in the human genome [112] revealed a mean length of 145 bp (median 122), with the majority falling between 50 bp and 300 bp and a sharp peak at just over 100 bp. Another analysis has shown this distribution to be close to log-normal [121]. It would seem therefore that there is value in more accurately reflecting the likely

lengths of exons in the penalty functions employed in GAZE_GeneID for exonic regions (i.e. by associating unlikely lengths with high penalties).

A source of exon-length data: GENSCAN

Many gene prediction methods (particularly those based around standard hidden Markov models) by their nature assume that the likelihood of a region of a particular type (e.g. exon) decays exponentially with length. Burge showed that this geometric model was an inaccurate representation for the lengths of each of the four different types of coding exon [21]. The Generalised Hidden Markov model framework employed by GENSCAN naturally accommodates length probability distributions based upon direct observation rather than an assumed (e.g. geometric) model (see chapter 1), but it was still non-trivial to obtain these distributions in the first place.

The main problem in obtaining realistic probabilities for the lengths of each of the four types of exons was small size of the training set (238, 1016, 238, 142 for each of initial, internal, terminal and single exons). This gave rise to spikey distributions for which the probability of many lengths was zero simply because they were not observed in the training set. A smoothing procedure was therefore employed, based on a simple underlying model of exon evolution [20].

Using the GENSCAN exon length distributions with GAZE

The GENSCAN parameter file for gene-finding in human sequences includes a probability for each length (up to a maximum) for each of initial, internal, terminal and single exons. The first task is to convert the given exon length probabilities into a form that can be presented to GAZE as a length penalty function. Taking the negative logarithm of each probability has the desired effect, converting low probabilities to high penalties[2].

[2]The base of the logarithm is unimportant, as this will effectively be determined by training a weight for the function.

In addition, a small change to the configuration of GAZE_GeneID is required, swapping out the constant length penalty functions for exonic regions and swapping in the new ones. No other changes to the model are required, and I refer to the revised configuration as GAZE_GeneID$_{\text{exo}}$.

Obtaining optimal weights for the exon length penalties

In optimising the parameters for GAZE_GeneID earlier, separate weighting factors were attached to the constant length-penalty functions for each of the four exon types. The same is done here, with the added qualification that what is meant by weighting a non-constant length penalty function is that the penalties for all lengths are subject to this same scaling factor. The assumption here therefore is that the penalty functions derived from the given probability distributions are of the correct *shape*, but need to be weighted appropriately in relation to the other elements involved in the scoring function (see previous chapter).

Table 5.6 shows the result of optimising the parameters of the GAZE_GeneID$_{\text{exo}}$ model on the training set. The model has the same 8 parameters as the standard configuration, but as with the GAZE_GeneID$_{\text{tss}}$ model, a choice must be made whether to re-estimate all 8 parameters from scratch, or to optimise only the 4 parameters for the exon length penalties, anchoring the others to their values obtained by the earlier optimisation (GAZE_GeneID$^{\text{MFD}}$). As before then, the model is trained using the MFD method in both ways, and the table shows the results gained to be almost indistinguishable. Again though, there is no justifiable reason for why this might be true in general.

Examining the relative difference in exon-level prediction accuracy between trained versions of the standard and explicit-exon-length configurations, it is evident that the degree by which the latter out-performs the former is more marginal in the test set than in the training set. This implies a degree of over-fitting has taken occurred, supported by the base-level results. Nevertheless, improvements on the test set at all levels are however still apparent.

(a)

Base level	H176 (training)			HMR195 (test)		
	Sn	Sp	CC	Sn	Sp	CC
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-4}$	0.86	0.88	0.85	0.81	0.90	0.83
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-10}$	0.86	0.88	0.85	0.80	0.90	0.82
GAZE_GeneID$^{\text{MFD}}$	0.86	0.85	0.83	0.81	0.88	0.82

(b)

Exon level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-4}$	0.73	0.74	0.73	0.15	0.14	0.67	0.72	0.69	0.19	0.13
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-8}$	0.73	0.74	0.73	0.15	0.14	0.66	0.71	0.69	0.20	0.13
GAZE_GeneID$^{\text{MFD}}$	0.69	0.70	0.69	0.17	0.16	0.64	0.69	0.66	0.20	0.14

(c)

Gene level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-4}$	0.34	0.28	0.31	0.03	0.20	0.29	0.24	0.27	0.05	0.16
GAZE_GeneID$_{\text{exo}}^{\text{MFD}-8}$	0.34	0.28	0.31	0.03	0.19	0.28	0.23	0.26	0.05	0.16
GAZE_GeneID$^{\text{MFD}}$	0.29	0.23	0.26	0.02	0.22	0.27	0.22	0.24	0.04	0.17

Table 5.6: The resulting accuracy of the GAZE model incorporating GENSCAN-derived exon length penalties (GAZE_GeneID$_{\text{exo}}$) trained using the Maximal Feature Discrimination method in two different ways as explained in the text. The results of using the same method to train the standard model (GAZE_GeneID), described earlier, are repeated here for comparison. (a) base-level accuracy; (b) exon-level accuracy; (c) gene-level accuracy. The accuracy measures are explained in section 1.4.1.

The ability to define arbitrary length penalty functions for any region type is perhaps the main advantages that GAZE has over other similar systems. It is therefore unfortunate that the treatment of penalty functions by the training procedure might be considered to be its weakest aspect, being reliant upon a pre-defined function with the correct shape. Ideally, the method would estimate an optimal shape for each penalty function directly from the training sequence. A possible way to address this would involve associating several weights to a single function, each being applied to a range of lengths (e.g. 1-50, 51-100 etc.), or at the extreme having a separate weight for each specific length.

An alternative approach would be to retain the binding of a single length function to a single weight, but to allow several functions to be associated to a particular $src \rightarrow tgt$ rule. This would require a simple extension to the scoring function, whereby the length-penalty component would be calculated as a sum of penalties taken from each penalty function specified in the rule. This approach is strictly more general than that above; a distinct weight for each distance can be represented by having a separate function for each distance which takes a positive number for the distance of interest and zero for all others[3]. Furthermore, it would allow the definition of composite functions, although it is not clear how such an approach would impact the search-space pruning strategy described in chapter 2, which assumes that it is possible to identify a point at which the penalty function for a rule becomes monotonically increasing (see section 2.6.3).

The main problem with both of these methods however is the large increase in the number of free variables in the optimisation. This will impact both the time taken to reach the function maximum, but also more importantly the number of training sequences that are required to obtain sensible estimates for this large number of parameters.

[3]although the definition of thousands or more penalty functions in a single $src \rightarrow tgt$ rule in the XML configuration file would be tedious at the very least!

5.5.3 Introducing C+G%-dependent model parameters

The natural way to address this problem is to divide the training sequences into pools (or *strata*) according to their C+G content, and then perform a separate training run for each, arriving at a distinct set of parameter values for each C+G% stratum. An early example of this approach is GENEPARSER [102] whereby separate neural networks were trained on each of three training sets containing respectively sequences with "low", "medium" and "high" C+G content. The boundaries for the categories were determined by calculating the mean C+G content of all full-length genes in Genbank, and classifying entries in the training set with C+G% more than one standard deviation away from this mean as "low" or "high" as appropriate.

By way of contrast, the training set of GENSCAN was initially divided into three according to the L1, L2, H1, H2 and H3 categories defined by Bernardi [7] and others: L1+L2 (less than 43% C+G), H1+H2 (43-51% C+G) and H3 (more than 51% C+G). Since the H3 group turned out to be excessively populated compared with the others, it was itself split into H3-1 (51-57% C+G) and H3-2 (more than 57% C+G). The parameters for the GHMM (e.g. the mean and variance of the geometric distributions for introns and intergenic regions, and certain transition and emission probabilities) were then derived separately from the sequences in each of the four datasets. The emission distribution for coding regions on the other hand was defined to have two parameterisations only, for low C+G% (L1+L2), and high (H1+H2 + H3-1 + H3-2).

The GENEID model for scoring coding regions differently according to C+G%

For all of the models so far, GENEID has been used as a source for the majority of the features and segments. It is fortunate therefore that GENEID is supplied with an additional parameter file for human gene finding that contains separate parameters for each of three C+G% strata: 0-45%, 45-55%, and 55-100% C+G. On inspection it was evident that the majority of parameters are the same between the

	I (0-45%)	II (45-55%)	III (55-100%)
H176	37	78	61
HMR195	38	107	50

Table 5.7: Breakdown of the training and test sequence sets by C+G content

three strata; only the parameters for the hexamer-based model for scoring coding regions display differences between strata. Just as the coding model defined in the original parameter file was used to make GAZE coding segments (see section 5.2.3), this time the C+G content of each sequence in the training and test sets was computed and coding segments were made by using the parameters corresponding to the appropriate C+G% stratum.

Table 5.7 shows the result of partitioning the training and test datasets according to the C+G% groupings of the three coding model used by GENEID. Interestingly, it shows the "medium" C+G% band to the most highly populated. Although apparently contrary to the GENSCAN findings outlined above, the difference can be explained by the higher boundary here between "medium" and "high" and strata.

Using GAZE with C+G%-stratified datasets

Unlike many other programs, there is no functionality in GAZE itself to account for varying parameters in sequences with different C+G content. However, one of the design aims of GAZE was to be sufficiently flexibility to allow different signal and content models, and even different models of gene structure, to be used in different situations. As a result of this design, it turns out to be straightforward to use GAZE with C+G%-specific parameters, and there are variety of ways in which this can be done.

The most natural approach is to compute the C+G content of the sequences in advance, and then construct GFF files of features and segments accordingly. This was done as explained above in obtaining C+G%-dependent coding segments for the

sequences. Then GAZE can be used exactly as before, with the same configuration file used for all the sequences. This method highlights one of the advantages of GAZE over many other existing systems in that it allows for variation across C+G% strata of not only the *parameters* of a set assumed underlying sequence-generating models for signal and content, but of the models themselves.

Another approach would be to pre-compute features and segments for *all* C+G% strata in advance and assess the C+G content at run-time. A simple Perl wrapper is then run in place of GAZE, which (i) computes the C+G content of the input sequence, and (ii) invokes GAZE with the appropriate GFF files. The method is more general in that it allows for the use of different GAZE configurations for the C+G% strata, allowing (among other things) model-element weightings, length penalty functions, and even gene structure models themselves that vary with C+G content.

I have used GAZE with the C+G%-stratified data in two ways. The first method was to use the default GAZE_GeneID configuration (my GAZE re-implementation of GENEID) on the H176 and H195 datasets, using for each sequence the set of coding segments that are appropriate for its C+G content. The results for this experiment are referred to as GAZE_GeneID$_{\mathrm{GC}}$ in table 5.8. An improvement in performance over the default model (i.e. that using the coding segments obtained using a global, C+G%-independent parameterisation of the GENEID coding model), at all levels of accuracy, is evident, most strikingly the correlation co-efficient.

Obtaining optimal weights for the C+G%-dependent models

The second way I have used GAZE with the C+G% stratified data is to use MFD training to optimise three specific sets of values for the model-element weights, one for each of the subsets of the sequences in the H176 (training) set having C+G content falling respectively into the ranges 0-45%, 45-55% and 55-100%. The result is three GAZE configurations, and the Perl wrapper mentioned above was used to obtain the results referred to in table 5.8 as GAZE_GeneID$_{\mathrm{GC}}^{\mathrm{MFD}}$.

(a)

Base level	H176 (training)			HMR195 (test)		
	Sn	Sp	CC	Sn	Sp	CC
GAZE_GeneID$_{\mathrm{GC}}$	0.92	0.86	0.88	0.92	0.87	0.87
GAZE_GeneID$^{\mathrm{MFD}}_{\mathrm{GC}}$	0.91	0.86	0.87	0.91	0.88	0.88
GAZE_GeneID	0.86	0.83	0.82	0.81	0.87	0.82
GAZE_GeneID$^{\mathrm{MFD}}$	0.86	0.85	0.83	0.81	0.88	0.82

(b)

Exon level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_GeneID$_{\mathrm{GC}}$	0.68	0.69	0.69	0.13	0.13	0.69	0.70	0.69	0.14	0.13
GAZE_GeneID$^{\mathrm{MFD}}_{\mathrm{GC}}$	0.72	0.73	0.73	0.14	0.13	0.70	0.74	0.72	0.17	0.13
GAZE_GeneID	0.64	0.67	0.66	0.17	0.14	0.61	0.65	0.63	0.19	0.13
GAZE_GeneID$^{\mathrm{MFD}}$	0.69	0.70	0.69	0.17	0.16	0.64	0.69	0.66	0.20	0.14

(c)

Gene level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_GeneID$_{\mathrm{GC}}$	0.11	0.11	0.11	0.01	0.09	0.16	0.14	0.15	0.01	0.09
GAZE_GeneID$^{\mathrm{MFD}}_{\mathrm{GC}}$	0.32	0.26	0.29	0.01	0.19	0.27	0.22	0.24	0.01	0.16
GAZE_GeneID	0.09	0.08	0.08	0.03	0.08	0.13	0.12	0.12	0.04	0.10
GAZE_GeneID$^{\mathrm{MFD}}$	0.29	0.23	0.26	0.02	0.22	0.27	0.22	0.24	0.04	0.17

Table 5.8: Accuracy of GAZE_GeneID using three C+G content dependent models for scoring coding regions; (a) base-level accuracy; (b) exon-level accuracy; (c) gene-level accuracy.

The same high correlation co-efficient is observed for the trained system as was seen in the untrained C+G%-stratified system. Marked improvements over the untrained system however are observed at the exon-level and the gene level, although these improvements are less striking when compared against the MFD-trained default, C+G%-independent model. These results suggest that the relatively high accuracy of the GAZE_GeneID^{MFD} system at all three levels is achieved at the base-pair level by the use of C+G%-stratified coding segments, and at the exon and gene levels by the MFD training of the model element weights.

The higher accuracy of gene prediction programs that take the C+G content of the underlying sequence into account has previously been explained by the observation that gene structural properties, such as average intron length and codon usage, vary according to this property (see chapter 1). However, the different sets of values for the weights obtained by training on sequences from each of the three C+G% strata are not representative of these structural differences, because the model itself is the same in each case (with the exception of the coding segments, explained above). Instead, different weight values for different C+G% strata implies that the relative importance of the model components varies according to C+G content. Although there is no biological reason to explain this result, it is consistent with previous observations that the difficulty with which localised signals for gene features can be detected also varies with C+G content. Burge has shown that the accuracy of discrimination between localised coding regions and non-coding regions is positively correlated with C+G content, and proposes this as a possible reason for the poor performance of gene prediction programs on A+T-rich sequences [20]. Levine on the other hand constructed a model for splice site detection and showed that the accuracy of discrimination between true and pseudo splice sites is poorest in sequences with high C+G content [74]. Based on these two observations, we might expect firstly the value for the splice site score weight obtained in the low C+G% optimisation to be higher than that obtained in the high C+G% optimisation, and vice-versa for the weights for the coding segments. This is indeed the case, with the

(a)

Base level	H176 (training)			HMR195 (test)		
	Sn	Sp	CC	Sn	Sp	CC
GAZE_GeneID$^{\text{MFD}}_{\text{all}}$	0.91	0.89	0.88	0.88	0.90	0.87
GENSCAN	0.97	0.86	0.90	0.95	0.86	0.89

(b)

Exon level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_GeneID$^{\text{MFD}}_{\text{all}}$	0.76	0.76	0.76	0.13	0.13	0.70	0.74	0.72	0.17	0.12
GENSCAN	0.82	0.75	0.79	0.06	0.15	0.77	0.73	0.75	0.08	0.14

(c)

Gene level	H176 (training)					HMR195 (test)				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_GeneID$^{\text{MFD}}_{\text{all}}$	0.37	0.31	0.34	0.02	0.18	0.33	0.27	0.30	0.02	0.17
GENSCAN	0.40	0.35	0.37	0.01	0.13	0.37	0.33	0.35	0.02	0.12

Table 5.9: Accuracy of GAZE_GeneID$_{\text{all}}$ (which includes transcription start sites, exon length penalties and C+G% dependency) when trained with MFD. The results for GENSCAN are repeated for ease of comparison. The accuracy measures are explained in section 1.4.1.

splice site weights for the low and high C+G% strata being 0.96 and 0.92, and the coding segment weights for these strata being 0.54 and 0.61.

5.5.4 Combining all three types of evidence

By way of postscript, it is interesting to consider the effect of including the innovations of the past three sections in one system, training the weights using Maximal Feature Discrimination. The trained system is referred to as GAZE_GeneID$^{\text{MFD}}_{\text{all}}$ in table 5.9.

Although the accuracy of the model as a whole is the best achieved by any GAZE model so far, it is still marginally out-performed by GENSCAN. One reason

for this could be the relative sophistication of the signal models used for donor and acceptor splice sites compared with those employed by GENEID. The splice acceptor model used by GENSCAN includes the branch-point region between 21 and 38 nucleotides upstream of the conserved AG. This specific region is modelled with a "windowed" weight array, allowing the capture of second-order dependencies. The GENEID splice acceptor model on the other hand does not extend upstream as far as the branch-point (see section 5.2.3). In addition, GENSCAN uses Maximal Dependence Decomposition for the modelling of the donor splice signal, allowing the capture of long-range dependencies, whereas GENEID uses a simple weight matrix (see section 5.2.3 and [20]).

It has been shown that these sophisticated models can be more accurate than simpler models (such as those used by GENEID) when judged by an "isolated" test of the discrimination between true and pseudo splice sites [20]. A natural extension to the work described here would involve implementing these models in a program that outputs scored splice site predictions in GFF. GAZE could then be used to investigate their accuracy when judged in an "integrated" test of their influence on the prediction of complete gene structures.

Chapter 6

Conclusions

In the previous chapters, I have proposed a semi-formal framework for the integration of gene component evidence from arbitrary sources into complete predictions of gene structure. I have shown that this concept is valid by implementing it in a program called GAZE, and applying it to the prediction of gene structures in worm and vertebrate genomic sequences. I envision GAZE to be useful both as a research tool for investigating signal and content recognition methods, and as the final stage in a genome annotation pipeline, drawing together the results of previous localised analyses.

One of the key aspects of the system is that it does not rely upon a fixed underlying model of gene structure. This allowed for example the modelling of *trans*-spliced genes in *C.elegans* with little effort (see chapter 3). GAZE therefore provides a platform for the modelling of structural properties that are not currently accounted for in existing programs. One example would be nested genes, i.e. genes situated in the introns of other genes. The constructs currently offered by GAZE do allow the definition of a model that allows a gene to occur in the intron of another gene. However, it is necessary to define an identical gene model for each of the six different introns that can occur (for each phase, on each strand). Extension of the system to allow sub-models to be defined once and referred to elsewhere in the configuration would therefore be a natural avenue for future work.

As well as providing an integrated tool for genome annotation and research, there are ideas in this thesis that might be applicable to to other integrated gene prediction approaches. One of these is the idea of *dominance* in the context of gene prediction, introduced in chapter 2. This was used as the basis for a search-space pruning strategy that allows GAZE to run in time that grows effectively linearly with the input sequence length. The primary advantage of the technique is that it is applicable under a wider range of conditions than is typically the case with other pruning methods. The linear run-time growth of GENEID for example is offset by the fact that it neither allows arbitrary length penalty functions nor reports posterior probabilities. GENSCAN on the other hand does compute posterior probabilities, but pseudo-linear run-time is achieved by restricting the use of arbitrary length probability distributions to alternating (in practice protein-coding) states. The GAZE pruning strategy is robust under arbitrary (although eventually monotonically increasing) length penalties for all types of gene region. It would therefore be interesting to see whether the improvements in accuracy resulting from explicit modelling of exon lengths (see [20] and chapter 5) are matched by explicit length modelling of introns and intergenic regions.

The other substantial new idea described in this thesis is the Maximal Feature Discrimination method for determining optimal weights for the scores of the different types of evidence employed in an integrated gene prediction system. This was shown to have better correspondence with the standard gene prediction accuracy assessment metrics than the classical maximum likelihood method. Another advantage that it offers is the ability to parameterise sophisticated models of gene structure when the location of certain features, for example the site of transcription initiation, are not known for the training sequences. Although presented in the context of GAZE, MFD can be applied to other probabilistic gene prediction systems. For Hidden Markov Models for example, the approach can be viewed as the maximisation of the posterior probabilities of the correct state transitions at specific positions, although this is only possible if such transitions can be unambigu-

ously defined. Because the posterior probabilities can be computed by the standard forward-backward algorithm, the method is likely to be directly applicable in many situations.

Given the current trend of systems that make use of genomic sequences from more than one organism, it is natural consider the applicability of GAZE to comparative gene prediction. One simple application would be to use TBLASTX matches of a genome of interest to a variety of other genomes to support coding regions. This is the approach essentially taken by an extension of GENEID called SGP-2 [83]. It has the apparent advantage that it is not limited to hits from a single genome. However, by treating matches to different genomes equally, we ignore the fact that they will share different levels of background genomic conservation to the target genome. This means that a relatively poorly scoring match to a distantly related organism can be more meaningful than a high scoring match to a closely related one. By supplementing genomic similarity with a phylogenetic tree, we can start to discriminate between those conserved regions that are functional, and those that can be explained by evolutionary distance alone. In this example, we could therefore adjust the scores of the conserved regions according to phylogeny, up-weighting matches to distantly related genomes and down-weighting matches to closely related ones. This suggests a less principled strategy that is immediately applicable, which involves attaching a separate weight to the matches to each genome and using MFD to optimise their values.

The true power of GAZE in the use of comparative information in gene prediction comes from its clean separation of feature detection and gene prediction. Multiple alignments of corresponding regions of several genomes can be used as the basis for models of the evolution of splice sites and coding regions (for example). Supplementing standard signal recognition methods (such as those described in chapter 1) with such models will improve their accuracy. In this sense, GAZE can therefore be viewed as an efficient and convenient way for sophisticated evolutionary models to be employed in the prediction of complete gene structures.

To end, it is interesting to consider the future of research into computational gene prediction. The recent appearance of comparative and similarity-based methods, and the accompanying lack of new *ab initio* single-sequence methods, is suggestive of a feeling that there is a ceiling to what is achievable with the latter, and that we are pretty close to it with current methods. However, the fact that a living cell is able to determine the precise gene structure of its genome with high fidelity, without the use of external sequences, suggests that continued research into single-sequence techniques is not in vain.

Bibliography

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *The Molecular Biology of the Cell*. Garland Publishing, New York, NY, 1989.

[2] 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 Res.*, 25:3389–3402, 1997.

[3] V. Bafna and D. Huson. The conserved exon method for gene finding. In Brunak et al. [18], pages 3–12.

[4] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Res.*, 10:950–958, 2000.

[5] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[6] R. Bellman. *Dynnamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[7] G. Bernardi. The isochore structure of the human genome. *Annual Review of Genetics*, 23:637–661, 1989.

[8] A. Bird. CpG islands as gene markers in the vertebrate nucleus. *Trends in Genetics*, 3:342–347, 1987.

[9] E. Birney and R. Durbin. Dynamite: a flexible code generating language for dynamic programming methods used in sequence comparison. In Gaasterland et al. [45], pages 56–64.

[10] E. Birney and R. Durbin. Using Genewise in the Drosophila annotation experiment. *Genome Res.*, 10:547–548, 2000.

[11] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, UK, 1995.

[12] T. Blumenthal, D. Evans, C. D. Link, A. Guffanti, D. Lawson, J. Thierry-Mieg, D. Thierry-Mieg, W. L. Chui, K. Duke, M. Kirali, and S. K. Kim. A global analysis of Caenohabditis elegans operons. *Nature*, 417:851–853, 2002.

[13] T. Blumenthal and K. Steward. RNA processing and gene structure. In D. L. Riddle, T. Blumenthal, B. J. Meyer, and J. R. Priess, editors, *C.ELEGANS II*, chapter 6. Cold Spring Harbor Laboratory Press, New York, NY, 1997.

[14] T. Blumenthal, O. White, and C. Fields. The C. elegans cleavage and polyadenylation signal. *The Worm Breeder's Gazette*, 13:62–63, 1993.

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

[16] R. P. Brent. An algorithm with guranteed convergence for finding the minimum of a function of one variable. In *Algorithms for function minimization without derivatives*, chapter 5, pages 61–80. Prentice Hall, Englewood Cliffs, NJ, 1973.

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

[18] S. Brunak, F. Galison, M. Gribskov, A. Krogh, A. G. Pederson, P. Rouze, G. Stormo, and A. Tramontano, editors. *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology*, Oxford, UK, 2001. Oxford University Press.

[19] P. Bucher. Weight matrix descriptions of four eukaryotic RNA polymerase elements derived from 502 unrelated promoter seqeunces. *J. Mol. Biol.*, 212:563–578, 1990.

[20] C. Burge. *The identification of genes in human genomic DNA*. PhD thesis, Stanford University, 1997.

[21] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268:78–94, 1997.

[22] C. Burge and S. Karlin. Finding the genes in genomic DNA. *Current Opinion in Structural Biology*, 8:346–354, 1998.

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

[24] The chromosome 21 mapping and sequencing consortium. The DNA sequence of human chromosome 21. *Nature*, 405:311–319, 2000.

[25] J. S. Chuang and D. Roth. Gene recognition based on DAG shortest paths. In Brunak et al. [18], pages 56–64.

[26] J.-M. Claverie. Computational methods for the identification of genes in vertebrate genomic sequences. *Hum. Mol. Gen.*, 6:1735–1744, 1997.

[27] R. Conrad, J. Thomas, J. Speith, and T. Blumenthal. Insertion of part of an intron into the 5' untranslated region of a Caenorhabditis elegans gene converts it into a trans-spliced gene. *Molecular Cell Biology*, 11:1921–1926, 1991.

[28] Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420:520–562, 2002.

[29] R. Davuluri, I. Grosse, and M. Zhang. Computational identification of promoters and first exons in the human genome. *Nature Genet.*, 29:412–417, 2001.

[30] P. Deloukas, L.H. Matthews, J. Ashurst, J. Burton, J.G. Gilbert, M. Jones, G. Stavrides, J.P. Almeida, A.K. Babbage, C.L. Bagguley, J. Bailey, K.F. Barlow, K.N. Bates, L.M. Beard, D.M. Beare, O.P. Beasley, C.P. Bird, S.E. Blakey, A.M. Bridgeman, A.J. Brown, et al. The DNA sequence and comparative analysis of human chromosome 20. *Nature*, 414:865–871, 2001.

[31] A. P Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–38, 1977.

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

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

[34] T. Down and T. Hubbard. Computational detection and location of transcription start sites in mammalian genomic DNA. *Genome Res.*, 12:458–461, 2002.

[35] I. Dunham, N. Shimizu, B.A. Roe, S. Chissoe, A.R. Hunt, J.E. Collins, R. Bruskiewich, D.M. Beare, M. Clamp, L.J. Smink, R. Ainscough, J.P. Almeida, A. Babbage, C. Bagguley, J. Bailey, K. Barlow, K.N. Bates, O. Beasley, C.P. Bird, S. Blakey, et al. The DNA sequence of human chromosome 22. *Nature*, 402:489–495, 1999.

[36] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK, 1998.

[37] L. Duret, D. Mourchiroud, and C. Gautier. Statistical analysis of vertebrate sequences reveals that long genes are scare in GC-rich isochores. *Journal of Molecular Evolution*, 40:308–317, 1995.

[38] S. R. Eddy. Non-coding RNA genes and the modern RNA world. *Nat. Rev. Genet.*, 2:919–929, 2001.

[39] J. W. Fickett. The gene identification problem - an overview for developers. *Computers and Chemistry*, 20:103–118, 1996.

[40] J. W. Fickett and A. G. Hatzigeorgiou. Eukaryotic promoter recognition. *Genome Res.*, 7:861–878, 1997.

[41] J. W. Fickett and C. S. Tung. Assessment of protein coding measures. *Nucleic Acids Res.*, 20:6441–6450, 1992.

[42] C. A. Fields and C. A. Soderlund. gm: a practical tool for automating DNA sequence analysis. *Comput. Applic. Biosci.*, 6:263–270, 1990.

[43] R. Fletcher and C. Reeves. Function minimisation by conjugate gradients. *Computing Journal*, pages 149–154, 1964.

[44] L. Florea, G. Hartzell, Z Zhang, G. M. Rubin, and W. Miller. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Res.*, 8:967–974, 1998.

[45] 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.

[46] M. Gelfand, L. Podolski, T. Astakhova, and A. Roytberg. Recognition of genes in human DNA sequences. *Journal of Computational Biology*, 3:223–234, 1996.

[47] M. S. Gelfand. Computer prediciton of the exon-intron structure of mammalian pre-mRNAs. *Nucleic Acids Res.*, 18:5865–5869, 1990.

[48] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93:9061–9066, 1996.

[49] M. S. Gelfand and M. A. Roytberg. Prediction of the exon-intron structure by a dynamic programming approach. *Biosystems*, 3:173–182, 1993.

[50] D. E. Goldberg. *Genetic algorithms in search, optimisation and machine learning*. Addison Wesley, Boston, MA, 1989.

[51] R. Guigo. Computational gene prediction - an open problem. *Computers and Chemistry*, 21:215–222, 1997.

[52] R. Guigo. Assembling genes from predicted exons in linear time with dynamic programming. *Journal of Computational Biology*, 5:681–702, 1998.

[53] R. Guigo, P. Agarwal, J. F. Abril, M. Burset, and J. W. Fickett. An assessment of gene prediction accuracy in large DNA sequences. *Genome Res.*, 10:1631–1642, 2000.

[54] R. Guigo and J.W. Fickett. Distinctive sequence features in protein coding, genic non-coding and intergenic human DNA. *J. Mol. Biol.*, 253:51–60, 1995.

[55] A. Hatzigeorgiou. Translation initiation start prediction in human cDNAs with high accuracy. *Bioinformatics*, 18:343–50, 2002.

[56] J. Henderson, S. Salzberg, and K. H. Fasman. Finding genes in DNA with a hidden Markov model. *Journal of Computational Biology*, 4(2):127–141, 1997.

[57] S. Henikoff, M. A. Keene, K. Fechtel, and J. W. Fristrom. Gene within a gene: nested Drosophila genes encode unrelated proteins on oppostie DNA strands. *Cell*, 44:33–42, 1986.

[58] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[59] K. L. Howe, T. Chothia, and R. Durbin. GAZE: a generic framework for the integration of gene-prediction data by dynamic programming. *Genome Res.*, 12:1418–1427, 2002.

[60] T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, T. Down, R. Durbin, E. Eyras, J. Gilbert, M. Hammond, L. Huminiecki, A. Kasprzyk, H. Lehvaslaiho, P. Lijnzaad, C. Melsopp, E. Mongin, R. Pettett, M. Pocock, S. Potter, A. Rust, E. Schmidt, S. Searle, G. Slater, J. Smith, W. Spooner, A. Stabenau, J. Stalker, E. Stupka, A. Ureta-Vidal, I. Vastrik, and Clamp M. The Ensembl genome database project. *Nucleic Acids Res.*, 30:38–41, 2002.

[61] W. J. Kent. BLAT - the BLAST-like alignment tool. *Genome Res.*, 12:656–664, 2002.

[62] W. J. Kent and A. M. Zahler. Conservation, regulation, synteny, and introns in a large-scale C.briggsae-C.elegans genomic alignment. *Genome Res.*, 10:1115–1125, 2000.

[63] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[64] I. Korf, P. Flicek, D. Duan, and M. R. Brent. Integrating genomic homology into gene structure prediction. In T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia, editors, *Proceedings of the Eigth International Conference on Intelligent Systems for Molecular Biology*, pages 140–148, Menlo Park, CA, 2002. AAAI Press.

[65] M. Kozak. Initiation of translation in prokaryotes and eukaryotes. *Gene*, 234:187–208, 1999.

[66] M. Krause and D. Hirsh. A trans-spliced leader sequence on actin mRNA in C.elegans. *Cell*, 49:753–761, 1987.

[67] 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.

[68] A. Krogh. Two methods for improving performance of a HMM and their application for gene finding. In Gaasterland et al. [45], pages 179–186.

[69] A. Krogh. Using database matches with HMMGene for automated gene detection on Drosophila. *Genome Res.*, 10:523–528, 2000.

[70] A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in E. coli DNA. *Nucleic Acids Res.*, 22:4768–4778, 1994.

[71] D. Kulp, D. Haussler, M. Reese, and F. H. Eeckman. Integrating database homology in a probabilistic gene structure model. In R. Altman, A. Dunker, and T. Klein L. Hunter, editors, *Proceedings of Pacific Symposium on Biocomputing*, pages 232–244, 1997.

[72] 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. [106], pages 134–142.

[73] A. N. Ladd and T. A. Cooper. Finding signals that regulate alternative splicing in the post-genomic era. *Genome Biol.*, 3:reviews0008, 2002.

[74] A. Levine. Bioinformatics approaches to RNA splicing. Master's thesis, The Sanger Centre, 2001.

[75] T. M. Lowe and S. R. Eddy. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res.*, 25:955–964, 1997.

[76] A. V. Lukashin and M. Borodovsky. GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Res.*, 26:1107–1115, 1998.

[77] C. Mathe, M. Sagot, T. Schiex, and P. Rouze. Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Res.*, 30:4103–4117, 2002.

[78] I. Meyer and R. Durbin. Comparative ab initio gene prediction of gene structures using pair HMMs. *Bioinformatics*, 18:1309–1318, 2002.

[79] R. Mott. EST_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA. *Comput. Applic. Biosci.*, pages 477–478, 1997.

[80] E. W. Myers and W. Miller. Optimal alignments in linear space. *Comput. Applic. Biosci.*, 4(1):11–17, 1988.

[81] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computing Journal*, 7:308–313, 1965.

[82] L. Pachter, M. Alexandersson, and S. Cawley. Applications of generalized pair hidden markov models to alignment and gene finding problems. *Journal of Computational Biology*, 9:389–400, 2002.

[83] G. Parra, P. Agarwal, J.F. Abril, T. Wiehe, J.W. Fickett, and R. Guigo. Comparative gene prediction in human and mouse. *Genome Res.*, 13:108–117, 2003.

[84] G. Parra, E. Blanco, and R. Guigo. GeneID in Drosophila. *Genome Res.*, 10:511–515, 2000.

[85] N. Pavy, S. Rombauts, P. Dehais, C. Mathe, D. V. V.Ramana, P. Leroy, and P. Rouze. Evaluation of gene prediction software using a genomic data set: application to Arabidopsis thaliana sequences. *Bioinformatics*, 15:887–899, 1999.

[86] M. Pertea, X. Lin, and S. Salzberg. GeneSplicer: a new computational model for splice site prediction. *Nucleic Acids Res.*, pages 1185–1190, 2001.

[87] E. Polak and G. Ribiere. Note sur la convergence de methodes de directions conjuguees. *Revue Francaise d'informatique et de recherche operationelle*, 16:35–43, 1969. In French.

[88] W. H. Press, S. A. Teukolsky, W. Vetterling T., and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1992.

[89] N. Drake R. Guigo, S. Knudsen and T. Smith. Prediction of gene structure. *J. Mol. Biol.*, 226:141–157, 1992.

[90] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[91] M. G. Reese, G. Hartzell, N. L. Harris, U. Ohler, J. F. Abril, and S. E. Lewis. Genome annotation assessment in Drosophila melanogaster. *Genome Res.*, 10:483–501, 2000.

[92] M. G. Reese, D. Kulp, H. Tammana, and D. Haussler. Genie : gene-finding in Drosophila melanogaster. *Genome Res.*, 10:529–538, 2000.

[93] E. Rivas and S. R. Eddy. Noncoding RNA gene detection using comparative sequence analysis. *BMC Bioinformatics*, 2:8, 2001.

[94] S. Rogic, A. Mackworth, and F. Oullette. Evaluation of gene-finding programs on mammalian sequences. *Genome Res.*, 11:817–832, 2001.

[95] A. Roytberg, T. Astakhova, and M. Gelfand. Combinatorial approaches to gene recognition. *Computers and Chemistry*, 21:229–235, 1997.

[96] A. A. Salamov and V. V. Solovyev. Ab initio gene finding in Drosophila genomic DNA. *Genome Res.*, 10:516–522, 2000.

[97] S. L. Salzberg, M. Pertea, A. L. Delcher, M. J. Gardner, and H. Tettelin. Interpolated markov models for eukaryotic gene finding. *Genomics*, 59(1):24–31., Jul 1 1999.

[98] M. Scherf, A. Klingenhoff, and T. Werner. Highly specific localization of promoter regions in large genomic sequences by PromoterInspector. *J. Mol. Biol.*, 297:599–606, 2000.

[99] T. Schiex, A. Moisan, and P. Rouze. EuGene: a eukaryotic gene finder that combines several sources of evidence. In O. Gascuel and M. F. Sagot, editors, *Lecture Notes in Computer Science*, volume 2006, pages 111–125. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 2001.

[100] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[101] E. E. Snyder and G. D. Stormo. Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucleic Acids Res.*, 21:607–613, 1993.

[102] E. E. Snyder and G. D. Stormo. Identification of protein coding regions in genomic DNA. *J. Mol. Biol.*, 248:1–18, 1995.

[103] V. V. Solovyev, A. A. Salamov, and C. B. Lawrence. Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Res.*, 22:5156–5163, 1994.

[104] V. V. Solovyev, A. A. Salamov, and C. B. Lawrence. Identification of human gene structure using linear discriminant functions and dynamic programming. In 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*, pages 367–375, Menlo Park, CA, 1995. AAAI Press.

[105] R. Staden. Methods to define and locate patterns of motifs in sequences. *Comput. Applic. Biosci.*, 4(1):53–60, 1988.

[106] 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.

[107] L. Stein, P. Sternberg, R. Durbin, J. Thierry-Mieg, and J. Spieth. WormBase: network access to the genome and biology of Caenorhabditis elegans. *Nucleic Acids Res.*, 29:82–86, 2001.

[108] G. D. Stormo. Gene-finding approaches for eukaryotes. *Genome Res.*, 10:394–397, 2000.

[109] G. D. Stormo and D. Haussler. Optimally parsing a sequence into different classes based on multiple types of evidence. In States et al. [106], pages 369–375.

[110] J. Tabaska, R. Davuluri, and M. Zhang. Identifying the 3'-terminal exon in human DNA. *Bioinformatics*, 17:602–607, 2001.

[111] J. Tabaska and M. Zhang. Detection of polyadenylation signals in human DNA sequences. *Gene*, 231:77–86, 1999.

[112] The International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[113] E. C. Uberbacher and R. J. Mural. Locating protein-coding regions in human DNA sequences by a multiple sensor-neural network approach. *Proc. Natl. Acad. Sci. USA*, 88:11261–11265, 1991.

[114] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, pages 260–269, 1967.

[115] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner. *Molecular Biology of the Gene*. Benjamin/Cummings, Menlo Park, CA, 1987.

[116] T. Wu. A segment-based dynamic programming algorithm for predicting gene structure. *Journal of Computational Biology*, 3:375–394, 1996.

[117] Y. Xu, J. R. Einstein, M. Shah, and E. C. Uberbacher. An improved system for exon recognition and gene modelling in human DNA sequences. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 376–383, Menlo Park, CA, 1994. AAAI Press.

[118] Y. Xu, R. J. Mural, and E. C. Uberbacher. Constructing gene models from accurately predicted exons: an application of dynamic programming. *Comput. Applic. Biosci.*, 10:613–623, 1994.

[119] R. Yeh, L. Lim, and C. Burge. Computational inference of homologous gene structures in the human genome. *Genome Res.*, 11:803–816, 2001.

[120] M. Zhang. Identification of protein coding regions in the human genome by quadratic discriminant analysis. *Proc. Natl. Acad. Sci. USA*, 94:565–568, 1997.

[121] M. Zhang. Statistical analysis of human exons and their flanking regions. *Hum. Mol. Gen.*, 7:919–932, 1998.

[122] M. Zhang. Computational prediction of eukaryotic protein-coding genes. *Nat. Rev. Genet.*, 3:698–709, 2002.

Appendix A

Some example GAZE configurations

A.1 GAZE_std

```
<?xml version="1.0" encoding="US-ASCII"?>
<gaze>

<declarations>
   <feature id="5ss_0" st_off="1" en_off="1"/>
   <feature id="5ss_1" st_off="1" en_off="1"/>
   <feature id="5ss_2" st_off="1" en_off="1"/>
   <feature id="stop" st_off="3" en_off="0"/>
   <feature id="3ss_0" st_off="1" en_off="1"/>
   <feature id="3ss_1" st_off="1" en_off="1"/>
   <feature id="3ss_2" st_off="1" en_off="1"/>
   <feature id="start" st_off="0" en_off="3"/>
   <feature id="start_rev" st_off="3" en_off="0"/>
   <feature id="3ss_0_rev" st_off="1" en_off="1"/>
   <feature id="3ss_1_rev" st_off="1" en_off="1"/>
   <feature id="3ss_2_rev" st_off="1" en_off="1"/>
   <feature id="stop_rev" st_off="0" en_off="3"/>
   <feature id="5ss_0_rev" st_off="1" en_off="1"/>
   <feature id="5ss_1_rev" st_off="1" en_off="1"/>
   <feature id="5ss_2_rev" st_off="1" en_off="1"/>

   <segment id="coding_seg" scoring="standard_max"/>
   <segment id="coding_seg_rev" scoring="standard_max"/>

   <lengthfunction id="intron_pen"/>
   <lengthfunction id="intergene_pen"/>
   <lengthfunction id="init_ex_pen"/>
   <lengthfunction id="term_ex_pen"/>
   <lengthfunction id="int_ex_pen"/>
   <lengthfunction id="sngl_ex_pen"/>
</declarations>

<gff2gaze>
```

```
   <!-- Features -->

   <gffline feature="atg" source="Genefinder" strand="+">
      <feat id="start"/>
   </gffline>

   <gffline feature="atg" source="Genefinder" strand="-">
      <feat id="start_rev"/>
   </gffline>

   <gffline feature="stop" source="Genefinder" strand="+">
      <feat id="stop"/>
   </gffline>

   <gffline feature="stop" source="Genefinder" strand="-">
      <feat id="stop_rev"/>
   </gffline>

   <gffline feature="splice5" source="Genefinder" strand="+">
      <feat id="5ss_0"/>
      <feat id="5ss_1"/>
      <feat id="5ss_2"/>
   </gffline>

   <gffline feature="splice5" source="Genefinder" strand="-">
      <feat id="5ss_0_rev"/>
      <feat id="5ss_1_rev"/>
      <feat id="5ss_2_rev"/>
   </gffline>

   <gffline feature="splice3" source="Genefinder" strand="+">
      <feat id="3ss_0"/>
      <feat id="3ss_1"/>
      <feat id="3ss_2"/>
   </gffline>

   <gffline feature="splice3" source="Genefinder" strand="-">
      <feat id="3ss_0_rev"/>
      <feat id="3ss_1_rev"/>
      <feat id="3ss_2_rev"/>
   </gffline>

   <!-- Segments -->

   <gffline feature="coding_seg" source="Genefinder" strand="+">
      <seg id="coding_seg"/>
   </gffline>

   <gffline feature="coding_seg" source="Genefinder" strand="-">
      <seg id="coding_seg_rev"/>
   </gffline>
</gff2gaze>


<dna2gaze>
   <dnafeat pattern="atg">
      <feat id="start" score="0.0"/>
   </dnafeat>

   <dnafeat pattern="cat">
      <feat id="start_rev" score="0.0"/>
   </dnafeat>

   <dnafeat pattern="taa">
      <feat id="stop" score="-100.0"/>
```

```
   </dnafeat>
   <dnafeat pattern="tag">
      <feat id="stop" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="tga">
      <feat id="stop" score="-100.0"/>
   </dnafeat>

   <dnafeat pattern="tta">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="cta">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="tca">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>

   <takedna id="5ss_1" st_off="0" en_off="1"/>
   <takedna id="3ss_1" st_off="1" en_off="-1"/>
   <takedna id="5ss_2" st_off="-1" en_off="1"/>
   <takedna id="3ss_2" st_off="1" en_off="0"/>
   <takedna id="5ss_1_rev" st_off="1" en_off="0"/>
   <takedna id="3ss_1_rev" st_off="-1" en_off="1"/>
   <takedna id="5ss_2_rev" st_off="1" en_off="-1"/>
   <takedna id="3ss_2_rev" st_off="0" en_off="1"/>
</dna2gaze>

<model>
   <target id="END">
      <source id="BEGIN">
         <output feature="no genes"/>
      </source>

      <source id="start">
         <useseg id="coding_seg" source_phase="0"/>
         <killfeat id="stop" source_phase="0"/>
         <output feature="CDS_end_not_found" strand="+" frame="0"/>
      </source>

      <source id="stop">
         <output feature="intergenic"/>
      </source>

      <source id="start_rev" mindis="0">
         <output feature="intergenic"/>
      </source>

      <source id="stop_rev">
         <useseg id="coding_seg_rev" source_phase="0"/>
         <killfeat id="stop_rev" source_phase="0"/>
         <output feature="CDS_start_not_found" strand="-"/>
      </source>

      <source id="3ss_0">
         <useseg id="coding_seg" source_phase="0"/>
         <killfeat id="stop" source_phase="0"/>
         <output feature="CDS_end_not_found" strand="+" frame="0"/>
      </source>

      <source id="3ss_1">
         <useseg id="coding_seg" source_phase="2"/>
         <killfeat id="stop" source_phase="2"/>
         <output feature="CDS_end_not_found" strand="+" frame="1"/>
      </source>
```

```
   <source id="3ss_2">
      <useseg id="coding_seg" source_phase="1"/>
      <killfeat id="stop" source_phase="1"/>
      <output feature="CDS_end_not_found" strand="+" frame="2"/>
   </source>

   <source id="5ss_0" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="+"/>
   </source>

   <source id="5ss_1" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="+"/>
   </source>

   <source id="5ss_2" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="+"/>
   </source>

   <source id="3ss_0_rev" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="-"/>
   </source>

   <source id="3ss_1_rev" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="-"/>
   </source>

   <source id="3ss_2_rev" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="-"/>
   </source>

   <source id="5ss_0_rev">
      <useseg id="coding_seg_rev" source_phase="0"/>
      <killfeat id="stop_rev" source_phase="0"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>

   <source id="5ss_1_rev">
      <useseg id="coding_seg_rev" source_phase="1"/>
      <killfeat id="stop_rev" source_phase="1"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>

   <source id="5ss_2_rev">
      <useseg id="coding_seg_rev" source_phase="2"/>
      <killfeat id="stop_rev" source_phase="2"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>
</target>

<target id="start">
   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="start_rev">
```

```
   <useseg id="coding_seg_rev" target_phase="0"/>
   <killfeat id="stop_rev" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="0"/>
   </source>

   <source id="stop_rev" mindis="6" len_fun="sngl_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_0_rev" mindis="3" len_fun="init_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="3" len_fun="init_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="3" len_fun="init_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="stop">
   <useseg id="coding_seg" target_phase="0"/>
   <killfeat id="stop" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="6" len_fun="sngl_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="3" len_fun="term_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="3" len_fun="term_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="3" len_fun="term_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="stop_rev">
   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="5ss_0">
   <useseg id="coding_seg" target_phase="0"/>
```

```
   <killfeat id="stop" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3" len_fun="init_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_0_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="-"/>
   </source>

   <source id="3ss_0_rev" mindis="39" len_fun="intron_pen">
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="5ss_1">
   <useseg id="coding_seg" target_phase="1"/>
   <killfeat id="stop" target_phase="1"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_1_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="-"/>
   </source>

   <source id="3ss_1_rev" mindis="39" len_fun="intron_pen">
      <killdna source_dna="ct" target_dna="a"/>
```

```
      <killdna source_dna="tc" target_dna="a"/>
      <killdna source_dna="tt" target_dna="a"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="5ss_2">
   <useseg id="coding_seg" target_phase="2"/>
   <killfeat id="stop" target_phase="2"/>

   <source id ="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_2_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <output feature="intron_end_not_found" strand="-"/>
   </source>
   <source id="3ss_2_rev" mindis="39" len_fun="intron_pen">
      <killdna source_dna="c" target_dna="ta"/>
      <killdna source_dna="t" target_dna="ca"/>
      <killdna source_dna="t" target_dna="ta"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="3ss_0">
   <source id="BEGIN" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_0" mindis="39" len_fun="intron_pen">
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_0_rev">
   <useseg id="coding_seg_rev" target_phase="0"/>
   <killfeat id="stop_rev" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="0"/>
   </source>

   <source id="stop_rev" mindis="3" len_fun="term_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="0">
```

```
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="3ss_1">
   <source id="BEGIN" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_1" mindis="39" len_fun="intron_pen">
      <killdna source_dna="t" target_dna="aa"/>
      <killdna source_dna="t" target_dna="ag"/>
      <killdna source_dna="t" target_dna="ga"/>
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_1_rev">
   <useseg id="coding_seg_rev" target_phase="2"/>
   <killfeat id="stop_rev" target_phase="2"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="1"/>
   </source>

   <source id="stop_rev" mindis="3"  len_fun="term_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="1"/>
   </source>
</target>

<target id="3ss_2">
   <source id="BEGIN" len_fun="intron_pen">
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_2" mindis="39" len_fun="intron_pen">
      <killdna source_dna="ta" target_dna="a"/>
      <killdna source_dna="ta" target_dna="g"/>
      <killdna source_dna="tg" target_dna="a"/>
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_2_rev">
   <useseg id="coding_seg_rev" target_phase="1"/>
   <killfeat id="stop_rev" target_phase="1"/>
```

```
      <source id ="BEGIN">
         <output feature="CDS_end_not_found" strand="-" frame="2"/>
      </source>

      <source id="stop_rev" mindis="3" len_fun="term_ex_pen" phase="1">
         <output feature="CDS" strand="-" frame="2"/>
      </source>

      <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="1">
         <output feature="CDS" strand="-" frame="2"/>
      </source>

      <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="2">
         <output feature="CDS" strand="-" frame="2"/>
      </source>

      <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="0">
         <output feature="CDS" strand="-" frame="2"/>
      </source>
   </target>
</model>

<lengthfunctions>
   <lengthfunc id="intron_pen" file="/tables/intron_penalty.klh"/>
   <lengthfunc id="init_ex_pen" file="/tables/exon_penalty.initial.klh"/>
   <lengthfunc id="term_ex_pen" file="/tables/exon_penalty.terminal.klh"/>
   <lengthfunc id="int_ex_pen" file="/tables/exon_penalty.internal.klh"/>

   <lengthfunc id="sngl_ex_pen">
      <point x="0" y="4"/>
      <point x="1" y="4"/>
   </lengthfunc>

   <lengthfunc id="intergene_pen">
     <point x="0" y="4"/>
     <point x="1" y="4"/>
   </lengthfunc>
</lengthfunctions>

</gaze>
```

A.2 GAZE_EST

```
<?xml version="1.0" encoding="US-ASCII"?>
<gaze>

<declarations>
   <feature id="trans_start" st_off="0" en_off="1"/>
   <feature id="5ss_0" st_off="1" en_off="1"/>
   <feature id="5ss_1" st_off="1" en_off="1"/>
   <feature id="5ss_2" st_off="1" en_off="1"/>
   <feature id="stop" st_off="3" en_off="0"/>
   <feature id="t_splice" st_off="1" en_off="1"/>
   <feature id="3ss_0" st_off="1" en_off="1"/>
   <feature id="3ss_1" st_off="1" en_off="1"/>
   <feature id="3ss_2" st_off="1" en_off="1"/>
   <feature id="start" st_off="0" en_off="3"/>
   <feature id="trans_stop" st_off="1" en_off="0"/>
   <feature id="trans_stop_rev" st_off="0" en_off="1"/>
   <feature id="start_rev" st_off="3" en_off="0"/>
   <feature id="3ss_0_rev" st_off="1" en_off="1"/>
   <feature id="3ss_1_rev" st_off="1" en_off="1"/>
   <feature id="3ss_2_rev" st_off="1" en_off="1"/>
```

```
   <feature id="t_splice_rev" st_off="1" en_off="1"/>
   <feature id="stop_rev" st_off="0" en_off="3"/>
   <feature id="5ss_0_rev" st_off="1" en_off="1"/>
   <feature id="5ss_1_rev" st_off="1" en_off="1"/>
   <feature id="5ss_2_rev" st_off="1" en_off="1"/>
   <feature id="trans_start_rev" st_off="1" en_off="0"/>

   <segment id="coding_seg" scoring="standard_max"/>
   <segment id="est_seg" mul="0.01"/>
   <segment id="est_intron" mul="0.05"/>
   <segment id="coding_seg_rev" scoring="standard_max"/>
   <segment id="est_seg_rev" mul="0.01"/>
   <segment id="est_intron_rev" mul="0.05"/>
   <segment id="est_span" mul="-1000.0"/>

   <lengthfunction id="intron_pen"/>
   <lengthfunction id="intergene_pen"/>
   <lengthfunction id="tsplice_pen"/>
   <lengthfunction id="init_ex_pen"/>
   <lengthfunction id="term_ex_pen"/>
   <lengthfunction id="int_ex_pen"/>
   <lengthfunction id="sngl_ex_pen"/>
</declarations>

<gff2gaze>
   <!-- Features -->

   <gffline feature="atg" source="Genefinder" strand="+">
      <feat id="start"/>
   </gffline>

   <gffline feature="atg" source="Genefinder" strand="-">
      <feat id="start_rev"/>
   </gffline>

   <gffline feature="stop" source="Genefinder" strand="+">
      <feat id="stop"/>
   </gffline>

   <gffline feature="stop" source="Genefinder" strand="-">
      <feat id="stop_rev"/>
   </gffline>

   <gffline feature="splice5" source="Genefinder" strand="+">
      <feat id="5ss_0"/>
      <feat id="5ss_1"/>
      <feat id="5ss_2"/>
   </gffline>

   <gffline feature="splice5" source="Genefinder" strand="-">
      <feat id="5ss_0_rev"/>
      <feat id="5ss_1_rev"/>
      <feat id="5ss_2_rev"/>
   </gffline>

   <gffline feature="splice3" source="Genefinder" strand="+">
      <feat id="3ss_0"/>
      <feat id="3ss_1"/>
      <feat id="3ss_2"/>
      <feat id="t_splice"/>
   </gffline>

   <gffline feature="splice3" source="Genefinder" strand="-">
      <feat id="3ss_0_rev"/>
      <feat id="3ss_1_rev"/>
```

```
   <feat id="3ss_2_rev"/>
   <feat id="t_splice_rev"/>
</gffline>

<!-- The following features will be added by the pre-processor to the GFF
     file, using the est_span objects. But perhaps there should be more
     expressiveness in the GFF2GAZE section to allow for the creation of
     features corresponding to the ends of the region given by the GFF line -->

<gffline feature="transcript_start" source="EST_derived" strand="+">
    <feat id="trans_start"/>
</gffline>

<gffline feature="transcript_start" source="BLAT_mRNA_BEST_derived" strand="+">
    <feat id="trans_start"/>
</gffline>

<gffline feature="transcript_start" source="EST_derived" strand="-">
    <feat id="trans_start_rev"/>
</gffline>

<gffline feature="transcript_start" source="BLAT_mRNA_BEST_derived" strand="-">
    <feat id="trans_start_rev"/>
</gffline>

<gffline feature="transcript_stop" source="EST_derived" strand="+">
    <feat id="trans_stop"/>
</gffline>

<gffline feature="transcript_stop" source="BLAT_mRNA_BEST_derived" strand="+">
    <feat id="trans_stop"/>
</gffline>

<gffline feature="transcript_stop" source="EST_derived" strand="-">
    <feat id="trans_stop_rev"/>
</gffline>

<gffline feature="transcript_stop" source="BLAT_mRNA_BEST_derived" strand="-">
    <feat id="trans_stop_rev"/>
</gffline>

<!-- Segments -->

<gffline feature="coding_seg" source="Genefinder" strand="+">
   <seg id="coding_seg"/>
</gffline>

<gffline feature="coding_seg" source="Genefinder" strand="-">
   <seg id="coding_seg_rev"/>
</gffline>

<!-- The following segments are derived by the pre-processor -->

<gffline feature="similarity" source="EST_GENOME_strand" strand="+">
   <seg id="est_seg"/>
</gffline>

<gffline feature="similarity" source="EST_GENOME_strand" strand="-">
   <seg id="est_seg_rev"/>
</gffline>

<gffline feature="intron" source="EST_derived" strand="+">
   <seg id="est_intron"/>
</gffline>
```

```
   <gffline feature="intron" source="EST_derived" strand="-">
      <seg id="est_intron_rev"/>
   </gffline>

   <gffline feature="EST_span">
      <seg id="est_span"/>
   </gffline>
</gff2gaze>

<dna2gaze>
   <dnafeat pattern="atg">
      <feat id="start" score="0.0"/>
   </dnafeat>

   <dnafeat pattern="cat">
      <feat id="start_rev" score="0.0"/>
   </dnafeat>

   <dnafeat pattern="taa">
      <feat id="stop" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="tag">
      <feat id="stop" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="tga">
      <feat id="stop" score="-100.0"/>
   </dnafeat>

   <dnafeat pattern="tta">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="cta">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>
   <dnafeat pattern="tca">
      <feat id="stop_rev" score="-100.0"/>
   </dnafeat>

   <takedna id="5ss_1" st_off="0" en_off="1"/>
   <takedna id="3ss_1" st_off="1" en_off="-1"/>
   <takedna id="5ss_2" st_off="-1" en_off="1"/>
   <takedna id="3ss_2" st_off="1" en_off="0"/>
   <takedna id="5ss_1_rev" st_off="1" en_off="0"/>
   <takedna id="3ss_1_rev" st_off="-1" en_off="1"/>
   <takedna id="5ss_2_rev" st_off="1" en_off="-1"/>
   <takedna id="3ss_2_rev" st_off="0" en_off="1"/>
</dna2gaze>

<model>
   <target id="END">
      <source id="BEGIN">
         <output feature="no genes"/>
      </source>

      <source id="start">
         <useseg id="coding_seg" source_phase="0"/>
         <useseg id="est_seg"/>
         <killfeat id="stop" source_phase="0"/>
         <output feature="CDS_end_not_found" strand="+" frame="0"/>
      </source>

      <source id="stop">
         <useseg id="est_span"/>
         <output feature="intergenic"/>
      </source>
```

```
<source id="start_rev" mindis="0">
   <useseg id="est_span"/>
   <output feature="intergenic"/>
</source>

<source id="stop_rev">
   <useseg id="coding_seg_rev" source_phase="0"/>
   <useseg id="est_seg_rev"/>
   <killfeat id="stop_rev" source_phase="0"/>
   <output feature="CDS_start_not_found" strand="-"/>
</source>

<source id="3ss_0">
   <useseg id="coding_seg" source_phase="0"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" source_phase="0"/>
   <output feature="CDS_end_not_found" strand="+" frame="0"/>
</source>

<source id="3ss_1">
   <useseg id="coding_seg" source_phase="2"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" source_phase="2"/>
   <output feature="CDS_end_not_found" strand="+" frame="1"/>
</source>

<source id="3ss_2">
   <useseg id="coding_seg" source_phase="1"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" source_phase="1"/>
   <output feature="CDS_end_not_found" strand="+" frame="2"/>
</source>

<source id="5ss_0" len_fun="intron_pen">
   <useseg id="est_intron" exact="source"/>
   <output feature="intron_end_not_found" strand="+"/>
</source>

<source id="5ss_1" len_fun="intron_pen">
   <useseg id="est_intron" exact="source"/>
   <output feature="intron_end_not_found" strand="+"/>
</source>

<source id="5ss_2" len_fun="intron_pen">
   <useseg id="est_intron" exact="source"/>
   <output feature="intron_end_not_found" strand="+"/>
</source>

<source id="3ss_0_rev" len_fun="intron_pen">
   <useseg id="est_intron_rev" exact="source"/>
   <output feature="intron_start_not_found" strand="-"/>
</source>

<source id="3ss_1_rev" len_fun="intron_pen">
   <useseg id="est_intron_rev" exact="source"/>
   <output feature="intron_start_not_found" strand="-"/>
</source>

<source id="3ss_2_rev" len_fun="intron_pen">
   <useseg id="est_intron_rev" exact="source"/>
   <output feature="intron_start_not_found" strand="-"/>
</source>

<source id="5ss_0_rev">
```

```
      <useseg id="coding_seg_rev" source_phase="0"/>
      <useseg id="est_seg_rev"/>
      <killfeat id="stop_rev" source_phase="0"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>

   <source id="5ss_1_rev">
      <useseg id="coding_seg_rev" source_phase="1"/>
      <useseg id="est_seg_rev"/>
      <killfeat id="stop_rev" source_phase="1"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>

   <source id="5ss_2_rev">
      <useseg id="coding_seg_rev" source_phase="2"/>
      <useseg id="est_seg_rev"/>
      <killfeat id="stop_rev" source_phase="2"/>
      <output feature="CDS_start_not_found" strand="-"/>
   </source>

   <source id="trans_start">
      <useseg id="est_seg"/>
      <output feature="UTR5_end_not_found" strand="+"/>
   </source>

   <source id="trans_stop">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="trans_stop_rev">
      <useseg id="est_seg_rev"/>
      <output feature="UTR3_start_not_found" strand="-"/>
   </source>

   <source id="trans_start_rev">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="t_splice" mindis="0" maxdis="50" len_fun="tsplice_pen">
      <useseg id="est_seg"/>
      <output feature="trans_splice_UTR5_end_not_found" strand="+"/>
   </source>

   <source id="t_splice_rev">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>
</target>

<target id="start">
   <source id="BEGIN">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="t_splice" mindis="0" maxdis="50" len_fun="tsplice_pen">
      <useseg id="est_seg"/>
      <output feature="trans_splice_UTR5" strand="+"/>
   </source>

   <source id="trans_start" mindis="0">
      <useseg id="est_seg"/>
      <output feature="UTR5" strand="+"/>
```

```
   </source>

   <source id="trans_stop" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="trans_start_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="t_splice_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>
</target>

<target id="start_rev">
   <useseg id="coding_seg_rev" target_phase="0"/>
   <useseg id="est_seg_rev"/>
   <killfeat id="stop_rev" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="0"/>
   </source>

   <source id="stop_rev" mindis="6" len_fun="sngl_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_0_rev" mindis="3" len_fun="init_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="3" len_fun="init_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="3" len_fun="init_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="stop">
   <useseg id="coding_seg" target_phase="0"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="6" len_fun="sngl_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
```

```
   </source>

   <source id="3ss_0" mindis="3" len_fun="term_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="3" len_fun="term_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="3" len_fun="term_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="stop_rev">
   <source id="BEGIN">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="trans_stop_rev" mindis="0">
      <useseg id="est_seg_rev"/>
      <output feature="UTR3" strand="-"/>
   </source>

   <source id="trans_start_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="t_splice_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="trans_stop" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>
</target>

<target id="5ss_0">
   <useseg id="coding_seg" target_phase="0"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3" len_fun="init_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>
```

```
   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>


<target id="5ss_0_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="target"/>
      <output feature="intron_end_not_found" strand="-"/>
   </source>

   <source id="3ss_0_rev" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="both"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="5ss_1">
   <useseg id="coding_seg" target_phase="1"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" target_phase="1"/>

   <source id="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_1_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="target"/>
      <output feature="intron_end_not_found" strand="-"/>
   </source>

   <source id="3ss_1_rev" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="both"/>
      <killdna source_dna="ct" target_dna="a"/>
      <killdna source_dna="tc" target_dna="a"/>
      <killdna source_dna="tt" target_dna="a"/>
      <output feature="intron" strand="-"/>
   </source>
```

```
</target>

<target id="5ss_2">
   <useseg id="coding_seg" target_phase="2"/>
   <useseg id="est_seg"/>
   <killfeat id="stop" target_phase="2"/>

   <source id ="BEGIN">
      <output feature="CDS_start_not_found" strand="+"/>
   </source>

   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_2_rev">
   <source id ="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="target"/>
      <output feature="intron_end_not_found" strand="-"/>
   </source>
   <source id="3ss_2_rev" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron_rev" exact="both"/>
      <killdna source_dna="c" target_dna="ta"/>
      <killdna source_dna="t" target_dna="ca"/>
      <killdna source_dna="t" target_dna="ta"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="3ss_0">
   <source id="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron" exact="target"/>
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_0" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron" exact="both"/>
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_0_rev">
   <useseg id="coding_seg_rev" target_phase="0"/>
   <useseg id="est_seg_rev"/>
   <killfeat id="stop_rev" target_phase="0"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="0"/>
   </source>

   <source id="stop_rev" mindis="3" len_fun="term_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
```

```
   <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="3ss_1">
   <source id="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron" exact="target"/>
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_1" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron" exact="both"/>
      <killdna source_dna="t" target_dna="aa"/>
      <killdna source_dna="t" target_dna="ag"/>
      <killdna source_dna="t" target_dna="ga"/>
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_1_rev">
   <useseg id="coding_seg_rev" target_phase="2"/>
   <useseg id="est_seg_rev"/>
   <killfeat id="stop_rev" target_phase="2"/>

   <source id="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="1"/>
   </source>

   <source id="stop_rev" mindis="3"  len_fun="term_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="1"/>
   </source>

   <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="1"/>
   </source>
</target>

<target id="3ss_2">
   <source id="BEGIN" len_fun="intron_pen">
      <useseg id="est_intron" exact="target"/>
      <output feature="intron_start_not_found" strand="+"/>
   </source>
   <source id="5ss_2" mindis="39" len_fun="intron_pen">
      <useseg id="est_intron" exact="both"/>
      <killdna source_dna="ta" target_dna="a"/>
      <killdna source_dna="ta" target_dna="g"/>
      <killdna source_dna="tg" target_dna="a"/>
      <output feature="intron" strand="+"/>
```

```
   </source>
</target>

<target id="3ss_2_rev">
   <useseg id="coding_seg_rev" target_phase="1"/>
   <useseg id="est_seg_rev"/>
   <killfeat id="stop_rev" target_phase="1"/>

   <source id ="BEGIN">
      <output feature="CDS_end_not_found" strand="-" frame="2"/>
   </source>

   <source id="stop_rev" mindis="3" len_fun="term_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="2"/>
   </source>

   <source id="5ss_0_rev" mindis="20" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="2"/>
   </source>

   <source id="5ss_1_rev" mindis="20" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="2"/>
   </source>

   <source id="5ss_2_rev" mindis="20" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="2"/>
   </source>
</target>

<target id="trans_start">
   <useseg id="est_span"/>

   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="trans_stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="trans_start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="t_splice_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="trans_stop">
   <useseg id="est_seg"/>

   <source id="BEGIN">
      <output feature="UTR3_start_not_found" strand="+"/>
   </source>

   <source id="stop" mindis="0" >
```

```
      <output feature="UTR3" strand="+"/>
   </source>
</target>

<target id="trans_start_rev">
   <useseg id="est_seg_rev"/>

   <source id="BEGIN">
      <output feature="UTR5_end_not_found" strand="-"/>
   </source>

   <source id="t_splice_rev" mindis="0">
      <output feature="TSL_UTR5" strand="-"/>
   </source>

   <source id="start_rev" mindis="0">
      <output feature="UTR5" strand="-"/>
   </source>
</target>

<target id="trans_stop_rev">
   <useseg id="est_span"/>

   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="t_splice_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="trans_start_rev" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="trans_stop" mindis="0" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="t_splice">
   <source id="BEGIN">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="trans_start" mindis="0">
      <useseg id="est_seg"/>
      <output feature="TSL_UTR5" strand="+"/>
   </source>

   <source id="trans_stop" mindis="0" len_fun="intergene_pen">
      <useseg id="est_span"/>
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="0" len_fun="intergene_pen">
```

```
         <useseg id="est_span"/>
         <output feature="intergenic"/>
      </source>

      <source id="trans_start_rev" mindis="0" len_fun="intergene_pen">
         <useseg id="est_span"/>
         <output feature="intergenic"/>
      </source>

      <source id="t_splice_rev" mindis="0" len_fun="intergene_pen">
         <useseg id="est_span"/>
         <output feature="intergenic"/>
      </source>

      <source id="start_rev" mindis="0" len_fun="intergene_pen">
         <useseg id="est_span"/>
         <output feature="intergenic"/>
      </source>
   </target>

   <target id="t_splice_rev">
      <useseg id="est_seg_rev"/>

      <source id="BEGIN"  mindis="0" maxdis="50" len_fun="tsplice_pen">
         <output feature="trans_splice_UTR5_end_not_found" strand="-"/>
      </source>

      <source id="start_rev" mindis="0" maxdis="50" len_fun="tsplice_pen">
         <output feature="trans_splice_UTR5" strand="-"/>
      </source>
   </target>
</model>

<lengthfunctions>
   <lengthfunc id="intron_pen" file="/tables/intron_penalty.klh"/>
   <lengthfunc id="init_ex_pen" file="/tables/exon_penalty.initial.klh"/>
   <lengthfunc id="term_ex_pen" file="/tables/exon_penalty.terminal.klh"/>
   <lengthfunc id="int_ex_pen" file="/tables/exon_penalty.internal.klh"/>
   <lengthfunc id="tsplice_pen" file="/tables/trans_splice_penalty.klh"/>

   <lengthfunc id="sngl_ex_pen">
      <point x="0" y="4"/>
      <point x="1" y="4"/>
   </lengthfunc>

   <lengthfunc id="intergene_pen">
     <point x="0" y="4"/>
     <point x="1" y="4"/>
   </lengthfunc>
</lengthfunctions>

</gaze>
```

A.3 GAZE_GeneID

```
<?xml version="1.0" encoding="US-ASCII"?>
<gaze>

<declarations>
   <feature id="5ss_0" st_off="1" en_off="1" mul="0.6"/>
   <feature id="5ss_1" st_off="1" en_off="1" mul="0.6"/>
   <feature id="5ss_2" st_off="1" en_off="1" mul="0.6"/>
   <feature id="stop" st_off="3" en_off="3" mul="0.6"/>
```

```
   <feature id="3ss_0" st_off="1" en_off="1" mul="0.6"/>
   <feature id="3ss_1" st_off="1" en_off="1" mul="0.6"/>
   <feature id="3ss_2" st_off="1" en_off="1" mul="0.6"/>
   <feature id="start" st_off="0" en_off="3" mul="0.6"/>
   <feature id="start_rev" st_off="3" en_off="0" mul="0.6"/>
   <feature id="3ss_0_rev" st_off="1" en_off="1" mul="0.6"/>
   <feature id="3ss_1_rev" st_off="1" en_off="1" mul="0.6"/>
   <feature id="3ss_2_rev" st_off="1" en_off="1" mul="0.6"/>
   <feature id="stop_rev" st_off="3" en_off="3" mul="0.6"/>
   <feature id="5ss_0_rev" st_off="1" en_off="1" mul="0.6"/>
   <feature id="5ss_1_rev" st_off="1" en_off="1" mul="0.6"/>
   <feature id="5ss_2_rev" st_off="1" en_off="1" mul="0.6"/>

   <segment id="cod_ini_0" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_ini_1" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_ini_2" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_ini_rev_0" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_ini_rev_1" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_ini_rev_2" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_0" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_1" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_2" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_rev_0" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_rev_1" scoring="standard_sum" partial="FALSE" mul="0.4"/>
   <segment id="cod_tr_rev_2" scoring="standard_sum" partial="FALSE" mul="0.4"/>

   <lengthfunction id="intron_pen" mul="0.0"/>
   <lengthfunction id="intergene_pen" mul="0.0"/>
   <lengthfunction id="init_ex_pen" mul="1.0"/>
   <lengthfunction id="term_ex_pen" mul="1.0"/>
   <lengthfunction id="int_ex_pen" mul="1.0"/>
   <lengthfunction id="sngl_ex_pen" mul="1.0"/>
</declarations>


<gff2gaze>
   <!-- Features -->

   <gffline feature="Start" source="geneid_v1.0" strand="+">
      <feat id="start"/>
   </gffline>

   <gffline feature="Start" source="geneid_v1.0" strand="-">
      <feat id="start_rev"/>
   </gffline>

   <gffline feature="Stop" source="geneid_v1.0" strand="+">
      <feat id="stop"/>
   </gffline>

   <gffline feature="Stop" source="geneid_v1.0" strand="-">
      <feat id="stop_rev"/>
   </gffline>

   <gffline feature="Donor" source="geneid_v1.0" strand="+">
      <feat id="5ss_0"/>
      <feat id="5ss_1"/>
      <feat id="5ss_2"/>
   </gffline>

   <gffline feature="Donor" source="geneid_v1.0" strand="-">
      <feat id="5ss_0_rev"/>
      <feat id="5ss_1_rev"/>
      <feat id="5ss_2_rev"/>
   </gffline>
```

```
   <gffline feature="Acceptor" source="geneid_v1.0" strand="+">
      <feat id="3ss_0"/>
      <feat id="3ss_1"/>
      <feat id="3ss_2"/>
   </gffline>

   <gffline feature="Acceptor" source="geneid_v1.0" strand="-">
      <feat id="3ss_0_rev"/>
      <feat id="3ss_1_rev"/>
      <feat id="3ss_2_rev"/>
   </gffline>

   <!-- Segments -->

   <gffline feature="cod_ini" source="GENEID" strand="+" frame="0">
      <seg id="cod_ini_0"/>
   </gffline>

   <gffline feature="cod_ini" source="GENEID" strand="+" frame="1">
      <seg id="cod_ini_1"/>
   </gffline>

   <gffline feature="cod_ini" source="GENEID" strand="+" frame="2">
      <seg id="cod_ini_2"/>
   </gffline>

   <gffline feature="cod_ini" source="GENEID" strand="-" frame="0">
      <seg id="cod_ini_rev_0"/>
   </gffline>

   <gffline feature="cod_ini" source="GENEID" strand="-" frame="1">
      <seg id="cod_ini_rev_1"/>
   </gffline>

   <gffline feature="cod_ini" source="GENEID" strand="-" frame="2">
      <seg id="cod_ini_rev_2"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="+" frame="0">
      <seg id="cod_tr_0"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="+" frame="1">
      <seg id="cod_tr_1"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="+" frame="2">
      <seg id="cod_tr_2"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="-" frame="0">
      <seg id="cod_tr_rev_0"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="-" frame="1">
      <seg id="cod_tr_rev_1"/>
   </gffline>

   <gffline feature="cod_tr" source="GENEID" strand="-" frame="2">
      <seg id="cod_tr_rev_2"/>
   </gffline>
</gff2gaze>

<dna2gaze>
```

```
   <dnafeat pattern="taa">
      <feat id="stop" score="-100"/>
   </dnafeat>
   <dnafeat pattern="tag">
      <feat id="stop" score="-100"/>
   </dnafeat>
   <dnafeat pattern="tga">
      <feat id="stop" score="-100"/>
   </dnafeat>

   <dnafeat pattern="tta">
      <feat id="stop_rev" score="-100"/>
   </dnafeat>
   <dnafeat pattern="cta">
      <feat id="stop_rev" score="-100"/>
   </dnafeat>
   <dnafeat pattern="tca">
      <feat id="stop_rev" score="-100"/>
   </dnafeat>

   <takedna id="5ss_1" st_off="0" en_off="1"/>
   <takedna id="3ss_1" st_off="1" en_off="-1"/>
   <takedna id="5ss_2" st_off="-1" en_off="1"/>
   <takedna id="3ss_2" st_off="1" en_off="0"/>
   <takedna id="5ss_1_rev" st_off="1" en_off="0"/>
   <takedna id="3ss_1_rev" st_off="-1" en_off="1"/>
   <takedna id="5ss_2_rev" st_off="1" en_off="-1"/>
   <takedna id="3ss_2_rev" st_off="0" en_off="1"/>
</dna2gaze>

<model>
   <target id="END">
      <source id="BEGIN">
         <output feature="no genes"/>
      </source>

      <source id="stop">
         <output feature="intergenic"/>
      </source>

      <source id="start_rev" mindis="0">
         <output feature="intergenic"/>
      </source>

      <source id="3ss_0_rev">
         <output feature="intron" strand="-"/>
      </source>

      <source id="3ss_1_rev">
         <output feature="intron" strand="-"/>
      </source>

      <source id="3ss_2_rev">
         <output feature="intron" strand="-"/>
      </source>

      <source id="5ss_0">
         <output feature="intron" strand="+"/>
      </source>

      <source id="5ss_1">
         <output feature="intron" strand="+"/>
      </source>

      <source id="5ss_2">
```

```
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="start">
   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="2000" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="2000" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="start_rev">
   <useseg id="cod_ini_rev_0" exact="target" />
   <useseg id="cod_tr_rev_0" target_phase="0" />
   <useseg id="cod_tr_rev_1" target_phase="1" />
   <useseg id="cod_tr_rev_2" target_phase="2" />

   <killfeat id="stop_rev" target_phase="0"/>

   <source id="stop_rev" mindis="60" len_fun="sngl_ex_pen" phase="0">
      <output feature="CDS_term" strand="-" frame="0"/>
   </source>

   <source id="5ss_0_rev" mindis="3" len_fun="init_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="3" len_fun="init_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="3" len_fun="init_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="stop">
   <useseg id="cod_tr_0" target_phase="0" />
   <useseg id="cod_tr_2" target_phase="1" />
   <useseg id="cod_tr_1" target_phase="2" />

   <killfeat id="stop" target_phase="0"/>

   <source id="start" mindis="60" len_fun="sngl_ex_pen" phase="0">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS_term" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="0" len_fun="term_ex_pen" phase="0">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS_term" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="0" len_fun="term_ex_pen" phase="2">
      <useseg id="cod_ini_1" exact="source" />
      <output feature="CDS_term" strand="+" frame="1"/>
   </source>
```

```
   <source id="3ss_2" mindis="0" len_fun="term_ex_pen" phase="1">
      <useseg id="cod_ini_2" exact="source" />
      <output feature="CDS_term" strand="+" frame="2"/>
   </source>
</target>

<target id="stop_rev">
   <source id="BEGIN">
      <output feature="intergenic"/>
   </source>

   <source id="start_rev" mindis="2000" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>

   <source id="stop" mindis="2000" len_fun="intergene_pen">
      <output feature="intergenic"/>
   </source>
</target>

<target id="5ss_0">
   <useseg id="cod_tr_0" target_phase="0" />
   <useseg id="cod_tr_2" target_phase="1" />
   <useseg id="cod_tr_1" target_phase="2" />

   <killfeat id="stop" target_phase="0"/>

   <source id="start" mindis="3" len_fun="init_ex_pen" phase="0">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="18" len_fun="int_ex_pen" phase="0">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="18" len_fun="int_ex_pen" phase="2">
      <useseg id="cod_ini_1" exact="source" />
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="18" len_fun="int_ex_pen" phase="1">
      <useseg id="cod_ini_2" exact="source" />
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_0_rev">
   <source id="BEGIN">
      <output feature="intron" strand="-"/>
   </source>

   <source id="3ss_0_rev" mindis="20" maxdis="25000" len_fun="intron_pen">
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="5ss_1">
   <useseg id="cod_tr_1" target_phase="0" />
   <useseg id="cod_tr_0" target_phase="1" />
   <useseg id="cod_tr_2" target_phase="2" />

   <killfeat id="stop" target_phase="1"/>
```

```
   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="1">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="18" len_fun="int_ex_pen" phase="1">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="18" len_fun="int_ex_pen" phase="0">
      <useseg id="cod_ini_1" exact="source" />
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="18" len_fun="int_ex_pen" phase="2">
      <useseg id="cod_ini_2" exact="source" />
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_1_rev">
   <source id="BEGIN">
      <output feature="intron" strand="-"/>
   </source>

   <source id="3ss_1_rev" mindis="20" maxdis="25000" len_fun="intron_pen">
      <killdna source_dna="ct" target_dna="a"/>
      <killdna source_dna="tc" target_dna="a"/>
      <killdna source_dna="tt" target_dna="a"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="5ss_2">
   <useseg id="cod_tr_2" target_phase="0" />
   <useseg id="cod_tr_1" target_phase="1" />
   <useseg id="cod_tr_0" target_phase="2" />

   <killfeat id="stop" target_phase="2"/>

   <source id="start" mindis="3"  len_fun="init_ex_pen" phase="2">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_0" mindis="18" len_fun="int_ex_pen" phase="2">
      <useseg id="cod_ini_0" exact="source" />
      <output feature="CDS" strand="+" frame="0"/>
   </source>

   <source id="3ss_1" mindis="18" len_fun="int_ex_pen" phase="1">
      <useseg id="cod_ini_1" exact="source" />
      <output feature="CDS" strand="+" frame="1"/>
   </source>

   <source id="3ss_2" mindis="18" len_fun="int_ex_pen" phase="0">
      <useseg id="cod_ini_2" exact="source" />
      <output feature="CDS" strand="+" frame="2"/>
   </source>
</target>

<target id="5ss_2_rev">
   <source id="BEGIN">
      <output feature="intron" strand="-"/>
```

```
   </source>

   <source id="3ss_2_rev" mindis="20" maxdis="25000" len_fun="intron_pen">
      <killdna source_dna="c" target_dna="ta"/>
      <killdna source_dna="t" target_dna="ca"/>
      <killdna source_dna="t" target_dna="ta"/>
      <output feature="intron" strand="-"/>
   </source>
</target>

<target id="3ss_0">
   <source id="BEGIN">
      <output feature="intron" strand="+"/>
   </source>

   <source id="5ss_0" mindis="20" maxdis="25000" len_fun="intron_pen">
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_0_rev">
   <useseg id="cod_ini_rev_0" exact="target" />
   <useseg id="cod_tr_rev_0" target_phase="0" />
   <useseg id="cod_tr_rev_1" target_phase="1" />
   <useseg id="cod_tr_rev_2" target_phase="2" />

   <killfeat id="stop_rev" target_phase="0"/>

   <source id="stop_rev" mindis="0" len_fun="term_ex_pen" phase="0">
      <output feature="CDS_term" strand="-" frame="0"/>
   </source>

   <source id="5ss_0_rev" mindis="18" len_fun="int_ex_pen" phase="0">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_1_rev" mindis="18" len_fun="int_ex_pen" phase="1">
      <output feature="CDS" strand="-" frame="0"/>
   </source>

   <source id="5ss_2_rev" mindis="18" len_fun="int_ex_pen" phase="2">
      <output feature="CDS" strand="-" frame="0"/>
   </source>
</target>

<target id="3ss_1">
   <source id="BEGIN">
      <output feature="intron" strand="+"/>
   </source>

   <source id="5ss_1" mindis="20" maxdis="25000" len_fun="intron_pen">
      <killdna source_dna="t" target_dna="aa"/>
      <killdna source_dna="t" target_dna="ag"/>
      <killdna source_dna="t" target_dna="ga"/>
      <output feature="intron" strand="+"/>
   </source>
</target>

<target id="3ss_1_rev">
   <useseg id="cod_ini_rev_1" exact="target" />
   <useseg id="cod_tr_rev_1" target_phase="0" />
   <useseg id="cod_tr_rev_2" target_phase="1" />
   <useseg id="cod_tr_rev_0" target_phase="2" />

   <killfeat id="stop_rev" target_phase="2"/>
```

```
      <source id="stop_rev" mindis="0"  len_fun="term_ex_pen" phase="2">
         <output feature="CDS_term" strand="-" frame="1"/>
      </source>

      <source id="5ss_0_rev" mindis="18" len_fun="int_ex_pen" phase="2">
         <output feature="CDS" strand="-" frame="1"/>
      </source>

      <source id="5ss_1_rev" mindis="18" len_fun="int_ex_pen" phase="0">
         <output feature="CDS" strand="-" frame="1"/>
      </source>

      <source id="5ss_2_rev" mindis="18" len_fun="int_ex_pen" phase="1">
         <output feature="CDS" strand="-" frame="1"/>
      </source>
   </target>

   <target id="3ss_2">
      <source id="BEGIN">
         <output feature="intron" strand="+"/>
      </source>

      <source id="5ss_2" mindis="20" maxdis="25000" len_fun="intron_pen">
         <killdna source_dna="ta" target_dna="a"/>
         <killdna source_dna="ta" target_dna="g"/>
         <killdna source_dna="tg" target_dna="a"/>
         <output feature="intron" strand="+"/>
      </source>
   </target>

   <target id="3ss_2_rev">
      <useseg id="cod_ini_rev_2" exact="target" />
      <useseg id="cod_tr_rev_2" target_phase="0" />
      <useseg id="cod_tr_rev_0" target_phase="1" />
      <useseg id="cod_tr_rev_1" target_phase="2" />

      <killfeat id="stop_rev" target_phase="1"/>

      <source id="stop_rev" mindis="0" len_fun="term_ex_pen" phase="1">
         <output feature="CDS_term" strand="-" frame="2"/>
      </source>

      <source id="5ss_0_rev" mindis="18" len_fun="int_ex_pen" phase="1">
         <output feature="CDS" strand="-" frame="2"/>
      </source>

      <source id="5ss_1_rev" mindis="18" len_fun="int_ex_pen" phase="2">
         <output feature="CDS" strand="-" frame="2"/>
      </source>

      <source id="5ss_2_rev" mindis="18" len_fun="int_ex_pen" phase="0">
         <output feature="CDS" strand="-" frame="2"/>
      </source>
   </target>
</model>


<lengthfunctions>
   <lengthfunc id="intron_pen">
      <point x="0" y="1.0"/>
      <point x="1" y="1.0"/>
   </lengthfunc>

   <lengthfunc id="intergene_pen">
```

```
      <point x="0" y="1.0"/>
      <point x="1" y="1.0"/>
   </lengthfunc>

   <lengthfunc id="init_ex_pen">
      <point x="0" y="4.5"/>
      <point x="1" y="4.5"/>
   </lengthfunc>

   <lengthfunc id="int_ex_pen">
      <point x="0" y="4.5"/>
      <point x="1" y="4.5"/>
   </lengthfunc>

   <lengthfunc id="term_ex_pen">
      <point x="0" y="4.5"/>
      <point x="1" y="4.5"/>
   </lengthfunc>

   <lengthfunc id="sngl_ex_pen">
      <point x="0" y="4.5"/>
      <point x="1" y="4.5"/>
   </lengthfunc>
</lengthfunctions>

</gaze>
```