Department of Computer Science
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Software for Annotation of Protein Coding Genes in Yeast Mitochondrial Genomes

(Master's Thesis)

Juraj Meššťánek

Supervisor: Mgr. Broňa Brejová, PhD.

Bratislava, 2010

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Software for Annotation of Protein Coding Genes in Yeast Mitochondrial Genomes

Master's Thesis

Study Programme: Informatics
Branch of Study: 9.2.1 Informatics
Supervisor: Mgr. Broňa Brejová, PhD.

Bratislava, 2010 Bc. Juraj Mešťánek

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

.....

Acknowledgements

I want to thank my supervisor Broňa Brejová for her invaluable help and guidance, and also to my family for their support.

Abstract

Author:	Bc. Juraj Meštánek
Title:	Software for Annotation of Protein Coding Genes in Yeast Mitochondrial Genomes
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Departement:	Departement of Computer Science
Supervisor:	Mgr. Broňa Brejová, PhD.
Number of Pages:	52
Year:	2010

In this thesis we present a software tool for automated computational prediction of protein coding genes in yeast mitochondrial genomes. Our tool is based on conditional random fields (CRFs).

To produce an accurate annotation, our tool combines information from several different sources. We use Exonerate to align reference proteins extracted from model organisms to the genome being annotated. We also use RNAWeasel to predict the positions of introns based on their characteristic structural motifs. Finally, we use multiple alignment of mitochondrial genomic sequences from several yeast species to look for evolutionary signatures typical for protein-coding regions. These three sources of information as well as the studied nucleotide sequence form a set of observations used in our CRF model to predict positions of exons and introns.

We have tested our tool on genes from 33 mitochondrial genomes. Currently, we predict 78% of genes and 70% of exons perfectly. In future we plan to make our tool available and easy to use for the life science community.

Keywords: gene finding, mitochondrial genomes, conditional random fields, external sources of evidence

Contents

List of Figures

Introduction

Cellular and genetic mechanisms are very complex and most of them have not been fully explained yet. However, there has been a great progress in recent years. Every year a database of annotated genes grows larger and new applications of genetic engineering are being discovered. This has a huge impact on quality of our everyday life. Genetically modified organisms and food became an answer to many problems. The understanding of processes inside cells help us with the fight against deadly diseases like cancer.

Proteins are encoded in DNA by segments called genes. A gene generally consists of two types of segments: exons and introns. Introns are non-coding parts of genes that need to be removed before the process of translation to protein can start. Coding segments, that are left after the removal of introns, are called exons. The problem of gene finding is to identify exons and introns in a DNA sequence. It is important to know the exact positions of exons in a gene because even a little change in a protein structure may lead to a significant change of its function.

The main aim of this work is to create a program for identifying exact gene structures in mitochondrial DNA based on a recently developed type of probabilistic models called conditional random fields (CRFs). One of the advantages of this model is the power to easily incorporate an external evidence such as output from intron identification algorithms. This would be problematic in a more traditional hidden Markov model approach. Size of mitochondrial DNA is much smaller than the size of the DNA sequence inside nucleus. This enables more computationally expensive strategies to take place.

In the past, the restricted number of genomic sequences available has limited comparative analysis and the development of more-sophisticated gene finding tools which require sufficient training data. But the situation has changed. Currently there are many phylogenetically related and relatively well-annotated sequences, but the need for effective gene finding algorithms remains. Large numbers of phylogenetically related sequences have opened new possibilities of incorporating evolutionary signatures into current methods of gene finding.

Chapter 1

Biological terms

This chapter explains the main principles of protein synthesis in living cells and points out characteristics of cell and gene structures that are relevant to this work. More concretely we describe the structure and functions of subcellular organelles called mitochondria and characteristics of introns - special sequences that are located between protein coding regions of DNA.

1.1 Genes and protein synthesis

A gene is a region of a genomic DNA sequence, which encodes one or possibly several proteins. DNA consists of four different molecules: adenine (A), cytosine (C), guanine (G) and thymine (T). Proteins are synthesized from the DNA template by a complex molecular machinery consisting of RNA and protein molecules. RNA is another type of a molecule that stores information. RNA consists of adenine (A), cytosine (C), guanine (G) and uracil (U) molecules.

1.1.1 Protein synthesis

Protein synthesis is done in three phases (see Figure 1.1):

1. transcription,
2. splicing,
3. translation.

In the transcription phase the two-stranded DNA double helix is unwound and one strand of DNA molecule is copied into a complementary preliminary messenger RNA

(pre-mRNA) by the protein complex RNA polymerase II.

Splicing removes some portions of the pre-mRNA, called introns; the remaining parts, called exons, are then joined together. Exons are therefore the part of a gene that code for proteins. The result of splicing is a mature mRNA molecule. Many eukaryotic genes are known to have different alternative splice variants, i.e., the same pre-mRNA producing different mRNAs.

And finally translation is the process of producing proteins according to the mRNA template. The mRNA sequences can be split up into a sequence of triplets called codons. Each codon codes for one amino acid. Because there are 64 possible codons and only 20 amino acids, some amino acids are encoded by several different codons. During translation, each codon is recognized by a specific transfer RNA (tRNA) molecule. Each tRNA molecule carries one amino acid which is added to the growing protein. The translation is terminated when a special stop codon is reached [8].

Protein is therefore a sequence of amino acids. Individual amino acids of the protein create molecular bonds which cause protein molecule to fold into a specific 3D structure. This 3D structure plays the key role in determining the function of a protein.

To sum it up, a genome is a sequence of alternating intergenic regions and genes, where each gene consists of alternating exonic and intronic segments. The first and the last segment of a gene needs to be an exon.

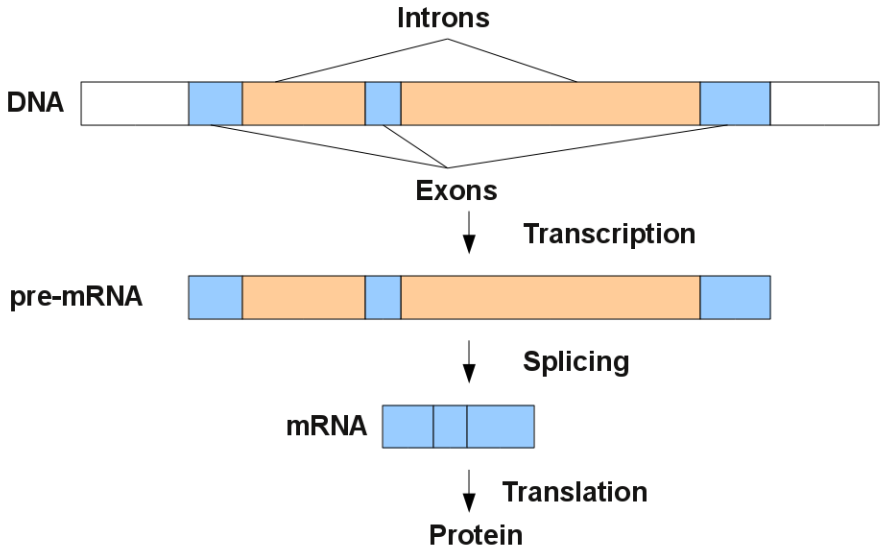

Figure 1.1: Schematic view of protein synthesis. Blue regions are exons, yellow regions introns and white are intergenic regions.

1.1.2 Gene homology

Genes in different species are called homologous if they share the same ancestor. The differences between the homologous genes are a consequence of the mutations that happened over a long time period. If a group of organisms of the same kind is separated from the rest and isolated for a long period of time, it will eventually evolve into a new species. This is because the mutations happen independently between individual isolated groups. When a species diverges into two separate species, the divergent versions of a single gene in the resulting species are said to be orthologous or orthologs. Genes can be also copied within the same genome. This process can speed up their evolution and back-up their functionality. These copies of a gene that occupy different positions in the genome are called paralogous or paralogs. Both orthologs and paralogs are homologs because they ultimately arose from a single gene either by speciation or by gene duplication. Homologous genes usually have the same or a very similar function because they share the same ancestor. For the same reason, they usually have a similar DNA sequence, and therefore homology between individual genes can be indentified by alignment algorithms (see Section 2.6).

1.2 Mitochondria

Mitochondria are subcellular organelles of eukaryotic organisms. They play an important role in the cell because they generate most of the cell's supply of adenosine triphosphate, which provides energy for many cellular processes. They also participate in other processes like aging by interacting with nuclear genes.

Mitochondria are thought to be descendants of bacteria engulfed by an eukaryotic cell and they contain their own genomes as well as systems for transcription, splicing and translation that are separate from those of the cytoplasm. Yeast mitochondrial genomes that we study in this work are circular and contain the same set of approximately 39 genes, encoding 14 proteins, 2 rRNAs, and around 23 tRNAs that can be distributed on both strands (see Figure 1.2). They are small, usually about 30 kb in size, therefore contain only a small number of introns.

Whole mitochondrial genomes are used as a powerful source of information for reconstructing phylogenetic trees, although mitochondrial sequences generally evolve more rapidly than nuclear sequences. Rapidly evolving portions of noncoding DNA are used for forensic identification and addressing population structure [7].

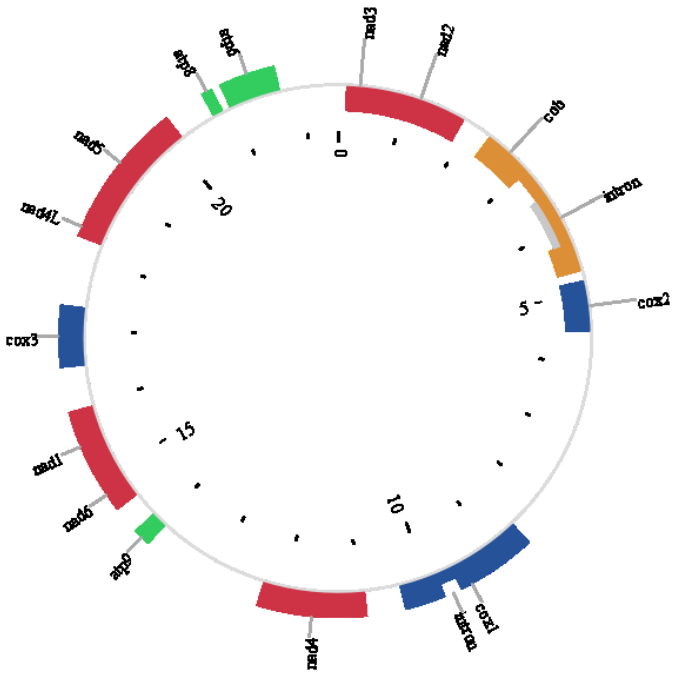

Figure 1.2: Protein coding genes of the mitochondrial genome of yeast Candida orthopsilosis. Two of the 14 genes have an intron. The image was created by software Circos [22].

1.3 Mitochondrial introns

Introns can be divided into four main types considering their splicing mechanism: spliceosomal introns, nuclear and archaeal tRNA introns, group I introns and finally group II introns [23]. Group I and group II introns are the only types that are present in mitochondria. They are characterized by their distinct, conserved RNA secondary structure. RNA secondary structure is a scheme, by which we can describe what molecules in the RNA sequence form a chemical bond (see Figure 1.3). Intron distribution is highly uneven among different types of species. For example, group I introns are abundant in fungal mitochondrial genes, but they are absent from most animal and protist mitochondrial genomes. Group II introns are common only in plant mitochondrial genomes, in which they are the predominant intron type. They are rare in the other genome types [23].

1.3.1 Characteristics of group I introns

Size of group I introns can differ from 68 to 3000 nucleotides but most are over 400 nucleotides long. Not all parts of the secondary structure are present in all group I introns. Splicing typically begins with a U that is paired with a G (see Figure 1.3). Splicing can rarely start with bases A or C. The last base is usually a G or in rare cases an A. Group I introns are able to propagate themselves in the genome. The DNA of some group I introns encodes a protein that can copy the intron elsewhere in a genome [5].

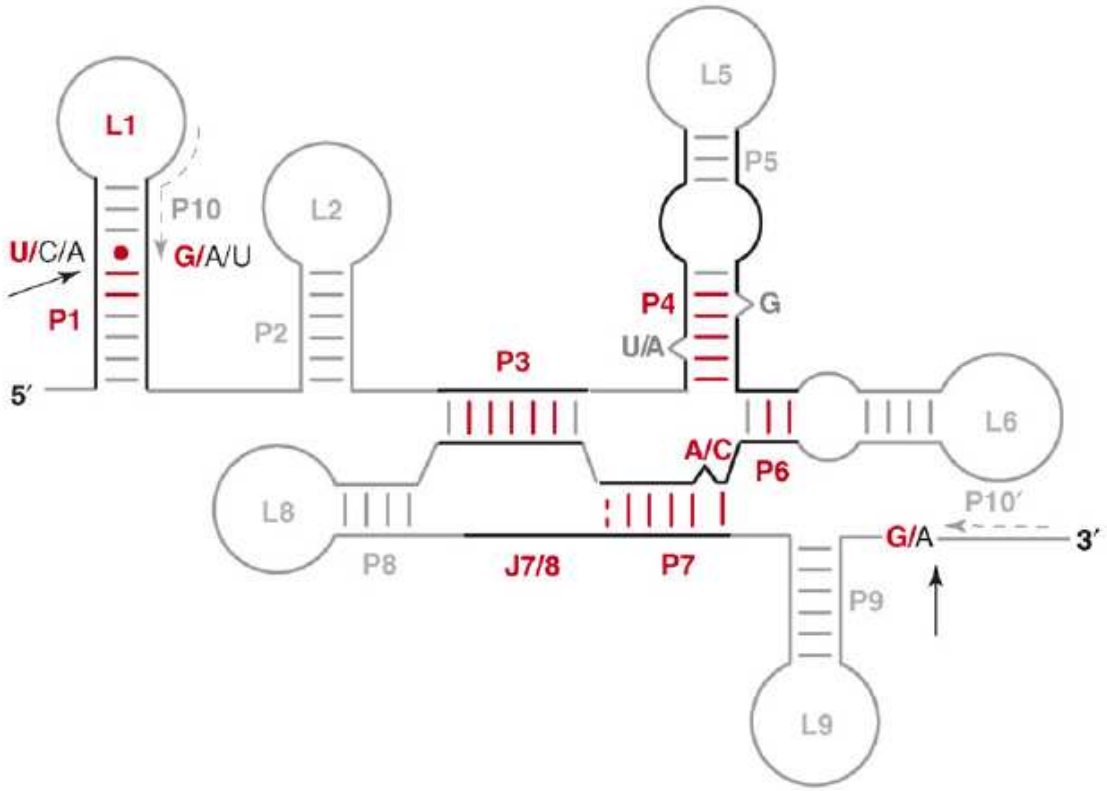

Figure 1.3: Secondary structure of group I intron. Splicing typically begins with a U that is paired with a G. The last base is usually a G. The more rare cases are shown in grey color. The splice sites are indicated by an arrow. P3 and P7 form a structure that is called a pseudoknot. (figure from reference [23])

1.3.2 Characteristics of group II introns

Group II introns are not very frequent. Their size can differ from 400 to 3400 nucleotids [6]. All group I introns have well defined secondary structure and fold into a specific 3D structure. Group II introns can be identified by domain V, which is the most distinctive and best-recognized secondary structure element of group II introns [23].

Splicing typically begins with a sequence GUGYG and ends with a sequence AY, where Y refers to bases C or T (see Figure 1.4). Some group II introns are mobile, but the mechanism by which they propagate differs from that of group I introns [23].

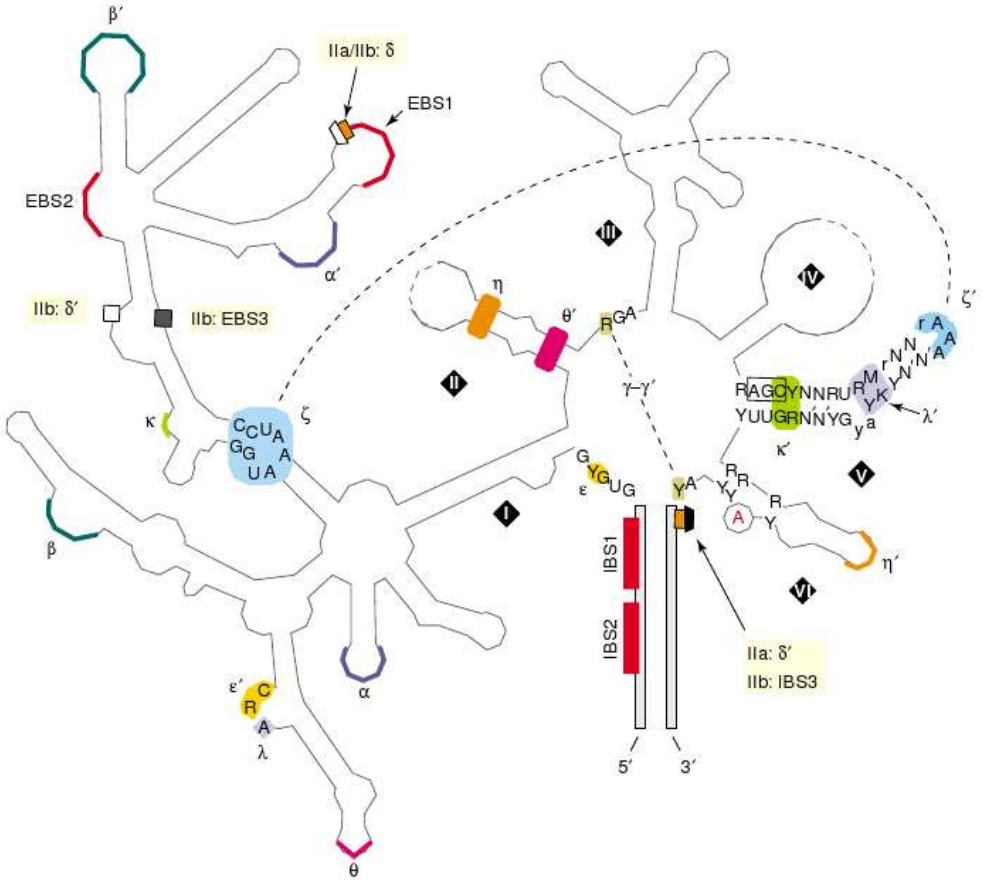

Figure 1.4: Secondary structure of group II intron. Splicing typically begins with a sequence GUGYG and ends with a sequence AY, where Y refers to bases C or T. The most distinctive element is domain V. (figure from reference [6])

Chapter 2

Models and algorithms

Our tool uses several existing tools for biological sequence analysis. This chapter gives a brief perspective on algorithms and models underlying these tools. In particular many of these tools are based on a class of probabilistic models called hidden Markov models (HMM). We will also use conditional random fields (CRF), a related class of models achieving better classification accuracy.

2.1 Hidden Markov models

Hidden Markov models are used for modeling sequence data in bioinformatics as well as other areas (natural language processing, speech recognition, etc.). In this section we will introduce them specifically in the context of gene finding.

Protein-coding genes consist of exons and introns as described in chapter 1. Differences between these elements can be manifested in many ways. For example at the nucleotide level bases G and C are more frequent in exonic regions than in intronic regions. We can also characterize gene structure itself. For example, the last exon of a gene has to be followed by a stop codon: a special sequence of three nucleotides that terminates the transcription process. Stop codon is then followed by an intergenic sequence. Nuclear genes usually contain many introns, and therefore it is more probable that a given exon will be followed by an intron than a stop codon.

HMMs can represent both the elements of a gene structure and the associated nucleotide sequence. Gene structure is described in an HMM by a probabilistic finite state automata. In context of our example, the states with their transition probabilities represent the structural elements such as exon, intron, intergenic sequence or a stop codon. For example the exon state most probably transitions to itself, because exons usually contain hundreds of nucleotides, but never transitions to the intergenic

state because only a stop codon state may transtition to the intergenic state. Upon transitioning to a state, the automata emits a symbol based on a probabilities assigned to this state. These probabilities are called emission probabilities. In our example, the emitted symbols represent nucleotides. The fact that bases G a C are more frequent in exons is taken into account in the emission probabilities of the states.

Each run of an HMM generates two sequences: a sequence of states and a sequence of emitted symbols. In gene finding, we are usually given the DNA sequence but we do not know which nucleotides belong to exons or introns. If we assume that this nucleotide sequence was generated from a model that is very similar to our HMM, we can use the HMM and ask different questions about the gene structure of our DNA sequence. For example, we can compute the most probable sequence of states, that generated our DNA sequence or the most probable state at a specific position of our sequence. The answers to these questions provide an information about the probable intronic and exonic regions and may be very close to the reality.

Because HMMs are basically finite state automata, the important algorithms run in linear time. HMMs have also some limitations that are a consequence of their design. Gene structures that have a more complicated secondary structure such as group I and II introns cannot be modelled by HMMs. A more complex models such as stochastic context-free grammars can be applied to identify such elements but at a cost of at least cubic time complexity. However in gene finding the studied sequences are usually very large and even quadratic algorithms are too slow.

We will now define HMMs more formally. An HMM is a probabilistic generative model that models joint probabilities of two sequences: a sequence of states S and a sequence of observations X. The random variables representing the ith symbols of sequences S and X are denoted S_i and X_i respectively. The probability of a state in the state sequence depends only on the previous state. More formally

$$P(S_n = s_n | S_{n-1} = s_{n-1}, S_{n-2} = s_{n-2}, ..., S_1 = s_1) = P(S_n = s_n | S_{n-1} = s_{n-1}). \quad (2.1)$$

Each state is given a set of transition probabilities. The probability of a state k transitioning to a state l is denoted by $T_{k,l}$.

$$T_{k,l} = P(S_i = l | S_{i-1} = k). \quad (2.2)$$

Additionally each state is assigned a probability T_k of starting the state sequence S. Finally, each state is associated with a set of emission probabilities $e_k(b)$ for each symbol b, that tell us what is a chance of seeing a symbol b when in state k. More

formally:

$$e_k(b) = P(X_i = b|S_i = k). \tag{2.3}$$

The previous parameters give a complete description of an HMM.

The probability of an HMM emitting a sequence of observations X from a sequence of states S (both of length N) can be written as:

$$P(X, S) = T_{S_1} e_{S_1}(X_1) \prod_{i=2}^{N} T_{S_{i-1},S_i} e_{S_i}(X_i). \tag{2.4}$$

The most probable path S^* for a sequence of observations X

$$S^* = \underset{S}{\operatorname{argmax}}\, P(X, S) \tag{2.5}$$

is usually the main question that we ask in gene prediction with an HMM, because we are usually interested in the most probable positions of introns and exons. The most probable path can be calculated by the Viterbi algorithm [12]. This algorithm is based on dynamic programming. Let $v_k(i)$ be the probability of the most probable path ending in state k at position i of the observation sequence. If we have these probabilities for all the states at position i, we can calculate them for the states at the next position by the following formula

$$v_l(i+1) = e_l(X_{i+1}) \max_k (v_k(i) T_{k,l}). \tag{2.6}$$

The algorithm fills in a table with M rows and N columns from left to right, where M is the number of states and N is the length of X. Each row corresponds to one state. The leftmost column contains values $v_l(1)$ that can be directly calculated as $e_l(1)T_l$. The algorithm simultaneously fills in a second table with the same number of rows and $N-1$ columns. Its ith column contains values $\operatorname{argmax}_k(v_k(i+1)T_{k,l})$ that represent the pointers to the states from which the values of the first table were calculated. The most probable path can be obtained by following the backpointers in the second table starting at the row which has the highest value in the rightmost column of the first table. The time complexity of the Viterbi algorithm is $O(M^2N)$ because each cell of the table takes $O(M)$ time to be filled.

Calculating a probability of a given sequence can be done by the forward algorithm. We can get the forward algorithm from the Viterbi by simply replacing maximizations with sums. Let $f_k(i)$ be the probability of the observed sequence $X_1...X_i$ where the

last symbol of this sequence is was emitted from state k:

$$f_k(i) = P(X_1...X_i, S_i = k). \tag{2.7}$$

Then the corresponding equation is

$$f_l(i+1) = e_l(X_{i+1}) \sum_k (f_k(i) T_{k,l}). \tag{2.8}$$

The algorithm works similarly as the Viterbi but does not use the second table. The leftmost column contains values $f_l(1)$ that can be directly calculated as $e_l(1)T_l$. The probability $P(X)$ can be calculated by summing the rightmost column of the table:

$$P(X) = \sum_k f_k(N). \tag{2.9}$$

The time complexity of the forward algorithm is $O(M^2N)$.

Finally, we can also compute the probability that the observation X_i of the sequence was emitted by a state k, i.e., $P(S_i = k|X)$. It is particularly useful when many paths have a similar probability as the most probable path. We can then ask what is the most probable state at each position and create a path from the answers. This problem can be solved by using a combination of the forward and backward algorithms. Probability $P(S_i = k|X)$ can be written as

$$P(X, S_i = k) = P(X_1...X_i, S_i = k)P(X_{i+1}...X_N, S_i = k), \tag{2.10}$$

where N is the length of the sequence X. The first factor is the forward probability $f_k(i)$ and we will denote the second term $b_k(i)$:

$$b_k(i) = P(X_{i+1}...X_N, S_i = k). \tag{2.11}$$

The recursion equation for computing the backward probabilities is

$$b_k(i) = \sum_l T_{k,l} e_l(X_{i+1}) b_l(i+1). \tag{2.12}$$

Finally the desired posterior probabilities can be obtained by formula

$$P(S_i = k|X) = \frac{f_k(i) b_k(i)}{P(X)}, \tag{2.13}$$

where $f_k(i)$ and $P(X)$ can be calculated by the forward algorithm and $b_k(i)$ by the

backward algorithm. The time complexity of the backward algorithm is $O(M^2N)$, and therefore calculating $P(S_i = k|X)$ has the same time complexity [12].

2.2 Generalized hidden Markov models

Although HMMs allow us to capture many aspects of gene structure, they have also limitations. For example it is not easily possible to incorporate arbitrary probability distribution of individual segments, such as exons into the model. We can in principle model state length distributions by chains of self-transitioning states, but this approach is not very flexible and has its own limitations.

A generalized hidden Markov model (GHMM) generalizes the HMM and enables the use of state lengths. Instead of emitting only one symbol, each state can emit a string of some finite length. The length of the emitted string as well as the output string itself are randomly drawn from a state-specific probability distribution.

Sequence of hidden states $S = S_1S_2...S_N$ can be written as a sequence of intervals $(t_i, u_i, v_i)_{i=1}^{p}$ with starts t_i, stops u_i, and labels v_i such that:

$$\begin{gathered} t_1 = 1;\ u_i \geq t_i;\ u_{i-1} + 1 = t_i;\ u_p = n; v_{i-1} \neq v_i \\ S_{t_i} = S_{t_i+1} = ... = S_{u_i} = v_i \end{gathered} \tag{2.14}$$

A GHMM models the the probability $P(S, X)$ of a segmentation $S = (t_i, u_i, v_i)_{i=1}^{p}$ and the observations X:

$$P_{GHMM}(S, X) = T_{v_1} \prod_{i=1}^{p-1} T_{v_i, v_{i+1}} \prod_{i=1}^{p} Q_{v_i}(X_{t_i,u_i}), \tag{2.15}$$

where T has the same meaning as in the previous section, and $Q_1, Q_2, \ldots, Q_M$ are the emission models, where $Q_v(X_{t,u})$ is the probability of hidden state v emitting the observed sequence X from t to u (including length distribution) [9].

The algorithms described in the previous section that compute specific probabilities for an HMM can also be used on a GHMM. However there is a significant change in the recursion step of all of the algorithms. The Viterbi algorithm used on a HMM creates a table with N columns. At each step it searches only the previous column. The Viterbi algorithm applied on a GHMM needs to search all of the previous columns in order to model all of the possible state lengths. This increases the running time to $O(M^2N^2)$. The same idea applies also to the forward and the backward algorithm.

2.3 Conditional random fields

A conditional random field is a probabilistic model that has a very similar field of application as HMMs. The main advantage of CRFs over HMMs is the power to easily incorporate information that does not have a probabilistic interpretation. Unlike HMMs, CRFs are not a generative model which means that they do not model the joint probability of the state labeling and the observations. Instead they directly model the state labeling conditional on a set of observations. We call such models discriminative. Another difference is in the training algorithms. HMMs can be simply trained by counting the transition and emission frequencies. This is not possible with CRFs because of their structure. Because there is no closed-form solution for assigning optimal parameters to a CRF, the training is done by numerical optimization techniques.

Given observation sequence X and a sequence of hidden states S (both of length N), conditional random fields model the conditional probability of $P(S|X)$. Let $\mathbf{f}$ be a set of k functions $f_1, f_2, \cdots, f_k$ that we will reference as "feature functions". Feature function f_j maps the sequence of observations X, an index i and the states at position i and $i-1$ (denoted as S_i and S_{i-1} respectively) to a measurement $f_j(i, S_{i-1}, S_i, X) \in \mathbb{R}$ [27]. Let $F_j(S, X)$ denote a sum over all of the measurements of feature function f_j:

$$F_j(S, X) = \sum_{i=1}^{N} f_j(i, S_{i-1}, S_i, X). \tag{2.16}$$

A CRF assigns a probability to the hidden states S by normalizing a weighted exponential sum of feature sums F_j:

$$P(S|X) = \frac{1}{Z_w(X)} \exp \sum_{j=1}^{k} w_j F_j(S, X), \tag{2.17}$$

$$Z_w(X) = \sum_{S} \exp \sum_{j=1}^{k} w_j F_j(S, X), \tag{2.18}$$

where w_j denotes the weight of the feature sum F_j, and $Z_w(X)$ is the normalizing constant [11].

An HMM can be simulated by a CRF using a single feature function f_{HMM} with weight 1.0:

$$f_{HMM}(i, S_{i-1}, S_i, X) = \begin{cases} \log e_{S_i}(X_i) + \log T_{S_i} \text{ if } i = 1 \\ \log e_{S_i}(X_i) + \log T_{S_{i-1}, S_i} \text{ if } i > 1. \end{cases} \tag{2.19}$$

Symbols T and e have the same meaning as in the section 2.1.

On the other hand, a CRF can loosely be understood as a generalization of an HMM that replaces the emission probabilities and the constant transition probabilities with a set of real functions that depend on the whole sequence. None of these functions need to have a probabilistic interpretation.

For the same reasons GHMMs are a good extension of HMMs, we would like to generalize CRFs to model a variable-length segmentations of state sequence S. We will split the state sequence S into a chain of segments similarly as we did in a GHMM. A sequence of hidden states $S = S_1 S_2 ... S_n$ can be written as a sequence of intervals $(t_i, u_i, v_i)_{i=1}^{p}$ with starts t_i, stops u_i, and labels v_i such that these segments satisfy the restrictions in 2.14. By restricting the interactions of the intervals only to the immediate neighbors we get a model called semi-Markov CRF (SMCRF). The feature sum F_j can be written as a sum of feature functions f_j:

$$F_j(S, X) = \sum_{i=1}^{p} f_j(v_{i-1}, t_i, u_i, v_i, X). \tag{2.20}$$

To use an SMCRF to annotate a genome sequence given observations X, one computes the segmentation S with the highest conditional probability $P(S|X)$. The inference algorithms to compute this are essentially the same and have the same complexity as those used in GHMMs.

For example the conditional probabilities of a GHMM are mathematically equivalent to an SMCRF using a single feature f_{GHMM} with weight 1.0:

$$f_{GHMM}(v_{i-1}, t_i, u_i, v_i, X) = \begin{cases} \log Q_{v_i}(X_{t_i,u_i}) + \log T_{v_i} \text{ if } i = 1 \\ \log Q_{v_i}(X_{t_i,u_i}) + \log T_{v_{i-1},v_i} \text{ if } i > 1. \end{cases} \tag{2.21}$$

We use the same notation as in the section 2.2.

The key issues in the application of CRFs and SMCRFs to gene prediction and other tasks are the design of the feature functions f_j and the selection of the weights w_j. Feature functions use the observations X to assign a real value to each state labeling of each possible interval and capture properties of the observation data relevant for classification. They are not required to be independent or have a probabilistic interpretation.

The traditional way of training the weights w_j of a CRF is by conditional maximum likelihood (CML). We are given a set of sequences and correct state paths

$(S^0, X^0), (S^1, X^1), \ldots, (S^N, X^N)$. For example, in gene finding we need to know both the sequence and its real gene structure. Our goal is to find values of the weights so that we maximize the likelihood of the training data. Assuming a single training sequence in training data (S^0, X^0), this is:

$$w_{CML} = \underset{w}{\operatorname{argmax}} \log P_w(S^0|X^0). \tag{2.22}$$

The function $\log P_w(S^0|X^0)$ is a concave function of w [30], because its Hessian is the negative of the covariance matrix of the random variables $F_j(S, X^0)$ when S is drawn from $P_w(S|X^0)$. Thus, $\log P_w(S^0|X^0)$ is guaranteed to have a single local maximum w_{CML}, which is also the global maximum. Because there is no closed-form solution, the optimization process of assigning weights is done by different numerical optimization techniques. The existence of a single local maximum guarantees, that these methods will converge to the optimal solution.

2.4 Gene prediction algorithms

Gene finding usually refers to a process of algorithmic identification of DNA regions that are biologically functional. These regions are usually protein coding genes but may also represent some other functional elements, e.g., RNA genes. Here we describe some of the tools that are used in prediction of protein-coding genes in nuclear genomes and are based on different probabilistic models described in the previous sections.

2.4.1 GENSCAN

GENSCAN [9] is one of the first gene finding programs. It uses a GHMM to predict genes in a target sequence. It uses only the target sequence as an input. GHMM model used by GENSCAN incorporates many of the gene structural properties of genomic sequences. GENSCAN uses different states that model exonic, intronic and intergenic regions as well as donor and acceptor splice sites. The typical gene density and the typical number of exons per gene are modelled through the state transition probabilities. Each state has its own length distrubution. Length of intronic states is modelled by exponential distributions. Separate empirically derived length distribution functions are used for initial, internal, and terminal exons and for single-exon genes. GENSCAN also incorporates many compositional properties of genes that are modelled in the emission probabilities of the individual states.

2.4.2 Conrad

Conrad [11] is a gene finder for nuclear genomes based on CRFs. It implements a variety of training and inference algorithms and several models with different levels of complexity. A core feature set forms a complete GHMM model somewhat similar to Genscan. Conrad also contains additional features that model external evidence, for example ESTs, phylogenetic features for nucleotide substitution models and a set of heuristics for multiple alignments. The feature weights together with discriminative training improve the accuracy compared to the GHMM-based tools. The accuracy can be also further improved by adding features that model additional external information. Most of these features are in detail described in the section 3.4.

2.5 Tools for finding RNA structural motifs

This section describes some of the currently used algorithms for identifying group I and group II introns in DNA sequences. These introns are characteristic by their conserved secondary RNA structure as described in the section 1.3. These structures can not be modelled by HMMs because the underlying finite state automata cannot model possible chemical bonds between distant complementary nucleotides. Most of the RNA structures can be modelled by a context-free grammar where each production is augmented with a probability [12]. Such a grammar is called a stochastic context-free grammar (SCFG). There exist several algorithms that infer the most probable secondary structure. These algorithms use dynamic programming and run in a cubic time, and therefore predicting secondary structure in longer sequences is very slow. Unfortunately group I and II introns also contain pseudoknots (see Figure 1.3) that can not be modelled by a context-free grammar and their detection is an NP-complete problem. Therefore they are usually not considered in the inference process.

2.5.1 CITRON and Rfam

CITRON [24] is one of the first tools for predicting group I introns. CITRON uses consensus matrices inferred from 143 group I introns. It uses a hierarchical search strategy. First it searches for the structural elements that are common in all group I introns and are easily recognizable. Then it extends its search to other structures.

Rfam [17] is a periodically updated collection of SCFG models and multiple alignments of different RNA families. Its models are usually used with the Infernal software [25] which is one of the best tools for identifying secondary structures in RNA sequences.

The major problem with CITRON and Rfam is that they consider only the common core features of group I introns. Group I introns are divided into many subgroups that significantly differ in both conserved motifs and peripheral elements of their overall secondary structure. Therefore incorporating different subgroup-specific models should lead to a substantial increase in sensitivity [23].

2.5.2 RNAweasel

RNAweasel [23] is a fast secondary structure predictor. It searches for secondary structure of group I and group II introns as well as tRNA a mRNA secondary structures. RNAWeasel uses a computationally efficient search engine ERPIN [15]. ERPIN utilizes RNA primary and secondary structure profiles that are trained from training sets that consists of RNA sequence alignments and secondary structure information. ERPIN searches for a precisely delimited structural elements both individually or in combination. This can be done by different search strategies which can substantially reduce the execution time. RNAWeasel automatically constructs training sets for ERPIN and sets various ERPIN parameters based on the users preferences. The main advantage of RNAweasel is the use of subgroup-specific intron predictors that can be easily developed and updated.

2.6 Alignment tools

Sequence alignment is a process of finding a possibly optimal pairing between symbols of two or more sequences taking into consideration possible insertions, deletions and substitutions based on our knowledge of evolutionary processes. We will refer to insertions and deletions as "gaps". Natural selection has an effect on the process of mutation so that some sorts of change may be seen more than others. The speed factor plays a very important role in design of alignment algorithms, because the sequences are usually very long.

2.6.1 Optimal local alignment

Given two sequences X of length M and Y of length N whose symbols are from a finite alphabet Σ (e.g., $\Sigma = \{A, C, G, T\}$), we want to insert gaps into both sequences, resulting in sequences X' and Y' of same length, such that we maximize the pairwise score of X' and Y'. A gap is represented by a symbol that is not in Σ, e.g., '-'. Because gaps represent insertions and deletions of symbols, each gap in the sequence X' needs to

be aligned with a symbol from the sequence Y and vice versa. The score we assign to an alignment will be a sum of terms for each aligned pair of original symbols plus terms for each gap. The score should correspond to the logarithm of the relative likelihood, that the sequences are related, compared to being unrelated. We expect, that identities and conservative substitutions (those that do not change function of the sequence by much) are more likely in alignments than we expect by a chance, therefore these terms will be assigned a positive score. On the other hand, gaps and non-conservative substitutions will contribute by a negative score, because they are more likely not to occur in real alignments [12]. An example of an alignment can be seen in Figure 2.1.

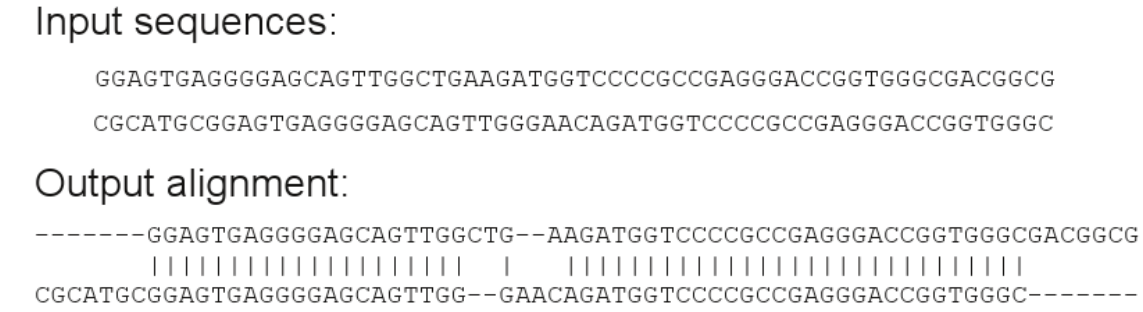

Figure 2.1: Example of a pairwise alignment. The resulting alignment was made by inserting gaps into the original sequences. Two gaps can not be paired.

A very common situation is where we are looking for the highest-scoring alignment between some contiguous subsequences of X and Y. This problem is called local alignment. Local alignment algorithms are best used with very diverged sequences that share short similar subsequences, because they search for all the possible combinations of these similar regions.

One of the algorithms that addresses the problem of local alignment is the Smith-Waterman algorithm [12]. The Smith-Waterman algorithm is an algorithm based on dynamic programming that searches for the local pairwise alignment. The algorithm returns optimal local alignments for the scoring scheme it uses.

The algorithm fills in a table with $M+1$ rows and $N+1$ columns using the following equation:

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j-1) + s(X_i, Y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d, \end{cases} \tag{2.23}$$

where $s(X_i, Y_j)$ is the pairwise score of symbols X_i and Y_i and $d > 0$ is the gap penalization. The table cell $F(i,j)$ holds the score of the best local alignment of the

subsequences $X_1X_2...X_i$ and $Y_1Y_2...Y_j$. First row of the equation corresponds to a start of a new local alignment, the second row corresponds to aligning symbols X_i and Y_j, and the third and fourth rows represent inserting a gap into sequences Y and X respectively. Values in the zeroth row and the zeroth column are initialized to 0. A second table is simultaneously constructed keeping pointers backward similarly as in the Viterbi algorithm in the section 2.1. When both of the tables are filled in, the algorithm searches the first table for cells whose score is above a given threshold. Starting from these cells, the algorithm reconstructs local alignments using the second table and ending in cells with score 0.

The Smith-Waterman algorithm time and space complexity is $O(NM)$ because each cell value can be computed in $O(1)$ time.

2.6.2 BLAST

BLAST [2] is one of the most widely used bioinformatics programs that searches for local alignments. The algorithm emphasizes speed over sensitivity. The speed factor is very important because genome databases are usually very large and it would be impossible to use algorithms that run in quadratic time, such as the Smith-Waterman algorithm.

BLAST first searches for a short high scoring local alignments called seeds between the query sequence and sequences in the database. For example if seeds are exactly matching substrings of length w, they can be found efficiently by hashing all substrings of length w in the query sequence. After seeding is done, BLAST continues to extend the seeds until the alignment score drops below the predetermined threshold. The BLAST algorithm uses a heuristic approach which may miss some high-scoring alignments, either because they do not contain high scoring seed or because the extension has failed to find an optimal alignment. Nonetheless tests on real data show that BLAST is over 50 times faster than the Smith-Waterman algorithm.

2.6.3 Exonerate

Exonerate [28] is a generic tool for pairwise sequence comparison. It is a collection of different nucleotide and protein alignment models that can produce both gapped and ungapped alignments. Exonerate implements exhaustive dynamic programming algorithms and many different heuristics for speeding up the search. In our work we use one of the more complex models, called protein2genome which aligns a DNA sequence to a homologous protein allowing long gaps in the DNA sequence corresponding to introns

in the gene. The model incorporates splice site restrictions and intron phases. A similar model of alignment was introduced earlier in Genewise [4]. It uses two pair-HMMs that emit symbols from two sequences at each step. The first pair-HMM models the translation of the DNA sequence to a protein sequence incorporating possible introns and the other pair-HMM models the alignment between the resulting protein and the homologous protein. These two pair-HMMs are then merged into one HMM.

2.6.4 MUSCLE

MUSCLE [13] is a tool for creating multiple alignments of protein sequences. The multiple alignment problem is a generalized version of the pairwise alignment problem described in the section 2.6.1. More than two sequences are allowed to be aligned and each column must contain at least one non-gap symbol, so that the gap insertions maximize the sum of aligned column scores. These scores have the same meaning as in the section 2.6.1. Accurate multiple alignments can significantly improve the accuracy of gene finding [29] because they provide a very strong evidence about the positions of coding regions as shown in the section 2.7. However computing an optimal multiple sequence alignment for arbitrary number of sequences is NP-hard, and therefore heuristic algorithms such as MUSCLE are usually applied to this problem.

MUSCLE works in three stages. In the first stage MUSCLE calculates a pairwise distance between the input sequences and clusters them into a tree according to these distances. This is done by heuristic algorithms that are fast but not very accurate. MUSCLE then constructs the multiple alignment from the leaves up. An alignment of all sequences in a subtree is represented as a profile which lists frequencies of individual characters in each column and is treated as a sequences of special symbols with a special alignment scoring function. At each internal node MUSCLE constructs a profile which is a pairwise alignment of children profiles. At the end of the first stage, the root contains the initial multiple alignment. In the second stage, MUSCLE uses the multiple alignment from the first stage as an input for the Kimura distance measure which is much more accurate than the heuristics used in the first stage. Then the algorithm constructs a new multiple alignment using the new tree similarly as in the first stage. In the third stage MUSCLE removes an edge from the tree. The tree is then divided into two subtrees. The profile of the multiple alignment in each subtree is then computed again and these profiles are re-aligned into a new multiple alignment of all sequences. If the score is improved, the new alignment is kept, otherwise it is discarded. Stage three is repeated until convergence or until a user-defined limit is reached.

2.7 Comparative analysis

In comparative analysis we use multiple sequence alignments to study various aspects of the sequences. For example, a multiple alignment provides information that is needed to construct phylogenetic trees. A phylogenetic tree is a rooted or unrooted binary tree where each node is a random variable. Leaf nodes correspond to the observed organisms in the multiple alignment, whereas internal nodes correspond to their ancestors. One column of the multiple alignment forms a complete set of leaf observations. Each tree branch is assigned a length corresponding to the evolutionary time between these two organisms. Multiple alignment, together with the tree and its branch lengths, help us to distinguish whether a region of the multiple alignment is more likely coding or noncoding because the rate of substitutions is usually much higher and has a different characteristics in noncoding regions than in coding regions. In order to quantify this likelihood, we use probabilistic models of substitution explained in the following text.

2.7.1 Substitution rate matrices

Given a set of M residues, we would like to model a probability of a residue a having being substituted by a residue b in time t denoted as $P(b|a,t)$. This can be represented as an $M \times M$ probability matrix that depends on time t, which we call a substitution matrix and denote by $S(t)$:

$$S(t) = \begin{pmatrix} P(A_1|A_1,t) & P(A_2|A_1,t) & \dots & P(A_M|A_1,t) \\ P(A_1|A_2,t) & P(A_2|A_2,t) & \dots & P(A_M|A_2,t) \\ \dots & \dots & \dots & \dots \\ P(A_1|A_M,t) & P(A_2|A_M,t) & \dots & P(A_M|A_M,t) \end{pmatrix}. \tag{2.24}$$

A natural expectation is that the probability $P(b|a,t+s)$ can be written as

$$P(b|a,t+s) = \sum_b P(a|b,t)P(b|c,s) \tag{2.25}$$

for all a,c,s and t. In other words matrix $S(t)$ is multiplicative, i.e., $S(t)S(s) = S(t+s)$.

A rate of substitutions between different pairs of residues is usually not the same. These rates can be modelled by a rate matrix. Given a rate matrix Q a substitution matrix for a short time $S(\epsilon)$ is approximately given by $S(\epsilon) \simeq (I + Q\epsilon)$, where I is the identity matrix.

An example of such a rate matrix is the K80 rate matrix [20] that has a following

form:

$$Q_{K80} = \begin{pmatrix} * & \kappa & 1 & 1 \\ \kappa & * & 1 & 1 \\ 1 & 1 & * & \kappa \\ 1 & 1 & \kappa & * \end{pmatrix}, \tag{2.26}$$

where symbol '*' is a value such that the corresponding row sums to 0.

The matrix models substitution rates between nucleotides T, C, A, G. The K80 model distinguishes between transitions and transversions. Transitions are substitutions between purines (nucleotides A and G) or between pyrimidines (nucleotides T and C). Transversions are substitutions between purines and pyrimidines and vice versa. Transitions are a more common form of mutation than transversions, so a parameter κ denoting the ratio between the transition and transversion rate is usually greater than 1. The K80 model assumes equal base frequencies ($\pi_T = \pi_C = \pi_A = \pi_G = \frac{1}{4}$).

Another example of a nucleotide rate matrix is the HKY85 model [19]. It is very similar to the K80 model with a difference that it allows unequal base frequencies ($\pi_T \neq \pi_C \neq \pi_A \neq \pi_G$):

$$Q_{HKY85} = \begin{pmatrix} * & \kappa\pi_C & \pi_A & \pi_G \\ \kappa\pi_T & * & \pi_A & \pi_G \\ \pi_T & \pi_C & * & \kappa\pi_G \\ \pi_T & \pi_C & \kappa\pi_A & * \end{pmatrix}. \tag{2.27}$$

We can also model substitution rates of codons. Such an example is the GY84 model [16]. It uses a 64×64 matrix where the substitution rate $q_{i,j}$ between the ith and jth codon is

$$q_{i,j} = \begin{cases} 0, \text{ if the two codons differ at more than one position,} \\ \pi_j, \text{ for synonymous transversion,} \\ \kappa\pi_j, \text{ for synonymous transition,} \\ \omega\pi_j, \text{ for nonsynonymous transversion,} \\ \omega\kappa\pi_j, \text{ for nonsynonymous transition,} \end{cases} \tag{2.28}$$

where π_i is the frequency if the ith codon, κ is the transition/transvertion ratio with the same meaning as in the K80 and HKY85 models, and finally ω is the ratio of nonsynonymous and synonymous substitution rates. Synonymous substitutions are those that do not change the amino acid encoded by the codon. Many of the protein-coding genes do not change very much in the process of evolution. This happens for example in genes that play a key role in the metabolic processes. Even a mutation of a single nucleotide could be potentially fatal to the organism, and therefore non-

synonymous mutations that change the structure of the encoded protein are very rare in these genes. Synonymous mutations do not change the resulting amino acids, and therefore they are seen much more often. Parameter ω for these types of genes is therefore much smaller than 1. In some genes non-synonymous mutations may happen faster than synonymous. This is especially true in genes that code for proteins that play a role in the immunity response, because the larger is the variety of proteins an organism can produce, the larger are the possibilities of fighting bacterial and viral diseases.

A substitution matrix $S(t)$ can be obtained from a rate matrix Q by matrix exponentiation

$$S(t) = e^{tQ}, \tag{2.29}$$

where matrix exponentiation is defined as

$$e^{Q} = \sum_{k=0}^{\infty} \frac{1}{k!} Q^k. \tag{2.30}$$

2.7.2 Felsenstein algorithm

Given a rooted binary phylogenetic tree T with N leaves and its branch lengths, rate matrix Q with M rows and columns, frequencies of symbols π in the root node and the observed symbols in the leaves of the phylogenetic tree denoted as C, one can compute the probability $P(C|T, Q, \pi)$ using the Felsenstein algorithm [14]. The Felsenstein algorithm is a dynamic programming algorithm. It fills in a table with $2N-1$ columns and M rows. The jth column corresponds to the jth node of the tree and the ith row corresponds to the ith symbol. The cell located in the ith row and the j column denoted as $p_{i,j}$ contains a probability $P(C'|T', Q, \pi)$ where T' is a subtree of T rooted in the jth node and $C' \subseteq C$ are the leaf observations of T'. Value $p_{i,j}$ can be calculated as follows:

$$p_{i,j} = \begin{cases} 1, \text{ if } j \text{ is a leaf and observes the symbol } i, \\ 0, \text{ if } j \text{ is a leaf and does not observe the symbol } i, \\ \prod_{\text{child } y} \sum_{\text{symbol } x} \left[e^{t_{j,y}Q}\right]_{ix}, \text{ if } j \text{ is an inner node,} \end{cases} \tag{2.31}$$

where y is a child of the jth node and $t_{j,y}$ is the edge length between the jth node and the yth node. If values of the root node are located in the rth column, then the probability $P(C|T, Q, \pi)$ can be written as

$$P(C|T, Q, \pi) = \sum_i \pi_i p_{ir}, \tag{2.32}$$

where π_i is the probability of the ith symbol.

2.7.3 PAML and PhyML

PAML (Phylogenetic Analysis by Maximum Likelihood) [31] is a package of programs for phylogenetic analysis of DNA and protein sequences using maximum likelihood. It implements many different codon and nucleotide substitution models. In this work we use PAML to train the parameters of the rate matrices we use from a given multiple alignment. However PAML does not include tools for determining the tology of trees with a larger number of species.

We use program PhyML to determine the tree topology [18]. PhyML is based on the maximum-likelihood principle which means it aims to construct a tree that would maximize the likelihood of the observed multiple alignment in some probabilistic substitution model. PhyML uses a simple hill-climbing algorithm that adjusts tree topology and branch lengths simultaneously. The algorithm builds an initial tree using a fast distance-based method and then iteratively modifies the tree to obtain a higher likelihood. The algorithm usually needs only a few iterations to reach an optimum.

Chapter 3

MtConrad: system for mitochondrial gene finding

3.1 Problem statement

The problem of gene finding is to identify exons and introns in a DNA sequence. Most of the research in gene finding concentrates on genes in nuclear genomes. Some of the programs that address this problem were described in the section 2.4. In this work we predict genes in yeast mitochondrial genomes which is a much less studied problem. Yeast mitochondrial genes lack the clear exon boundary rules typical for nuclear genes. More precisely nuclear spliceosomal introns are bounded by a specific pair of nucleotides on both ends (nucleotides GT at the 5' end and AG at the 3' end). Lack of such rules makes identifying precise exon boundaries in yeast mitochondrial genomes much harder. On the other hand, mitochondrial genomes are short and contain only a small set of well-conserved genes which allows us to use methods that are more difficult to apply on nuclear genomes.

A commonly used method for finding genes in mitochondrial genomes is aligning a known protein from a close organism to the studied genome. The resulting alignment then serves as a good starting point as it gives a good global view of the probable exonic and intronic locations. The protein alignments are then usually manually corrected to obtain the desired annotation. However this method alone cannot be always trusted. For example, larger exonic insertions in the studied genome can be mistaken for an intron. The alignment algorithm also has a high chance of missing the precise exon boundaries if there is a nonsynonymous mutation at the splice site.

Another approach is to specifically look for introns in the sequence. This is possible because yeast mitochondrial introns have a very conserved secondary structure. There

exist several tools for prediction of these structures, e.g., RNAWeasel [7] or Infernal [25]. The problem with this approach is that in many cases these tools predict only a partial structure of the intron and usually do not indicate the exact position of the splice sites.

In this work we present a software tool for automated computational prediction of protein coding genes in yeast mitochondrial genomes that combines these two previously used methods as well as other techniques originally developed for nuclear gene finding.

Sequences of many yeast mitochondrial genomes are already available and constitute another valuable source of information. In particular, multiple alignment of mitochondrial genomic sequences from these species in combination with different evolutionary models may provide a valuable information about the exact position of exons.

This year a new program called MFannot [1] was released, however it has not yet been published, and therefore the details of its methods are not known. It is the first program that directly addresses the problem of mitochondrial gene finding. We compare its results with MtConrad in the section 4.3.

3.2 Overall approach

Our tool is based on the Conrad gene finder described in the section 2.4.2. We use algorithms for training and inference with SMCRF implemented in Conrad, but modify the model extensively to adapt it to characteristics of mitochondrial genomes and available sources of information. We build on the basic state model that allows exons and introns on both strands but does not include more complicated situations such as alternative splicing. To use the information in the target DNA sequence, we use the core GHMM features and modify them to fit the structure of mitochondrial genomes. These changes consist of exon and intron boundary structure modifications, different state length limits and different caching settings.

In addition to the core features we use several sources of external information: protein aligned to the target DNA sequence, position of introns based on their secondary structure and comparative analysis of multiple alignment.

We use Exonerate [28] to align a protein from a model organism to the target sequence. We analyzed the accuracy of the alignments and created a set of features that correspond to the information that the alignments provide.

We use RNAWeasel [23] to predict the positions of introns based on their secondary

structure. We ran RNAWeasel on our data set and analysed the output similarly as we did with protein alignments. Based on the results of our analysis we created a set of features for this information source.

We have transformed the output from these programs into sequences of symbols called "evidence tapes". Evidence tape has the same size as nucleotide sequence it originated from. Each symbol on the evidence tape gives a specific information about the nucleotide at the same position in the nucleotide sequence (see Figure 3.1).

Finally, we use several nucleotide and codon substitution models on the multiple alignment. We do not use evidence tapes for this source of evidence instead we use log-likelihood values from the model.

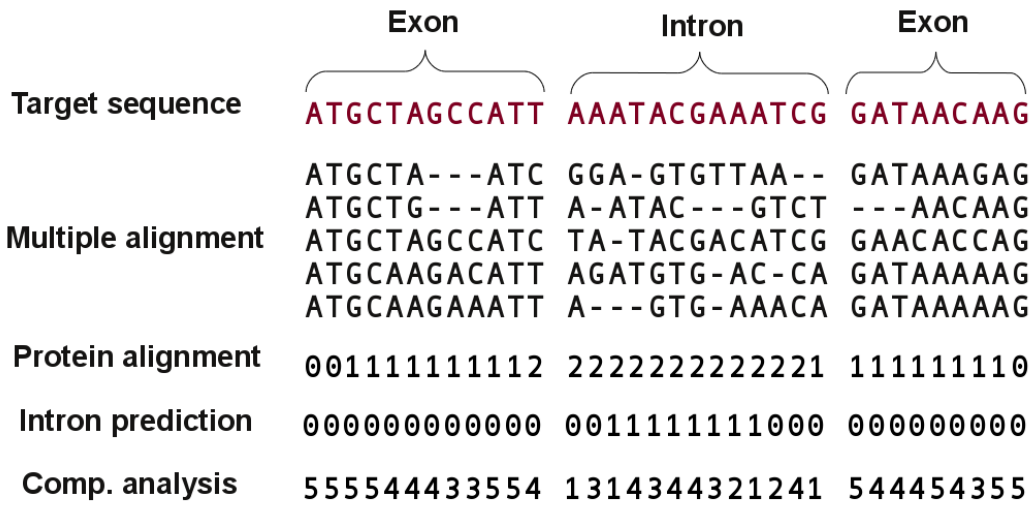

Figure 3.1: Example of evidence tapes.

To compare and check the results of our model we have also implemented a simple HMM that uses the same evidence tapes as MtConrad.

3.3 Sources of evidence

Here we describe the characteristics of the evidence we used. First we discuss and compare protein alignments and intron RNA structure prediction. Second we examine the trained parameters of the rate matrices used in comparative analysis.

3.3.1 Protein alignment and intron RNA structure

Since there are only few distinct ortholog groups in the genome we study and proteins in each group have similar sequences, we can use the knowledge of some representative

Test name	TP	FP	TN	FN	sens	spec
Exonerate coding	257084	1881	309661	10792	0.96	0.993
Exonerate intron	181390	3456	394295	277	0.998	0.981
RNAWeasel intron	66513	1010	396741	115154	0.366	0.985

Table 3.1: The table shows how good are individual tapes in predicting positions of intronic and exonic nucleotides. (TP - true positives, i.e., a predicted positive match that is as a matter of fact true; FP - false positives, i.e., a predicted positive match that is as a matter of fact false; TN - true negatives; FN - false negatives; sensitivity(sens) = TP/(TP+FN), specificity(spec) = TP/(TP+FP))

gene and its protein to predict the probable positions of coding nucleotides. To select a representative protein from each group, we used a gene from Saccharomyces cerevisiae if the group contained it, otherwise we used a gene from Candida albicans. We then aligned each nucleotide sequence to its representative protein using Exonerate. Because the sequences are short, we used quadratic time algorithm to predict the optimal alignment. We took the alignment with the highest score and converted it to an evidence tape. The resulting evidence tape consisted of symbol 0 - 4. Symbols 1 and 2 represent positions of aligned protein on plus and minus strand respectively. Symbols 3 and 4 represent positions between aligned sections on plus and minus strand respectively thus forming potential introns. Symbol 0 represents a sequence surrounding the alignment, presumably located in an intergenic region.

Another evidence tape can be created by looking for RNA secondary structures typical for group I and II introns. We predicted positions of these structures with program RNAWeasel. The program searches for secondary structure patterns of group I and II introns. The evidence tape for this program consists of symbols 0-2. Symbols 1 and 2 represent predicted intron positions on plus and minus strand respectively. Symbol 0 represents no evidence. RNAWeasel predicted 134 secondary intron structures in our dataset out of which 13 were group II introns and 121 were group I introns.

We individually analyzed the strength of each tape by comparing them against the real annotation. Then we measured how good were the tapes at correcting bad predictions of other tapes simply by counting the corrected nucleotides. Both tapes proved to be very informative. Exonerate tape showed a very good approximative power in both intron and exon locations (see Table 3.1).

Exonerate successfully predicted 101 out of 282 splice sites. RNAWeasel prediction was even better in specificity. The only setback of RNAWeasel prediction is that the true information it provides is a subset of Exonerate prediction. Particularly Exonerate corrected 1009 positions with incorrect RNAWeasel predictions. These were nucleotides that RNAWeasel predicted as introns but Exonerate labeled them correctly as coding.

Test name	Corrected predictions
Exonerate coding corrected Weasel	1009 nucleotides
Exonerate intron corrected Weasel	0 nucleotides
Weasel corrected Exonerate	10 nucleotides

Table 3.2: Number of corrected nucleotides.

On the other hand RNAWeasel has managed to correct only 10 of Exonerate's incorrectly predicted nucleotides (see Table 3.2).

All RNAWeasel false positive errors have occured in three sequences where RNAWeasel predicted a large intron significantly overlapping an exon (see Figure 3.2).

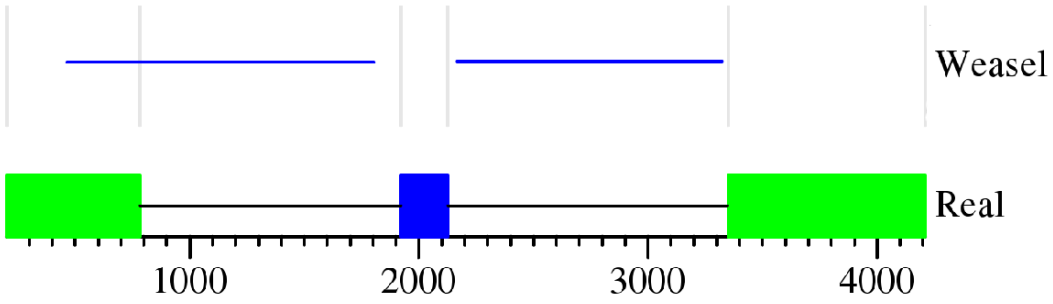

Figure 3.2: Example of an error produced by RNAWeasel.

There was no overlapping with exons in any other cases. This result is very good, because RNA secondary structure of group I a II introns tend to overlap with exons on their edges. RNAWeasel successfully predicted only one out of 282 splice sites which is not a good result, but many of the predicted introns had boundaries very close to the splice site, usually within 5-20 nucleotides (see Figure 3.3). We also tried to use Rfam [17] for intron prediction, but the results were much worse.

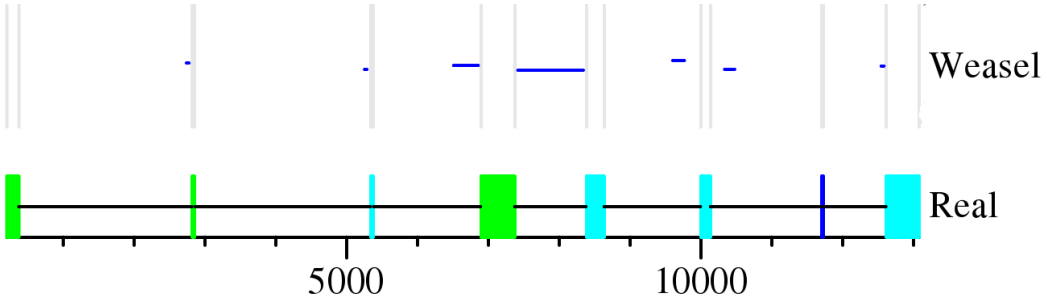

Figure 3.3: Typical output produced by RNAWeasel. Rectangles in the lower part represent exon positions. Lines in the upper part represent predicted introns. Most of the predicted introns are very close to one of the exons.

Position	π_A	π_C	π_G	π_T	K80			HKY85		
					ts	tv	ts/tv	ts	tv	ts/tv
Exon 0	0.34	0.09	0.22	0.35	0.51	0.34	1.50	2.37	1.42	1.66
Exon 1	0.24	0.19	0.13	0.44	0.39	0.28	1.39	1.74	1.24	1.40
Exon 2	0.50	0.06	0.04	0.40	0.44	0.69	0.65	6.68	2.73	2.44
Intron	0.44	0.08	0.12	0.36	0.64	0.94	0.69	7.16	3.97	1.8
Intergenic	0.44	0.07	0.07	0.44	0.55	0.98	0.56	9.03	4.03	2.24

Table 3.3: Table shows the nucleotide frequencies (π_A, π_C, π_G, π_T) as well as transition (ts) and transversion (tv) rates of the K80 and HKY85 models trained on the exonic (phase specific), intronic and intergenic regions of our input data.

3.3.2 Comparative genomics

We have trained a separate substitution rate matrix for introns, intergenic regions and three positions in a codon. Details of the training process are described in the section 4.1.4. The parameters of the K80 and HKY85 rate matrices are shown in Table 3.3. Nucleotides located in the zeroth and first exon phase have a higher GC content than the rest of these genomes which are very AT rich. Introns have a slightly elevated GC content because some of the introns contain an open reading frame (a sequence that can possibly code for a gene) in their DNA sequence. Transition and transverion rates correspond to the rate of mutation in individual gene structure elements. We can see that the rate is higher in intronic and intergenic positions and lower in codon positions.

The parameters of the GY94 for exonic regions were $\kappa = 0.65$ and $\omega = 0.09$. The codon frequencies were estimated from frequencies of exonic nucleotides in different exon phases (see Table 3.3). The parameter ω that models the nonsynonymous/synonymous rate ratio was similar to our expectations that the synonymous rate should be much higher than nonsynonymous rate (see Section 2.7.1). The parameter $\kappa = 0.65$ is not a standard value because the transition rate is usually higher than the transversion rate in exonic regions.

3.4 MtConrad model

In this section we describe details of the probabilistic model underlying our tool. It is based on the basic model called Interval13 used in Conrad. Conrad model was designed for nuclear genomes so modifications had to be done to adjust the core features for prediction of mitochondrial genes and to adapt it for new sources of external evidence. Recall from the section 2.3 that probability distribution over gene structures in a SMCRF is defined as $P(S|X) = \frac{1}{Z_w(X)} \exp \sum_{j=1}^{k} w_j F_j(S, X)$ and each feature sum F_j can be expressed as a sum of terms $F_j(S, X) = \sum_{i=1}^{p} f_j(v_{i-1}, t_i, u_i, v_i, X)$. We will first describe the set of states and allowed transitions between them and then individual features in detail.

3.4.1 State model

The state model consists of 13 states. One state models intergenic regions, 3 states model coding regions of genes located on plus strand, 3 states model intronic regions on plus strand and similarly 6 states model the genes on minus strand. More complicated cases such as genes within introns were not modelled. The states and transitions between them are depicted in Figure 3.4.

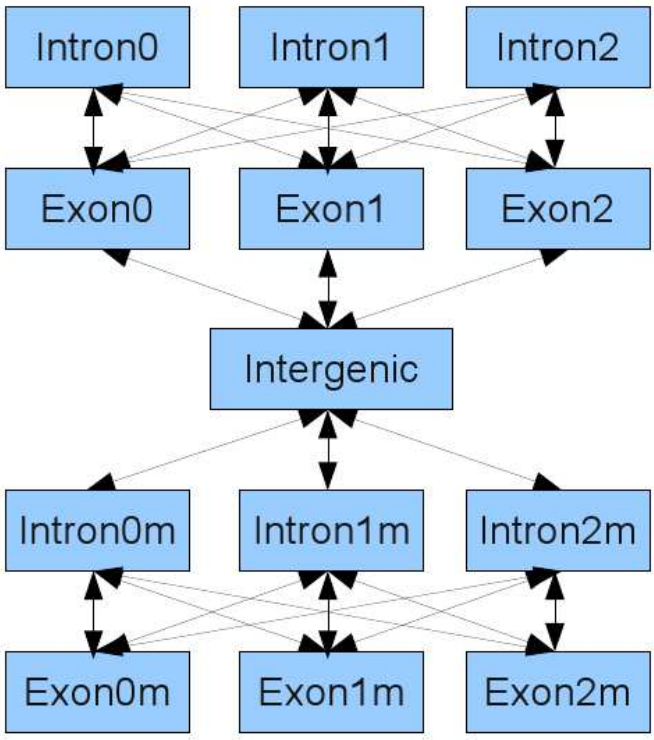

Figure 3.4: MtConrad hidden state model.

States. We use Interval13 state model. All of the positions below are 0-based so the first nucleotide of the sequence has index 0. Each state of the model generates the whole segment, thus the over all model is SMCRF described in the section 2.3.

1. Intergenic

   Intergenic state represents bases between adjacent genes including the stop codon but not including the start codon of genes.

2. Exon_i, $i = 0, 1, 2$

   This state represents an exon located on a plus strand where the first base of each codon is located at position k in the nucleotide sequence such that $k = i(\mod 3)$, the second base of codon at position $k = i + 1(\mod 3)$ and the third base at position $k = i + 2(\mod 3)$, more generally the nucleotide at position k in the sequence is at position $(k - i)(\mod 3)$ in a codon.

3. Intron_i, $i = 0, 1, 2$

   This state represents an intron of a plus strand gene, splitting a codon with i nucleotides to the right and $(3 - i)$ nucleotides on the left. We need three intron states to ensure that the next exon continues in the correct reading phase.

4. Exonm_i, $i = 0, 1, 2$

   This state represents an exon located on a plus strand where the third (leftmost) base of each codon is located at position k in the nucleotide sequence such that $k = i(\mod 3)$, the second base of codon at position $k = i + 1(\mod 3)$ and the first (rightmost) base at position $k = i + 2(\mod 3)$, more generally the nucleotide at position k in the sequence is at position $(-k + 2 + i)(\mod 3)$ in a codon.

5. Intronm_i, $i = 0, 1, 2$

   An intron of a minus strand gene, splitting a codon with i nucleotides to the left and $(3 - i)$ nucleotides to the right.

Transition constraints. Transitions between states ensure that only valid gene structures can be produced. For example, the sum of exon lengths has to be dividable by three. We use the same transition constraints as in Conrad. Because states in the model are defined by their position in the sequence the transition constraints are also dependent on the position. In the following S_k represents state at position k in the sequence. We describe the transitions only for the plus strand, the transitions for the minus strand are analogous. The allowed transitions in the model are as follows:

- Start of a gene: if S_{k-1} is intergenic and $k = i(\mod 3)$ there is a possible transition to S_k=Exon$_i$.

- End of a gene: if S_{k-1} is Exon$_i$ and $k = i(\mod 3)$ there is a possible transition to S_k=integenic.

- Donor site (start of an intron): if S_{k-1} is Exon$_i$ and $k = i - j \pmod 3$ there is a possible transition to S_k=Intron$_j$.

- Acceptor splice site (end of an intron): if S_{k-1}=Intron$_i$ and $k = j - i \pmod 3$ there is a possible transition to S_k=Exon$_j$.

Length constraints. Individual states in Conrad have minimum and maximum lengths. It is important because the algorithms used for training and inference of SMCRFs (see sctions 2.2 and 2.3) run in $O(N^2)$ time, where N is the length of the sequence. Using maximum state lengths decreases the time complexity to $O(NM)$, where M is the maximum length used among states. Minimal length of each state should be at least of the size of the boundary model used on the state ends. These boundary models represent typical sequences occurring on exon boundaries. We used a minimum exon length of 9, a minimum intron length of 15, and a minimum intergenic length of 18 bases.

We had to increase the maximum length of the intron states from 600 to 3500 bases because mitochodrial introns are larger than nuclear introns. We lowered the maximum exon length from 5000 to 3000. The largest mitochondrial exon in our data set was around 2000 bases long.

Base constraints. The model requires specific bases at transitions around intergenic state. In particular, all genes must start with ATG codon and end either with TAG or TAA codon. We decided to use only the start codon ATG although the genetic codes we were working with (Code 3: Yeast Mitochondrial Code and code 4: The Mold, Protozoan, and Coelenterate Mitochondrial Code) contain a larger set of start codons. We encountered only 5 cases of ATA start codon in our data set so we decided not to include other start codons in the model. We have removed the nuclear splice site constraints from the Conrad model. These constraints also speed up the SMCRFs inference and training algorithms.

3.4.2 Core features

Core features model functions that depend only on the reference sequence and current and previous state. Together they are equivalent to a complete GHMM model for gene prediction as shown in equation 2.21. These functions were used in every test we made (in combination with some subset of evidence features).

Reference features. Reference features correspond to emission probabilities of an HMM. They return the log probability of each nucleotide based on the surrounding sequence and the current state. For each nucleotide the probability depends on the three previous nucleotides, or in case of minus strand states on the three following nucleotides. The dependence on the adjacent nucleotides allows us to model frequencies of 4-tuples of nucleotides.

We use the model from Conrad without any changes. The model uses five sets of log probabilities: one for intergenic regions, one for introns and one for each position of codon. Log probabilities are trained from the training data and initialized with pseudocounts of 1 to prevent zero probabilities. Plus and minus strands are trained together and any training or evaluation on the minus strand is done on the reverse-complement counterpart of the examined sequence. This reduces the amount of free parameters and the amount of required training data. Bases near the boundary of the segment are evaluated to 0 because they are evaluated by separate boundary features.

A total of five features are defined for the reference features. These features do not correspond directly to the five log probability tables. Instead they are for intergenic, plus strand introns, minus strand introns, plus strand exons, and minus strand exons. Each feature has its own weight.

Length features. The three length features compute the log probability for state length distributions of introns, exons, and intergenic regions. We used the same model for length distribution as used in Iterval13 model. Intergenic distances are modeled using an exponential distribution, and the exon and intron distributions are modeled as a mixture of two gamma distributions.

Transition features. The transition feature models the frequency of various state transitions between segments, and corresponds to the transition matrix of a GHMM. This feature is represented by one weight. Because we use the same model as Conrad

We use the transition features from Conrad without any changes. Intergenic state has six possible transitions to exon state but only two are valid at a given position

in the sequence (one on each strand). Both of these transitions have a probability of 0.5. This is to prevent nonsense parameters in the situations where the training data contains a vast majority of plus strand genes.

Each exon state has four possible transitions: three to intron state and one to an intergenic state. At every position only one of the transitions to intronic states is valid. The only free parameter used in this model is the probability P of transitioning to intron state. Transitioning to intergenic state has a probability of $1 - P$. This probability is determined from the average number of introns in each gene.

Three possible transitions exist out of each intron state, but at any given position, only one of these will be valid. This transition probability is therefore set to 1.0.

Boundary features. The boundary features model nucleotide signals on the state boundaries. We use the same boundary features as Conrad. These features are trained from the training data and represented by position weight matrices (PWMs) and are initialized with pseudocount of 1 to prevent zero probabilities. PWMs model position-specific frequencies of nucleotides on the boundaries. We use them to capture the boundary rules described in the section 1.3. The number of nucleotides taken into consideration is consistent with the reference features. There are four types of signals:

- Splice donor extends 3 bases into the exon and 6 into the intron
- Splice acceptor extends 6 bases into the exon and 9 into the intron
- Start signal extends 9 bases into the intergenic region and 6 into the exon
- Stop signal extends 3 bases into the exon and 9 into the intergenic region

Each of these signals has its own feature both for plus and minus strand. This gives together eight weights.

3.4.3 Evidence features

Evidence features incorporate evidence gained from external programs or from models that use multiple alignments.

Exonerate features. Exonerate provides a very strong evidence of intron and exon locations. In contrast to the core features, we do not estimate any frequencies from training data for these features and we use only simple indicator functions which are

incorporated by appropriate weights. Such an approach would not be possible in an HMM. We split the features into two subsets with ten features total: features evaluating nucleotides and features evaluating edges (exon boundaries). It is reasonable to use edge features because Exonerate successfully predicted a substantial amount of splice sites.

We use five nucleotide features each evaluating to 1 under specific conditions and zero otherwise. Four features represent a situation when the current labeling of the position agrees with the Exonerate labeling. This includes labellings for exons/introns on the plus/minus strands. The last feature represents a negative situation when Exonerate did not predict exon or intron but the current label is not an intergenic state. We would expect that this feature will have a negative weight while the previous four positive.

Similarly we model features evaluating edges. Four features evaluate to 1 if the donor/acceptor splice site agrees with the exonerate prediction. If none of the four features was evaluated at a given boundary then the last feature is evaluated to 1.

Conrad contains a set of features called EST features. These features use a set of similar indicator functions as we do. We extend these features by adding indicator functions for different strands.

Intron features. RNAWeasel provides a very specific information about intron nucleotides. However it does not predict any splice sites, therefore we did not model edge features.

We use tree features to model output from RNAWeasel that are similar to Exonerate features. Two features (one for each strand) correspond to the situation when the current state agrees with the prediction of RNAWeasel and one feature corresponds to the situation when RNAWeasel predicted an intron nucleotide but the current state is intergenic. Each of these features evaluate to 1.

Nucleotide comparative features. We use comparative features to evaluate different nucleotide substitution models whose input is a multiple alignment column. Before these features can be used, they need to be initialized with a specific nucleotide substitution rate matrices. We have used Kimura80 [20] and HKY85 [19] substitution rate matrices described in the section 2.7.1.

We use five position-specific phylogenetic features: one feature models intergenic regions, one intronic regions and three features model different codon phases. Each features is assigned its own nucleotide substitution model that was trained from the

training data a fits the evolutionary characteristics of the modelled region (see Table 3.3).

These features evaluate to log probabilities of the the multiple alignment columns in the given evolutionary model. The evaluation is carried out by the Felsenstein algorithm (see Section 2.7.2). Complementary nucleotides are used for the minus strand states.

Nucleotide comparative features are based on the phylogenetic features used in Conrad. We had to change the implementation of inputs and evolutionary models because the Conrad implementation did not allow to use different target organisms in both training and inference. We have changed the implementation of the K80 model training algorithm so it can now also train the HKY85 model.

Codon comparative features. We have extended the nucleotide-based phylogonetic features to be also applicable with codon substitution models. Such models are not present in Conrad. In particular we use the GY94 codon substitution model [16] (see Section 2.7.1). Its parameters were trained using the PAML software [31]. The process of trainig the GY94 model is described in the section 4.1.4. We use the GY94 model for the exonic states and the HKY85 model for the intronic and intergenic states. We have trained the HKY85 model from intronic data.

At each exon state we calculate the positions of the three codon phases in the reference sequence and use the multiple alignment columns at these positions as an input for the Felsenstein algorithm so that alignment columns with gaps in the reference sequence are not used. The feature then returns the log probability of this alignment. We calculate the log probability of a codon alignment only if the current position in the sequence is in the third codon position (second phase). We use complementary bases for the minus states.

The second feature is evaluated for every intronic and intergenic state using the HKY85 model. The evaluation process is identical to one used in the nucleotide comparative features so that we again evaluate triplets of columns together and omit columns with a gap in the reference sequence.

We do not evaluate boundary positions of the exonic segments. For the plus exonic segments we do not evaluate first two nucleotides and for the minus exonic segments we do not evaluate the last two nucleotides. This is to prevent the intronic columns to be used in the codon alignment when the splicing did not occur between two codons. We have created two separate features that evaluate these boundary exonic positions. They use the same GY94 model. They take the exonic boundary columns and fill in

the missing codon columns with columns that consist of gaps. The first feature models the log probability of exonic boundaries with one missing column and the other one with two missing columns.

3.5 Evidence-HMM

To compare the performance of our SMCRF system with some baseline and to evaluate strength of individual evidence we have created a simple HMM that incorporated the probabilistic information that could be obtained from our information sources.

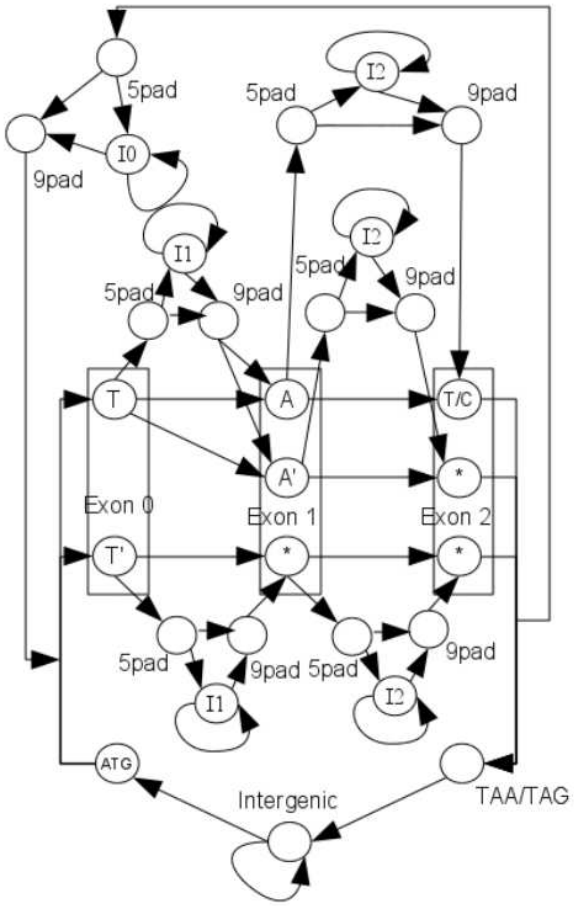

Figure 3.5: HMM states for plus strand. Three columns of states in the middle correspond to three possible codon positions. All intron states can be separated into three groups (I0, I1 and I2) based on the position in the codon. Generating stop codon (TAA/TAG) is forbidden. Letters inside the states represent nucleotides that they emit. Letters with an apostrophe are forbidden.

We modelled both plus and minus strands. Exon states were forbidden to generate stop codons. This is achieved by phase-specific exon states that emit different nu-

cleotides and adding different intron states between these exon states (see Figure 3.5). Minimal length of introns was set to 14 nucleotides with starting pad of length 5 and ending pad of length 9. This is achieved by adding a chain of states before and after each intron state. Intron states that are in the same group (I0, I1 and I2) have the same emission and transition probabilities. We used only start codon ATG, because the other one (ATA) was very rare in the data.

Each state is associated with four random variables:

1. a nucleotide of the predicted sequence (X_1),
2. a multiple alignment column from 33 organisms (X_2),
3. symbols 0-5 for Exonerate output (X_3),
4. symbols 0-4 for RNAWeasel output (X_4).

We consider all of the variables conditionally independent so the probability of a given observation at a given state is:

$$P(X_1, X_2, X_3, X_4|S) = \prod_{\imath=1}^{4} P(X_i|S), \tag{3.1}$$

therefore the emission probabilities of each state are a product of four probabilities. The probabilities of the Exonerate and RNAWeasel symbols as well as probabilities of the nucleotide symbols were trained simply by counting the frequencies from the training data. For comparative analysis (to compute $P(X_4, S)$) we used the K80 nucleotide substitution model and trained the parametes (transition and transversion rate, root nucleotide probabilities) from the training data. We used the Felsenstein algorithm (see Section 2.7.2) to calculate the probability of the multiple alignment column and multiplied it with the other probabilities. Similar evidence tapes were used in gene finder TWINSCAN [21].

Chapter 4

Results

In this chapter we present the accuracy results of our system on a set of yeast mitochondrial genomes. To conduct such tests, we had to first assemble a set of yeast mitochondrial genomes and their annotations, and estimate parameters of the model. We describe details of the data preparation and parameter estimation first, followed by the test results.

4.1 Data preparation

This section describes techniques that were used to prepare and analyze the data. The goal is to gain a perspective on the quality of the annotated sequences. Prepared data will be later used by our program for training and testing. The preparation can be divided into multiple parts:

1. collecting data,
2. extraction of desired parts,
3. generating the phylogenetic tree,
4. generating evidence from external programs,
5. analysis of the evidence.

4.1.1 Collecting data

We have obtained a set of 33 annotated mitochondrial genomes in Genbank format from different yeast species (Saccharomycetaceae taxonomy group). A majority of

these genomes is from the Genbank database [3], the rest are unpublished genomes provided by prof. Nosek from Faculty of Natural Sciences, Comenius University in Bratislava. The file of each genome consists of the DNA sequence and an annotation in a machine-readable text format describing location of protein-coding genes and other functional elements, their names and function. The level of annotation was different in almost every sequence which made automated processing of these files complicated. These are the most important differences between annotations that we had to deal with:

- missing annotation of introns,
- incorrect protein translations (not agreeing with the DNA sequence),
- nonstandard use of annotation tags (files not adhering to the format).

We had to manually correct the problematic files. Some of the files were not usable so we omit them from our analysis.

4.1.2 Extraction of desired parts

Using Bioperl library we have extracted protein-coding genes with the surrounding sequence of fixed length. The length of this sequence on each side of a gene was set to 200 nucleotides. The surrounding sequence was allowed to contain functional elements (parts of adjacent genes, tRNA genes and others).

We have kept only genes that had an orthologous gene at least in three other organisms. Orthology between genes was easily identified by the same gene names or gene name synonyms in the annotation. Proteins of genes with unique names were aligned against predicted orthologous groups to find any additional orthologies. By this procedure we have managed to extract 323 genes that could be sorted into 15 orthologous groups. The smallest group contained genes from 13 organisms.

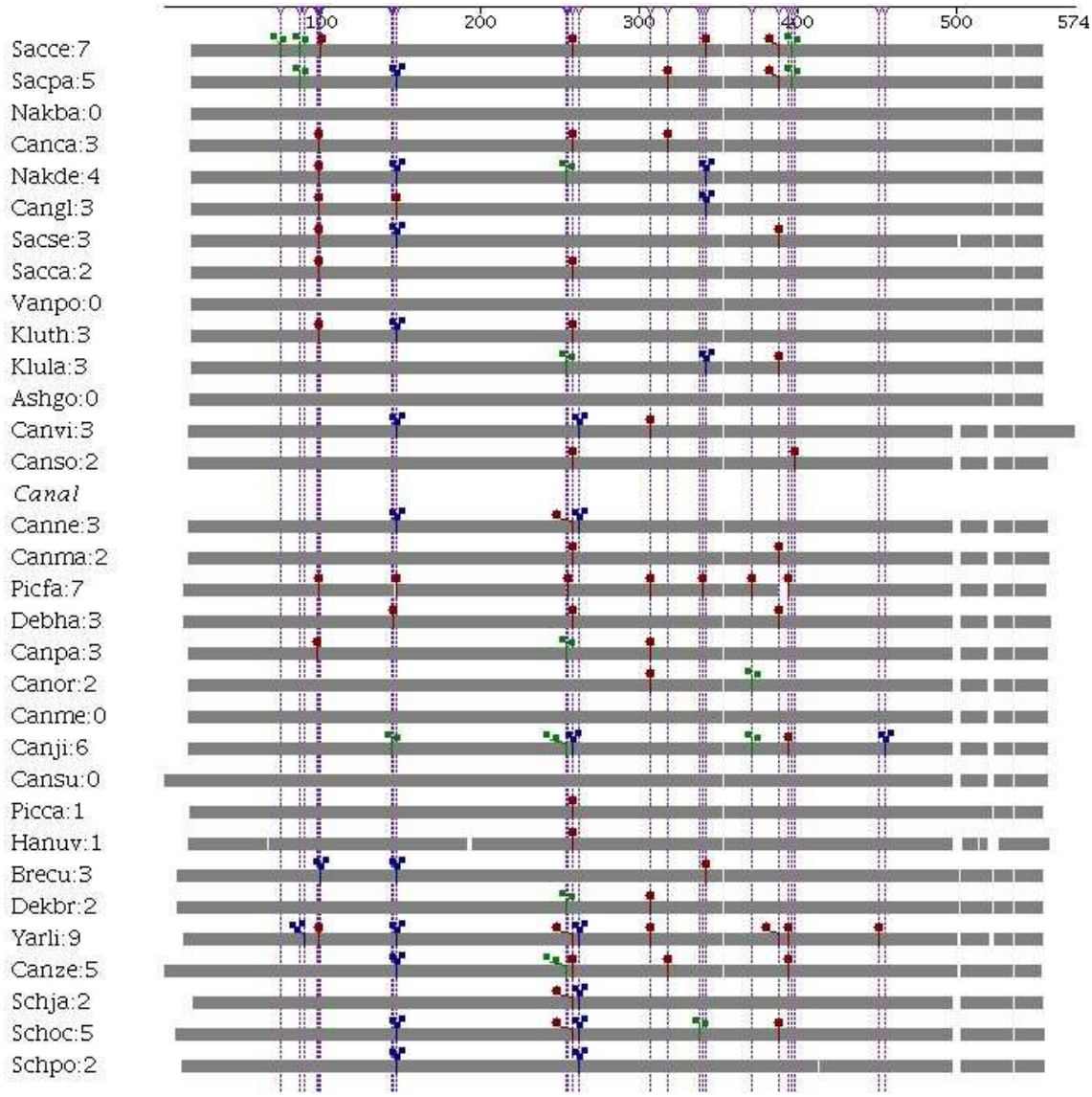

Figure 4.1: Graphical alignment of COX1 proteins from 33 organisms with positions of introns. Each row represents a protein from one organism. Positions of introns in different codon positions are labeled as "flags" of different color. The sequences are very similar and most of the introns are located in the same positions. The picture was generated in program Malin [10].

256 genes were located on the plus strand and 67 on the minus strand. Average number of exons per gene was 1.44 with 141 introns in the whole set. Average length of an exon was 577 nucleotides. Average length of an intron was 1289 nucleotides. Most of the introns were located in COX1 and COB genes. Aligned COX1 genes and positions of their introns can be seen in Figure 4.1. The largest gene was approximately 13000 nucleotides long.

4.1.3 Generating the phylogenetic tree

Substitution models we use in MtConrad require a phylogenetic tree with branch lengths representing evolutionary distances between different organisms and their ancestors. Because we did not have a suitable phylogenetic tree of species in our set we needed to create one from our data.

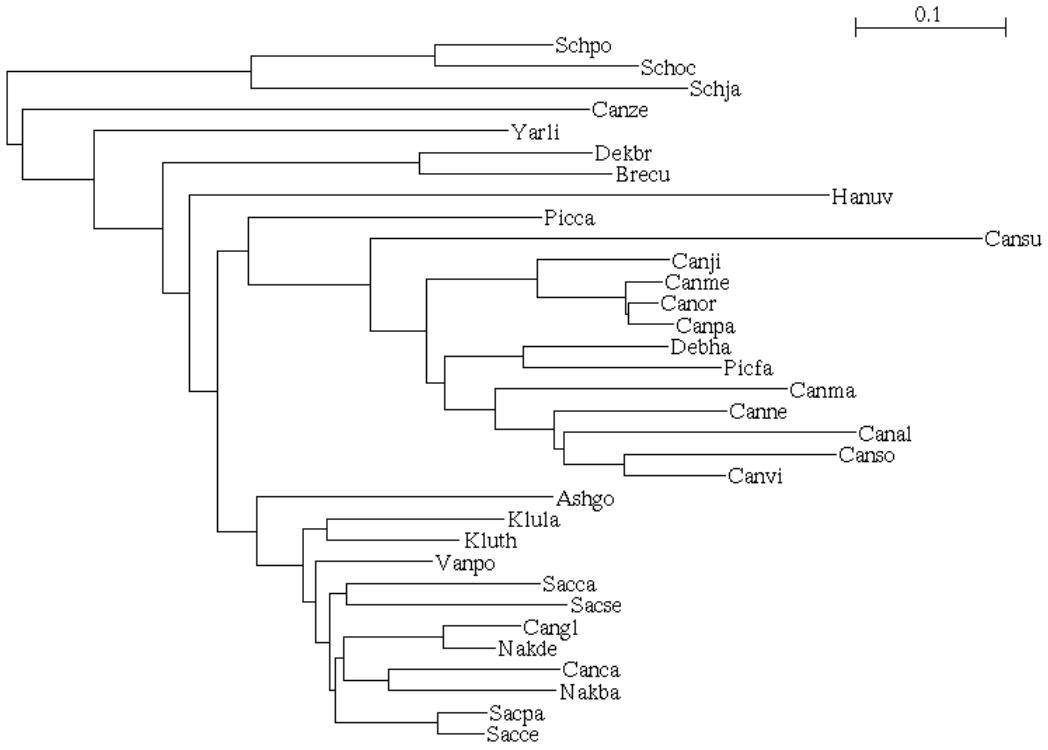

Figure 4.2: Phylogenetic tree of input species.

First, we created a multiple alignment of protein sequences in each orthologous group using MUSCLE (see Section 2.6.4). Then we merged all the created alignments into one alignment by concatenating their columns. Using this alignment and program PhyML (see Section 2.7.3) we constructed the final phylogenetic tree in Newick format. This tree can be seen in Figure 4.2.

4.1.4 Estimating parameters of the rate matrices

We used a Nelder-Mead simplex method [26] implemented in Conrad to train the parameters of the K80 model. Nelder-Mead simplex method is used to minimize a function of multiple variables without derivatives. We have also modified the implementation of this method to estimating parameters of the more complex HKY85 model.

We used PAML [31] to generate parameters of the GY94 model (see Section 2.7.1).

First we extracted coding DNA of each gene. Then we translated these sequences and aligned them using MUSCLE (see Section 2.6.4). Finally we created a codon multiple alignment by mapping the codons back to the protein multiple alignment and replacing each gap in the protein alignment by three gaps. Using this codon multiple alignment and the phylogenetic tree (see Section 4.1.3) as an input, we ran PAML with parameters icode=4, model=0, NSites=0 and CodonFreq=2. We extrated the parameters of the GY94 model from the PAML output. PAML also generated new lengths for individual tree edges because the definition of the edge length for codon substitution models differs from the definition used for nucleotide substitution models.

4.2 Evidence-HMM results

In this section we describe the accuracy results of the Evidence-HMM with different combinations of evidence sources. These results will provide a baseline for the strength of individual tapes. Another reason is that the HMM can be trained and tested very efficiently in comparison with CRFs, and therefore we can test different configurations much more faster.

Test	**State match**	**Perfect**	**Exon**		**Coding nucl.**	
			sens	**spec**	**sens**	**spec**
1 H, E, W, C	74.5%	57.7%	0.452	0.267	0.961	0.993
2 H, W, C	62.5%	21.0%	0.237	0.070	0.872	0.859
3 H, E, W	74.6%	59.6%	0.463	0.274	0.959	0.994
4 H, E, C	82.7%	56.1%	0.482	0.326	0.960	0.995
5 H, E	83.4%	56.7%	0.493	0.360	0.961	0.995
6 H, C	59.1%	15.0%	0.193	0.049	0.841	0.832
7 H, W	65.0%	57.6%	0.483	0.298	0.908	0.752
8 H	66.8%	55.1%	0.499	0.351	0.916	0.728

Table 4.1: Results for different evidence sets used by Evidence-HMM. The first column contains the used evidence sources in the test (H = emission + transition in HMM; E = Exonerate; W = RNAWeasel; C = comparative). The second column contains the number of perfectly predicted states. The third column contains the number of perfectly predicted sequences. Columns four and five contain sensitivity and specificity of exon prediction. Finally the sixth and the seventh columns contain the sensitivity and specificity of coding nucleotides (i.e., we are not considering codon positions)

To train and test the evidence-HMM, we have used 319 out of 323 gene sequences. Four sequences were discarded because they contained an intron with length of 1. The model was tested using a 6-fold holdout testing, which means that the data set was divided into six parts and iteratively five parts were used for training and one for testing. The statistics of all six tests were then joined together. We tested the model

Test	State match	Perfect	Exon		Coding nucl.	
			sens	spec	sens	spec
1 H, E, W, C	75.9%	71.4%	0.576	0.520	0.982	0.992
2 H, W, C	68.9%	59.2%	0.523	0.393	0.925	0.817
3 H, E, W	75.6%	70.5%	0.568	0.514	0.979	0.992
4 H, E, C	79.3%	71.7%	0.589	0.541	0.982	0.993
5 H, E	78.9%	70.8%	0.586	0.536	0.979	0.993
6 H, C	68.9%	58.6%	0.517	0.370	0.920	0.801
7 H, W	68.1%	55.4%	0.508	0.376	0.921	0.773
8 H	66.8%	55.1%	0.499	0.351	0.916	0.728

Table 4.2: Test with transition and nucleotide emission log probabilities multiplied by 30.

on all combinations of the given evidence (8 tests total) and found the most probable annotation using the Viterbi algorithm (see Table 4.1).

The model had a good sensitivity and specificity of coding nucleotides. The number of perfectly predicted exons was not very good. The number of predicted exons in the basic model was almost two times higher than in the correct annotation. The more information we used, the more jumps between exon states and intron states occurred because of the lowered weight of the transition probabilities. This is why comparative analysis did not improve in the basic model. This happened especially in the exon regions. We tried to fix this problem by multiplying selected log probabilities. The results were very positive as can be seen in the second test although the modifications were not optimal.

We have performed a set of alternative tests with modified log probabilities of state transitions and nucleotide emissions (see Table 4.2). This method partially simulates weights used by CRFs.

These tests clearly show that Exonerate is the most valuable source of evidence. This can be seen in tests 1 and 2 of table 4.2 where the accuracy of the predictions notably drops when Exonerate tape is omitted. Comparative analysis positively improved the prediction in every aspect compared to using only the testing sequence (tests 8 and 6). Evidence provided by RNAWeasel seems to have the smallest positive effect. This is mainly because the evidence effects relatively a small number of nucleotides. RNAWeasel combined with Exonerate gives worse predictions in this model than Exonerate alone. This is probably because RNAWeasel's true predictions are a subset of Exonerate's true predictions. All of these results are consistent with our previous conclusions.

We have also tested a state model with allowed ATA start codon (see Table 4.3).

Test	State match	Perfect	Exon		Coding nucl.	
			sens	prec	sens	spec
1 H, E, W, C	75.8%	69.5%	0.563	0.513	0.980	0.991
2 H, W, C	68.4%	56.7%	0.518	0.394	0.923	0.808
3 H, E, W	75.5%	68.9%	0.557	0.509	0.978	0.991
4 H, E, C	79.1%	69.9%	0.576	0.535	0.979	0.992
5 H, E	78.8%	69.2%	0.575	0.531	0.978	0.992
6 H, C	68.2%	56.1%	0.517	0.367	0.919	0.792
7 H, W	67.4%	52.3%	0.499	0.367	0.918	0.763
8 H	66.0%	51.7%	0.490	0.334	0.915	0.719

Table 4.3: Allowed start codon ATA. Transition and nucleotide emission log probabilities are multiplied by 30.

Only 5 out of 323 sequences contained start codon ATA. The results were quite similar although it is clear that in this model it is not optimal to allow it.

4.3 MtConrad results

We have tested MtConrad on a set of 323 genes. The model was tested using a 6-fold holdout (see Section 4.2) testing with different configurations of evidence features (see Table 4.4). The combination of Exonerate + RNAWeasel + HKY85 outperformed every other MtConrad test. This test predicts 78% of sequences perfectly and has 70% exon sensitivity. It has also the best intron sensitivity reaching 55%. The tests show that adding either RNAWeasel, or comparative analysis improves the prediction accuracy. This is in contrast with the results from Evidence-HMM where adding RNAWeasel tape has lowered the accuracy of the prediction. The accuracy of the prediction is notably better than with Evidence-HMM although characteristics for coding nucleotides are very similar (see Table 4.2). We have compared our results also with MFannot [1]. Because MFannot has not yet been published the results are hard to interpret. MFannot has a much better sensitivity and specificity of both exons and introns. MtConrad has a better sensitivity of coding nucleotides and perfectly predicted more sequences, which implies that MtConrad has a better accuracy on intronless genes.

Figure 4.3 illustrates improvements in accuracy achieved by MtConrad compared to the evidence tapes it uses.

Test name	Match	Perfect	Exon		Intron		Coding nucl.	
			sens	spec	sens	spec	sens	spec
1(Exonerate)	87.6%	75.2%	0.645	0.716	0.444	0.562	0.968	0.993
2(Exoner.+RNAW)	90.0%	75.9%	0.669	0.729	0.503	0.566	0.973	0.993
3(K80)	91.6%	76.5%	0.702	0.734	0.552	0.577	0.989	0.991
4(HKY85)	91.2%	78.0%	0.700	0.742	0.552	0.594	0.985	0.993
5(GY94)	90.0%	77.4%	0.680	0.729	0.483	0.538	0.979	0.990
6(MFannot)	93.0%	74.3%	0.766	0.809	0.820	0.820	0.936	0.998

Table 4.4: MtConrad test results and results of MFannot. The first column contains name of the test, the second column contains the number of state matches and the third column shows the number of perfectly predicted sequences. Columns 4-9 show sensitivity and specificity of exons, introns and coding nucleotides. Test 1 shows results for Exonerate evidence features. Test 2 is a combination of Exonerate and RNAWeasel. Tests 3, 4 and 5 use both Exonerate and RNAWeasel as well as one of the substitution models (whose name is in brackets).

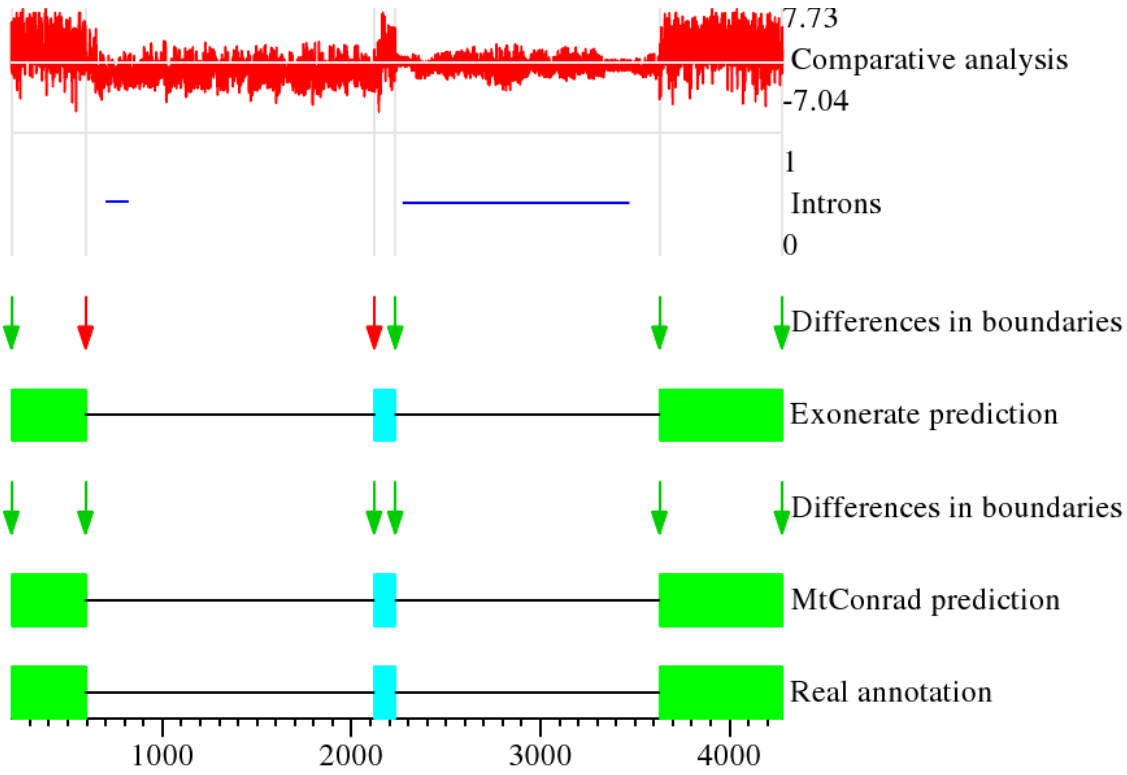

Figure 4.3: Visualization of MtConrad's input and output for gene COB from yeast Saccharomyces pastorianus. Exonerate tape provides a very good information about the positions of exons although it did not predict the splice sites very well (red arrows represent the incorrectly predicted splice sites). RNAWeasel tape (introns) is less sensitive but very specific. We can see that the information from comparative analysis approximates the positions of introns very well (higher values are more likely exon positions). In this instance MtConrad predicted the annotation perfectly.

Conclusion

In this thesis we have created and tested a tool for annotation of protein coding genes in yeast mitochondrial genomes called MtConrad. MtConrad is based on conditional random fields. Conditional random fields allow us to incorporate information from many information sources, even if it does not have a probabilistic interpretation. To produce accurate annotation, our tool combines information from several different external sources. We use Exonerate to align reference proteins extracted from model organisms to the genome being annotated and RNAWeasel to predict the positions of introns based on their characteristic structural motifs. We also use multiple alignment of mitochondrial genomic sequences from several yeast species to look for evolutionary signatures typical for protein-coding regions.

To estimate parameters of our model and to test its accuracy, we have assembled a set of 33 yeast mitochondrial genomes. First we test the accuracy of RNAWeasel and Exonerate on their own and then in various combinations in a simpler tool based on hidden Markov models called Evidence-HMM, which we have developed for this purpose. Finally, we test our main tool MtConrad and show that it predicts 78% of genes and 70% of exons perfectly.

We demonstrate that each of our external sources increases the prediction accuracy. We also show that MtConrad has a better accuracy than the Evidence-HMM. We have also compared our tool to MFannot, a recently developed tool that has not yet been published. We show that MFannot has a better exon and intron sensitivity than MtConrad while MtConrad has a better sensitivity of coding nucleotides and is better in predicting intronless sequences.

In future we plan to further improve prediction accuracy of our tool. We want to use support vector machine to improve the accuracy of the splice site prediction. We also want to incorporate additional information from RNAWeasel, in particular the categorization of the predicted introns to increase the accuracy of our position weight matrices.

The website of our tool is `http://compbio.fmph.uniba.sk/mtconrad/`. In future we plan to run our tool as a web service so that it can be conveniently used by life science researchers.

Bibliography

[1] MFannot. `http://megasun.bch.umontreal.ca/cgi-bin/mfannot/mfannotInterface.pl`.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 36(Database issue):D25–30, 2008.

[4] E. Birney, M. Clamp, and R. Durbin. GeneWise and Genomewise. *Genome Research*, 14(5):988–995, 2004.

[5] E. Bon, S. Casaregola, G. Blandin, B. Llorente, C. Neuveglise, M. Munsterkotter, U. Guldener, H. W. Mewes, J. Van Helden, B. Dujon, and C. Gaillardin. Molecular evolution of eukaryotic genomes: hemiascomycetous yeast spliceosomal introns. *Nucleic Acids Research*, 31(4):1121–1125, 2003.

[6] L. Bonen and J. Vogel. The ins and outs of group II introns. *Trends in Genetics : TIG*, 17(6):322–331, 2001.

[7] J. L. Boore, J. R. Macey, and M. Medina. Sequencing and comparing whole mitochondrial genomes of animals. *Methods in Enzymology*, 395:311–318, 2005.

[8] A. Brazma, H. Parkinson, T. Schlitt, and M. Schojatalab. A quick introduction to elements of biology - cells, molecules, genes, functional genomic, microarrays. `http://www.ebi.ac.uk/microarray/biologyintro.html`.

[9] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78–94, 1997.

[10] M. Csuros. Malin: maximum likelihood analysis of intron evolution in eukaryotes. *Bioinformatics*, 24(13):1538–1539, 2008.

[11] D. DeCaprio, J. P. Vinson, M. D. Pearson, P. Montgomery, M. Doherty, and J. E. Galagan. Conrad: gene prediction using conditional random fields. *Genome Research*, 17(9):1389–1398, 2007.

[12] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[13] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[14] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[15] D. Gautheret and A. Lambert. Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles. *Journal of Molecular Biology*, 313(5):1003–1011, 2001.

[16] N. Goldman and Z. Yang. A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Molecular Biology and Evolution*, 11(5):725–726, 1994.

[17] S. Griffiths-Jones, S. Moxon, M. Marshall, A. Khanna, S. R. Eddy, and A. Bateman. Rfam: annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33(Database issue):D121–124, 2005.

[18] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003.

[19] M. Hasegawa, H. Kishino, and T. Yano. Dating of human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 2:160174, 1985.

[20] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111120, 1980.

[21] I. Korf, P. Flicek, D. Duan, and M. R. Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17 Suppl 1:S140–148, 2001.

[22] M. Krzywinski, J. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra. Circos: an information aesthetic for comparative genomics. *Genome Research*, 19(9):1639–1645, 2009.

[23] B. F. Lang, M. J. Laforest, and G. Burger. Mitochondrial introns: a critical view. *Trends in Genetics : TIG*, 23(3):119–125, 2007.

[24] F. Lisacek, Y. Diaz, and F. Michel. Automatic identification of group I intron cores in genomic DNA sequences. *Journal of Molecular Biology*, 235(4):1206–1207, 1994.

[25] E. P. Nawrocki, D. L. Kolbe, and S. R. Eddy. Infernal 1.0: inference of RNA alignments. *Bioinformatics*, 25(10):1335–1337, 2009.

[26] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308313, 1965.

[27] S. Sarawagi and W.W. Cohen. Semi-markov conditional random fields for information extraction. *Adv. Neural Inf. Process. Syst.*, 17:11851192, 2005.

[28] G. S. Slater and E. Birney. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6:31, 2005.

[29] M. J. van Baren, B. C. Koebbe, and M. R. Brent. Using N-SCAN or TWINSCAN to predict gene structures in genomic DNA sequences. *Curr Protoc Bioinformatics*, Chapter 4:Unit 4.8, 2007.

[30] H. Wallach. Efficient training of conditional random fields. *In Proceedings of the 6th Annual CLUK Research Colloquium. CLUK, Edinburgh, Scotland.*, 2003.

[31] Z. Yang. PAML 4: phylogenetic analysis by maximum likelihood. *Molecular Biology and Evolution*, 24(8):1586–1591, 2007.

Abstrakt

Autor:	Bc. Juraj Meštánek
Názov diplomovej práce:	Software for Annotation of Protein Coding Genes in Yeast Mitochondrial Genomes
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci diplomovej práce:	Mgr. Broňa Brejová, PhD.
Rozsah práce:	52
Rok:	2010

V tejto práci prezentujeme softvér na hľadanie proteín-kódujúcich génov v mitochondriálnych genómoch kvasiek. Náš nástroj je založený na pravdepodobnostnom modeli zvanom conditional random fields.

Pomocou tohoto modelu kombinujeme informácie z rôznych zdrojov , aby sme dosiahli zvýšenú presnosť predikcie. Po prvé, zarovnávame proteíny z modelových ogranizmov k anotovanému genómu pomocou programu Exonerate. Po druhé, používame program RNAWeasel na predikciu charakteristických štrukturálnych motívov mitochondriálnych intrónov a pomocou tejto informácie určujeme ich pozíciu. Ako posledné používame viacnásobné zarovnania mitochondriálnych DNA sekvencií z rôznych druhov kvasiniek, v ktorých hľadáme regióny, ktoré sa vyznačujú vlastnosťami charakteristickými pre kódujúce časti génov. Tieto tri zdroje informácií spolu so skúmanou DNA sekvenciou tvoria množinu pozorovaní, ktorú používame v našom pravdepodobnostnom modeli na hľadanie exónov a intrónov.

Náš nástroj sme otestovali na sade 33 mitochondriálnych genómov, kde predpovedáme 78% génov a 70% exónov úplne správne. V budúcnosti plánujeme sprístupniť tento nástroj tak, aby sa jednoducho používal.

Kľúčové slová: hľadanie génov, mitochondriálne gény, conditional random fields, externé zdroje informácie