Gene Prediction with Conditional Random Fields

by

Matthew K. Doherty

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2007

Author .....
Department of Electrical Engineering
and Computer Science
June 1, 2007

Certified by..... .....
James Galagan
Assoc. Director,
Microbial Genome Analysis, Broad Institute
Thesis Supervisor

Certified by..... .
David DeCaprio
Director of Informatics Development,
Chemical F tute
/isor

Accepted by ....
nith
Chairman, Department Committee on Graduate Students

Gene Prediction with Conditional Random Fields

by

Matthew K. Doherty

Submitted to the Department of Electrical Engineering
and Computer Science
on June 1, 2007, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The accurate annotation of an organism's protein-coding genes is crucial for subsequent genomic analysis. The rapid advance of sequencing technology has created a gap between genomic sequences and their annotations. Automated annotation methods are needed to bridge this gap, but existing solutions based on hidden Markov models cannot easily incorporate diverse evidence to make more accurate predictions. In this thesis, I built upon the semi-Markov conditional random field framework created by DeCaprio et al. to predict protein-coding genes in DNA sequences. Several novel extensions were designed and implemented, including a 29-state model with both semi-Markov and Markov states, an N-best Viterbi inference algorithm, several classes of discriminative feature functions that incorporate diverse evidence, and parallelization of the training and inference algorithms. The extensions were tested on the genomes of *Phytophthora infestans*, *Culex pipiens*, and *Homo sapiens*. The gene predictions were analyzed and the benefits of discriminative methods were explored.

Thesis Supervisor: James Galagan
Title: Assoc. Director,
Microbial Genome Analysis, Broad Institute

Thesis Supervisor: David DeCaprio
Title: Director of Informatics Development,
Chemical Biology and Novel Therapeutics, Broad Institute

Acknowledgments

Thanks to Dave DeCaprio, without whose generous support at every stage this thesis would not have happened. Thanks to James Galagan as well for supervising it. This thesis was made possible by the resources provided to me by the Broad Institute.

Thanks to Elliot Fleming, Ross Glashan, and John Nham for interesting discussions that led to several of the ideas I present. Thanks to Mariya Kovalenko for catching all my mistakes.

This thesis is dedicated to my loving family, without whose unconditional love, joy, encouragement, and affection none of my endeavors would be possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The accurate annotation of an organism's protein-coding genes is crucial for subsequent genomic analysis. While the genomes of many different organisms have been sequenced nearly completely to date, the annotations of these genomes are not nearly so mature. There are many reasons that sequencing has outpaced annotation; among them is that sequencing technology has rapidly advanced since the start of the 21st century. In contrast, there has not been an increase in fast, accurate end-to-end annotation tools, and so this step in translating genomes into knowledge remains largely a manual effort.

In this thesis, we address the problem of protein-coding gene annotation. A modern annotation pipeline for such a task consists first of predicting genes using mRNA, protein, or other species' genome homologies [18]. In eukaryotes, these methods can quickly find at least half the protein-coding genes, but this extrinsic evidence is expensive to procure, and its availability determines the sensitivity of predictions. *Ab initio* programs, which use the DNA sequence alone, are used to predict the remaining genes. The extensions to the gene finding program described in this thesis aim to improve the state-of-the-art in gene annotation by (i) increasing the number of correctly identified genes found with extrinsic evidence and (ii) improving the reliability of *ab initio* gene prediction to find the remaining genes.

In particular, we build upon the work of [34] and [7] to create conditional random fields (CRFs) that are capable of predicting genes more accurately than their hidden

Markov model (HMM) counterparts. We begin with an overview of the problem domain and existing solutions, and a thorough description of our CRF infrastructure. We then describe the specific extensions developed in this thesis to address the problems inherent in scaling CRFs to larger, more complex genomes. Specifically, our hybrid 29-state model reduces algorithmic complexity; parallelization and accurate starting points for training decrease the time needed for a train/test cycle; an N-best Viterbi algorithm helps us tune the CRFs by showing suboptimal predictions; and a number of probabilistic and non-probabilistic features improve prediction accuracy using the genome sequence, sequence alignments with other species, and experimental evidence. We test our models on the genomes of *Phytophthora infestans*, *Culex pipiens quinquefasciatus*, and *Homo sapiens*, comparing several CRF and HMM-equivalent models using various and diverse sets of evidence. We conclude by analyzing the results and proposing directions for future work.

1.1 Genomics

An organism's genome is the means by which it passes inheritable traits to its offspring. The genome is composed of deoxyribonucleic acid, or DNA, tightly wound in a double helix. DNA can be viewed as a sequence of various discrete monomers called nucleotides. There are four different nucleotides in DNA, each differing in its base. These four bases are the purines, adenine and guanine, and the complementary pyrimidines, thymine and cytosine. Adenine is the complement of thymine, and guanine the complement of cytosine. The two strands' sequence bases are complements of each other, where each pair of complementary bases forms a base-pair (bp).

The central dogma of transcription and translation is the process by which a cell maps a set of instructions (encoded as nucleotides) from its genome to functional macromolecules, called proteins, which play essential roles in every cell process and structure. A protein-coding gene is a sequence of DNA bases from which one or more RNA transcripts are created that are ultimately translated into proteins. In general, a gene may not be ultimately translated into protein, but non-protein-coding genes

are not the targets of the gene prediction methods presented in this thesis.

On the other hand, protein-coding genes are those genes that the cell decodes to create a protein by the process shown in figure 1-1. In the first step of the decoding process, known as transcription, the cellular machinery creates a strand of pre-messenger ribonucleic acid (pre-mRNA), which is a sequence of nucleotides complementary to those in the DNA. The bases are the same complements as found in the double-stranded DNA, except that adenine in DNA is complemented by uracil in RNA.

In eukaryotes, which are the focus of our study, a typical pre-mRNA strand consists of an alternating series of exons and introns. The pre-mRNA is cut at donor and acceptor sites, excising the introns in a process known as splicing. The remaining exons are glued together to form mRNA.

A protein is created from the finished mRNA in a process known as translation. Translation reads codons, which are consecutive triplets of nucleotides, and by the genetic code converts them into amino acids. Translation starts at a start codon, and stops at a stop codon. The amino acids are the monomers that form a protein. The regions between start and stop codons are translated, and the others (known as untranslated region or UTR) are not. Finally, the protein folds in a variety of ways into its final configuration, and is used by the cell.

1.2 Comparative Genomics

Genomes are not identical among individual organisms. There are many sources of genetic variation, including recombination, errors made while copying DNA for offspring, mutations that occur during an organism's development, or DNA damage caused by chemical reactions from environmental factors. There are different types of variation as well: point mutations are changes of a single base, and indels are insertions or deletions of a tract of nucleotides. Variation is very common, and is fairly evenly distributed throughout the genome. Changes to the functional regions of the genome, including its genes, usually decrease the fitness of the organism, putting

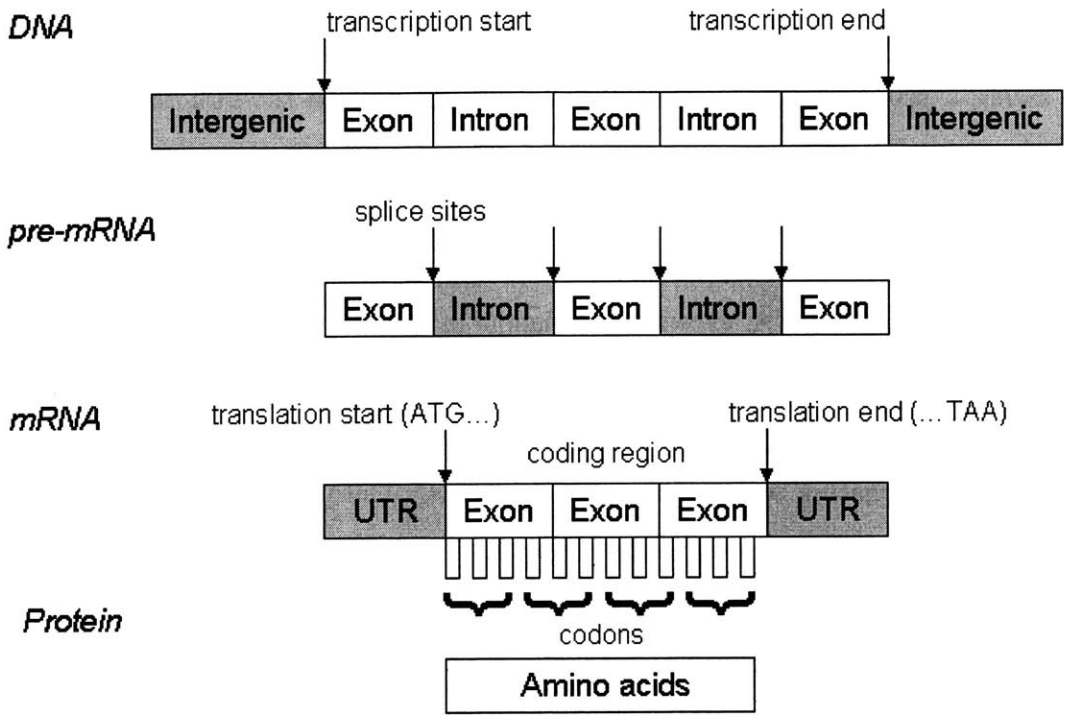

Figure 1-1: Transcription, splicing, and translation.

pressure on its lineage that can gradually build up over generations. A less-fit lineage will not as readily survive and reproduce to pass on its genome as a more fit lineage. Thus, changes to functional regions are usually evolutionarily selected against. Much less commonly, mutations to functional regions in the genome can actually increase the fitness of the organism, and so its lineage will be selected for.

Because changes to functional regions usually decrease the organism's fitness, they are not as commonly passed on from generation to generation. This means that changes to nonfunctional regions are more likely to be passed on, and after many generations the functional regions are conserved while the nonfunctional regions bear little resemblance to the ancestors'. That is, the functional regions leave a footprint of conservation in the genome.

Thus, a significant insight into genomics is that the genomes of evolutionarily-related organisms are themselves related, especially in functional regions [15]. In particular, the similarity between two species' genomes is correlated with the amount

of time that has passed since the species branched from a common ancestor. A multiple sequence alignment among evolutionarily-related organisms will bring to light their genomes' regions of similarity, and provide evidence for the locations of functional regions.

A number of multiple sequence alignment (MSA) tools, such as MULTIZ, MLAGAN, TBA, and MUSCLE, have been developed to find these regions of similarity among genomes using sophisticated alignment algorithms [19]. Closely-related species may show near-perfect sequence similarity, while genomes of distantly-related species may bear little resemblance. But by choosing two species at an appropriate evolutionary distance (that is, the amount of genetic change that occurred since their common ancestor), their sequence alignments can be used to identify conserved, and likely functional, genomic regions. Thus, MSAs are immensely useful as extrinsic evidence (evidence other than the DNA sequence) for the presence of protein-coding genes.

1.3 Gene Prediction using Hidden Markov Models

Gene prediction can be defined as a labeling problem. Given a sequence of nucleotides $X = \{x_t\}$, the sequence of annotating labels $Y = \{y_t\}$ must be populated to show where the genes are located (e.g. $y_t = 1$ if the nucleotide x_t is part of a gene, or 0 otherwise). Currently, the most advanced gene prediction tools predict such labelings using a probabilistic model.

In particular, the hidden Markov model (HMM) has proven to be an invaluable resource for a variety of labeling tasks including gene prediction. It is where we begin our research into improved gene finding methods. An HMM is a Markov model in which the sequence of states (the labels) corresponding to a sequence of observations must be determined. Therefore, the model emits the input nucleotide sequence X, and the sequence of labels Y are the states corresponding to each emission. The model is parameterized by initial probabilities π_y, transition probabilities $T_{y',y}$ for all states y' and y, and the probability $Q_y(x)$ of each emission x at each state y. Thus,

the HMM defines a probability distribution over observations and labels:

$$\Pr(X, Y) = \pi_y \prod_{t=1}^{L-1} T_{y',y} \prod_{t=1}^{L} Q_y(x_t).$$

Since this distribution can be sampled to generate an observation sequence, an HMM is referred to as a generative model.

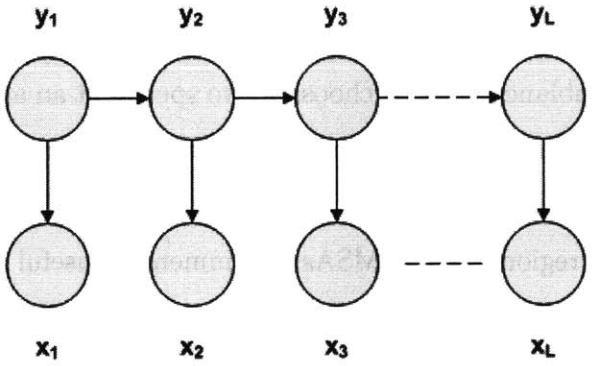

Figure 1-2: An HMM's graph.

In a generalized hidden Markov model, or GHMM, an emission is not an individual nucleotide but rather a segment (a consecutive subsequence) of nucleotides with the same label. Therefore, the length of the segment can be explicitly modeled. GHMM's underlie many of the successful gene prediction tools available today, such as AUGUSTUS [30], EHMM [22], Genscan [3], Fgenesh [24], and GeneID [21].

Several well-known algorithms operate on HMMs. The forward algorithm finds the model's joint probability $\Pr(X, Y)$ of a sequence of observations and the states that emitted them (or, for gene prediction, nucleotides and their labels). An HMM can be used to solve the inference problem: given a sequence of observations X, find the most likely corresponding sequence of labels Y. This problem is solved for HMMs with the Viterbi algorithm, which finds $\max_Y \Pr(Y|X)$, the most-likely Y given X. For gene prediction, the model is trained with a straightforward supervised algorithm that counts the occurrences of the various transitions and emissions to assign probabilities to each.

It can be readily seen that an HMM is a roundabout solution to some problems,

such as gene prediction, that involve inference and diverse observed evidence. Indeed, any generative model is a roundabout solution to a labeling problem. First, in order to find $\Pr(Y|X) = \Pr(X,Y)/\Pr(X)$, generative models require a complete and accurate joint probability distribution $\Pr(X,Y)$ of the observations and their labels. In particular, because it must model the observation distributions, extrinsic observations are difficult to incorporate. Moreover, all the effort used to find these distributions is not even directly useful, as the distribution of concern is $\Pr(Y|X)$. Finally, in an HMM each observation is assumed to be independent of all others given its label, which makes it difficult for the model to consider remote observations at any particular position.

1.4 Linear-Chain Conditional Random Fields

A conditional random field (CRF) is an alternative probabilistic model. A CRF models the probability $\Pr(Y|X)$ of a sequence of hidden variables Y (the labels) given observations X with a weighted sum of feature functions evaluated over X and Y [16]. Graphically, nodes represent the individual y_t's, as well as all of X together. The graph's edges represent dependencies among the observations and hidden variables, where features are defined over fully-connected subgraphs, or cliques. For gene prediction, a specific case of the CRF called a linear-chain CRF finds use [7] [34]. In a linear-chain CRF, features are evaluated over all of X and consecutive hidden variables (y', y).

Because a CRF directly models the conditional probability of a hidden sequence Y given observations X, it is referred to as a discriminative model [31] [32].

In a CRF, this conditional probability can be written as

$$\Pr(Y|X) = \frac{1}{Z_\lambda(X)} \exp\left(\sum_j \lambda_j F_j(Y,X)\right),$$

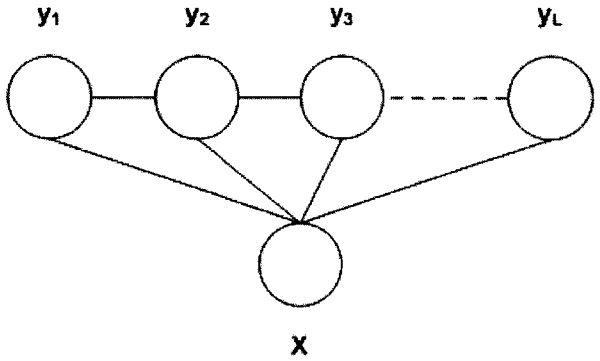

Figure 1-3: A linear-chain CRF's graph.

or in vector notation,

$$\Pr(Y|X) = \frac{1}{Z_\lambda(X)} \exp\left(\lambda \cdot F(Y, X)\right).$$

$Z_\lambda(X)$ is the normalization factor,

$$Z_\lambda(X) = \sum_Y \exp\left(\sum_j \lambda_j F_j(Y, X)\right).$$

For a linear-chain CRF, the j^{th} feature sum F_j is defined to be the feature function f_j evaluated over the entire sequence:

$$F_j(Y, X) = \sum_{t=1}^{L} f_j(y_{t-1}, y_t, X, t).$$

A linear-chain CRF can reproduce the predictions of an HMM if several constraints are placed on the model and its features. First, the hidden variables Y can be states from a Markov model where the edges indicate allowed transitions. For gene prediction, the same state models as HMMs can be used. Furthermore, feature functions can model the log-probability of a transition between particular consecutive states (transition probabilities), as well as the log-probability of an emission corresponding to a particular state (emission probabilities). When the weights are each set to 1.0, the features each output a log-probability. By the equation for $\Pr(Y|X)$, these

feature sums are exponentiated, ultimately yielding probabilities each corresponding to an HMM property. Thus, when the model and feature functions are defined in this way, the CRF exactly reproduces an HMM, so the HMM can be viewed as a special case of the CRF [31].

But the linear-chain CRF can extend the above HMM-like model as well. The feature weights do not have to be set to 1.0; instead, they can be discriminatively trained by maximizing a criterion of our choosing. In addition, because of the normalization factor Z_λ, the features do not have to produce well-defined conditional probability distributions. Instead, they can be arbitrary real-valued functions of the observations and hidden variables, which makes incorporating additional observed evidence a much simpler task. Finally, distant observations do not have to be independent of each other as assumed by an HMM. The similarities between linear-chain CRFs and HMMs are readily apparent, and we will see this theme explored throughout this thesis.

1.4.1 Semi-Markov Linear-Chain CRFs

The strict Markov model-like CRF described above is parameterized by emission and transition probabilities. Let a segment of the sequence of hidden variables be defined by a triplet of start position, stop position, and label (t_i, u_i, v_i). When multiplied over the length of a single segment (recall the Markov assumption), the transition probabilities implicitly form an exponential (or, equivalently, geometric) probability distribution over the possible lengths of that label. But for some problems we would like to explicitly model this length using some other distribution.

Thus, while the goal is to assign labels y_t at individual positions t, it is useful to consider explicitly the different commonly-labeled segments in a labeling. A sequence's segmentation Y into p segments can be written as

$$Y = (t_i, u_i, v_i)_{i=1}^{p}.$$

A Markov model assigns a distinct label y_t at a particular position t. The semi-

Markov CRF (SMCRF), analogous to a GHMM, assigns a label v_i to a segment (t_i, u_i, v_i) [25]. Feature functions are now evaluated over the entire segment, not just individual positions, so the sums are redefined to be

$$F_j(Y, X) = \sum_{i=1}^{p} f_j(v_{i-1}, v_i, X, t_i, u_i),$$

where each feature function is parameterized by the previous segment's state v_{i-1}, the current segment's state v_i, the observation sequence X, and the segment's start and end positions t_i and u_i. Since feature functions are given the segment length as an input, they can explicitly model the length distributions.

1.4.2 Using an SMCRF

Traditionally, a CRF's weights are trained by maximizing the conditional log-likelihood of the correct labeling Y^0 given observations X^0, which are both provided in a training set.

$$\lambda_{\text{CML}} = \text{argmax}_\lambda(\log(\Pr(Y^0|X^0)))$$

This log-likelihood is a concave function of the weights because its Hessian is the negative covariance matrix of the feature sums [7]. Thus, its only maximum is a global maximum. It is typically maximized using a gradient-based optimization method. For an SMCRF, the conditional maximum likelihood gradient is computed using a dynamic programming algorithm consisting of a forward and backward pass in some ways similar to that used by a GHMM to compute marginal probabilities.

Likewise, the inference problem is solved using a variant of the Viterbi algorithm similar to a GHMM's [23], which consists of a forward pass to compute best partial paths and a backward trace through the most-likely path.

1.4.3 Training

The weights λ of a linear-chain CRF are trained using a gradient descent algorithm. Gradient descent algorithms use an objective function F_λ and its gradient with re-

spect to λ to iteratively increase the value of the objective function. L-BFGS, a limited-memory variant of the traditional Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, is one such gradient descent algorithm [37]. The BFGS method computes an approximate Hessian matrix at each step, using these second derivatives to find a new set of feature weights which are used to compute the objective value and its gradient. The L-BFGS algorithm was chosen because it has been shown to perform well on CRFs [35].

Conditional Maximum-Likelihood (CML)

Traditionally, weights are found by maximum CML. Given training data (X^0, Y^0) consisting of a sequence of observations and corresponding labeling Y^0, the objective function is the conditional probability $\Pr(Y^0|X^0)$ of the labeling given the observations. The gradient of this objective function with respect to the feature weights is found with a dynamic programming algorithm similar to that used by HMMs to calculate marginal probabilities. The gradient G_j for each feature is

$$G_j = F_j - E[F_j].$$

The expected feature sums expand to

$$E[F_j] = \sum_{t=1}^{L} f_j(y', y, X, t) \Pr(y', y|X),$$

where $\Pr(y', y|X^0)$ are the marginal probabilities of consecutive labels (y', y) in Y^0.

The marginal probabilities are found using a forward and backward pass. In the forward pass, one computes the forward feature sums $\alpha(y, t)$ into the label y at position t given the observations $(x_1, x_2, ..., x_t)$ from X^0. These are found by the recurrence

$$\alpha(y, t) = \sum_{y'} \exp\left(\lambda \cdot f(y', y, X^0, t)\right) \alpha(y', t-1)$$

using the vector notation from above. In the backward pass, one computes the back-

ward feature sums $\beta(y', t)$ out of the label y' at position t given the observations $(x_t + 1, x_t + 2, ..., x_L)$. These are found by the similar recurrence

$$\beta(y', t) = \sum_y \exp\left(\lambda \cdot f(y', y, X^0, t)\right) \beta(y, t + 1)$$

Once the α's and β's have been computed, the marginals are

$$\Pr(y', y|X^0) = \frac{1}{Z_\lambda(X^0)} \alpha(y', t - 1) \exp\left(\lambda \cdot f(y', y, X^0, t)\right) \beta(y, t),$$

where

$$Z_\lambda(X^0) = \sum_y \alpha(L, y).$$

The objective value and the feature gradients are normalized to per-position values, which keeps them all in roughly the same order of magnitude and allows standard optimization tolerances and easier human debugging.

To extend the algorithm to handle semi-Markov states, the expectation of the feature sums must include all possible segment lengths $l \in 1...L'$:

$$E[F_j] = \sum_{u=1}^{L} \sum_{l=1}^{L'} f_j(v', v, X^0, u - l, u) \Pr(v', t, u, v|X^0).$$

The new marginal probabilities are from a previous state v' to an entire segment (t, u, v), and are found using modified α's and β's. Namely, let $\alpha(v, u)$ be the forward feature sum over all lengths l into the label v ending at position u. Likewise, let $\beta(v', t)$ be the backward feature sum over all l into v' ending at t. They are found by the recurrences

$$\alpha(v, u) = \sum_{l=1}^{L'} \sum_{v'} \exp\left(\lambda \cdot f(v', v, X^0, u - l, u)\right) \alpha(v', u - l - 1)$$

and

$$\beta(v', t) = \sum_{l=1}^{L'} \sum_{v} \exp\left(\lambda \cdot f(v', v, X^0, t, t + l)\right) \beta(v, t + l + 1).$$

Finally, the segment marginal probabilities are

$$\Pr(v', t, u, v|X^0) = \frac{1}{Z_\lambda(X^0)}\alpha(v', u-1)\exp\left(\lambda \cdot f(v', v, X^0, t, u)\right)\beta(v, t),$$

where

$$Z_\lambda(X^0) = \sum_v \alpha(v, L).$$

Maximum Expected Accuracy (MEA)

The CML gradient optimizes gene prediction accuracy indirectly by maximizing the likelihood of the correct labeling. Ideally, we would optimize the feature weights by maximizing the model's performance using the same metrics we use to evaluate real predictions: in particular, gene prediction tools are typically evaluated by their nucleotide, splice site, and gene accuracies. But changing the feature weights causes different labelings to be inferred, resulting in discontinuous jumps in the accuracy with respect to the feature weights. This accuracy would would not be concave, and since it is not continuous, it is not differentiable either. Thus, gradient-descent would be impossible.

In MEA, we instead compute an alternative objective function that maximizes the model's expected accuracy over the distribution of possible segmentations that can be produced by the CRF [7]. Using a single training sequence (Y^0, X^0) for simplicity, the accuracy of the path Y is defined to be the similarity S between Y and Y^0 given X^0. In turn, the similarity function S is defined to be the sum of individual comparisons of the edges taken by the hidden paths.

$$S(Y, Y^0, X^0) = \sum_{i=1}^{p} s(y_{i-1}, y_i, y^0_{i-1}, y^0_i, X^0, i).$$

The objective function A is the expected value of S given weights λ.

$$A_{\text{MEA}} = E_\lambda[S(Y, Y^0, X^0)].$$

Finally, the weights λ are found that maximize A:

$$\lambda_{\text{MEA}} = \text{argmax}_{\lambda}(A_{\text{MEA}}(\lambda)).$$

This objective function is at least differentiable because it uses only local comparisons. But since it is not concave, we first seed the weights by maximizing CML, and then improve them using the MEA gradient. Thus, by construction, the new weights will improve the expected accuracy compared to the weights found by CML alone.

The gradient is defined to be the covariance of F_j and S [34]:

$$G_j = E[F_j S(Y, Y^0, X^0)] - (E[F_j])(E[S(Y, Y^0, X^0)]).$$

The expectation of the product expands to

$$E[F_j S(Y, Y^0, X^0)] = \sum_{u=1}^{L} \sum_{l=1}^{L'} f_j(y', y, X, u-l, u) \Pr(y', y|X) s(y', y, y^{0'}, y^0, X^0, u),$$

which can be computed efficiently using an additional set of forward and backward passes similar to those used to find $E[F_j]$.

To date, we have tried two different scoring functions, both giving a single point for each correct nucleotide, and one adding in various bonuses for correct splice sites. Empirically [7], the most interesting scoring functions are those that give a significant bonus to splice sites, and the most successful gives 1 point for a correct nucleotide and a 200-point bonus for a correct splice site.

1.4.4 Inference

Viterbi

The Viterbi algorithm used for inference is a straightforward variant of that used for HMMs. We iterate over all the positions t in the sequence. For each state y at position t we find the most likely previous state by taking the evaluation of the best partial path to each of the possible previous states y' with the evaluation of the edge

(y', y). The value $V(t, y)$ of the best partial path to y at position t is thus defined by the recurrence

$$V(t,y) = \begin{cases} \max_{y'} V(t-1, y') + \lambda \cdot F(y, y', X, t), & \text{if } t > 0 \\ 0, & \text{if } t = 0 \\ -\infty, & \text{if } t < 0 \end{cases}$$

In the semi-Markov case, we must expand the search over possible segment lengths $l \in 1...L'$, where L' is the largest allowed segment length:

$$V(t,v) = \begin{cases} \max_{v', l=1...L'} V(u-l-1, v') + \lambda \cdot F(v, v', X, u-l-1, u), & \text{if } t > 0 \\ 0, & \text{if } t = 0 \\ -\infty, & \text{if } t < 0 \end{cases}$$

v' then is the previous state that is part of the most likely partial path to the segment (t, u, v) [25]. We store a back pointer from the segment to this most likely previous state. When we reach the end of the sequence, we choose the ending segment with the highest partial path probability (where the partial path is now the complete path), and follow its back pointers to recover the Viterbi path.

1.5 Conrad

To research the application of CRFs to gene prediction, we developed Conrad (CONditional RAnDom field), a modular and extensible software CRF framework implemented in Java and gene caller built on top of this framework. The following components were developed for gene prediction, but are all easily substituted with others for use in different problem domains or to leverage future theoretical advances.

1.5.1 Caching Feature Evaluations

During training, we use iterative algorithms that evaluate every feature at each position many times. But in order to simplify the process of encoding evidence as feature functions, we prefer not to have to spend time designing efficient feature function implementations. So instead of evaluating the features each time their values are needed by the algorithms, we use a cache to evaluate them once and store them in one of several cache structures. At startup, a cache processor evaluates and caches the feature evaluations for all their possible inputs, and then the features are never evaluated again. Thus, feature caching simplifies the process of encoding new evidence by abstracting away the evaluation of the features from the optimization of their weights.

We distinguish between feature managers, or composite features, and their constituent feature functions. Each feature manager handles a class of related feature functions. At initialization, feature managers are each assigned a sequence of feature indices $\{j\}$ where each j corresponds a feature handled by the feature manager. A single function call is made to the feature manager, which then evaluates all of the features it manages.

Although a feature is always defined at edges of the model, if the previous state is not used in a particular evaluation we can optimize by evaluating the feature only at the current state. Therefore, since features can be evaluated at edges (using the "from" and "to" states), and also at nodes (using only the current state), we use the general term "potential" to refer to some state or transition between states. Thus, a feature's evaluations for all potentials at a position fully describes its output for all possible input states and transitions at that position.

The cache processor uses several different policies, and each policy corresponds to a feature behavior that is optimized in a different way. Each feature manager expressly informs the cache processor of its behavior. The different cache strategies used are as follows.

The constant strategy is used for features that do not change over different posi-

tions in the sequence. Thus, we can simply store for each feature a triplet consisting of the potential, feature index, and feature value. The state transitions feature manager, for example, uses this cache strategy. To cache P potentials, a feature function using this cache strategy needs $O(P)$ space.

The sparse strategy is used for features that do not have evaluations at every position. We build upon the constant strategy by storing the evaluation's position alongside its potential, value and feature index. To cache evaluations for P potentials at S different positions, a feature function using this strategy needs $O(SP)$ space.

In contrast, the dense strategy stores the feature's evaluation for every position in a lookup table indexed by position. To cache evaluations for P potentials in a sequence of length L, a feature function using this strategy needs $O(LP)$ space. While this is the same space complexity as the sparse strategy for L positions, the table structure allows considerably faster lookups for densely-packed evaluations, and the constant is smaller by four bytes per position.

The dense node boundary strategy is additionally parameterized with left and right pad values l and r. These instruct the cache processor not to evaluate the feature within l positions to the left of a state transition and r positions to the right. The reference predictor feature manager uses this strategy because it is used in concert with the boundary features: since we want only one emission probability per position, the reference predictor features must not give any evaluations for the boundary region handled by the boundary features. To cache evaluations for P potentials in a sequence of length L with T transitions, a feature function using this strategy needs $O(P(L - T(l + r)))$ space.

Finally, the lengths strategy stores quadruplets consisting of a length, state, feature index and value. This strategy stores a feature's evaluations for different states over all the possible lengths a segment of that state can take. This strategy is used by the explicit lengths features to model state lengths' distributions explicitly. To cache Q states of maximum duration L', each explicit lengths feature thus requires $O(QL')$ space.

1.5.2 Engineering Concerns

The problem of representing small probabilities cannot be understated. In order to find the probability of a label segment, the probabilities of that label at individual positions (which are already small fractional quantities) must be multiplied along the entire length of the segment. These prdocuts quickly go below the smallest allowed value using a standard floating point representation. This problem is solved by normalizing these values using a floating point-like data structure that uses two integers, a base and an exponent, to represent extremely small values.

Second, forward and backward feature sums must be evaluated quickly and with low overhead. For this problem we use a circular buffer that supports constant time inserts, deletes, and array accesses while limiting the amount of storage space required to hold the intermediate values of the gradient computation.

Finally, transitions must be invalidated whenever possible to prune the search space. To determine which transitions are invalid as quickly as possible, we use a 1D lookup bit-array indexed by the sequence position and all the possible state transitions. These software features keep the inference and gradient algorithms tractable even when using semi-Markov states.

1.5.3 13-State Semi-Markov Model

The semi-Markov 13-state model Interval13 reproduces a GHMM's emission, transition, and length probabilities [7], and serves as a discriminative semi-Markov platform for the incorporation of additional evidence. Conrad was first applied to gene prediction on the fungus *Cryptococcus neoformans* [7] using Interval13. In this thesis, the Interval13 model finds use for other relatively small eukaryotes as well, including *Phytophthora infestans* and *Culex pipiens*. Its states and transitions are as follows. The `intergenic` state has valid transitions to any of the six exonic states: three $\texttt{exon}_i$ on the positive strand, and three $\texttt{exonm}_i$ on the negative strand. The exonic states have transitions back to the `intergenic` state as well as to the intronic states on their respective strands: three intron states $\texttt{intron}_i$ on the positive strand, and

three $\texttt{intronm}_i$ on the negative.

Given a position t, the indices $i \in \{0, 1, 2\}$ are used to preserve the reading frame as follows. The indices of the positive strand exonic states are such that the first base in a codon satisfies $t \equiv i \mod 3$; for negative strand exonic states the last base of the codon satisfies $t \equiv i \mod 3$. Both positive and negative strand intronic states put their $3'$ exon in frame i. The model's transitions are constrained so that these requirements are always satisfied. While the model technically allows its states to be either semi-Markov or Markov (self-transitions are implicitly allowed), combining the two causes double-counting by emission probability features at state boundaries; therefore, it is only useful as a semi-Markov model.

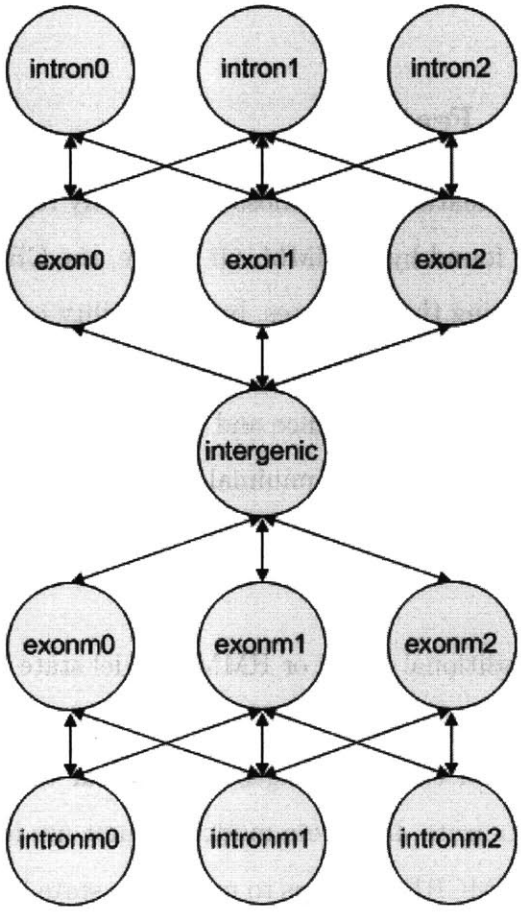

Figure 1-4: The Interval13 model for gene prediction.

Gene Constraints

In order to limit the inference search space, this module imposes constraints on the reference sequence at state transitions. On the positive strand, the Pstart constraint requires that the sequence ATG (indicating the start of an open reading frame, or ORF) is present at the start of a gene. The Pstop constraint closes the ORF, requiring the one of TAA, TAG, or TGA. Within the gene, this stop sequence appears only at the end of the gene. Finally, the Pdon and Pacc constraints require the sequences GT and AG to start and stop an intron, respectively. The constraints on the negative strand are analogous, using reverse-complementary nucleotide sequences. Thus, this feature is designed to improve both compute time and accuracy by invalidating uninteresting paths.

1.5.4 Generative Features

We define the generative features to be those that exactly reproduce the emission and transition probabilities found by an HMM. Of course, the CRF can go a step farther by discriminatively training these features, but the ability to compare the CRF to an HMM directly is valuable. The following classes of feature functions encode various information about the reference sequence and its labeling. They find widespread use in a variety of problem domains with minimal tuning.

Explicit Lengths

As described above, traditional CRFs or HMMs model state lengths implicitly using the probability of a self-transition or transition out at each node, resulting in an exponential distribution of the state lengths. It is clear that for many applications this distribution is inaccurate, but these models cannot use any other.

Like a GHMM, the SMCRF allows us to model the state lengths explicitly. Specifically, the state lengths composite feature provides several feature functions that use the state length as an explicit input. With this input, we can model arbitrary state length distributions.

We use a mixture of gamma distributions to model accurately a variety of state length distributions. A gamma distribution, parameterized by α and β, represents a sum of α exponential distributions with mean $1/\beta$ (where β is known as the rate constant). Its probability density function (PDF) is given by

$$\Pr_{\Gamma_i}(l) = l^{\alpha_i - 1} \frac{\beta_i^{\alpha_i} \exp(-\beta_i l)}{\Gamma(\alpha_i)}$$

The two-component mixture model consists of a distribution for the observations l parameterized by (α_1, β_1), a distribution for the background values parameterized by (α_2, β_2), and a bias p toward distribution 1. Together, let

$$\theta = (\alpha_1, \beta_1, \alpha_2, \beta_2, p).$$

It is trained using the usual expectation maximization algorithm for mixture models, finding

$$\text{argmax}_\theta \Pr_{Mix}(l|\theta) = \frac{p \Pr_{\Gamma_1}(l)}{p \Pr_{\Gamma_1}(l) + (1-p) \Pr_{\Gamma_2}(l)}$$

Then, the feature at l evaluates to the log probability the length l comes from the distribution of observations:

$$f_i = \log \left(p \Pr_{\Gamma_1}(l) + (1-p) \Pr_{\Gamma_2}(l) \right).$$

Transition Features

This feature models the probability distribution of per-gene exon count. First, we compute the probability of transitioning from exon to intron states by finding the average number of introns per gene,

$$\Pr(\text{intron} \mid \text{leaving exon}) = 1/ \sum_{g \in \text{genes}} |g_{\text{introns}}|/|\text{genes}|.$$

Next, we compute the probability that a given exon is the final exon in the gene,

which is the same as the probability of a transition from an exon to intergenic space:

$$\Pr(\text{intergenic} \mid \text{leaving exon}) = 1/\sum_{g \in \text{genes}} |g_{\text{exons}}|/|\text{genes}|.$$

So on a positive-strand donor transition or negative-strand acceptor transition, the feature evaluates to log Pr(intron | leaving exon). On a positive-strand start or negative-strand stop, the feature evaluates to log Pr(intergenic | leaving exon). Note that these probabilities sum to 1 because they describe all the possibilities given that the model is exiting the exon state.

Reference Features

The reference features evaluate the probability of a particular nucleotide in the observed sequence appearing in a certain state s given the k previous nucleotides (where k is known as the lookback). In other words, the reference predictor is a k^{th} order Markov model of nucleotides. It is the CRF equivalent of an HMM's emission probabilities. Conrad uses a lookback of three bases for gene prediction.

Boundary Features

The nucleotide context of start, stop, and splice sites is often well-conserved. The boundary features learn a position weight matrix (PWM) describing the probability of finding each of the four bases at each position in these contexts. We have eight boundary features total: one feature each of the start and stop codons per strand, and one feature for each of the donor and acceptor splice sites per strand. At the appropriate state boundary, the feature outputs the evaluation of the PWM for the context surround the transition.

Optionally, this feature can be parameterized with a threshold c, which is the fraction of valid splice sites to invalidate. Let T be the number of scored transitions. All transitions that score below the cT^{th} lowest score will be invalidated. This option is designed to eliminate less interesting search paths.

Phylogenetic Features

The phylogenetic features evaluate to the log probability of a multiple alignment column given a nucleotide in the reference sequence using a probabilistic model of nucleotide evolution [26]. The Kimura-80 model is used by default, but others can be substituted. The feature requires a multiple alignment and a phylogenetic tree, with branch lengths, of all the species. Felsenstein's algorithm (a description of which can be found in [27]) is used to train the overall length of the tree, keeping relative branch lengths the same. Notably, the corresponding I/O components support the popular Fasta multiple alignment and Newick tree formats.

1.5.5 Discriminative Features

SMCRFs allow us to encode arbitrary evidence as feature functions in order to extract additional signal and so make better-informed predictions. Incorporating additional evidence into a GHMM requires altering the existing emission and transition probability functions modeling the joint probability $\Pr(X, Y)$ in order to find the conditional probability $\Pr(Y|X)$. But a SMCRF models this conditional probability directly. A feature function can be a valid probability distribution, but this feature design is not required. A discriminative feature function is an arbitrary real-valued function that provides evidence that the solver can use to discriminate among possible paths.

In order to improve the model's performance, each feature must extract some correlation of the observed sequence with the hidden sequence that is not extracted by other features. Increased correlation leads directly to increased performance. This requirement is often quite difficult to satisfy in practice, as is evident in the poor performance of *ab initio* HMM gene finders compared to those that use extrinsic evidence. Thus, the most effective use of discriminative features is to incorporate extrinsic or non-probabilistic evidence, such as a multiple sequence alignment or experimental data, which frequently contains information that cannot be found by looking only at probabilistic models of the nucleotide sequence.

Gaps

For each aligned species S, this composite feature manages two indicator feature functions for the exon regions, two for introns, and two for intergenic regions . Each pair consists of one indicator function that captures a frameshifting gap with a boundary at the current position, and one that captures a non-frameshifting gap with a boundary at the current position.

Gap features capture gaps in the genomic alignment of the query species with an otherwise similar species. Gaps that would shift the reading frame (of length 1 or 2 (mod 3)) rarely occur conserved exons [15], but this information is not used by the probabilistic phylogenetic features as their model interprets gaps as missing information [26]. Thus, this feature captures effects not used by the phylogenetic features.

Footprints

For each informant species S, we have a footprint feature each for exon, intron, and intergenic states. Each feature is an indicator function that captures the presence of an alignment of S with the query species at each position. Exonic regions are more likely to be aligned than nonexonic regions because they are less likely to have changed over the evolution of the species and also because their conservation makes them easier to align with MSA tools. Thus, this feature as well captures effects not used by the phylogenetic features.

1.5.6 I/O

Conrad's input facilities permit the use of a variety of file formats to inform the feature functions. The I/O abstraction allows us to convert domain-specific file formats, such as Fasta and GTF, to a common internal sequence representation suitable for CRF training and inference. We use two types of internal sequences: an input sequence contains a sequence representing observations, while a training sequence contains the observations' correct labeling as well. As such, training sequences are used for

training, and input sequences for inference.

Chapter 2

Extensions to Conrad

Several extensions to Conrad were developed in this thesis. First, the hybrid 29-state model reduces the runtime complexity of the training and inference algorithms. Second, an N-best Viterbi algorithm aids in feature function tuning. Third, combination features allow existing features to be combined to increase specificity. Next, novel feature functions increase overall gene prediction accuracy. Finally, parallelization and accurate starting weights reduce the time per train/test cycle.

2.1 Hybrid 29-State Model

Predicting genes in mammalian genomes is a much different task than for many other organisms. Mammalian genomes, especially the human's [8], contain many short exons and long introns within a gene as well as vast intergenic regions. Significant changes were made to the model developed for small eukaryotes to maximize prediction accuracy.

The first problem the SMCRF models had to overcome in order to predict genes in mammals is that of scaling the solutions to the training and inference problems for larger genomes. Both the gradient and Viterbi algorithms are linear in the size of the data set L and length of the longest allowed segment L', which may be acceptable for smaller genomes, but is intractable for 200-megabase mammalian chromosomes with 10-kilobase introns.

To maintain tractability, we use a hybrid model that incorporates Markov self-transitioning states for the long intergenic and intronic sequences, and semi-Markov state segments for the relatively short exons. These state types give the training and inference algorithms runtime complexity linear in LL' in the short exonic regions but only L in the significantly longer intronic and intergenic regions.

2.1.1 Preventing Emission Collisions

In order to combine these two types of states, we considered the behavior of the features at transitions between two states of different types. The boundary features as well as the reference features find probabilities for the emission of a nucleotide given a position and state. To maintain equivalence with the GHMM, it is ensured during feature evaluation that only one of these emission probability features is evaluated for any given position. The Interval13 model uses boundary pads within which the reference features are inactive; by construction the boundary features will be active exactly within these pads.

However, when the introns and intergenic regions are represented by Markov states, the model has no notion of these states' durations. As a result, the solver cannot keep track of the number of positions we are from a boundary in order to check whether the current position is within a pad. If Markov and semi-Markov states were to be combined within the Interval13 model, then emission probabilities would be double-counted at state transitions. This double-counting misrepresents the desired model, and strays from the GHMM equivalence that we use for benchmarking. This is undesirable because the model must extract as much signal as possible from these core features to be competitive with *ab initio* gene predictors. Furthermore, it is helpful to be able to recreate the GHMM to demonstrate the benefits of the discriminatively-trained SMCRF versus the generative GHMM,

The Interval29 model was developed to eliminate the double-counting at boundaries between Markov and semi-Markov states. This model incorporates several boundary states of length two which act as pads for the emissions features. They are intergenic-exonic (ig-e), exonic-intergenic (e-ig), three exonic-intronic (e-i_j's), and

three intronic-exonic (i-e_j's). These boundary states occur between Markov and semi-Markov states where the boundary features are active, but during them the reference features have no output. Essentially, we get higher-level behavior using additional states. Figure 2-1 shows how the Interval29 and Interval13 state models map to donor and acceptor splice sites GT and AG, respectively.

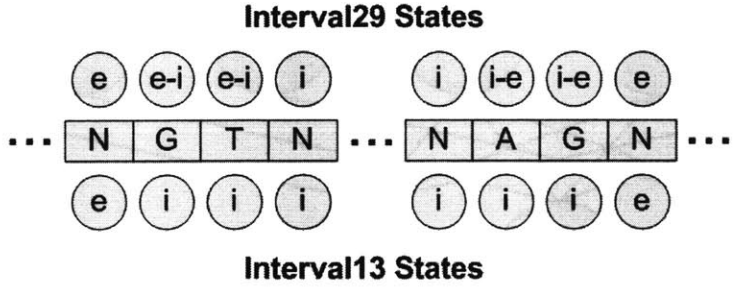

Figure 2-1: Mapping from reference nucleotide sequence to Interval29 and Interval13 state models at donor and acceptor splice sites GT and AG. (N is the IUPAC code for any base.)

While not shown in figure, the exon-to-intron and intron-to-exon boundary states must keep track of the reading frame. By encoding the reading frame in the state, the model mimics behavior that could have been achieved by increasing its order, which would severely hinder performance. Figure 2-2 shows the model's full positive strand. This state model is crucial for inference and gradient tractability in large genomes.

2.1.2 Demonstrating Equivalence to the 13-State Model

We have two strong high-level tests to confirm equivalence of the Interval29 and Interval13 models. For the first test, we configure an Interval13 model (with feature sums F_j) to use all semi-Markov states. We find the sum of its reference features F_{Ref} and boundary features F_{Bdary}. The test asserts that an Interval29 model (with feature sums F'_j) configured with Markov intronic and intergenic states and semi-Markov exonic states has $F_{\text{Ref}} = F'_{\text{Ref}}$ and $F_{\text{Bdary}} = F'_{\text{Bdary}}$. To contrast, note that if we configure another Interval13 model (with feature sums F''_j) with Markov intronic and intergenic states and semi-Markov exonic states, we find that $F_{\text{Ref}} \neq F''_{\text{Ref}}$ and $F_{\text{Bdary}} \neq F''_{\text{Bdary}}$.

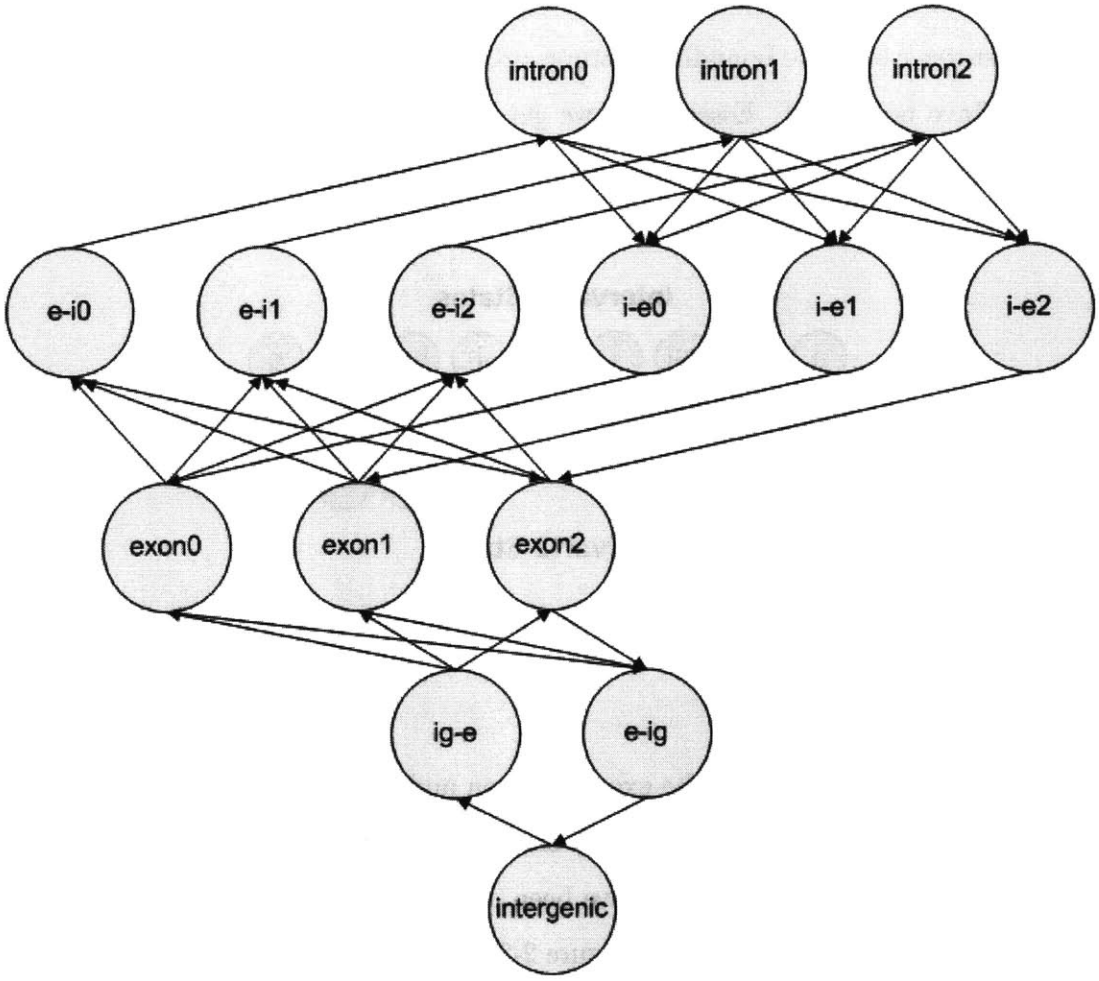

Figure 2-2: The positive strand of the Interval29 model for gene prediction.

For the second test, we set an Interval13 model (with feature sums F_i) with state lengths features to model intronic and intergenic lengths explicitly as exponential distributions, and exonic lengths as a mixture of gammas. We configure an Interval29 model (with feature sums F_i') with Markov intronic and intergenic states and semi-Markov exonic states with lengths modeled by a mixture of gammas. Note that Interval29's explicit length features cannot model durations for intronic or intergenic states because they each necessarily have unit length. However, simply by outputting self- and exit-transition probabilities, Interval29's implicit length features model intronic and intergenic lengths as exponential distributions. Found within the package calhoun.analysis.crf.test, Interval29BaselineTest asserts

(1) $F_{\text{ExpLens-exonic}} = F'_{\text{ExpLens-exonic}}$,

(2) $F_{\text{ExpLens-intronic}} = F'_{\text{ImpLens-intronic}}$, and

(3) $F_{\text{ExpLens-intergenic}} = F'_{\text{ImpLens-intergenic}}$.

Thus, Interval29 can find state duration probabilities corresponding to an exponential distribution but at the same time eliminates linear dependence on intronic and intergenic lengths.

2.2 N-Best Viterbi

Often, it is useful to examine not only the single most-likely path, but all of the N most-likely paths. Such examination is useful for accuracy tuning because it allows us to see how close the correct labeling was to the best, and from there to adjust the feature functions to improve the likelihood of the correct labeling.

Existing N-best algorithms [4] [1] were developed in the context of real-time speech recognition (and, in fact, [4] was implemented for CRF inference in [6]). They are designed to fit into heuristic-based "beam" searches, wherein the algorithm throws away paths that score poorly in a local region according to a set of heuristics. In contrast to speech recognition, bioinformatics applications' state models are much more tightly constrained, eliminating the need (at present time) for a beam search.

To meet the need for a simpler N-best algorithm that works well for few paths and small N, we developed our own based on that for a beam search but that returns optimal solutions. We present a novel N-best Viterbi algorithm that fits bioinformatics more easily than existing N-best algorithms. In contrast to existing approaches that find the N-best paths in backward passes or use a "grow and prune" strategy [4], we keep track of the N-best paths during the forward pass.

As a result, the modifications to the traditional Viterbi algorithm are minor. Basically, for every position $t-1$, we know for each state y' at $t-1$ the N-best paths from position 1. To extend to t, we find the N-best paths to each state y at position t by considering all the $|S|N$ paths (where $|S|$ is the number of states) to the possible previous states y' in combination with edges (y', y) to the current state.

In more detail, the algorithm works as follows. The base case, $t = 1$, is trivial. By the invariant, at step t we have the N-best partial paths to each previous state y'. As shown in 2-3, the traditional Viterbi algorithm uses the fact that the 1st best path to each state y always comes from a previous 1st best path to a y'.

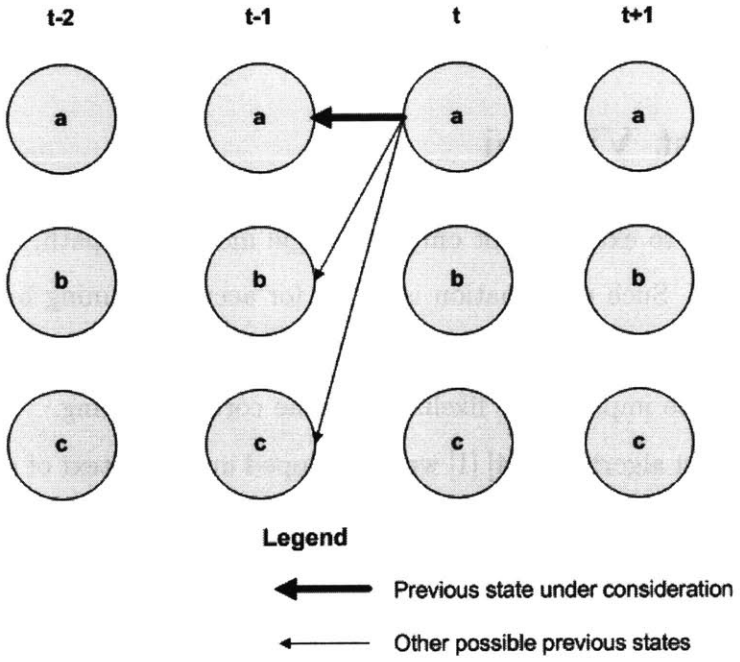

Figure 2-3: The Viterbi algorithm considers each possible previous state in turn.

Now we show the extension to N-best. To find the N-best paths to y, we need to look not at each possible previous state y', but at each of the N-best paths to each y'. Namely, we define the set $\{P_{y'}\}$ to be the set of size N of the N-best partial paths to state y' at position $t-1$–ultimately, the algorithm returns $\{P_y\}$ for position L. We define $\{Q_y\}$ to be the union over y' of all the sets $\{P_{y'}\}$, with the edge (y, y'):

$$\{Q_y\} = \bigcup_{y'} \{P_{y'}\} + (y', y).$$

Then, the N-best partial paths to a state y are the N-best paths of length t, taken from the set $\{Q_y\}$. Thus, to find $\{P_y\}$, we simply take the N-best scoring paths from $\{Q_y\}$. Let the function $\max_v^i f(v)$ be the value of $f(v)$ of rank i over the different

inputs v. Then the recurrence for the value of the i^{th}-best Viterbi path is

$$V_i(t,y) = \begin{cases} \max^{i}_{y',j=1\ldots N} V_j(t-1,y') + \lambda \cdot F(y,y',X,t), & \text{if } t > 0 \\ 0, & \text{if } t = 0 \\ -\infty, & \text{if } t < 0. \end{cases}$$

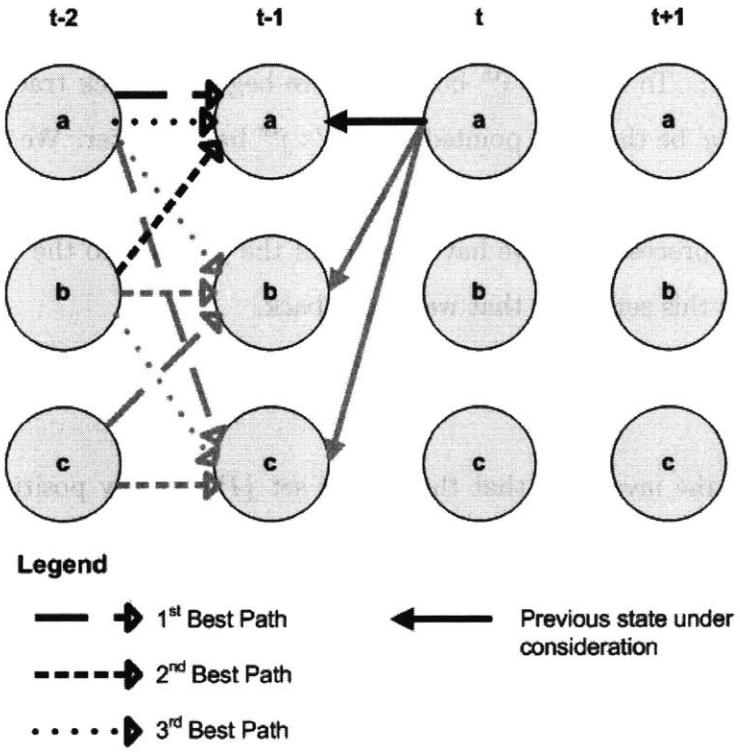

Figure 2-4: N-Best Viterbi considers all paths to all possible previous states simultaneously.

For example, in figure 2-4, the N-best paths to the state a_t come from the N-best paths to a_{t-1}, b_{t-1}, and c_{t-1}. Where the Viterbi algorithm considers only each previous state, the N-best Viterbi algorithm considers all N paths to all previous states together. The a_{t-1} state is highlighted for clarity, but the algorithm considers all the paths to all previous states together.

The semi-Markov extension is identical to that for the traditional Viterbi algo-

rithm. We expand the search over possible lengths $l = 1...L'$:

$$V_i(t,y) = \begin{cases} \max^i_{y',j=1...N,t=1...L} V_j(u-l,y') + \lambda \cdot F(y,y',X,u-l,u), & \text{if } t > 0 \\ 0, & \text{if } t = 0 \\ -\infty, & \text{if } t < 0. \end{cases}$$

The backward trace is somewhat more sophisticated than in the traditional Viterbi algorithm. First, we define an **end** state that has edges into it from all the states at the last position. To find the i^{th} best path, we begin the back trace with $y = \textbf{end}$ and $j = i$. Let y' be the state pointed to by y's j^{th} back pointer. We compute j', the number of paths of rank $1...j-1$ that end in state y'. Then we set $y = y'$ and $j = j'$ and repeat the process until we have traced all the y's back to the start state. The i^{th} best path is this set of y's that we traced back.

Correctness

We now prove the invariant, that the correct set $\{P_y\}$ at any position t is a subset of $\{Q_y\}$, and so can be found from the N-best partial paths $\{P_{y'}\}$ to each previous state y'. Let $P_{y'}$ a the partial path that coincides with a P_y until its terminal state y' at position $t-1$. If P_y is among the N-best to y, then it must be the case that P'_y is among the N-best to y'; otherwise, we would have at least N better paths to y that traverse y', so P_y could not be among of them. Thus, each of the N-best paths P_y is a strict (trivial) super-path of one of the N-best paths $P_{y'}$. Each trivial super-path is included in $\{Q_y\}$, so we have shown that the N-best paths $\{P_y\}$ must be a subset of $\{Q_y\}$.

Runtime Analysis

The runtime analysis can be extended from the traditional Viterbi algorithm. The key to maintaining a runtime linear in N is to store each $\{Q_y\}$ and $\{P_y\}$ not as sets, but as sorted lists Q_{y_i} and P_{y_i}. Let S be the set of all states, and let L be the length of the sequence. The traditional Viterbi algorithm has a time complexity of

$O(L|S|^2)$. In this algorithm, for each y' that we want to check, we cannot just look at the value of its best partial path and the edge to y as Viterbi does. Instead, we must look at the probabilities of all the N-best partial paths to y', combined with the edge to y. Assembling the sorted list Q_{y_i} from the individual sorted N-best lists $P_{y_i'}$ requires $O(N|S|)$ time, where the step takes just $O(|S|)$ time in the traditional Viterbi algorithm. Then extracting the N-best from Q_{y_i} can be done in constant time. So the new runtime complexity has an additional factor linear in N, giving an overall runtime of $O(NL|S|^2)$. In the semi-Markov case, we must search over possible segment lengths as well, so the algorithm has a runtime complexity of $O(L'NL|S|^2)$.

Discussion

To store the N-best partial paths, we keep N back pointers from each state at each position. The key insight is that more than one of the N back pointers from a state can actually point to the same previous state. Although the paths can be identical locally, over the entire sequence they will be different.

This runtime complexity is better than the worst case for [4], but worse than their empirically-found runtime complexity. But our existing Viterbi algorithm does not use a beam search in the forward pass anyway, so it is useful that the new N-best algorithm does not need one either. Thus, this algorithm is a better fit for our applications. However, this method was implemented but never used in this thesis.

2.3 Combining Signals To Increase Specificity

Combinatorial features multiply two or more traditional features to find stronger correlation between the features' evaluations and the hidden sequence. They are useful because the optimizer considers each feature sum independently, but this assumption can be limiting. This class of features was implemented but not used. As a simple

example, consider the hidden sequence

$$y_t = \begin{cases} 0 & , 0 < t \leq a \\ 1 & , a < t \leq b \\ 0 & , b < t \leq c \end{cases}$$

Now suppose we have two features, f_1 and f_2, which evaluate to

$$f_1 = \begin{cases} 1 & , 0 < t \leq a \\ 1 & , a < t \leq b \\ 0 & , b < t \leq c \end{cases}$$

and

$$f_2 = \begin{cases} 0 & , 0 < t \leq a \\ 1 & , a < t \leq b \\ 1 & , b < t \leq c \end{cases}$$

Clearly, neither feature is particularly well-correlated with the hidden sequence. Both are sensitive, but neither is specific. But their product is both sensitive and specific:

$$f_1 f_2 = \begin{cases} 0 & , 0 < t \leq a \\ 1 & , a < t \leq b \\ 0 & , b < t \leq c \end{cases}$$

The combination features that we support are each the product of k features where $2 \leq k \leq |f|$. For each product size k, there are $|f|$ choose k possible combination features. These features were implemented but never used in this thesis.

2.4 Generative Features: Implicit Lengths

The implicit lengths features are the CRF-equivalent of the transition probabilities inherently modeled by an HMM. While explicit state length models yield more accurate predictions, they come at significant computational cost. For long segments the model must assume Markov independence among its positions even though doing so prevents the use of explicit state lengths. The implicit length features measure state length implicitly using the probability of various state transitions.

In particular, this feature finds probabilities of self-transitions and exit-transitions for intronic and intergenic states. The probability of a self-transition $s \rightarrow s$ is the ratio of the count of transitions $s \rightarrow s$ to the count of all transitions $\forall s', s \rightarrow s'$. Similarly, the probability of an exit-transition from $s \rightarrow s', s \neq s'$, is the ratio of the count of transitions $\forall s'$ s.t. $s \neq s', s \rightarrow s'$ to the count of all transitions from $\forall s', s \rightarrow s'$. By giving these probabilities at the corresponding transitions, the features effectively create implicit exponential distributions over the state lengths.

2.5 Discriminative Features

There are a number of additional signals that we can consider to improve *ab initio* gene prediction performance, as well as performance with extrinsic evidence. We use discriminative features to exploit them.

2.5.1 Sequence Edges

As the continuity of a genome assembly increases, so does the likelihood that each sequence begins and ends in an intergenic state. So for many of the genomes we study, it would make little sense for the gene predictor to try to start in a genic state, but by default Conrad has an equal probability of starting in any state. Therefore, we would like to bias the beginning and end of each sequence to be intergenic. The sequence edges feature, parameterized by k, returns 1 for each position in $[1, k]$ and $[L - k, L]$ whose state is intergenic, and 0 otherwise. k is chosen by taking the mean

of the lengths of the intergenic tracts at the beginning and end of each sequence, then dividing by two to be more conservative.

2.5.2 GC Content

GC content is defined to be the average number of G or C bases per position over a sequence or region. It has been observed that GC content tends to be higher in exonic regions than surrounding intergenic and intronic regions [33]. For example, the following figure shows the GC content of transcript RP11-328M4.1-003 (Forkhead box P4, isoform 3) from ENCODE sequence ENr334 averaged over each exonic, intronic, and intergenic region, with exonic regions shaded.

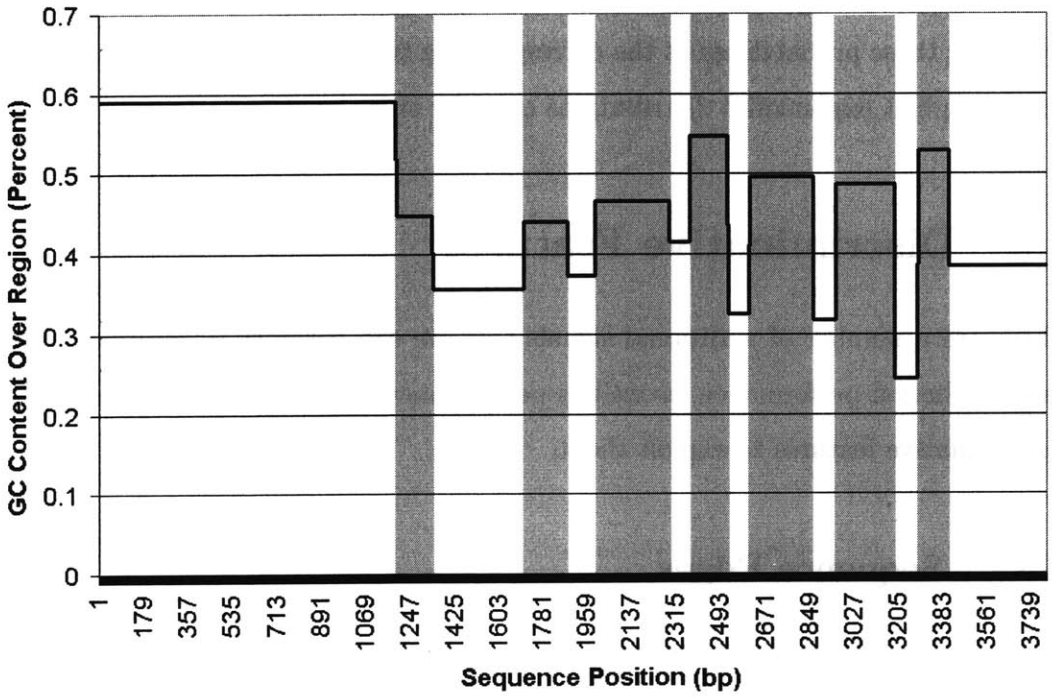

Figure 2-5: GC content averaged over intergenic, exonic, and intronic regions in a region of an ENCODE sequence. Shaded regions are exonic.

Figure 2-5 shows that we can use GC content as an indicator of exonic regions. The simplest approach to encoding this signal in a feature function is to average GC content over a relatively small window (say, 200 bases, the average exon length). But

there are a number of problems with this approach. First, while this sequence does not show it, an exon's GC content is typically only higher than that of its neighboring introns, not all of the introns across every sequence. Second, a windowed average will peak at the middle of the exon, which is not where we want it; we would prefer the feature's output to correlate with the start and end of exons so we can more accurately identify the entire exon.

To resolve these problems, we take the ratio of two windowed averages of GC content, finding the GC content's rate of change. Specifically, we average the GC content over a leading window of 200 bases (approximately the length of an exon) and the average over a trailing window of 1000 bases (approximately the length of an intron), and divide the two. This ratio effectively gives us the rate of change of GC content, which correlates well with the start of an exon. A similar argument shows that we can find exon stops by taking the ratio of a 200-base tailing window and a 1000-base leading window. The GC content features output these ratios. We have one feature for nonexonic to exonic transitions, and another for the exonic to nonexonic transitions.

Figure 2-6 shows that in this sequence the rate of change of GC content is generally well-correlated with exon starts. The GC content peak in the leading intergenic region is worth noting as well, as it may indicate a number of genetic features, including promoters or an exon from an alternative transcript or nearby gene.

2.5.3 Branch Sites

A number of signals are necessary for cellular machinery to recognize and to splice introns from pre-mRNA. An individual intron is spliced between its 5′ donor site and 3′ acceptor site by a spliceosome protein/RNA complex, which consists of several small nuclear ribonucleic proteins (snRNPs). The spliceosome's snRNPs recognize three major pre-mRNA sequence components: the U1 snRNP binds to the 5′ donor site, U2 binds to the branch site, and U5 binds to the 3′ acceptor site.

This feature detects branch sites, which are a major mRNA sequence component (along with the 5′ and 3′ splice junctions) needed for splicing to occur [13]. (Recall that

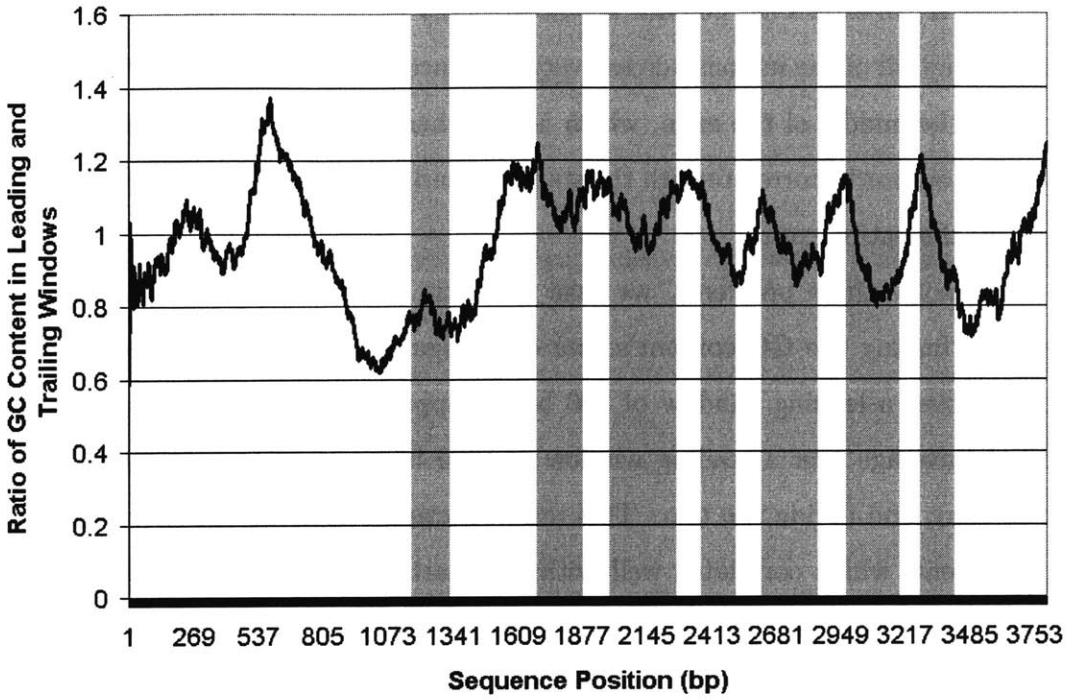

Figure 2-6: Ratio of GC content in leading $[0, 200]$ window over GC content in trailing $[-1000, -1]$ window in a region of an ENCODE sequence. Shaded regions are exonic.

the 5′ and 3′ splice junction consensus sequences are enforced by the gene constraints, and their contexts are scored with the boundary features.)

Branch sites, which have the consensus sequence CU{A|G}AC in vertebrates [11], generally occur 18-40 bases upstream of the acceptor site. The branch site feature scores the region 18-40 bases upstream of the acceptor site by summing PWM evaluations of the branch site consensus sequence along this segment.

Figure 2-7 shows evaluations of a branch site PWM summed over a trailing window in the same sequence of ENr334. If the introns had well-conserved branch sites, we would see that the peaks were correlated with exon starts. However, instead we see that branch sites are almost entirely not present in this ENCODE region. This is an expected result, as it has been shown that vertebrate branch sites are generally poorly-conserved [11]. Still, as the later results show, this feature can improve gene prediction accuracy.

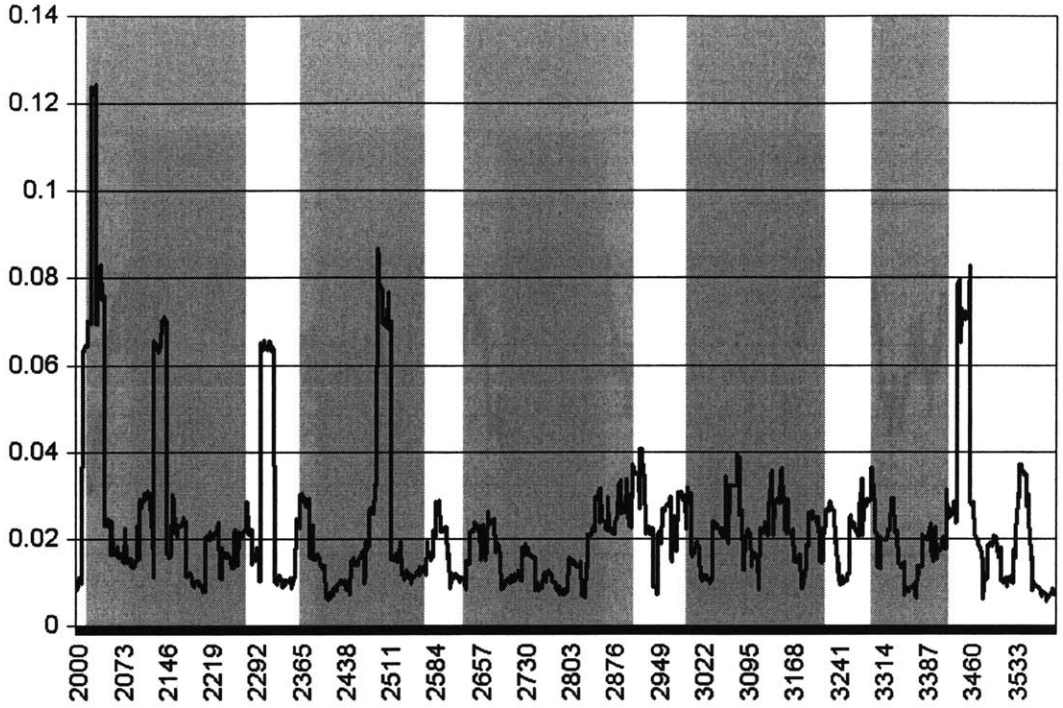

Figure 2-7: Sum over reverse window $[-40, -18]$ of branch site PWM evaluations in a region of an ENCODE sequence. Shaded regions are exonic. The peaks do not correspond with anything useful.

2.5.4 Polypyrimidine Tracts

As it has been shown that branch sites in vertebrates are not always well-conserved, we must consider other splicing signals as well. In cases where the branch site is degenerate or occurs more than 40 bases upstream of the 3′ splice junction, a polypyrimidine (poly(Y), where Y is the IUPAC ambiguity code for a pyrimidine) tract is often located in its place [5]. A poly(Y) tract is an mRNA sequence of approximately 15-20 bases consisting of a large proportion of pyrimidines, cytosine and uracil, located 5-40 bases upstream of the acceptor site.

There are two poly(Y) tract features. The first measures pyrimidine content in the region 5-40 bases upstream of the acceptor site on the positive strand, and the second measures purine content in the reverse region on the negative strand. Both features measure nucleotide content in a manner similar to that in which GC content is measured. It uses a sliding window that counts the occurrences of C and T bases 5-40

bases upstream, then divides by the number of occurrences for 35 bases downstream.

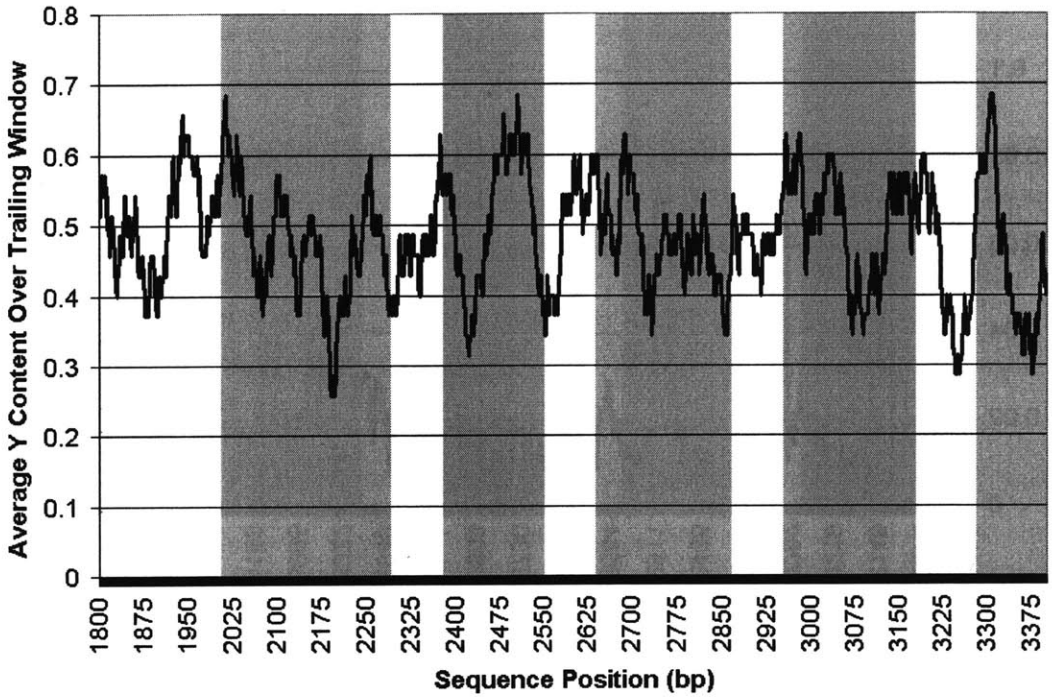

Figure 2-8: Average pyrimidine content over trailing window $[-40, 5]$. Shaded regions are exonic.

In figure 2-8, we expect peaks to correlate with exon starts becasue the feature is averaging over a trailing window. The figure corroborates previous results [5] [20] that show that poly(Y) tracts are often found in place of branch sites in vertebrates. Although there are many false positives, we do generally see that exon starts occur at higher values.

2.5.5 Information Content

It has been shown that regions of high information content tend to be associated with exons [9]. We measure the information content of a region using Fisher's measure, $I = 1/\operatorname{Var}(n_i)$, where n_i is a random variable indicating the proportion of base $i \in \{A, C, G, T\}$. We approximate the variance due to a sliding window of 200 bases

with

$$\mathrm{Var}(n_i) \approx \sum_{i \in \{A,C,G,T\}} (n_i - E[n_i])^2,$$

where $E[n_i]$ is found for each sequence.

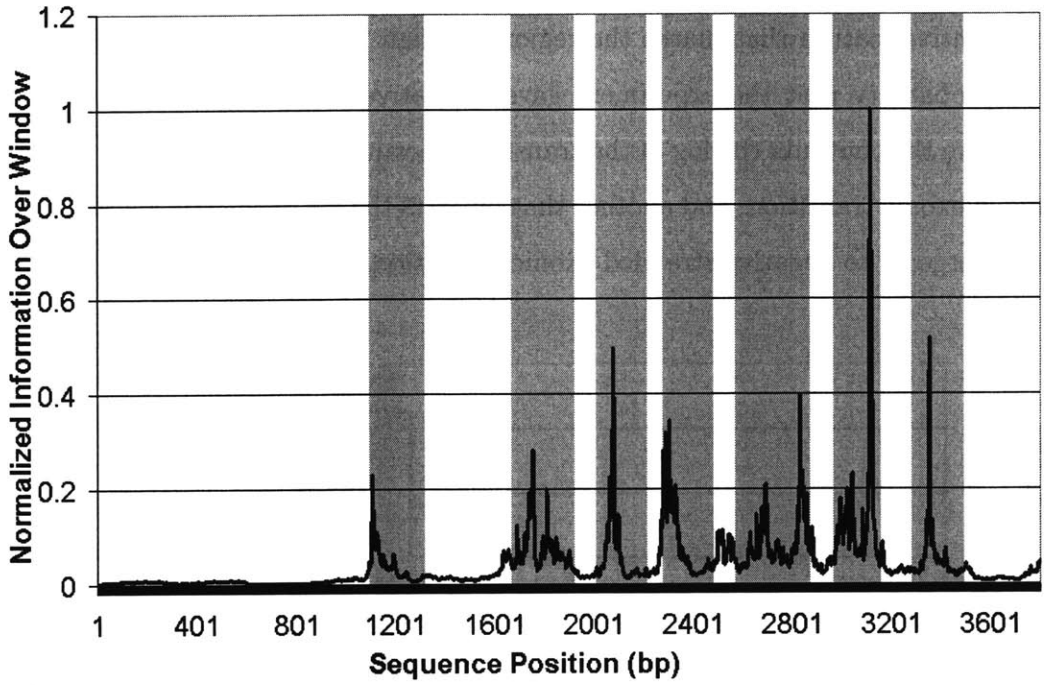

Figure 2-9: Normalized sum of information content over 200bp window centered at each position in an ENCODE transcript. Shaded regions are exonic.

Since we are summing information content over a window centered at each position, we expect values to be higher for exonic states, and indeed figure 2-9 shows that this is true for the ENCODE sequence. Note that not all sequences behave as well as this one.

2.5.6 DNAase Hypersensitive Sites

DNAase hypersensitive sites are short regions of DNA found upstream of an expressed gene that are sensitive to cleavage by DNAase enzymes. This sensitivity is due to the arrangement of the structural elements of chromatin, which also enables binding of transcription factors in the region. The data consists of ranges of sequence positions

that the experiments showed to be sensitive to DNAase cleavage and the corresponding p-value for each experiment. These sites tend to occur upstream of promoters, which are upstream of the start of transcription, so to prepare this data for Conrad we translated the data downstream 2,000 bases.

By giving the p-values of the DNAse hypersensitive site experiments, we have a discriminative feature that marks the regions of high DNAase sensitivity according to the probability that the experiment gave a positive result. Specifically, we have one feature that outputs the log of the translated p-value at an intergenic to positive-stranded exonic transition, and another that outputs the log of the translated p-value at an intergenic to negative-stranded exonic transition.

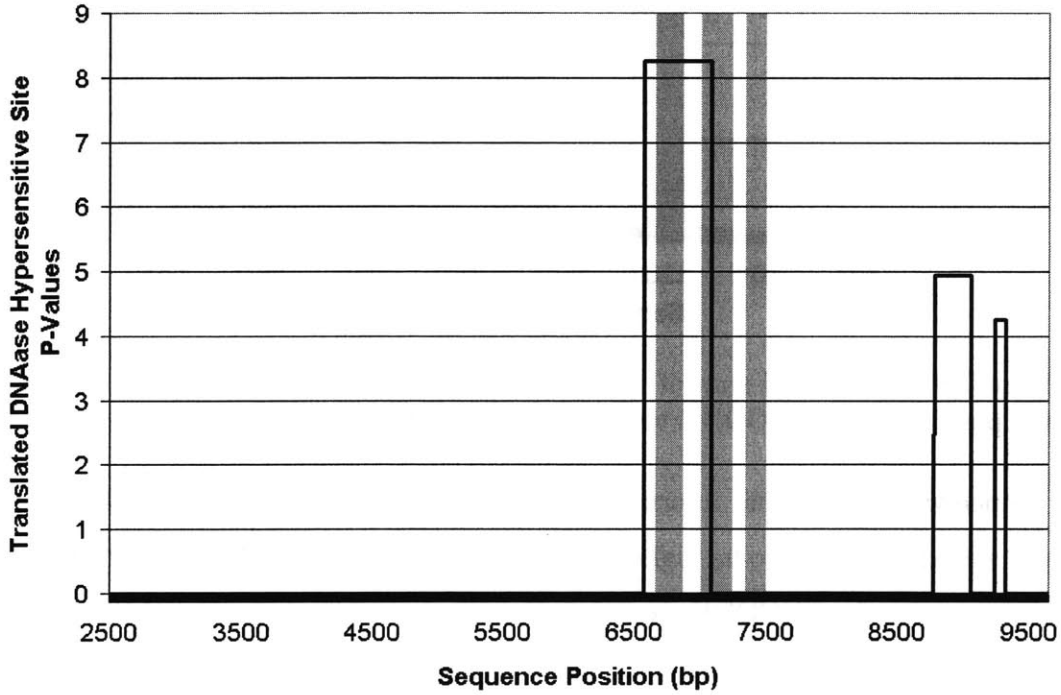

Figure 2-10: Translated DNAase hypersensitive site p-values. Shaded regions are exonic.

Figure 2-10 shows that this transcript's start codon occurs in the middle of a DNAase hypersensitive site. The second and third DNAase hypersensitive sites are either false positives, or may be related to another transcript. This feature is a very reliable indicator of a nearby gene start, and so is especially useful for prediction in

mammalian genomes with vast intergenic spaces.

2.6 Rapid Iteration

Conrad encourages rapid iteration because evidence is so easy to encode. However, some useful training and testing sets are expansive, requiring hundreds of CPU hours to train the model and test it. The following methods were developed to test improvements to Conrad much more quickly.

2.6.1 Parallelization

As is typical in computational biology [17], the algorithms used by Conrad are very parallelizable. Running different sets of sequences on different machines requires no communication among execution paths in either the gradient or inference algorithm. Smaller data sets reduce compute time directly because there is less data to process as shown in figure 2-11. Since the algorithms are linear in the length of the sequence, parallelization provides linear speedup. In some cases, parallelization can make problems tractable that were not before because memory requirements are split among the slaves.

Both training and inference parallelization were used heavily in developing the gene prediction models. In particular, for prediction of human genes the gradient descent algorithm is unmanageable on a single commodity machine when run on all 1,000 known coding genes from GenBank in the ENCODE region, but can be completed in a few hours when run in parallel on several machines. Since it was shown in [12] that 800 training genes are needed for a well-generalized model, parallelization is crucial to meet the end goal of accurately predicting new genes.

Training

To parallelize training onto n machines (plus one master host), the sequences are divided into n sets where the sum of positions of the sequences in each set are roughly equal. Load balancing is performed using the obvious greedy algorithm: first, we sort

the sequences from longest to shortest; then, we assign to the least-loaded machine each sequence in this decreasing order.

For each iteration, the master host sends to the n slaves a set of weights λ for which they calculate the gradient for their assigned sequences. After the slave finds the gradient, it simply reports it to the master, which aggregates all of the results. The communication overhead of the parallelized training method is minimal, as each server must send to the master only a few bytes per feature per iteration.

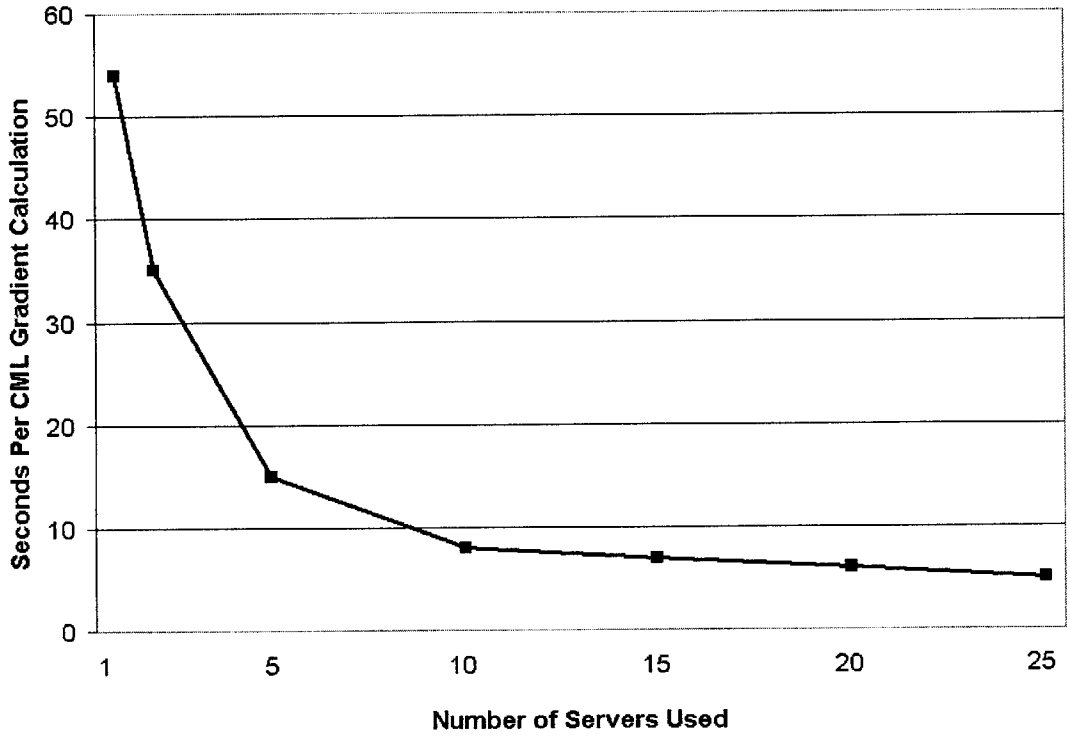

Figure 2-11: Time per training iteration for 100 human genes.

Since we expect the slaves to find the gradient using the same function as in the serial case, we need to renormalize the output. Let L_i be the total length of the sequences considered by slave i, and so the total length of sequences is $\sum_i L_i$. The master aggregates these gradients, renormalizing the values for slave i by multiplying by the sum of the lengths of the sequences used by slave i and then dividing again

by the sum of the lengths of all sequences:

$$G_j = \sum_i G_{j_i} \frac{L_i}{\sum_k L_k}.$$

The master then proceeds with L-BFGS in the usual manner, obtaining from L-BFGS a new set of weights to send to the slaves next, and so on, until convergence is reached. Since the runtime of L-BFGS is insignificant compared to that of the gradient computation, training parallelizes very well.

Inference

The algorithm is identical to parallel training, except the slaves send back only a dense representation of the most likely hidden states for the sequences that they were each assigned (and there is no back-and-forth iteration). The communication overhead in this case is minimal as well, as the master sends only the indices of the sequences each host will analyze, and the servers send only the exon starts and stops for each sequence back to the master.

2.6.2 Finding Accurate Starting Points

When starting the training from $\forall j, \lambda_j = 1.0$, the number of iterations required for L-BFGS convergence rapidly increases with the number of features. This starting point can be improved for the CML gradient.

If a model's core features do not often change, but other features are added or changed, then we can reuse the old feature weights and expect that the optimizer will converge more quickly. A simple script was developed to generate new Conrad configurations based on a template, the previous optimal weights, and a new set of features. In practice, this technique reduces the number of iterations required for convergence by 80-90%, thus reducing the overall runtime of the training algorithm by the same amount. It is worth noting that the optimizer will always converge to the same set of weights regardless of its starting points because the CML gradient is globally convex.

Chapter 3

Results

We trained and tested various models, features, and gradients on the organisms *Phytophthora infestans*, *Culex pipiens*, and the ENCODE regions of *Homo sapiens*. Our results show with success that Conrad can be extended to use a different state model (Interval29) and many new feature functions incorporating diverse genic evidence.

3.0.3 Data Format

All models use a naming scheme similar to [7]. The format is Conrad[number of states]([gradient])[discriminative features]-[number aligned species], e.g. Conrad29(MEA)SCBPI-2. Following are possibilities for each field and their explanations.

Number of states	
13	The Interval13 model was used.
29	The Interval29 model was used.

Gradient	
CML	Weights were trained using the CML gradient.
MEA	Weights were seeded using CML, then improved upon using MEA.
Gen	Weights were fixed at 1.0.

Discriminative features	
G	Gap features
F	Footprint features
S	Sequence edge features
C	GC content features
B	Branch site features
P	Polypyrimidine tract features
I	Information content feature
D	DNAase hypersensitive site features

Number of aligned species	
2	One informant species
3	Two informant species

Training set size refers to the number of transcripts used for training. It is assumed that testing is performed on all other transcripts available in the curated reference set. All transcripts in the curated set are flanked by 200bp of intergenic space on either end. Nucleotide (NT) and exon (Ex) sensitivity (Sn) and specificity (Sp) are as defined in [12]. Correct transcripts (CT) is the fraction of complete transcripts that were predicted correctly. Finally, since all scores are fractions, they are shown as percentages for easier viewing.

3.1 Gene Prediction in Small Eukaryotes

3.1.1 *Phytophthora infestans*

P. infestans is an oomycete (a fungus-like organism) responsible for the potato disease that caused the Irish potato famines of the mid-18^{th} century. We tested Conrad on the genome of this organism using comparative and *ab initio* models. We randomly selected a specific number of genes for training and tested against the rest of the curated set of 536 transcripts. For each model, we did ten replicates each of 50, 100, 200, 300, 400 and 500 training transcripts.

All CRF models use the Interval13 state model. The comparative models use MSAs with *P. ramorum* (strains of which are responsible for Sudden Oak Death [10]) and *P. sojae*. The MSAs were created with MUSCLE, and the Newick tree was created from the results in [10]. The *ab initio* discriminative models (Conrad13(MEA) and Conrad13(CML)) use all of Interval13's log-likelihood *ab initio* features with the specified discriminative training method. The generative model again uses all of the *ab initio* features, but weights are tied to 1.0 to reproduce an equivalent GHMM. Figure 3-1 shows transcript accuracies for the cross-validation experiments on all models. Table 3.1 shows scores for the best training set size for eaech model.

The high *P. infestans* accuracy results are not surprising given the models' impressive performance on the similar *Cryptococcus neoformans* [7]. It is clear that SMCRFs perform well for gene prediction on small eukaryotes. Specifically, the results show that discriminative training of equivalent GHMM features improves the full transcript accuracy by over 10%. Furthermore, the incorporation of multiple alignment data improves it another 3%, and training with MEA or using GC content improves accuracy 2% more. With these experiments, the theoretical advantages of an SMCRF over an equivalent GHMM are shown to be effective in practice.

Model	NT Sn	NT Sp	Ex Sn	Ex Sp	CT
Conrad13(CML)GFC-3	97.9	94.4	78.1	80.6	58.3
Conrad13(MEA)GF-3	96.7	93.6	76.8	81.7	55.5
Conrad13(CML)GF-3	94.9	94.0	69.6	80.2	52.7
Conrad13(CML)	92.5	93.2	61.7	80.0	50.0
Conrad13(Gen)	95.5	87.9	77.1	76.7	40.9

Table 3.1: Best accuracy scores of various models and training methods for gene prediction on *P. infestans*.

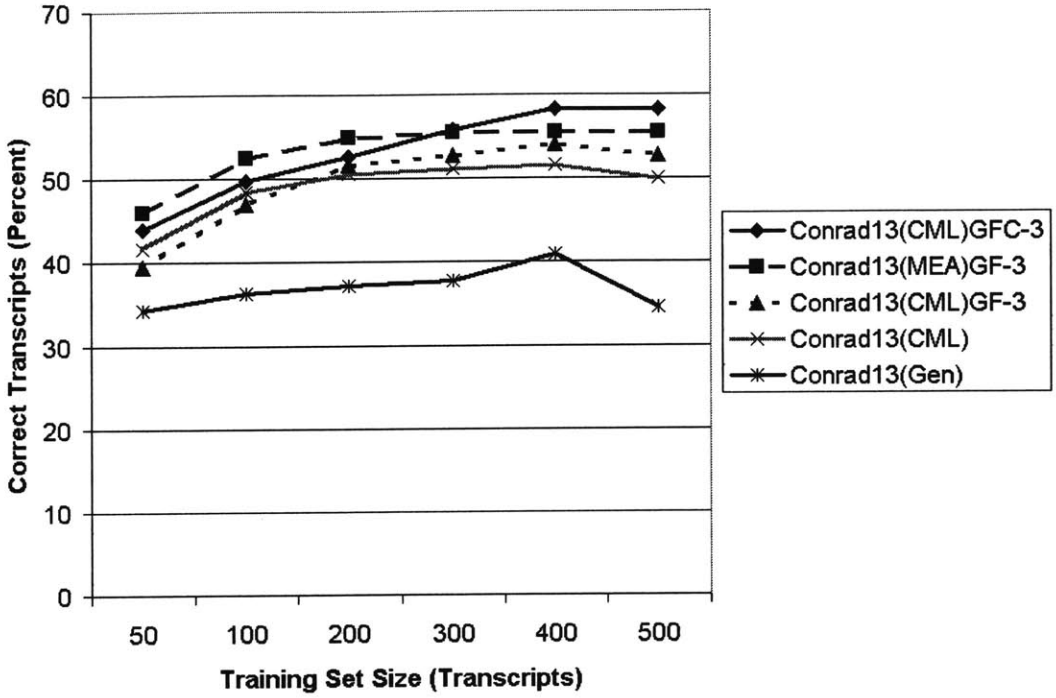

Figure 3-1: Transcript accuracy on *P. infestans*.

3.1.2 *Culex pipiens quinquefasciatus*

C. pipiens is a household mosquito that serves as a vector for a number of infections diseases including West Nile Virus. It features genes up to 32 kb in length with a mean of 2.5 kb. For this set of experiments, we used discriminative and generative Interval13 models as well as a discriminative Interval29 model. For Conrad13(Gen), Conrad13(CML), and Conrad29(CML) we did ten replicates each of 50, 100, 200, 300, 400, and 500 training transcripts randomly selected from the curated set of 613 transcripts. For the additional Conrad29(CML) models, we did only 10 replicates at 500 training transcripts because the model clearly overfits for smaller training set sizes (as shown in figure 3-2).

The discriminative Interval13 model (Conrad13(CML)) uses all available *ab initio* Interval13 features with weights trained using CML. The Interval29 model Conrad29(CML)'s features are equivalent or as close to equivalent as possible to those used by Conrad13(CML), and its weights are trained using CML as well. The gen-

erative Interval13 model Conrad13(Gen) again uses the same features, but weights are fixed at 1.0 to reproduce an equivalent GHMM. Figure 3-2 shows the accuracy scores from the cross-validation experiments over different training set sizes. Table 3.2 shows the best accuracy scores over the training set sizes for each model.

While the *C. pipiens* results indeed show that an SMCRF can accurately predict genes in insects, they are also notable for their direct comparison of the different Interval13 and Interval29 models. In particular, we see that the discriminatively-trained Interval29 model performs comparably to the generative Interval13 model given enough data to avoid overfitting, but the discriminative Interval13 model outperforms both. There are a few reasons that we expect Interval13 to outperform Interval29. First, Interval29 uses an implicit exponential probabilistic model of state lengths, which is limiting compared to Interval13's mixture of gamma distributions. Second, splice site contexts are modeled using shorter PWM in Interval29 than the Interval13, which further limits accuracy. Later analysis in chapter 4 discusses this accuracy gap. The tradeoff is that the entire cross-validation experiment across all replicates and training set sizes required just four hours with Interval29 but nearly two days with Interval13.

In addition, we see that adding discriminative features incrementally improves accuracy in the Interval29 model. The GC content features show improvement beyond the noise floor, but the improvement due to the others is unclear. The improvement due to these *ab initio* features tails off as we add more of them, suggesting either (i) that there is limited remaining signal in the raw nucleotide sequence that can be used to improve accuracy further, or (ii) the features have little value. Further testing is required to make this determination.

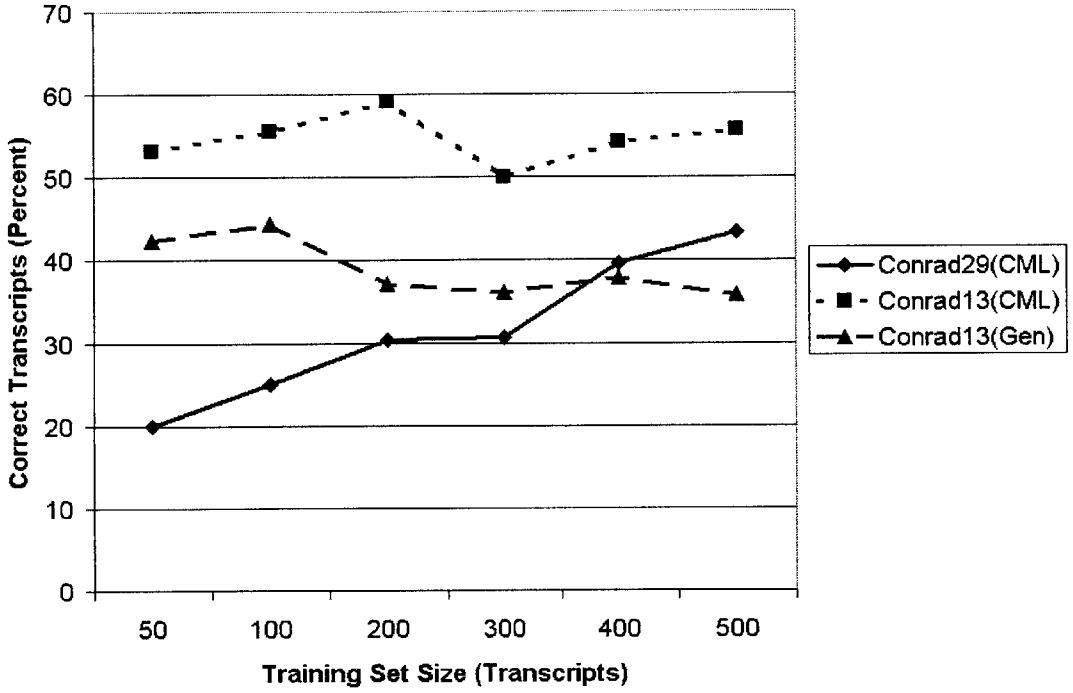

Figure 3-2: Transcript accuracy on *C. pipiens.*

Model	NT Sn	NT Sp	Ex Sn	Ex Sp	CT
Conrad29(CML)-CBPI	94.4	95.6	66.4	76.2	51.6
Conrad29(CML)-CBP	95.3	95.4	66.8	75.2	50.8
Conrad29(CML)-CB	94.0	95.3	65.5	75.8	50.5
Conrad29(CML)-C	95.2	95.4	65.4	75.5	50.1
Conrad29(CML)	93.7	95.1	62.4	73.4	43.4
Conrad13(CML)	95.1	98.2	63.0	81.8	55.6
Conrad13(Gen)	94.9	97.4	66.7	71.2	44.3

Table 3.2: Best accuracy scores for *ab initio* gene prediction on *C. pipiens* for a variety of models.

3.2 Gene Prediction in the *Homo sapiens* ENCODE Regions

The ENCODE (ENCyclopedia of DNA Elements) project aims to annotate the functional elements of the human genome in a comprehensive manner. The project uses

a standard set of 44 regions of 1% of the *Homo sapiens* (human) genome to focus researchers' efforts.

EGASP (ENCODE Genome Annotation aSsessment Project) was a gene prediction competition held in 2005. [12] The goal of the project was to test gene predictors head-to-head. For this set of experiments, we tested Conrad on transcripts from EGASP. In particular, we train on 364 transcripts from the 11 regions specified by the competition, and test on the remaining 33 used for evaluation of the participants' submitted predictions. All the CRF models use the Interval29 state model. We used one model that incorporates DNAase hypersensitive sites, several different *ab initio* models, and one comparative model with several different percentages of homology support. Each was trained with both CML and MEA gradients for comparison. The average transcript length is 30kb, while the longest transcript length is 804 kb . Most of this length occurs in intronic regions, which have an average length of 3.4 kb but can be up to 272 kb long. Table 3.3 shows accuracy scores for the different training methods and feature sets. Figure 3-3 and table 3.4 show the effects of increased CF1 homology support in the training and test data.

Because the Interval29 model is so much faster than the equivalent Interval13 model, it was necessary for the *H. sapiens* experiments to be performed only with variants of the former. The low accuracy scores are expected, as gene prediction in mammals, and human in particular, is a much different task than gene prediction in small eukaryotes. There are many reasons for this, including the human genome's extremely long introns and highly-variable transcription and splicing signals. Still, as expected from the Phytophthora and Culex experiments, we see improved accuracy by adding discriminative features and training with MEA. The increases due to the MEA gradient are especially noteworthy, as the MEA gradient used in [7] shows mixed results. Incorporating DNAase hypersensitive site evidence improves accuracy as well. Perhaps most interesting is the improvement due to increased proportions of CF1 alignment, which strongly indicates that additional alignments will improve accuracy even further.

Model	NT Sn	NT Sp	Ex Sn	Ex Sp	CT
Conrad29(MEA)SCBPID	60.3	72.9	26.7	60.1	11.1
Conrad29(CML)SCBPID	52.8	83.6	26.7	65.0	8.2
Conrad29(MEA)SCBPI	62.0	73.5	26.6	58.2	9.8
Conrad29(CML)SCBPI	50.1	81.8	22.4	61.0	6.6
Conrad29(MEA)	56.9	70.1	22.2	52.4	6.1
Conrad29(CML)	42.6	78.1	15.8	55.3	5.5
Conrad29(Gen)	47.0	58.5	9.1	37.2	1.8

Table 3.3: Accuracy scores for gene prediction on the ENCODE regions using a variety of models and training methods.

Model	CF1 Support	NT Sn	NT Sp	Ex Sn	Ex Sp	CT
Conrad29(MEA)	0	62.8	73.1	28.3	58.0	10.2
GFSCBPI-2	20	59.2	69.3	21.9	54.4	11.2
	40	68.5	75.2	31.1	57.4	15.2
	60	73.6	81.7	36.4	61.5	18.5
	80	78.4	86.4	41.6	64.3	21.0
	100	91.0	87.9	57.5	71.1	30.7
Conrad29(CML)	0	50.1	81.8	22.4	61.0	6.6
GFSCBPI-2	20	56.2	79.9	24.3	60.7	12.6
	40	61.8	84.9	26.5	61.0	14.5
	60	60.7	85.6	22.8	61.1	16.0
	80	60.5	89.0	18.6	59.2	19.8
	100	64.4	92.7	20.1	66.0	27.4

Table 3.4: Accuracy scores for best comparative model on ENCODE transcripts versus proportion of sequences supported by CF1 homology.

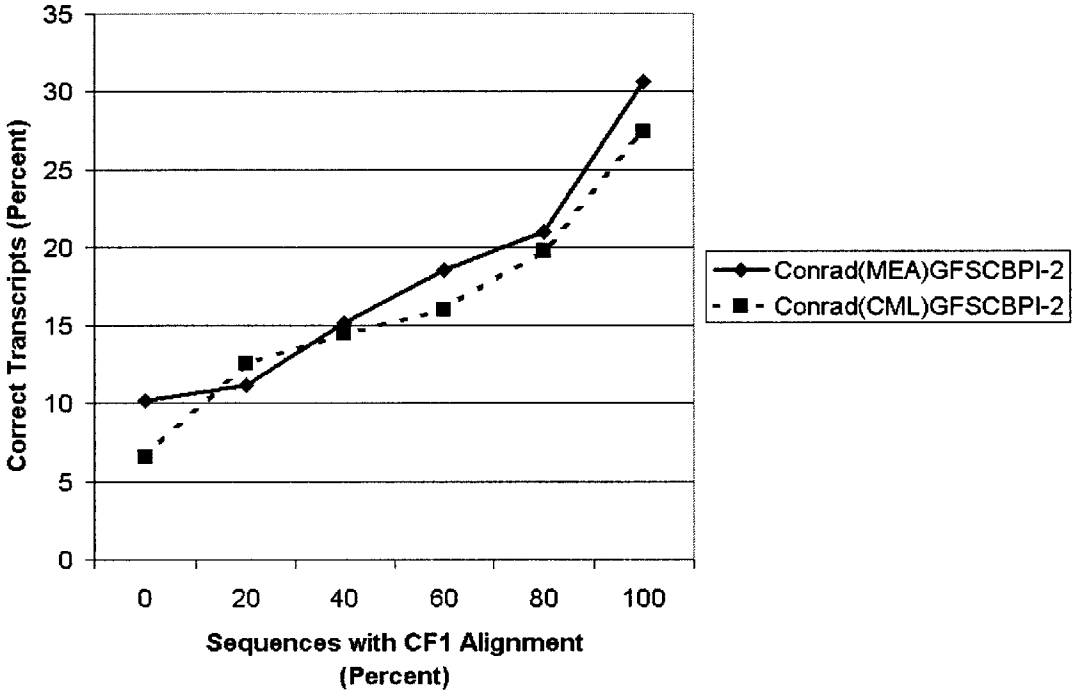

Figure 3-3: Transcript accuracy on ENCODE regions versus percent of sequences supported by CF1 homology. Sequences with CF1 alignments are three times more likely to be predicted correctly.

Chapter 4

Discussion

We showed that a CRF can reproduce the results of an HMM, and that it can improve these results by training its weights. Moreover, the CRF is useful in incorporating diverse evidence with no known probabilistic interpretation encoded as feature functions. We have used a wide variety of methods to extract signal from this data in the feature functions.

1. We can find a probability distribution for the data (as an HMM does) using a Markov model over the reference sequence, a trained PWM at state boundaries, or a phylogenetic model.

2. We can pass the data through a simple filter, as we did for GC content and others.

3. We can evaluate the output of other programs or methods, as we did for DNAase hypersensitive sites' p-values.

Although not explored in this thesis, we can also combine sets of feature functions using combination features. There is preliminary evidence that these features improve performance, but the current weight optimization methods do not scale well for the exponential number of combination features. To measure the combination features' usefulness, we will either have to try other training methods (as suggested in [7] and [31]) or choose these features very specifically.

There remain many other questions to be answered as well. First, the performance gap between the Interval13 and Interval29 models needs to be fully explained. Although the implemented tests appear to demonstrate high-level equivalence, it is not clear that this is the case. The two proposed differences, the differing state length distributions and splice site models, may not fully explain the gap. Instead, it is possible that bugs exist either in the solver or model that are preventing Interval29 from attaining better accuracy scores. Before the model is extended further, it is necessary to explore these issues.

The limited cross-validation results prevent us from directly determining the marginal value of particular models and features. The GC content features, for instance, clearly add value, but the branch sites, polypyrimidine tracts, and information content features may not. We must test additional combinations of these features to determine the value of each. The limited results are due to compute time limitations, but they will be expanded as the parallelization code is improved.

It is interesting that the MEA gradient consistently improves accuracy scores, as the results for this gradient in [7] are mixed. One possible explanation is that, for the tests performed in this thesis, the L-BFGS algorithm was allowed to iterate until floating point precision prevented it from improving the objective value any further. This method was made possible by the training parallelization, as the number of iterations it requires is much larger than is needed to meet the obtainable eps value used in [7]. Another possible explanation is that the objective function's "landscape" is significantly different for varying organisms. These explanations are speculative until comprehensive tests can be performed.

The difference in accuracy due to comparative data in *H. sapiens* and *P. infestans* is noteworthy. Transcripts are three times more likely to be correctly predicted with full alignment support on *H. sapiens*. But on *P. infestans*, with 1.5x support from two different species, the comparative models improve transcript accuracy only 3%. We are unsure of the cause of this, and need to examine the properties (especially the homology support) of the individual train and test sets to find an explanation.

In addition, a number of future directions for CRF gene prediction stem from results presented in this thesis. First, it would be interesting to develop a long intron state model similar to that used by AUGUSTUS. [30] Since the long introns in human genes are difficult to model probabilistically, such a state model should improve accuracy. It will also be interesting to compare the performance of Interval13's mixture of gamma distributions versus the AUGUSTUS distribution.

Second, it would be useful to parameterize features using properties of the sequence. For instance, it was found in [36] that genes with higher GC content tend to have more and longer exons. Such properties are trivial to encode in feature functions, and there is substantial evidence that they will improve gene calling accuracy.

While the signals extracted in the discriminative feature functions were found using custom solutions, results from external programs can be used as well. For example, we use the information content feature to detect repeats and other regions of low complexity, but this signal could easily be incorporated using the output from a program such as RepeatMasker [28]. In this way, we can use discriminative features to leverage the wealth of available programs that produce evidence for the presence of genes.

Indeed, gene prediction is not the only application that would benefit from combining output from external programs. Motif-finding is a rapidly maturing problem for which a vast number of programs have been written. Conveniently, these motif-finding programs generally output their predictions as consensus sequences or PWMs, which can be easily incorporated into a CRF simply by evaluating them. The CRF's supervised discriminative training algorithm will then evaluate them each for significance as a theoretically-sound "ensemble" algorithm [14]. We hypothesize that such a CRF-based prediction model would be at least as accurate as its most accurate constituent motif-finding program.

CRFs can go even farther for motif-finding. A skip-chain CRF (SCCRF) extends the linear-chain CRF by including skip edges between remote hidden variables [31]. These skip edges allow one to define features on multiple instances of a single label, such as, for instance, "motif." Motifs are notoriously difficult to find because there

is such little apparent signal available that we can use to detect their presence. By combining evidence from multiple instances of the same motif, we can more accurately label it. However, there is no polynomial-time exact solution to the inference problem for SCCRFs, so it requires approximation algorithms to solve efficiently. Still, this is an interesting path for future work.

Finally, predicting alternative transcript poses problems for gene prediction tools, which typically predict only to make one transcript per gene. However, evidence indicates that returning suboptimal solutions to the inference problem can accurately predict alternative transcripts [2] [29]. Since this is exactly what the N-best Viterbi algorithm is designed to do, this is a worthwhile application to explore.

We have demonstrated the ability to extend Conrad and therefore validated its use as a platform for research in gene prediction. These results, as well as those from [7] and [34], show that many applications of CRFs and related models are likely to appear in the near future. Conrad is one of the first in what will be a wave of discriminative approaches to varied and interesting problems in the field of bioinformatics and beyond.

Bibliography

[1] O.E. Agazzi and D. Burton. Trellis-based postprocessing of viterbi decisions for noisedecorrelation and compensation of nonlinear distortion (with application to magnetic recording channels). In *IEEE International Conference on Communications*, 1996.

[2] M. Alexandersson, S. Cawley, and L. Pachter. Slam: Cross-species gene finding and alignment with a generalized pair hidden markov model. *Genome Research*, 13:496–502, 2003.

[3] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic dna. *J. Mol. Biol.*, 268, 1997.

[4] Y. Chow and R. Schwartz. The n-best algorithm: An efficient procedure for finding top n sentence hypotheses. In *International Conference on Acoustics, Speech, and Signal Processing*, 1990.

[5] C.J. Coolidge, R.J. Seely, and J.G. Patton. Functional analysis of the polypyrimidine tract in pre-mrna splicing. *Nucleic Acids Research*, 25(4):888–896, 1997.

[6] A. Culotta and A. McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings of AAAI*, 2005.

[7] D. DeCaprio, J.P. Vinson, M.D. Pearson, P. Montgomery, M. Doherty, and J.E. Galagan. Conrad: Gene prediction using conditional random fields. *In Preparation*, 2007.

[8] E.S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[9] M. Farach, M. Noordewier, S. Savari, L. Shepp, A. Wyner, and J. Ziv. On the entropy of dna: algorithms and measurements based on memory and rapid convergence. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 48–57, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[10] M. Garbelotto, D.M. Rizzo, K. Hayden, M. Meija-Chang, J.M. Davidson, and S. Tjosvold. Phytophthora ramorum and sudden oak death in california: Iii. preliminary studies in pathogen genetics. In *Proceedings of the Fifth Symposium on Oak Woodlands: Oaks in California's Challenging Landscape*, 2002.

[11] M.R. Green. Biochemical mechanisms of constitutive and regulated pre-mrna splicing. *Annu Rev Cell Biol*, 7:559–599, 1991.

[12] R. Guigó and P. Flicek et al. Egasp: the human encode genome annotation assessment project. *Genome Biol*, 7 Suppl 1, 2006.

[13] N.L. Harris and P. Senapathy. Distribution and consensus of branch point signals in eukaryotic genes: a computerized statistical analysis. *Nucleic Acids Research*, 18(10):3015–3019, 1990.

[14] J. Hu, B. Li, and D. Kihara. Limitations and potentials of current motif discovery algorithms. *Nucleic Acids Research*, 33(15):4899–4913, 2005.

[15] M. Kellis, N. Patterson, M. Endrizze, B. Birren, and E.S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423:241–254, 2003.

[16] J. Lafferty and A. McCallum et al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

[17] M.F. Lin. Comparative gene identification in mammalian, fly, and fungal genomes. Master's thesis, Massachusetts Institute of Technology, 2006.

[18] C. Mathé and M. Sagot et al. Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Research*, 30(19), 2002.

[19] D.W. Mount. *Bioinformatics: Sequence and Genome Analysis.* Cold Spring Harbor Laboratory Press, 2001.

[20] P.A. Norton. Polypyrimidine tract sequences direct selection of alternative branch sites and influence protein binding. *Nucleic Acids Research*, 22(19):3854–3860, 1994.

[21] G. Parra, E. Blanco, and R. Guigó. Geneid in drosophila. *Genome Research*, 10:511–515, 2000.

[22] J.S. Pedersen and J. Hein. Gene finding with a hidden markov model of genome structure and evolution. *Bioinformatics*, 19(2), 2003.

[23] L.R. Rabiner. A tutorial on hidden markov model and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, 1989.

[24] A.A. Salamov and V.V. Solovyev. Ab initio gene finding in drosophila genomic dna. *Genome Research*, 10:516–522, 2000.

[25] S. Sarawagi and W. Cohen. Semimarkov conditional random fields for information extraction, 2004.

[26] A. Siepel and D. Haussler. Combining phylogenetic and hidden markov models in biosequence analysis. In *RECOMB '03: Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 277–286, New York, NY, USA, 2003. ACM Press.

[27] A. Siepel and D. Haussler. Phylogenetic estimation of context-dependent substitution rates by maximum likelihood. *Molecular Biology and Evolution*, 21(3):468–488, 2004.

[28] A.F.A Smit, R. Hubley, and P. Green. *RepeatMasker Open-3.0.*

[29] E.E. Snyder and G.D. Stormo. Identification of protein coding regions in genomic dna. *Journal of Molecular Biology*, 248:1–18, 1995.

[30] M. Stanke, A. Tzvetkova, and B. Morgenstern. Augustus at egasp: using est, protein and genomic alignments for improved gene prediction in the human genome. *Genome Biology*, 7 Suppl 1, 2006.

[31] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, 2006.

[32] I. Ulusoy and C.M. Bishop. Generative versus discriminative methods for object recognition. *cvpr*, 2:258–265, 2005.

[33] A.E. Vinogradov. Dna helix: the importance of being gc-rich. *Nucleic Acids Research*, 31(7):1838–1844, 2003.

[34] J.P. Vinson, D. DeCaprio, M.D. Pearson, S. Luoma, and J.E. Galagan. Comparative gene prediction using conditional random fields. NIPS, 2006.

[35] H. Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

[36] X. Xia, Z. Xie, and W.H. Li. Effects of gc content and mutational pressure on the lengths of exons and coding sequences. *J Mol Evol.*, 56(3):362–370, 2003.

[37] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.