# A SIMPLE AND FAST HOMOLOGY-BASED GENE PREDICTION IN MITOCHONDRIAL GENOMES

Amirhossein Hajianpour

Thesis submitted to the University of Ottawa in partial fulfillment of the thesis requirement for the degree of

Master of Computer Science Specialization in

Bioinformatics

Faculty of Engineering

University of Ottawa

**Examining Committee**

The following served on the Examining Committee for this thesis.

| | |
|---|---|
| Internal Member(s): | Ashkan Golshani<br>Professor, Department of Biology, Carleton University<br>Hussein Al Osman<br>Associate Professor, School of Electrical Eng. and Computer Science,<br>University of Ottawa |
| Supervisor(s): | Marcel Turcotte<br>Professor, School of Electrical Eng. and Computer Science,<br>University of Ottawa |

## Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

**Abstract**

With the abundance of genomic data after the Human Genome Project, the need for analysis, and annotation of these data arise. Annotation of genomes helps us understand the functionality of different parts of the genomes of various species. In this thesis, we propose a simple, and fast homology-based gene prediction method called Exon Hunter (EH) that achieves a performance comparable with state-of-the-art methods in mitochondrial genomes. Mitochondria are crucial for a eukaryotic cell, and mutation in its DNA has connections with disorders such as Alzheimer and cancer. We used Hidden Markov Model (HMM) Protein Profile of a number of genes to search for protein-coding genes in different genomes. Our method forms every subset of the hit set, and calculates a score for each subset according to an objective function. Then it chooses the subset with the highest score. Finally, we analyze the codon usage bias of our dataset, and we discuss how it can help us improve this prediction. ExonHunter is written in Python and is publicly available on github.com/amirh-hajianpour/ExonHunter.

**Acknowledgements**

I would like to firstly thank Professor Marcel Turcotte for all he has done for me. Everything would be impossible without his help from my admission to the University of Ottawa, to my graduation. I hope that some day I pay back what I owe to you. I also want to show appreciation to Professor Franz Lang for introducing the problem, providing the data, and fruitful discussions. I finally want to thank Alibek Kruglikov, Sajad Norouzi, and Ali Sattari for their kind help throughout the thesis.

## Dedication

This is dedicated to those who give hands to the ones in need.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background and Motivations

Since 2004, when the *Human Genome Project* published the first version of a human genome, the efforts to produce more and more sequences increased. The 1000 Genomes Project launched in 2008 [17], and shortly after that, the 1000 *Plant Genomes Project* started [44]. Similarly, after only one year, 10,000 *Vertebrate Species Genome* [50] sequencing began. GenBank reports that the base pair database increases about 40% each year [9]. Therefore, a huge amount of raw and unannotated data became available [77]. Today, the genomes of thousands of species are ready to be analyzed, and every day new species and genomes are being added to the genome databases (e.g. ncbi.nlm.nih.gov and genomesonline.org). This abundance of genomic data is the result of continuous improvement of sequencing techniques. Over the past decades, genome sequencing has improved dramat-

ically from *First-Generation DNA Sequencing*, which used to cost about 100 million US dollars and took almost 15 years, to *Second-Generation DNA Sequencing* (Next-Generation Sequencing a.k.a. NGS) that decreased the cost by 100 fold, and can be achieved in almost two months [42]. Recently, *Third-Generation DNA Sequencing* (Single Molecule real-time Sequencing a.k.a. SMRT) has been proposed. Whenever a new technique is designed, its error rate, time, and cost are all diminishing [35].

Sequences, alone, do not provide much information for any deduction. This amount of uncharacterized data necessitates annotation. Genome annotation is the process of labelling different segments of a genome [29]. This step is crucial for understanding the role of each part of a genome. In other words, genome annotation is identifying functional elements of a genome. Genome annotation can be classified in three different levels: nucleotide, protein, and process [70]. The focus of this thesis is the nucleotide-level genome annotation. At nucleotide-level, gene prediction is one of the main keys to get a better understanding of a genome and its functionality. Many computational gene prediction methods have been proposed in the last decades in order to automate the annotation process.

In order to understand the processes that we will design and implement, it is necessary to become familiar with the basic biology of the problem. Here, we cover the definition of a number of terms in biology, and also talk about the properties of some of the biological mechanisms that are important to us to solve the problem of genome annotation.

### 1.1.1 Genome, Chromosome and DNA

A *genome* is the collection of all the genetic materials of an organism. This means, it contains all the genetic information that is needed to be transferred from a parent, to its offspring [29].

A genome consists of the DNA of an organism. It includes both, the coding sequence (genes) and non-coding sequences. Different species have different forms of DNAs. Some have nucleus DNA, some species have mitochondrial DNA, and photosynthetic organisms have chloroplast DNA. There are organisms that have more than just one form of DNA (e.g. eukaryotes have both nucleus and mitochondrial DNA). Mitochondrial DNAs are smaller than nuclear DNAs. The smallest eukaryotic genomes are about 10 Mb long, and the largest ones can reach 100,000 Mb in length, while prokaryotic genomes are smaller than 4 Mb [11].

*Chromosomes* are the organizing units where the nuclear DNA is located. Chromosomes are stored in the nucleus of a cell, see Figure 1.1. In eukaryotes, they contain the larger portion of the genetic information compared to mitochondrial DNA. While many prokaryotes have a single chromosome, human cell has 22 autosome pairs and 2 sex chromosomes [11].

*Deoxyribonucleic Acids* or DNAs are the molecules that collectively form the genome. In eukaryotic organisms they are linear, and in prokaryotic organisms they typically have a circular shape. These long molecules are composed of two polynucleotide chains and fold around each other. This folding forms a special structure that is called a double helix [78].

[1] https://socratic.org/questions/how-are-dna-chromosomes-genes-and-alleles-related

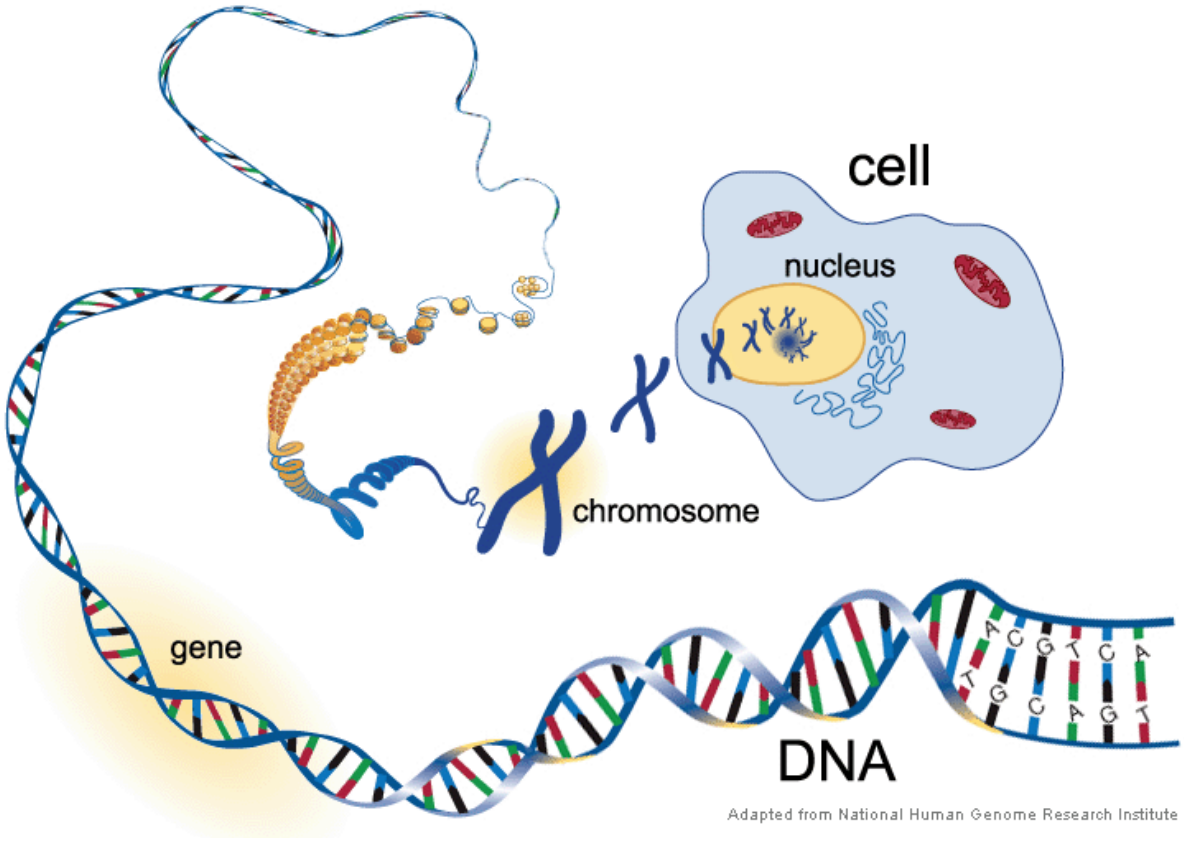


Figure 1.1: A eukaryotic cell. The nucleus, chromosomes, histones, DNA, and nucleotide base pairs are shown in the picture.[1]

This double helix shape has two strands that are placed in the opposite directions. Each strand is a sequence of nucleotides. A base pair is two nucleotides that are connected with hydrogen bonds in opposite strands, and face each other [11]. There are four types of nucleotides that form a DNA sequence, each of which is represented by a character: *Adenine* (A), *Thymine* (T), *Cytosine* (C), *Guanine* (G). These two strands are complementary, and they run in opposite direction. It means that A with T, and C with G constitute a base pair. This property of DNA makes it possible for a cell to infer the sequence of the opposite

strand, using only one strand, according to the base pairing rule that is mentioned above. Therefore, in order to represent a DNA sequence, the sequence of one of the strands is sufficient. As mentioned before, DNA segments can be divided into non-coding regions, and coding regions (genes).

### 1.1.2 Gene

Gene annotation involves finding the starting and ending position of genes. This is because not all the regions of DNA code for genes. Gene annotation can also be defined as separating coding regions (exons) from non-coding regions (intron). Here, we talk about the structure of genes, and what they are made of.

There are multiple definitions for what constitutes a gene [27, 53]. But in simple words, a gene is a discrete segment of a DNA that has the instructions for making proteins or non-coding RNA molecules. This thesis focuses on protein-coding genes on Since proteins and RNA molecules are responsible for a vast majority of processes, such as catalytic, structural, signalling, and regulatory roles in a cell, genes are extremely important. Genes are the units that possess the genetic traits and information [31].

In 1970, *Francis Crick* published the revolutionary paper, *Central Dogma of Molecular Biology* [18], and explained the relationships between DNA, RNA, and Protein. Gene expression is the process of reading the instructions from DNA in order to synthesize a protein [52]. The first step in reading this instruction is *Transcription*. Transcription is the process of copying the information of a gene into a molecule called *Ribonucleic Acid* or RNA, since nucleotides in RNA are ribonucleotides. RNA polymerase is the main protein

that transcribes DNA to build RNA. With a promoter followed by a start codon being at the beginning of a gene and acting as a start site and a terminator followed by a stop codon that is at the end of a gene and specifies the stop site, *RNA polymerase* is able to know where to begin and where to stop. The window that starts from a start codon and ends with a stop codon in the same frame is called *Open Reading Frame* (ORF) [2]. The complementary sequence of nucleotides will be copied to RNA from DNA with only one exception; instead of Thymine (T) there is *Uracil* (U) in RNA. RNA is a single strand while DNA is a double helix strand. Since DNA has two strands that are complementary, and the fact that RNA is built using only one strand of DNA, RNA will have the opposite sequence of the strand that is built on (Template Strand); it will have the same sequence of the other strand (Coding Strand). It is also worth mentioning that RNA is just a copy of a gene of a DNA; genes are just a very small region compared to the whole genome.

The RNA that is just transcribed and has not been modified is called *pre-mRNA* which stands for *precursor messenger RNA*. It is a precursor because it is not modified yet to be used in Transcription, and it is an intermediary compound that will eventually become mRNA. An RNA might be the final product of the gene expression process, or it might go through the next step which is the synthesis of a protein.

Before the beginning of the translation step, pre-mRNA needs to be modified. As mentioned before, a pre-mRNA consists of both non-coding regions called *introns* (intervening sequences), and coding regions called *exons* (expressed sequences). Although it might seem wasteful to have regions that will not code into protein and will be removed, introns play an important role in synthesizing different proteins from the same RNA sequence through a mechanism called *Alternative Splicing* [28]. Introns should be removed, and exons should

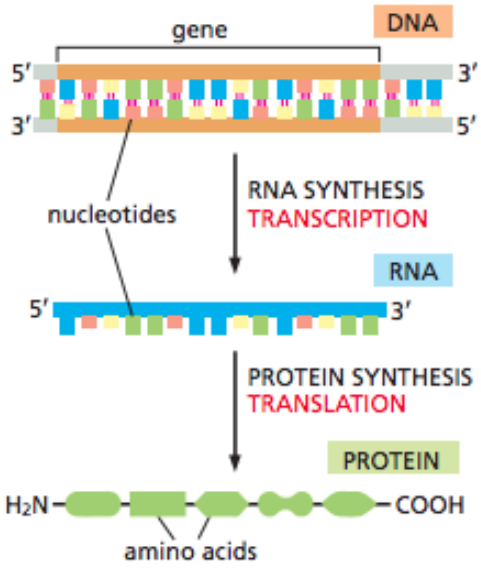


Figure 1.2: The two main steps of protein Synthesis: Transcription and Translation [2]

be stitched together in a process called RNA splicing, see Figure 1.3. After the RNA splicing mechanism, mRNA will leave the nucleus [2].

The transformation of mRNA to protein is called *Translation*. As we know already, there are only 4 nucleotides; but the number of amino acids that are units of proteins is 20. This shows that this translation cannot be a one-to-one translation. According to the *Pigeonhole Principle*, at least three nucleotides are needed for this translation. It is obvious that two nucleotides can only code for $4 \times 4 = 16$ amino acids. If translation is from three consecutive nucleotides, which is called a *codon* (triplet) [81], to one amino acid, multiple codons should code for the same amino acids. This is because a three-nucleotide sequence has $4 \times 4 \times 4 = 64$ different combinations, but there are only 20 amino acids. This 3-to-1 translation follows a set of rules which is called the genetic code [19], see Figure 1.4.

[2] https://www.genome.gov/genetics-glossary/Intron

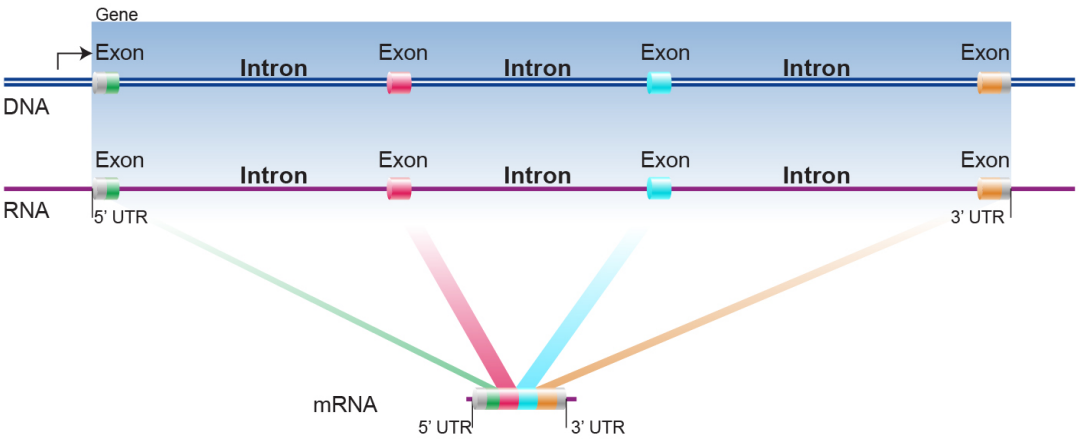


Figure 1.3: RNA Splicing[2]

Small RNA molecules called *tRNA* are responsible for binding to codons and linking their corresponding amino acids together, and in a sense translating the codons. When the process reaches the end of the mRNA, the protein has been made and will be released [2]. Figure 1.2 illustrates the two main steps of protein synthesis.

The coding sequence of an mRNA can be read in three different Reading Frames. It can start from the first, second, or third nucleotide of the coding sequence of an mRNA. If it starts from the fourth nucleotide, it means it just skipped the first codon, but started from the first nucleotide. It is obvious that each frame can code for very different proteins, but only one of the frames produces the right one. Therefore, genome annotation also includes the specification of the reading frame of each exon.

[3] https://openstax.org/books/biology/pages/15-1-the-genetic-code

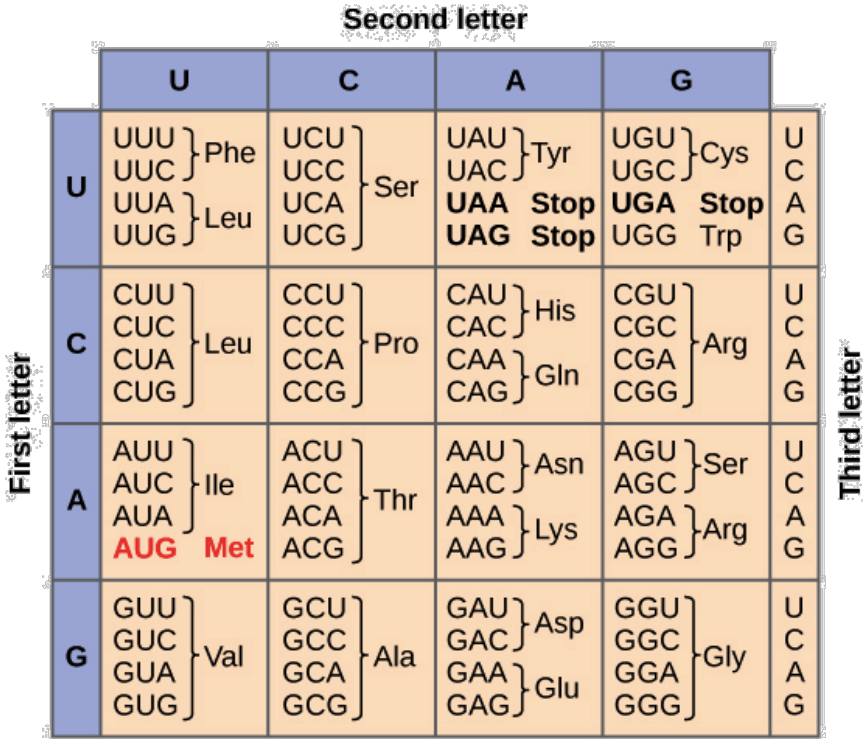


Figure 1.4: The standard genetic code. The genetic code for Fungi is different from this code. In Fungi "UGA" codes for Tryptophan (Trp) instead of coding for Stop Codon [3].

### 1.1.3 Mitochondrial DNA

Since the type of DNA that we will be using as our dataset is Mitochondrial DNA, we will give a brief explanation of what mitochondrial DNA is and why we are interested in studying it.

Mitochondria are organelles that exist in almost all the eukaryotic cells. Mitochondria play an important role in many biological processes, and its dysfunction leads to serious consequences. Since mitochondria are passed down from mother to offspring, mutations in their DNA are transmitted to the next generation as well, and this adds to their impor-

tance. Recent studies show the relationship between differences in mitochondrial genome and multiple common diseases. For every 100,000 humans, there are more than 10 people with mitochondrial diseases and disorders [15]. In addition, mitochondria and mitochondrial DNA (mtDNA) have been shown to have a connection with many common diseases such as Cardiovascular Diseases, Alzheimer, Cancer, Parkinson, and Kidney diseases. Furthermore, mtDNA is useful in tracking patterns of gene flow and anthropological genetics. In another word, mtDNA is used as an additional evidence to build a more accurate image of evolution [5]. Therefore, studies and developing methods to analyze mitochondrial genomes can help scientists to propose treatments and preventative strategies that will improve the health of many individuals [2, 15]

## 1.2 Markov Chain, HMM

There are many genome annotation programs that use Markov Models, and Hidden Markov Models (HMM). In addition to that, HMMER which plays a fundamental role in this thesis is based on HMM. Here, we present the definition, and formulation of the aforementioned terms.

Markov Model (MM) [56] is one of the most widely used statistical approaches especially in modelling biological sequences. A Markov Model is a stochastic model in which the future states only depend on current states. In bioinformatics, it can be defined as a stochastic process in which the probability of a particular nucleotide occurring at a specific position is dependent on the $k$ previous nucleotide(s). The order of such a model is $k$. An order 0 MM assumes that each nucleotide is not dependent on any previous nucleotide and

occurs at an arbitrary frequency. If $k$ is large, MM can model the dependence of farther neighbouring positions. It is obvious that the basic assumption of such an approach is that in a sequence of stochastic events in a process, a random variable is predictable using $k$ previous random variable. Many gene prediction methods are built using different versions of MM. GeneMark, GENESCAN, FGENESH, GENIE, TWINSCAN and Glimmer are some of the well-known programs that use this family of approaches [21, 25, 67]. We will thoroughly talk about gene prediction methods in chapter 2.

A Markov chain is a special case of a Markov process where the state space is finite. A Markov process is a stochastic process in which:

- The number of states or outcomes is finite
- The probabilities are constant over time
- It satisfies the memory-less property:

$$P\{X(t_{n+1}) = x_{n+1}|X(t) = x_n...X(t_1) = x_1\} = P\{X(t_{n+1}) = x_{n+1}|X(t) = x_n\} \qquad (1.1)$$

A Markov chain is defined as a process that starts from one state and moves to another step, consecutively. Each move is called a step. If a Markov chain is in state $s_i$, then it transits from that state to $s_j$ with the probability of $p_{ij}$. Each of these probabilities is called transition probabilities. They determine how probable it is to move from one state

$$\begin{bmatrix} A & 0.3 & 0.2 & 0.2 & 0.3 \\ T & 0.1 & 0.2 & 0.4 & 0.3 \\ C & 0.2 & 0.2 & 0.2 & 0.4 \\ G & 0.1 & 0.8 & 0.1 & 0 \end{bmatrix}$$

Figure 1.5: An example of a transition probability matrix for the above Markov chain. As can be seen, the summation of every row is 1.

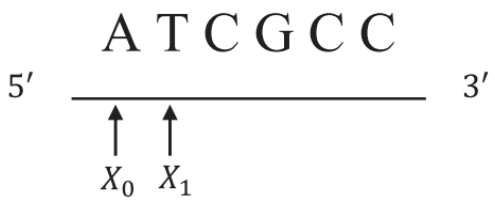


Figure 1.6: A first-order Markov chain for a DNA sequence.

to another. The matrix of all these $p_{ij}$ probabilities is called the transition probability matrix of the Markov chain, see Figure 1.5.

DNA sequences consist of 4 different bases A, T, C, G. One state for each base can be considered, which means our Markov chain has four states. Again, the underlying assumption is that a nucleotide in a position is determined by the adjacent nucleotides. Below, an example of a first-order Markov chain is presented. In the first-order Markov chain, every nucleotide is only dependent on the previous nucleotide.

In the example of Figure 1.5 and Figure 1.6, the nucleotide located at the $X_1$ position can be calculated by only knowing the nucleotide at the position $X_0$. It means, the probability of finding a sequence like $z = \{X_1 = T, X_0 = A\}$ can be calculated as follows:

$$P\{z\} = P\{AT\} = P(A|s_0) \times P(T|A) = 0.25 \times 0.2 = 0.05 \tag{1.2}$$

, where $s_0$ is the initial state and have uniform distribution.

An HMM [6] can be characterized using 4 elements (S, V, A, B) as below:

- S=$\{S_1, ..., S_N\}$ are the N states. The states are hidden and cannot be observed.
- A=$(a_{ij})_{i \in S, j \in S}$ is the transition probability of moving from state $S_i$ to state $S_j$. It is expected that $a_{ij} \in [0, 1]$ for each $S_i$ and $S_j$ , and that $\sum_{i \in S} a_{ij} = 1$, for each $S_j$.
- V=$\{V_1, ..., V_M\}$ is the vocabulary, the set of symbols that may be emitted
- $(b_{ij})_{i \in V, j \in S}$ is the emission probability that symbol $v_i$ is seen when we are in state $S_i$ [6]

HMM Profiles (HMM Protein Profile) are typically used for finding significant protein sequence similarity. In other words, they can find homology relations between sequences. In this application of HMMs, a Multiple Sequence Alignment (MSA) is represented as a Hidden Markov Model, see Figure 1.8. In a state-based model like HMM, the evolutionary fingerprints of a multiple sequence alignment can represented effectively. Besides modelling the conserved columns (positions) of a multiple sequence alignment, HMM Profiles also model insertions and deletions. They use position dependent gap and substitution probabilities which are claimed to model the real biological feature in a better way [22].

[4] https://en.wikipedia.org/wiki/Hidden_Markov_model
[5] https://www.ebi.ac.uk/training-beta/online/courses/pfam-creating-protein-families/

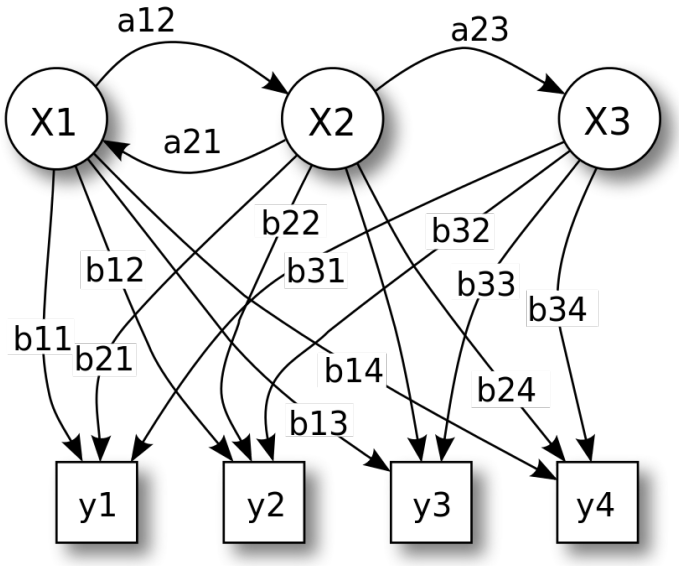


Figure 1.7: An example of a Hidden Markov Model (HMM) with 3 hidden states [x1-x3], 4 observations [y1-y4], state transition probabilities ($a_{ij}$) and output probabilities ($b_{ik}$).[4]

## 1.3 Synonymous Mutation and Codon Usage Bias

In Chapter 6, we will use codon usage as a feature of matches to reach a better classification of true hits and false hits. Therefore, in this section, we will explain what is needed to be understood about codon usage bias.

A synonymous mutation is a mutation that does not change the encoded amino acid. As Figure 1.4 shows, if a point mutation happens in the third nucleotide of Phenylalanine codon (e.g. "UUU"), the mutated codon would still code for Phenylalanine (e.g. "UUA"). These mutations have no effect on the sequence or functionality of proteins. However the frequency that they are used is different. This difference in usage of codons is called Codon

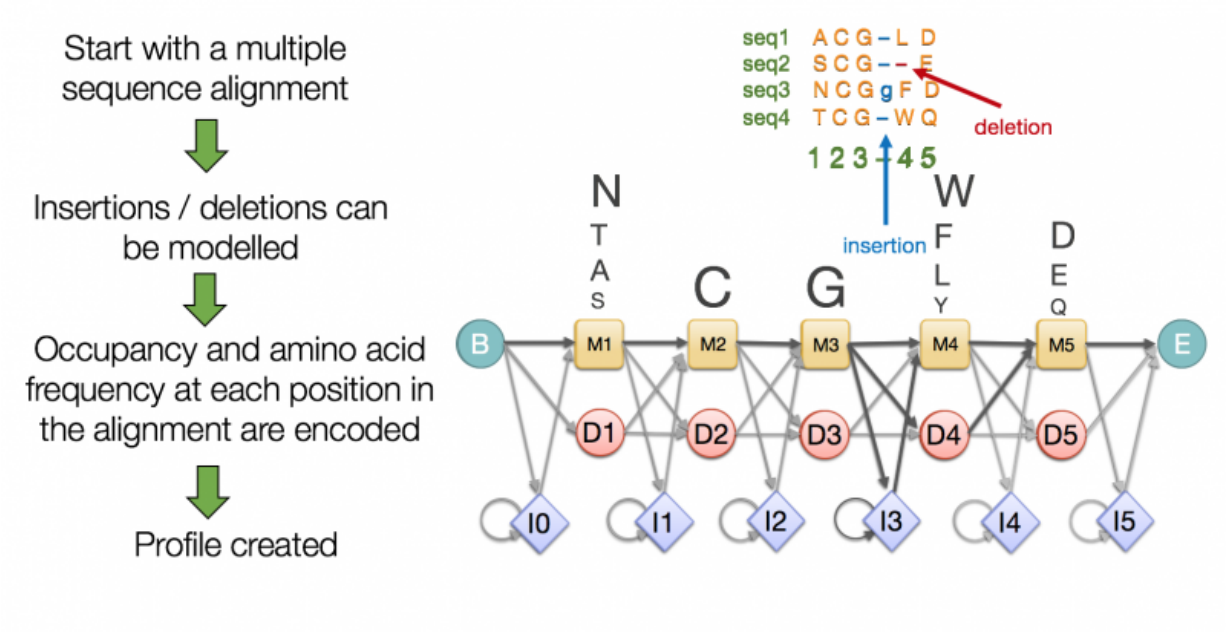


Figure 1.8: An example of modelling a multiple sequence alignment of 4 sequences with 5 consensus nucleotide positions with an HMM Protein Profile.[5]

Usage Bias (CUB) or shortly Codon Bias.

There are two different hypotheses why codon usage bias exists. Mutational Hypothesis claims that the underlying mutation processes are the reason for such bias [4]. Selective hypothesis, on the other hand, assumes an advantage for this preference [63]. According to this hypothesis, organisms that have such preference will be promoted, and the other group of organisms will be suppressed.

CUB is very different in various organisms. However, closely related species typically tend to have approximately similar profiles [64]. CUB can also vary throughout the genome, or even genes of a species [54]. Also, there are studies which show that CUB can be tissue specific [55]. There have been many studies that suggest a significant positive correlation

between Codon Usage Bias and expression level of a gene [30, 34]. Likewise, there is a remarkable negative correlation between expression level of a gene and the rate of synonymous substitutions between divergent species [8, 62, 65]. These correlations are justified by the selective hypothesis. Evolution explains that highly expressed genes are more important for the species. Therefore, these genes, and their CUB are under a strong selective pressure. This argument leads to the fact that the translation process is selective [54, 64].

### 1.3.1 Measures of Codon Usage Bias

Measuring the Codon Usage Bias of any given sequence is the task of representing how biased the amino acids are coded in that sequence. Many CUB measuring methods have been proposed over the last several years. The most straightforward way of calculating how the codon usage of a sequence is biased might be to calculate the Euclidean distance between codon usage of that sequence, and a uniform distribution.

Let $U$ be a uniform distribution, consisting 64 equal data points, and $Q$ be the frequency of the 64 codons of the query sequence. CUB can be calculated as:

$$Q = [q_1, q_2, ..., q_{64}] \tag{1.3}$$

$$U = [u_1, u_2, ..., u_{64}] = [1/64, 1/64, ..., 1/64] \tag{1.4}$$

$$CUB_i = \sqrt{\sum_{i=1}^{64} (u_i - q_i)^2} \tag{1.5}$$

But of course, since longer genes would have higher frequency values for each codon, it

is necessary to normalize the query codon frequency. So, normalized query codon frequency that is denoted by $Q_n$ can be defined as follows:

$$Q_n = [\frac{q_1}{\sum_{i=1} q_i}, \frac{q_2}{\sum_{i=1} q_i}, ....\frac{q_6 4}{\sum_{i=1} q_i}] \tag{1.6}$$

The number of each amino acid can also be a problem in this calculation. If an amino acid has a high frequency, the codons that code that amino acid would be more than other codons. Therefore, this would lead to the bold impact of frequent amino acids in calculating this measure. Let $o_{ac}$ be the count of codon $c$ in amino acid $a$, and $C_a$ be the set of codon coding for amino acid $a$. Then, the bias of codon $c$ in amino acid $a$, that is represented as $b_{ac}$ can be calculated as the statement below:

$$b_{ac} = \frac{o_{ac}}{\sum_{c \in C_a} o_{ac}} \tag{1.7}$$

Relative Synonymous Codon Usage (RSCU) that is one of the widely used CUB measurements uses this approach with one difference. RSCU divides each codon count with the average value of codon count for each amino acid. Let $ka$ be the number of synonymous codons that code for amino acid a, then RSCU is defined like this:

$$b_{ac} = \frac{o_{ac}}{\frac{1}{k_a} \times \sum_{c \in C_a} o_{ac}} = \frac{O_{ac}}{\overline{o_{ac}}} \tag{1.8}$$

Another approach is to use entropy. Shannon's Entropy [60] is the average amount of information or uncertainty that a discrete random variable possesses. Let $X$ be a discrete

random variable, with possible outcomes of $x_1$, $x_2$, ..., $x_n$ that have probabilities $p_1$, $p_2$, ..., $p_n$, respectively. Entropy of this random variable is obtained as follows:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \times \log_2 P(x_i) \tag{1.9}$$

An intuitive way of obtaining CUB using entropy is to calculate entropy for codon counts of each amino acid separately, and then get the average value of entropy. There are known methods that use entropy for calculating CUB. Weighted sum of relative entropy (Ew) [72] is one of these methods. $E_w$ is the sum of the relative entropy of each amino acid, weighted by its relative frequency in the:

$$E_w = \sum_{a \in A} F_a E_a \tag{1.10}$$

$$E_a = \frac{H_a}{max(H_a)} = \frac{H_a}{\log_2 k_a} \tag{1.11}$$

The other method is Synonymous Codon Usage Order (SCUO) [76]. This method is almost identical to EW, but instead of relative entropy, it uses the normalized difference between the maximum entropy and observed entropy:

$$SCUO = \sum_{a \in A} F_a E_a \; E_a = \frac{max(H_a) - H_a)}{max(H_a)} = \frac{\log_2 k_a = H_a}{\log_2 k_a} \tag{1.12}$$

The methods that we described do not have any biological foundation. There is another group of methods that use a type of biological information. Most of these methods select

a set of genes and on this specific set they calculate how biased the codon usage is. One example is the Codon Adaptation Index (CAI) [64] which is the most used codon usage index. CAI is a geometric mean-based algorithm. CAI defines the codons that are frequently found in highly expressed genes as translationally optimal codons. Then it forms a subset of these translationally optimal codons. After that, it calculates the deviation of codon usage with a method called Relative Adaptiveness, between the set of translationally optimal codons and codon usage of a query sequence. This deviation is called CAI. Surely this method needs extra information which is the expression levels of genes in the sequences. However, gene expression levels can be different in genes and genomes of different species. Therefore one of the drawbacks of using such a method is that the reference profile that is used might not be the pattern that needs to be in the query sequence [14].

For this work, we used the average of entropy over the amino acids to calculate how biased the codon usage is. We used the entropy package of scipy[6]. One implementation detail that is worth mentioning is that if a sequence (typically a small sequence) lacks one or multiple amino acids, there will be some amino acids with codon usage vectors of 0. The package returns 'nan' for vectors of 0, therefore, we eliminated these amino acids in our work. Moreover, in all the codon usage bias studies start codon, stop codon, and amino acids with only one codon are removed. Therefore, in addition to start codon and stop codon, Methionine was removed because it was the only amino acid with one codon.

[6] https://www.scipy.org/

### 1.3.2 Codon Usage Distance

Other than CUB, the distance between two codon usages can be of importance. In order to compare CUB between different genes of a genome, or even CUB in different species a distance measure can be helpful.

One of the advantages of entropy is that its definition can be extended to be considered as a distance. Before explaining how we can use entropy as a distance, let's define what a Cross Entropy is. Cross Entropy is basically just the quantification of the difference between two probability distributions, and can be calculated as follows:

$$\text{Cross Entropy, } H(p,q) = -\sum p(i)\log(q(i)) \tag{1.13}$$

Where $p$ is the Probability Distribution, and $q$ is the Predicted Probability Distribution. Obviously, if our prediction is perfect, $q$ would be $p$ and Cross Entropy would be the same as Entropy. But if the prediction is different, cross entropy will be greater than entropy. This amount of difference between entropy and cross entropy is called Relative Entropy or Kullback-Leibler Divergence (KL-Divergence).

$$\text{Cross Entropy} = \text{Entropy} + \text{KL} - \text{Divergence} \tag{1.14}$$

$$D_{KL}(p||q) = H(p,q) - H(P) = -\sum_i p_i \log(q_i) - (-\sum_i p_i \log(p_i)) \tag{1.15}$$

$$D_{KL}(p||q) = -\sum_i p_i \log(q_i) + \sum_i p_i \log(p_i) = \sum_i p_i \log\frac{p_i}{q_i} \tag{1.16}$$

As the last term suggests, KL-Divergence is not symmetric. It means that:

$$D_{KL}(p, q) \neq D_{KL}(q, p) \tag{1.17}$$

The same entropy package provides an implementation of KL-Divergence, as well. There's only one problem that needs to be addressed. If the second vector (q) contains 0s, KL-Divergence will be infinity. In order to resolve such problem, we add an $\epsilon = 10^{-10}$ (epsilon) to the second parameter of this function. So that the denominator is never 0.

## 1.4 Problem Description

The main problem that we will address in this thesis is the task of finding genes in a given genome. In this problem, finding a gene is in fact finding the starting and ending positions of the coding regions (exons) of a protein-coding gene in mitochondrial genomes. Of course, finding the coding regions also means finding the non-coding regions (introns), as well.

Gene prediction is a challenging process. Only a small part of the genome is coding. That is why generally gene prediction methods suffer from high False Positive rates. On the other hand, some genes have small regions that disturbs the prediction. Small introns might make the algorithms account for only one exon. However that exon is in fact two exons separated by a small intron. In addition, the detection of small exons is hard because the signals and signatures that make this identification possible are difficult to identify for these methods. A big fraction of gene prediction programs have difficulty in specifying

the starting and ending regions (splice sites) precisely [45]. They typically either miss the junction, and under-predict these sites (a part of the coding part is left out of the boundary specification) or consider a part of non-coding regions and over-predict those junctions (a part of non-coding regions is added to the predicted coding segment). Identification of fragmented genes is another challenge that many programs do not address. These genes are composed of multiple fragments that are far from each other.

## 1.5 Contributions

We received 12 mitochondrial genomes of Fungi from our collaborator at the University of Montréal, Professor Franz Lang. We aligned 83% of the protein sequences that their HMM profile were missing, using T-Coffee. We built HMM Protein Profile from the Multiple Sequence Alignment that we obtained from T-Coffee, using the "hmmbuild" command of HMMER. Then we translated the genome of each species into the 6 reading frames. We ran the "hmmsearch" command of HMMER on the 6 reading frames of each genome, to search for the protein sequence that HMM represents. Then a search tree was built in order to calculate the best set of hits (candidate exons) that "hmmsearch" gives, according to a scoring function. A software system called ExonHunter (EH) was developed to automate this process. This procedure was done for all the genomes and their genes. Moreover, we studied the effect of using codon usage bias information in order to improve the performance of Exon Hunter. And, finally, we examine how splice-site prediction methods can help the gene prediction process.

# Chapter 2

# Gene Prediction

In this chapter, gene prediction approaches, programs and their performance have been described. This gives us a good view of what the state of the art is in the genome annotation literature, and what is considered a good or a bad performance and how it can be measured.

## 2.1 Challenges

As mentioned before, with the abundance of genomic sequences, the need for computational methods that detect genes increased. Gene prediction is a very important problem in computational biology, since it can provide information about the product of gene expression. The fact that the final product of gene expression is also the beginning of many other analyses in biology adds to the importance of this task [77].

In prokaryotes, this is a relatively easy problem because these organisms don't have

introns in their mRNAs and typically mRNA translates into protein without being modified. Genes in these organisms usually start with a start codon (AUG) and end with a stop codon (UGA, UAG, UAA) in the longest ORF of the area [77].

In eukaryotes, on the other hand, with the presence of introns in mRNAs that causes arbitrary stop codon sequences in introns to be accounted as an exonic stop codon, this problem becomes more challenging. One of the other features of eukaryotic genomes that makes it harder to find genes is the existence of short regions. Short introns are more probable of not being recognized as non-coding regions. Similarly, since the signals of short exons are less significant, they might be missed too. In addition, the collective coding region of a genome is remarkably smaller than the whole sequence of DNA. In a prokaryote like Haemophilus influenzae, 85% of the genome is coding. In contrast, in fly and worm this number decreases to 25% of their genome [1]. The non-coding bulk of DNA is sometimes called junk DNA.

## 2.2 Approaches

There are some assumptions that many gene prediction programs have: (1) No exon has overlap with any other exon in the same genome. (2) Length of any exon is divisible by 3. (3) There is only one stop codon in any exon, and it is located at the end of the exon. The first assumption does not have an exception, and it applies to every program. It should be mentioned that alternative splicing, which is the mechanism to synthesize more than one protein from the same gene, does not contradict with this assumption, although it is harder for gene prediction programs to address this matter. The second assumption is sometimes

not true because of a phenomenon called split codon that will be described later. Finally, the last supposition is not globally correct and there are cases in which stop codons happen in the middle of an exon. This assumption will be addressed later, as well.

There are two main classes of computational gene prediction methods: similarity-based methods that look for similar sequences in different genomes; and signal-based methods that focus on nucleotide compositions and characterization of the structure of a sequence, and are also referred as *ab initio*.

### 2.2.1 Ab Initio Gene Prediction

For deciding if a region is coding or not, *ab initio* algorithms look for sequence compositions and signals that exist in either of these regions. Therefore, they try to extract features and tune them in a way that helps them make this decision [1]. One of the main reasons that make *ab initio* gene prediction methods important is the fact that only about half of the genes can be found from homology to other known genes or proteins [45].

One of the features that can be used for this distinction is Splice Site. Splice site is the junction of the intron-exon or exon-intron. *Canonical splice site* is a sequence of a junction that is highly conserved. The canonical splice site for the donor site (exon-intron junction) is [AG—GTRAGT, where R = A or G] and for acceptor (intron-exon junction) is [YYTTYYYYYYNCAG—G, where Y = C or T and N = A, C, T or G]. In these sequences, R represents *Purines* and Y represents *Pyrimidines*, and N is for any nucleotide. Among these conserved positions, GT at the beginning of an intron, and AG at the end of an intron are stronger [45].

Another deciding factor in this group of methods is sequence composition. Generally, the regions that are GC-rich are probably coding regions (exons). On the other hand, intronic regions are AT-rich [51].

The codons that encode for the same amino acid are called synonymous codons. In different species and different genes, these synonymous codons are not used equally. A preference ratio that is called codon bias (or codon usage) has been seen. Because of mutational biases, and differences in the strengths of natural selection on translation optimization, codon usage has been used to find prokaryotic genes in methods such as GENEMARK [43], GLIMMER [20], Prodigal [33]. The percentage of predicted genes in early prokaryotic gene predictors was about 95%, although their accuracy in finding the starting point of a gene was around 80%. In eukaryotes, predictors like GENEID [32] and GENESCAN [13] were the first programs that used start, end and splice site signals, as well as order five Markov Chains, and codon bias to find genes. The first eukaryotic gene predictors did not have more than 80% true coding nucleotide, while a large part of what they were predicting was non-coding regions [45]. We will thoroughly explain the performance measures in section 2.4.1.

Some other *ab initio* programs tried to increase their performances by using fragments or full/length mRNAs, and proteins and aligning them to the target genome, in addition to the signals that were mentioned previously. The alignment was being done using algorithms such as: Smith-Waterman [68], Needleman-Wunsch [47] and BLAST [3]. EXONERATE [66], PROCRUSTES [26], and GENEWISE [10] were three prevalent algorithms that used this strategy [45].

### 2.2.2 Homology-Based

The second group of the gene prediction methods is Similarity-Based or Homology-Based approach. Similarity-based programs are making use of cDNA, EST (Expressed Sequence Tag), proteins, or even genomes of the same species or closely related ones. The programs that use proteins rely on the fact that protein/coding regions are conserved between species. Multiple studies show that closely related species have more similar genome structure. In other words, these algorithms are looking for homologous genes. *Homologous genes* are the genes that share a common origin. They are mainly divided into two categories: *Orthologous Genes* are genes that are originated from a single ancestral gene in the last common ancestor, and *Paralogous Genes* are the ones that are related via gene duplication [36]. The similarity-based methods compare a target genome with a query genome, cDNA, EST, mRNA or protein sequences. The target genome is the genome that needs to be annotated. The query sequences have been derived from the manual annotation of the same organism or a closely related species. BLASTX was one of the first programs that used this methodology by translating the query genome to all six possible frames, and searching for the proteins to find the coding regions of the genome. Later, similarity-based methods tried to use *ab initio* content and signals in order to specify gene structures. Although genomes structure are proved to be similar in close species, they are not completely conserved; therefore *ab initio* information could help similarity-based methods, especially to find junctions. EXONERATE, GENEWISE, and PROJECTOR [46] employed this process for annotating a genome. GENOMESCAN [82] is a program that uses a full *ab initio* step as well as a similarity-based step. The score of a candidate exon will increase if it finds

a similarity between the candidate exon and the protein. GRAILEXP [79], CRASA [16], and AUGUSTUS [69] are algorithms that use ESTs or mRNAs instead of protein [45].

## 2.3 Advantages and Disadvantages

*Ab initio* gene prediction programs only use the DNA sequence that should be annotated. They extract features that can discriminate coding regions (exons) versus non-coding regions (introns). They consider features such as GC and AT rich regions, codon bias, start and stop codon, splice sites, etc. They are suitable for finding novel genes, since homology cannot be defined for a novel gene. In addition, when homology information is limited in a scenario, these programs are used to predict genes. But as mentioned above, these programs need to be trained on annotated sequences, in order to set their hyperparameters. The availability of annotated sequences limits the applicability of these programs. Clearly, the training set of these algorithms can directly impact their performance on the target sequence. If the training sequences are relatively close to the query sequence, the program should perform reasonably well; otherwise, its performance is limited. As noted previously, due to some exceptions, the start and ending points are hard to identify using these methods. Another difficulty that these methods have is the identification of short exons and introns. When the regions are small, the signals are statistically weak and do not qualify for discrimination. *Ab initio* methods might also detect a gene as multiple genes, and the other way around. Lastly, these algorithms have poor performance in large genomes due to low gene density and large non-coding regions. As an example, only 1.1% of the human genome consists of protein/coding sequences, and the remaining sequences

are introns or intergenic DNA (previously known as junk DNA) [75].

Generally, Homology-based methods perform better than pure *ab initio* methods because of the experimental biological data that they directly take as parameters. Their main limitation is that these methods can only find homologous genes. This means that firstly, a gene should be common in multiple species, and secondly, annotated sequences should be available. Likewise, these methods fail to find short exons and introns. In addition, boundary specification in homology-based algorithms is typically wrong due to not fully conserved splice sites.

## 2.4 Evaluation of Gene Prediction Programs

Since there are many programs that aim to find genes in unannotated genomes, the urge for evaluating these algorithms becomes necessary. It is also possible that a program performs better in specific situations. This can be very helpful for scenarios where a specific aspect of gene finding is more important. Of course in order to calculate these statistics, a true genome annotation specifying the exonic and intronic regions of the whole genome for each gene is needed.

### 2.4.1 Performance Measures

For evaluating the performance of a program, quantitative measures are used. Different measures might consider different aspects of a method. Therefore, many performance measures have been proposed by scientists in order to cover various angles of an algorithm.

The performance of gene prediction programs are measured in three different levels: nucleotide-level, exon-level, and protein-level.

In the protein level, different distance measurements are defined between the amino acid sequence that is produced using the candidate exons that are the result of the gene prediction method and the query protein. One can define different penalties for amino acid insertion, deletion, or any rearrangement discrepancies in the sequences. These measurements are not the ones that are used commonly in the literature.

Exon level accuracy statistics are important performance measures that indicate how many exons have been missed, and how many non-coding regions have been wrongly identified as exons. Surely, this type of statistics can be crucial for authors of the programs to know the limits of their methodologies. It is worth mentioning that in this definition, positive class is coding regions (exons) and the negative class belongs to non-coding regions (introns).

- True Positive or TP (in this level Correct Exons or CE) is the number of exons that are correctly identified as coding regions. It can also be read as the number of data in the positive class that are truly classified.
- True Negative or TN (Correct Intron or CI) is the number of introns that are correctly identified as non-coding regions.
- False Positive or FP (Wrong Exon or WE) is the number of introns that are identified as coding regions.

- False Negative or FN (Missed Exon or ME) is the number of exons that are identified as non-coding regions.

- Sn (Sensitivity):

  Sn = CE / AE

- Sp (Specificity):

  Sp = CE / PE

- Miss Rate (MR):

  MR = ME / AE

- Wrong Rate (WR):

  WR = WE / PE

In the last 4 statistics, AE is Actual Exons (TP + FN), and PE is Predicted Exons (TP + FP). It is quite clear that TP and TN are high and Sn, and Sp are close to 1 in a good gene prediction program. Likewise, FP and FN are desired to be low and MR and WR should be close to 0.

And the last level measures the performance of the gene prediction programs at nucleotide level. The same statistics that were mentioned in the previous level can be easily applied to this level. For example, True Positive in the nucleotide level can be shown by CB (Correct Base) and defined as the number of bases that are predicted to be in the coding region and are actually in the coding region [77].

[1] https://en.wikipedia.org/wiki/Sensitivity_and_specificity

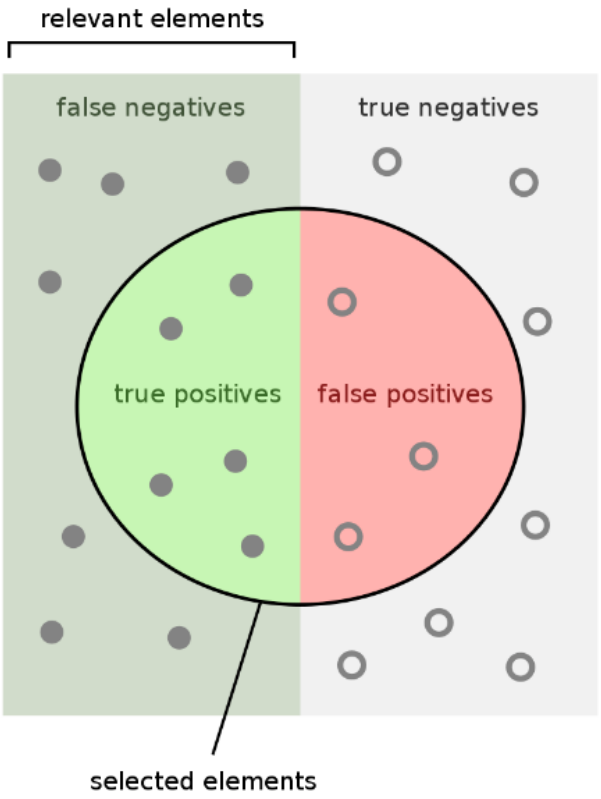


Figure 2.1: An example of the performance of a prediction. The class of each data point is shown in the figure.[1]

- Sn (Sensitivity): It measures the portion of coding nucleotides that are properly predicted as coding. Sn = TP/TP+FN = TP/P

- Sp (Specificity): It measures the portion of non-coding nucleotides that are properly predicted as non-coding. Sp = TN/TN+FP = TN/N

Where:

- TP is the number of coding nucleotides predicted as coding

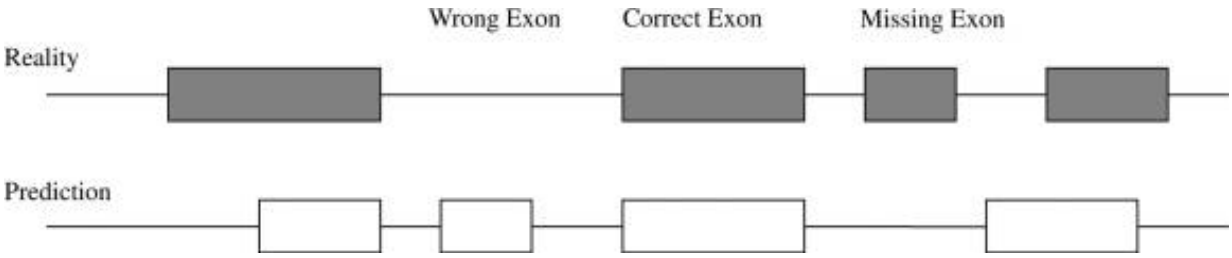


Figure 2.2: Performance measures specification of a gene prediction program in exon-level [77].

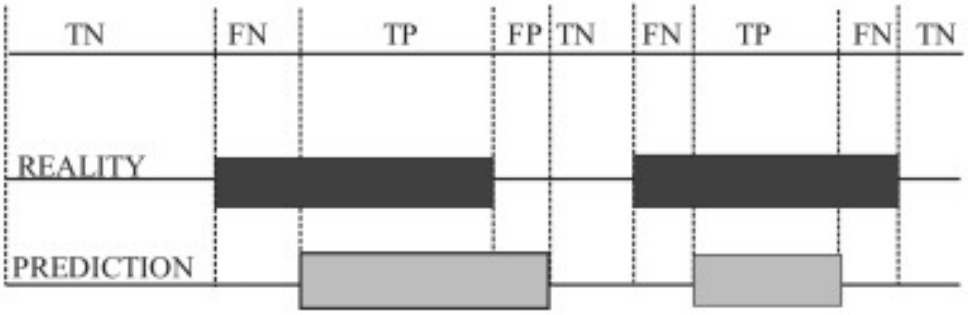


Figure 2.3: Performance measures specification in nucleotide level.

- TN is the number of non-coding nucleotides predicted as non-coding
- FP is the number of non-coding nucleotides classified as coding
- FN is the number of coding nucleotides classified as non-coding

Sn is a relatively good performance measure for this level of assessment, although Sp is not suitable because of the class imbalance problem. The reason is that in a typical genome, a very small fraction of the genome belongs to the coding part, which means TN is generally very high. Therefore, Sp is often close to 1, no matter how the program performs in the prediction of genes or its boundaries.

### 2.4.2 Gene Prediction Evaluation

Although there have been many reviews on the performance of different gene prediction programs, a fair comparison is almost impossible. This is because every gene prediction method trains on and applies to a certain dataset. They might be even more specific in a way that they only have an acceptable performance on a small subset of species. On the other hand, the true annotation which these programs are being compared with are not fully correct, due to limited knowledge and experimental facilities. One of the common problems in *ab initio* programs is the novel gene prediction. If a gene is False Positive, many programs claim that this gene is a novel gene that is not annotated, yet. Of course, by laboratory assay this proposition can be endorsed or declined. Without such experiment, this claim cannot be supported, or rejected. Therefore, one should not consider FPs as an advantage of their method. In conclusion, a comprehensive evaluation is not easy to achieve for all the available programs.

In a review [59] that was done in 2020, the performance of five different gene prediction algorithms is compared using the G3PO benchmark. This benchmark has 1793 proteins from a variety of organisms that can be used for evaluation of the performance of gene prediction algorithms. The proteins in this benchmark are from a diverse set of features such as protein length, gene length, exon number, intron length, GC content. The paper also adds sequences ranging from 150 to 10,000 nucleotides from both upstream and downstream to make the task more similar to a genome annotation task. Although, for the following evaluation, only 889 proteins were chosen, with a 150 bp constant flanking sequence from both upstream and downstream sequence. Adding 10,000 nucleotides to

some of the algorithms might result in very long computation time.

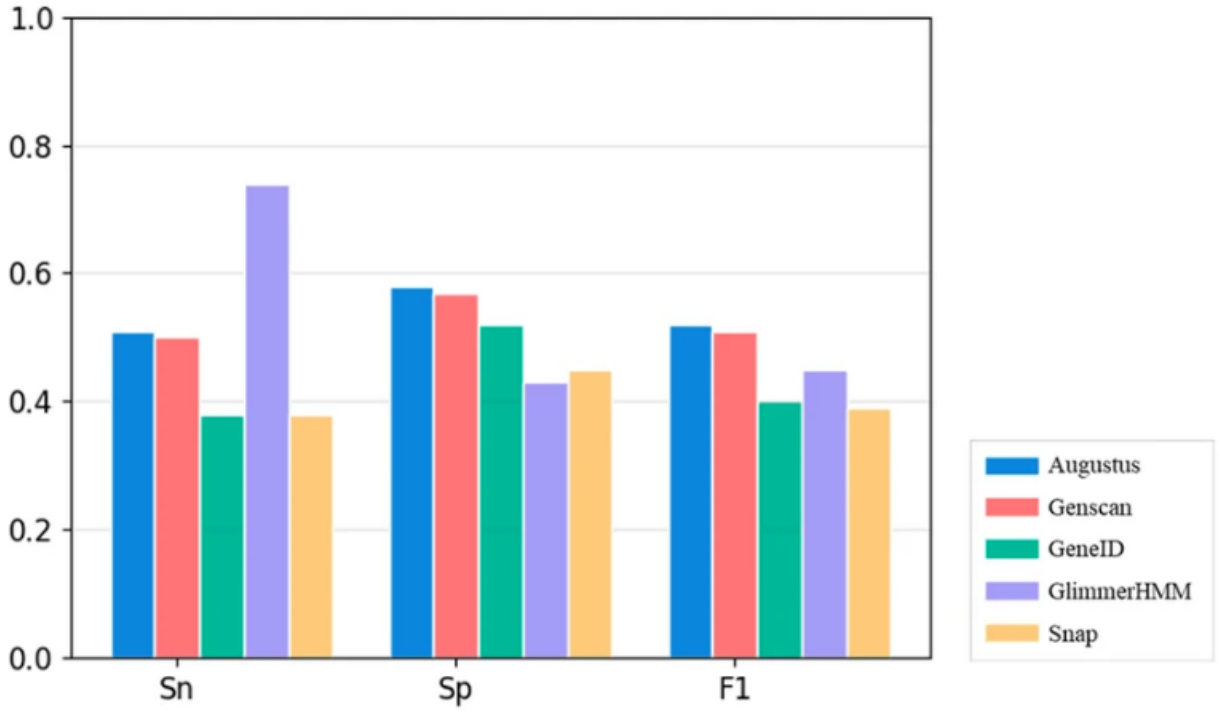


Figure 2.4: Nucleotide-level performance comparison between 5 of the widely used gene prediction programs. The figure shows Sensitivity, Specificity, and F1 Score [69]

There are two versions of AUGUSTUS [69]. The first version of AUGUSTUS is a fairly complex *ab initio* gene prediction program that was built in 2006 for gene prediction in eukaryotes. It is a collection of different methods and models. It uses an HMM (Hidden Markov Model), an intron model, a new donor splice site model, and a novel GC content estimation method. In the paper, the sensitivity and specificity for nucleotide level are reported 97% and 59%, and for exon level they are 80% and 49%, respectively. But in the review, it can be seen that because these two experiments are conducted using two different datasets, the results are substantially different. It is quite obvious that the performance

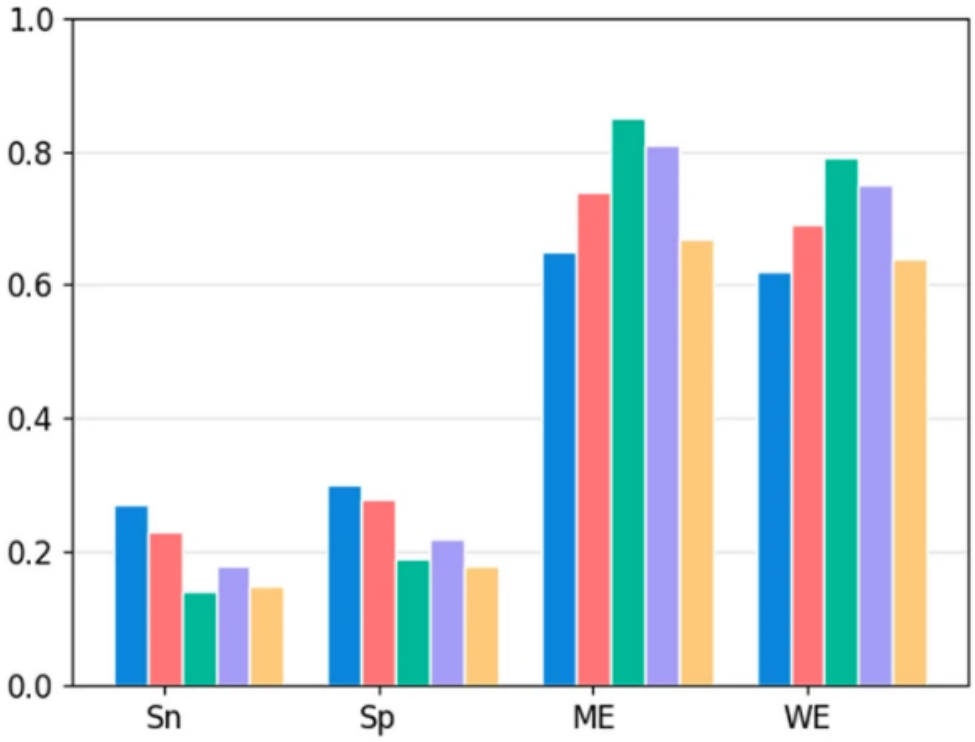


Figure 2.5: Exon-level performance comparison of the 5 gene prediction programs [69].

of AUGUSTUS is slightly better compared to other 4 gene prediction methods, although it is not satisfactory, overall.

GENESCAN [13] was proposed in 1997, and is one of the most used gene prediction programs to this date. It uses length and compositional features of exons and introns, in addition to models for donor and acceptor splice sites, gene density and regions with different GC content. Likewise, GENESCAN reports very high sensitivity and specificity. However the review shows a contrast.

GENEID [32] is one of the first used programs for gene prediction that was built in 1992. It is mainly based on start and stop codon, in addition to the donor and acceptor

splice sites.

GlimmerHMM [58] is the fourth program that is used in the review. It was programmed in 1999. GlimmerHMM uses a Markov model plus a splice site prediction model, and using a dynamic programming technique, it tries to assemble a gene.

Lastly, Snap [37] is another HMM-based gene prediction program that was built in 2005. It is fairly similar to GENESCAN with some modifications. Snap is reported to be the second fastest compared to other gene predictions after GENEID.

AUGUSTUS outperforms the other 4 gene prediction methods by a small margin. The only exception is the significant superiority of sensitivity of GlimmerHMM in the nucleotide level which is not clear why such difference is seen.

# Chapter 3

# Exon Hunter (EH)

In this chapter, we will describe ExonHunter (EH), which is a homology-based gene finding program. Later in the next chapters, we will add codon usage bias to ExonHunter in order to increase the performance of the method. In addition, in future works we will be adding splice site prediction information for the same reason. These two features do not belong to homology-based approaches. Therefore, like many other methods, ExonHunter falls into the hybrid categories. We will formulate the objective function that will be optimized. In addition, a few implementation details will be mentioned.

## 3.1 Pipline

Figure 3.1 illustrates the pipleline of our study. The input for ExonHunter is a genome and a profile hidden Markov model. For some protein families, we were not provided an HMM. For these cases, we used T-Coffee to align sequences, then `hmmbuild` to produce

the necessary HMM. The predictions made by ExonHunter were compared against the annotation provided to us by our collaborators at the University of Montréal. Further information on data preparation can be found in Chapter 4.

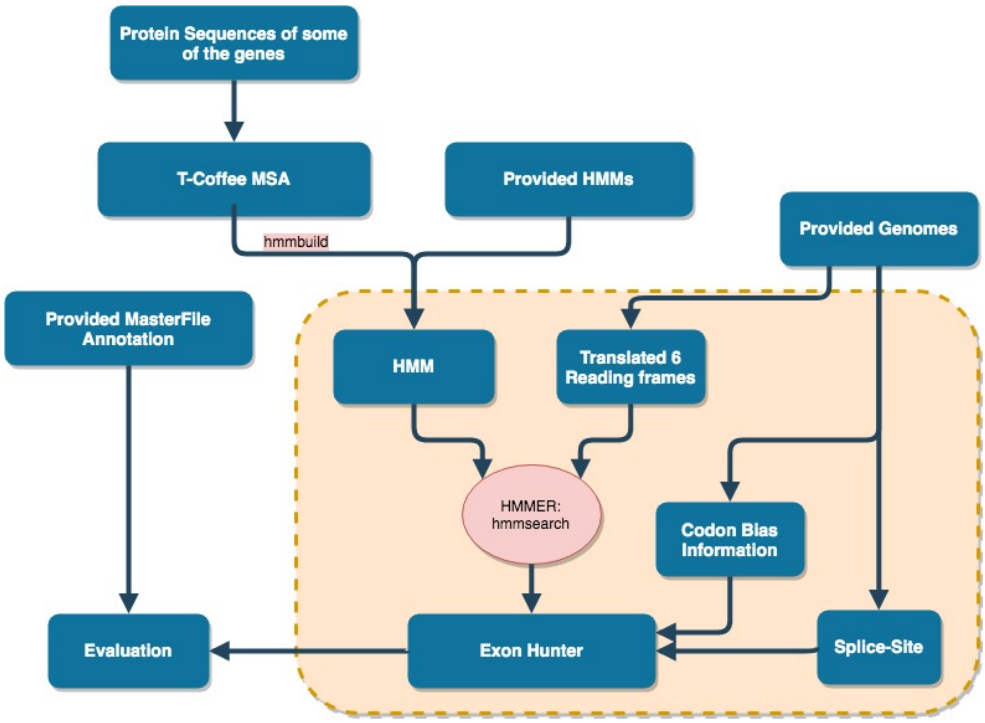


Figure 3.1: The pipeline of our study. The orange frame in the diagram contains the building blocks of ExonHunter. An experiment is finding a gene in a target genome by taking one HMM protein profile, and one genome.

## 3.2 Methodology

The first step of ExonHunter is to translate the given genome into all 6 possible reading frames. Next, ExonHunter uses HMMER to match the given HMM model against all 6 possible reading frames. In a perfect scenario, the task of gene finding is to find a set of hits

(candidate exons) that each one of them perfectly matches an exon, and the whole set can synthesize the gene. Exon Hunter (EH) uses this simple idea, and tries to maximize the bases in the gene that are covered by hits. There's a penalty for hits that have overlapped with each other. Exon Hunter builds a search search tree where each leaf has a different set of hits. Then it calculates a score for each leaf based on how the hits in that set cover the gene, and how much the hits have overlapped. Finally, it chooses the set that has the highest score. Suppose there are $n$ hits. Each hit can either be included in a set, or excluded. For $n$ hits, $2 \times 2 \times ... \times 2 = 2^n$ possible unique sets can be formed. Building a tree with $2^n$ leaves can be very time consuming. However, we will see that the number of hits that will be selected as candidate exons is low. Furthermore, has explained below, ExonHunter uses non-crossing constraints to prune the search tree.

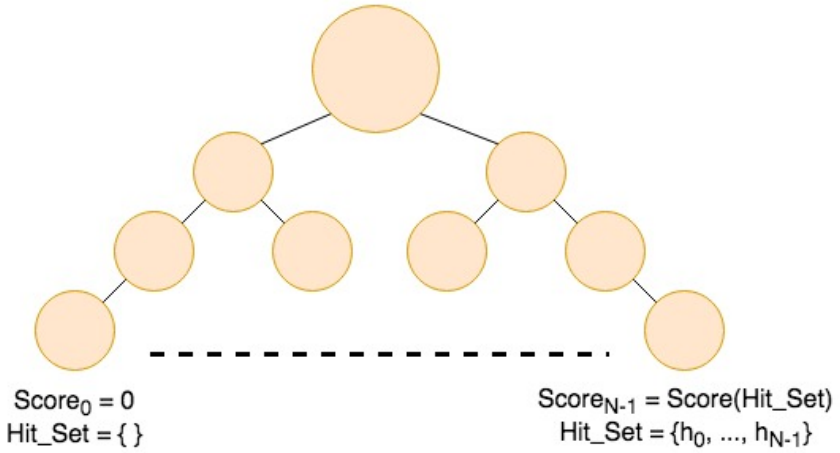


Figure 3.2: The search tree that Exon Hunter creates to find the best hit set with the highest score.

Coverage can also be considered as TPR: the amount of nucleotides/amino acids in the exons/protein that are covered by the hits. Overlap, on the other hand, prevents the

inclusion of possible hits that are extended to introns. Obviously, high levels of coverage and low levels of overlap is desired. The score of each hit set can be defined as follows:

$$\text{Score} = \text{Coverage} - \text{Overlap} \tag{3.1}$$

If $L_i$ is the length of $i$th hit and $O_{i,j}$ is the overlap of hit $i$ and hit $j$ in an arbitrary set of hits called $S$:

$$\text{Coverage} = \sum_{i \in S} L_i - \sum_{i \in S, j \in S} O_{i,j} \tag{3.2}$$

If there is an overlap between the hits, summing lengths of hits will result in counting the overlapping positions multiple times. That is why the overlap is subtracted in this equation.

$$\text{Overlap} = \sum_{i \in S, j \in S} O_{i,j} \tag{3.3}$$

One can add a dependent coefficient to the term for setting different importance level to each of the criteria, where $0 \leq \alpha \leq 1$:

$$\text{Score} = \alpha \text{ Coverage} - (1 - \alpha) \text{ Overlap} \tag{3.4}$$

If $\alpha$ is closer to 1, overlap would have a lower impact on the decision of inclusion of the hits. If it is closer to 0, the importance of coverage would fade. The reason for having

dependent coefficients for coverage and overlap is that ideally coverage and overlap should not increase or decrease together. By having coefficients that change in opposite directions firstly, we are reducing the number of parameters by one (33% percent), and also implicitly we are adding the obvious constraint which is the fact that increasing one term would result in decreasing the other, and vice versa. This equation can be extended more by taking the e-value of the hits into account, where $0 \leq \beta$ :

$$\text{Score} = \alpha \text{ Coverage} - (1 - \alpha) \text{ Overlap} - \beta \sum_{i \in S} \log e\text{-}value_i \tag{3.5}$$

The $e$-value that should be used is the i-evalue that HMMER reports for every hit. E-value, as a matter of fact, can be interpreted as the number of hits expected by chance. As the score of an alignment/match increases, $e$-value drops exponentially. It can also be considered as a significance measure where lower $e$-values are desired. We used a base 10 log to make typically very small values of $e$-value sensible for the equation. If it is close to 0, obviously the $e$-value wouldn't have an impact on the score of a hit. If it is close to 1, each 10 fold of $e$-value would have the same impact as one position coverage (or overlap). The minus behind the logarithm term is because low $e$-values are desired, and log of a good $e$-value is negative.

Ideally, this formula would give high scores to true positive hits, and lower scores for false positive hits. However, in the real world, this scoring would not perform perfectly. In order to maximize the ability of the score to produce the desired result, so that it differentiates true hits from false hits more effectively, it is possible to consider these coefficients as hyperparameters of the approach, and adjust them using an optimization

technique. Efficiency of the score can have many definitions. One can calculate the AUC (Area Under Curve) of a ROC (Receiver Operating Characteristic) graph, and consider it as the performance measure of the score. We will see in the next chapter how these coefficients are optimized. Eventually, we set $\alpha = 0.6$ , $\beta = 0.8$ after a number of experiments.

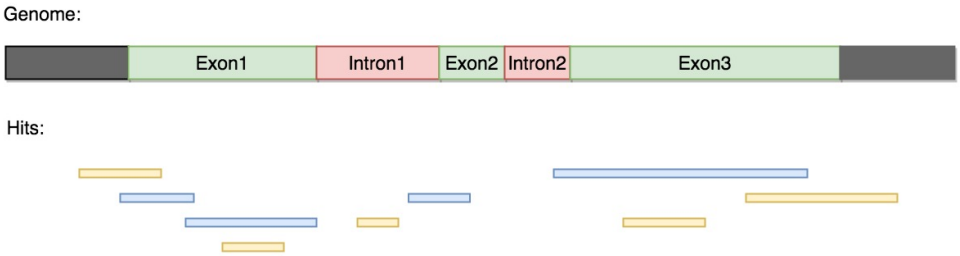


Figure 3.3: An example of a gene with three exons and two introns and a list of hits (candidate exons). The blue hits are the ones that will be selected according to our objective function. They cover a good amount of the exons, and there is no hit that can be added to this set that adds more coverage than it adds overlap. Different levels of hits do not characterize different frames. In this figure, candidate exons of each exon are located below the exons. In other words, hits are aligned with the genome.

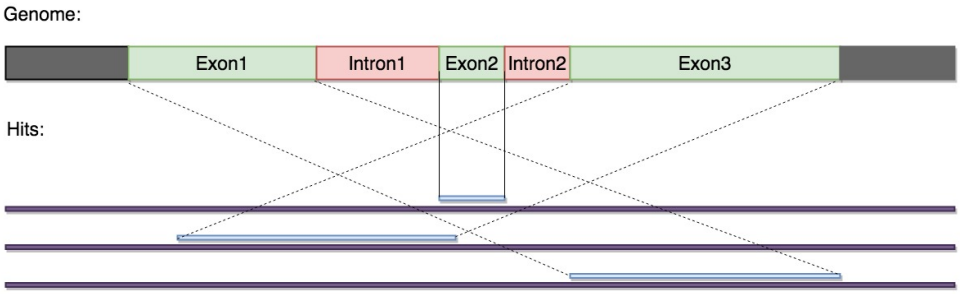


Figure 3.4: An example of crossing happening between two hits (Left hit and Right hit) from the same direction. Such a set of candidate exons cannot form a valid set to constitute a protein.

A set of hits has to satisfy one constraint, and if it does not, that set cannot be a valid set that is acceptable as a set of candidate exons. Suppose $h_i$ and $h_j$ denote two hits, and

$e_{i'}$ and $e_{j'}$ are their corresponding exons. The following statement should always be true :

$$\forall_{i,j} s_{h_i} < s_{h_j} \rightarrow s_{e'_i} < s_{e'_j} \tag{3.6}$$

Where $s_x$ is the starting position of $x$. And $x$ can be either an exon or a hit.

In other words, if a candidate exon $i$ hits an exon $i'$, no candidate exon in that set that happens before candidate exon $i$ should hit any exon that happens after $i'$. The same rule should be true for candidates after $i$, and exon before $i'$. Figure 3.4 shows a visual representation of this rule which we call a Crossing. It is important to mention that it does not matter if crossing hits are from different reading frames, but in the same direction. In case of a forward and backward hit, the valid hits are the ones that cross each other.

## 3.3 Implementation

There are details on implementing Exon Hunter (EH) that need to be stated. EH is an open-source package that is programmed in Python and is available on GitHub [1]

HMMER has two different output formats where one of them (the Tabular Output Format) is suitable for parsing, and the other is easy to read. EH parses the tabular format to extract necessary information such as HMM and target sequence coordinates, and $e$-values of each hit, and also HMM length. Obviously, having HMMER is necessary for running ExonHunter, since EH runs it as a sub-process.

[1] https://github.com/amirh-hajianpour/ExonHunter

Reading MasterFile annotation needs some consideration. The obvious thing is that Fragmented Genes need to be attached to each other in order to form only one gene. Since we are looking for homology of proteins in a genomic sequence, only protein-coding genes need to be considered, and RNAs, mRNAs, tRNAs and ORFs (Open Reading Frame) should be ignored. There are some introns that are inserted in other introns. These regions are called *twintron*, and are clearly ignored.

# Chapter 4

# Data Preparation

This chapter is dedicated to the preparation of the data that will be fed to Exon Hunter. A short description of the dataset is provided. Then we will feed HMMER an HMM protein profile to find exons of the query gene in the reference genome, and finally we will filter false positive to make the future step of our study feasible.

## 4.1 Dataset

The dataset that we used for our analysis is a set of mitochondrial genomes of fungi. The following table shows the name of the taxonomy ID of the species used in this study.

It is necessary to know how these organisms are distant from one another. Therefore, we used Taxonomy Browser [1] of NCBI to build a common tree.

[1] https://www.ncbi.nlm.nih.gov/Taxonomy/CommonTree/wwwcmt.cgi

| Index | Organism Name | Taxonomy ID |
|---|---|---|
| 1 | Allomyces macrogynus | NCBI:txid28583 |
| 2 | Brettanomyces custersianus | NCBI:txid13368 |
| 3 | Cantharellus cibarius | NCBI:txid36066 |
| 4 | Ceratocystis cacaofunesta | NCBI:txid386457 |
| 5 | Ciboria shiraiana | NCBI:txid984309 |
| 6 | Coccidioides posadasii | NCBI:txid199306 |
| 7 | Coemansia braziliensis | NCBI:txid61391 |
| 8 | Dactylella tenuis | NCBI:txid383872 |
| 9 | Microbotryum violaceum | NCBI:txid5272 |
| 10 | Podospora anserina | NCBI:txid2587412 |
| 11 | Smittium culicis | NCBI:txid133412 |
| 12 | Stenocarpella maydis | NCBI:txid238245 |

Table 4.1: The list of organisms that are used in our analysis

The Tree Viewer[2] of NCBI was then used to give a visual representation of the same common tree. This tree is sorted in an ascending order. It means that species at the top are more distant from the next species, and the organisms at the bottom are closer to their neighbouring nodes. This tree is gives a acceptable representation of how similar the species are. Knowing how far each organism is from the others in terms of similarity should reflect in our analysis. In other words, similar organisms (organisms that have similarity in their features such as gene set and codon usage) are expected to be closer in the taxonomy tree. For example, if fungus A is close to fungus B but far from fungus C according to the tree, it is expected organism A and B to have more similar set of genes, compared to the genes of organism C.

The annotation of these fungi organisms are provided by the Department of Biochemistry of Université de Montréal, via private communications with Dr. B. Franz Lang. This

[2] https://www.ncbi.nlm.nih.gov/projects/treeview/

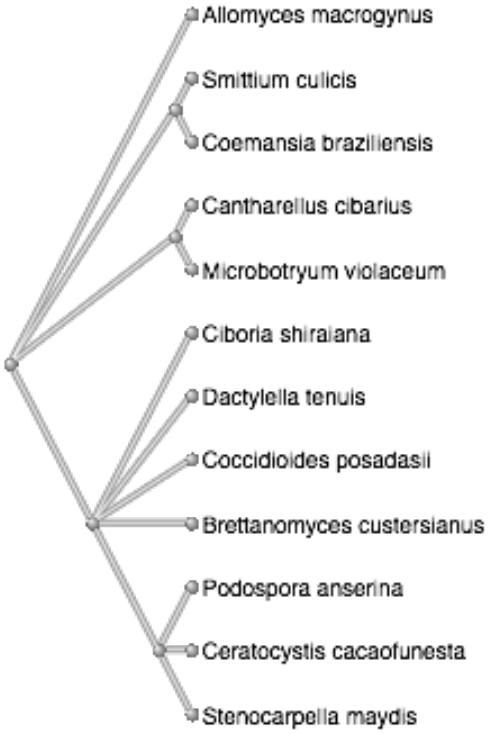


Figure 4.1: Taxonomy Tree represented in Slanted Cladogram using the Tree Viewer of NCI for the 12 fungi of the dataset. Allomyces macrogynus is the most distant species from the rest of the organisms. Ceratocystis cacaofunesta and Stenocarpella maydis are the closest species in our dataset.

annotation is done using MFannot[3], which is an annotation pipeline for mitochondrial genome. MFannot uses an RNA/inton detection tool, an intron-exon boundary model, and a procedure in order to detect small introns. This pipeline particularly performs well on mtDNA with many introns. MFannot is typically followed by a manual annotation corrections that is made by an expert using a gene sequence comparison with a closely related species, in addition to a validation step that utilizes RNA-seq data. The description of how a MasterFile format annotation is formed is mentioned in the Appendix A section.

[3] https://megasun.bch.umontreal.ca/cgi-bin/mfannot/mfannotInterface.pl

In our dataset, each fungal mitochondrial genome has coding and non-coding genes. Since our approach is a homology-based gene prediction, we are only interested in protein-coding genes.

There are genes in these organisms that have no introns. If a gene has no intron, it is called an intronless gene and in the MasterFile format there would be no intron or exon line.

There are genes in these species that are fragmented. A fragmented gene is a gene that is divided into segments that are extraordinarily distant from each other but code for a single functional unit. One of the reasons for this fragmentation can be an insertion between a protein-coding region.

Another phenomenon that needs to be addressed is split codons. Splicing and translation are two independent steps in gene translation occurring sequentially. Splicing is unaware of the codon structure. Therefore, there is no reason for splicing to necessarily occur before or after a codon. When a splicing event occurs inside a codon, the result is called a split codon. The RNA splicing mechanism does not always pick exons with a length that is divisible by 3. Sometimes the length of an exon is not divisible by 3, which implies that a part (one or two nucleotides) of the next (or the previous) codon is located in the current exon. In other words, the RNA splicing machinery does not separate the coding region from the non-coding region in the amino acid level, but rather in nucleotide level.

## 4.2 T-Coffee and HMMER

Typically, HMM Protein Profiles derive their structure from a multiple sequence alignment. The HMM Protein Profile of some of the genes were provided to us, but for the rest of them, only some protein sequences of the genes that had to be aligned, and then built into HMM profiles were available.

There are multiple benchmarks and studies that suggest the superiority of T-Coffee [49] over other multiple sequence alignment algorithms [24, 48, 57, 73]. Therefore, we used T-Coffee to align the multiple protein sequence that we had for each gene. T-Coffee performs pairwise alignments firstly. And then using these alignments, it builds a library of the alignment information that can be helpful for the next step which is a progressive alignment. Then in the progressive alignment step, the intermediate alignment occurs which is based on the order of the alignments that should be done next, and also based on the information that tells how the sequences should be aligned with each other.

The most recent version[4] of T-Coffee is available online[5] on the website of the Centre for Genomic Regulation. T-Coffee has different settings, and this specific version has a setting that provides most accurate alignments in exchange for using a larger memory. The output of this online tool is a colour-coded MSA on an HTML page. It can also output a variety of formats such as ClustalW and FASTA [41] alignment format. We downloaded the FASTA alignment format, which we are going to use in the next step.

After retrieving the FASTA alignment format of the MSA from T-Coffee, we had to

[4]The official website of T-Coffee in the FAQ section claims this: http://www.tcoffee.org/Projects/tcoffee/index.html#DOWNLOAD

[5]http://tcoffee.crg.cat/apps/tcoffee/do:regular

build the HMM Protein Profile. In order to build the profile and search the HMM in the target sequences, we used HMMER[6] [23]. HMMER is a free, open-source package for sequence analysis. The main usage of it is to find homologous proteins or nucleotide sequences, in addition to the sequence alignment. The full list of its commands, and functionalities are described in its User's Guide[7]. HMMER uses an HMM Profile to look for similar or homologous sequences in a query sequence, or in a database of sequences. HMMER has different commands for different purposes of biological sequence analysis. The "hmmbuild" program is used to build HMM Profile from a multiple sequence alignment in FASTA format. We used this command to generate the HMM profile that we needed to build. We used the default setting of the command, and ignored any extra adjustable option to create our models.

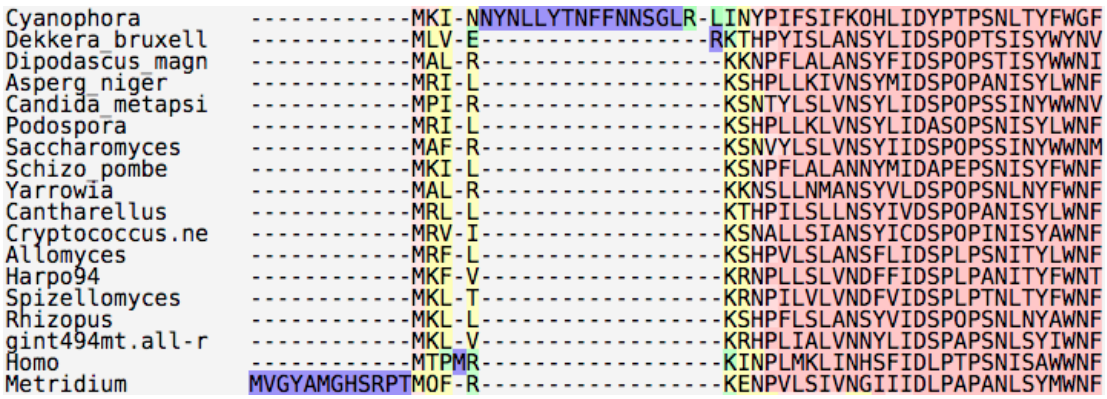


Figure 4.2: A portion of the beginning of the multiple sequence alignment of proteins using T-Coffee. Purple and green areas are aligned bad, yellow is average, and pink is good alignment positions. This portion of alignment tells us that some genes in some of the genomes might be longer or shorter than the consensus. This means that a general HMM won't have the same performance in predicting one gene in different genomes as we will see in the result chapter.

[6] http://hmmer.org/
[7] http://eddylab.org/software/hmmer/Userguide.pdf

The next step was to use the HMM profile to search for homology in query sequences. The program that takes an HMM profile and a target sequence or a set of sequences is "hmmsearch". "hmmsearch" has many parameters that should be mentioned here. The first group of options is for the control of the output format.

```
Query:       cob_MSA  [M=384]
Scores for complete sequences (score includes all domains):
   --- full sequence ---   --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence
    ------- ------ -----    ------- ------ -----   ---- --  ---------
    3.5e-68  218.9  14.9    2.6e-56  179.8   5.5    4.3  5  1_orf_A. macro
    1.3e-49  157.7   9.2    3.2e-27   84.0   0.2    3.6  3  3_orf_A. macro
    6.1e-41  129.2   6.0    1.8e-40  127.6   6.0    1.8  1  2_orf_A. macro

Domain annotation for each sequence (and alignments):
>> 1_orf_A. macrogynus gc=4, 57473 Residues.
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to
 ---   ------ ----- --------- --------- ------- -------    ------- -------
        envfrom  env to     acc
        ------- -------    ----
   1 !   57.9   1.9   1.4e-19   2.7e-19     163     200 ..    3046    3083
    ..     3042    3086 .. 0.93

   2 !  179.8   5.5   1.3e-56   2.6e-56     249     379 ..    3880    4010
    ..     3856    4013 .. 0.91

   3 ?   -5.3   3.4       2.2       4.4     281     373 ..    6814    6907
    ..     6808    6914 .. 0.70

   4 ?   -4.7   4.1       1.4       2.9     320     379 ..   12097   12158
    ..    12085   12162 .. 0.78

   5 ?   -3.0   1.0      0.44      0.87     163     185 ..   16520   16540
    ..    16516   16549 .. 0.82

  Alignments for each domain:
  == domain 1  score: 57.9 bits;  conditional E-value: 1.4e-19
   cob_MSA  163 ewlWGGfsvdnatlnrffslhyllPfiiaalavvhlia 200
                  +WGGfsvdnatlnrffslhyllPfi+aal+vvhlia
  1_orf_A. 3046 IVVWGGFSVDNATLNRFFSLHYLLPFILAALVVVHLIA 3083
                579**********************************98 PP
```

Figure 4.3: A truncated part of the output of "hmmsearch" command on 6 target sequences: only 1_orf_A, 2_orf_A, 3_orf_A, and r1_orf_A have hits that are above the set threshold. The query HMM profile belongs to the gene cob. The length of the HMM profile is 384 amino acids long. The number under column N indicates the number of hits that are found in the corresponding target sequence. The alignment of the first found hit in the first target sequence is shown.

The second group of options is inclusion threshold control. One can change the inclusion threshold of "hmmsearch" to include more hits (candidate exon: a segment of the target sequence that matches a segment of HMM profile to some extent) by setting a higher threshold, or vice versa. Some of these options are described changing the sensitivity of the search in exchange with increased searching time. HMMER has three different filters: MSV, Viterbi, and Forward filter. The common inclusion threshold configuration for this command is to set the p-value of the third filter to a higher value: "–F3 = 1" or "–F3 = 0.1". This is because it is the main filter that removes a substantial number of hits. By adjusting this specific option, we will later see that sensitivity can be increased even more. Finally, we decided to use the "–max" option where all the filters are off. Ultimately, it will be shown that this option introduces many false positives to the search. However a new filtering technique will be used in order to eliminate many false positive hits. Surprisingly, this configuration does not increase the computation time of the algorithm significantly in our dataset, although it increases the sensitivity. "hmmsearch" reports the result in a specific format. The report includes some basic information about the target sequences and the HMM. Then a summarized report of the found hits and some relevant statistics is presented. The next section of the output is a list of each target sequence in which at least one hit is found, with the positions in both the HMM profile and the target sequence, e-values, and the alignment of the hit with the target sequence. There are some symbols that can possibly be seen in an alignment such as the one in the Figure 4.3. Surely, the query HMM profile is shown by its consensus sequence. Capital letters represent highly conserved positions. A dot '.' in the HMM profile line shows that an insertion has happened in that position in the target sequence. A plus '+' in the line between the two

indicates a 'conservative substitution' as the manual suggests. And a dash '-' in the target sequence line means a deletion with respect to the HMM profile. The bottom line of the figure is the posterior probability of each position where lower digits show low, and high value digits indicate high posterior probability. A start '*' means the maximum posterior probability for that specific position. Shortly, in this chapter we will talk about the target sequences that we fed to HMMER. And finally at the end of the output the numbers of target sequences that have passed the three filters are presented, in addition to the total computation time of the "hmmsearch" process.

We started with collecting MasterFile genome annotation, FASTA genomic mitochondrial sequences of 12 fungi, and HMM profiles of genes that exist in these species, in addition to some sets of protein sequences that their HMM profile was not available. After aligning protein sequences of each gene using T-Coffee, we used the "hmmbuild" program of HMMER to generate the HMM profile that we needed.

The ultimate goal is to run "hmmsearch" on an HMM protein profile and a genome as the target sequence, get the hits that are reported by the command, carry out some filtering to remove a large number of false positive hits, possibly perform some local search in order to match the starting and ending positions of exons, and finally synthesize candidate exons to report as the full-gene sequence of the corresponding protein. The language of an HMM protein profile is obviously amino acids. Therefore, to search for homologous sequences, the genome needs to be translated. A gene or an exon can be in either of 6 reading frames. So, we translate each target genome into 3 forward and 3 backward reading frame sequences. We should mention that the genetic code that is used for translating genomes is not identical

to the Universal genetic code. A specific genetic code[8] was used to do this translation. This code has just one different codon compared with the standard genetic code. Every codon codes for the same amino acid, except that "UGA" codes for Tryptophan (Trp) instead of coding for Stop Codon. Exon Hunter has the option of using three different genetic code: human, vertebrate, and fungal. The translated genomes are saved in a fasta format file. In order to differentiate between 6 reading frames, we added index to the start of the genome name. Sequence column of Figure 4.3 shows how the different reading frames are named. First forward frame starts with "1_" and the name of the genome follows the string. Second and the third frames follow the same rule. If the frame is backward, the same structure applies, but there is an "r" letter at the beginning to represent the reverse direction of the frame.

By running the following code, for every gene, and every genome[9], the gene prediction step starts:

hmmsearch –max [HMM_Protein_Profile] [Target_Genome]

In chapter 5, we will see that how Exon Hunter will be fed by the hits that HMMER produces.

[8] https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?chapter=cgencodes#SG4

[9] Every genome has a specific set of genes. There are some genes that are not present in all the target genomes. If only a small fraction of a gene is found in a genome, it is considered that the genome does not contain the query gene.

# Chapter 5

# Performance of Exon Hunter on selected Mitochondrial Genomes

We now analyze the performance of ExonHunter (EH). We chose GeneWise as the program that we will compare our results with. GeneWise is a similarity-based program, therefore we can provide it with the same models (HMMs or sequences) that we feed EH. GeneWise is one of the widely used, and well-known algorithms among the gene annotation programs. The outputs of the GeneWise are produced using the online version of the program that is available on European Bioinformatics Institute website[1].

[1] https://www.ebi.ac.uk/Tools/psa/genewise/

| | Rel. TPR | Abs. TPR | Precision (PRC) |
|---|---|---|---|
| HMMER | 0.91 | 0.91 | 0.17 |

Table 5.1: Out of 447 exons, 406 of them are detected by HMMER. Since HMMER is the first step of our methodology, absolute and relative TPR are the same. As we mentioned several times, false positives in the HMMER output is abundant, which is shown by the low precision.

## 5.1 HMMER and its Limits

In this section, we will show how Exon Hunter is different from HMMER. Obviously, there has to be some advantages in the usage of ExonHunter over using only HMMER. Since HMMER has multiple parameters, different settings will result in different outputs. This means that with conservative values for parameters and filters that HMMER has, it is more probable that HMMER outputs fewer hits, and therefore fewer exons will be found, and consequently fewer portions of the genes will be identified. This means lower True Positive Rate (TPR) and higher Precision (PRC). On the other hand, with a more liberal setting, more exons will be predicted. This will result in higher TPR, but lower PRC. The optimal setting can be assumed problem-dependent. Although there's a default setting for "hmmsearch" command of HMMER, the optimal setting which will maximize TPR and PRC at the same time depends on the problem.

We ran HMMER for all the genes and all the species of our dataset. As we mentioned earlier, optimizing the performance of HMMER is necessary. There can be two possible solutions for this problem. One way is to optimize the built-in parameters that HMMER has, such as: target inclusion $e$-value threshold (–incE), or conditional domain inclusion $e$-value threshold (–incdomE). The problem with tweaking these parameters is that they are

tuned by a specific dataset, therefore the reported expectation might not be true for every dataset. For example, target inclusion threshold claims 1 false positive for incE = 0.01, however, for different datasets this report might not be correct. The other solution is to force HMMER to produce as many hits as possible, and use the $e$-values of hits to compute an optimal threshold. In order to achieve the best result, we chose the most liberal setting (which is setting off all the filters by command "–max") to get as many hits as HMMER is capable of. The idea is to filter out the false positives from this pool of hits that probably have a considerable number of false positives, by setting a threshold for $e$-values of hits. For doing so, we paired the $e$-value of each hit with a class label which is whether a hit belongs to an exon or not. Then, we performed a 10-fold cross validation on this data of paired values. A $k$-fold cross validation is a data resampling method to evaluate the generalization of a predictive model, or to tune model parameters. K-fold cross validation is done by first dividing the dataset into $k$ randomly selected equal subsets, where in each turn ($k$ times), one fold (subset) is left out of the model execution, and performance is measured each time. In the training set of cross validation (9 folds) we sorted the data according to the $e$-values. As going through hits, from the hit that has the lowest $e$-value, we calculate the accuracy. We intended to have 90% accuracy. When the accuracy reaches 90% we stop and record the threshold. We do this for each fold, therefore we have a vector of thresholds that result in 90% accuracy, and a vector of accuracies which corresponds to the accuracy of each test fold. As Table 5.2 depicts, the average accuracy is more than 90%, which is a desirable performance. On the other hand, standard deviation is low, which tells us that each fold, however they have been selected randomly, is being similarly classified as desired. In conclusion, we could set the threshold that we want by either choosing the

| | Min | Max | Average | Standard Deviation |
|---|---|---|---|---|
| Threshold | 0.0005 | 0.0120 | 0.0058 | 0.0029 |
| Accuracy | 0.89 | 0.95 | 0.93 | 0.02 |

Table 5.2: Statistics of 10-fold cross validation on $e$-value of hits and their corresponding outcome.

average or maximum of thresholds. We chose maximum threshold which is 0.012. Later in this section we will show why maximum threshold has been chosen.

We have previously mentioned that the number of false positives in this pool of hits that HMMER produces is high. Therefore, using normal cross validation can lead to poor result. Since the folds are created from the random selection of each data point, if a significant number of false hits are located in a small number of the folds, cross validation would not produce a useful result. So, we used stratified cross validation so that each fold has a similar false and true hit distribution.

The whole reason for having this filtering step is to reduce the computation time of the future steps. Since one of the crucial steps of EH is to form every possible subset of the found hits and calculate a score for each hit set, a high number of hits can make this calculation impossible to complete. Forming every subset of a set (power set) is of $O(2^n)$, and in some cases, with "–max" setting, there are more than 200 hits for each experiment (one gene and one genome), which makes this filtering strategy inevitable. By choosing the maximum threshold of the threshold vector, we are being more careful to include as many hits as we can. This is because we will present strategies to remove false positives. As shown in Table 5.1, PRC is low due to high number of false positive hits that HMMER produces. However, in this study, we have, yet, no mechanism to add to the hits that

| | Min | Max | Average | Standard Deviation |
|---|---|---|---|---|
| Lengths of Exons | 3 | 42 | 21 | 10.74 |

Table 5.3: Lengths statistics of the missing exons from HMMER output. These 41 small exons cannot be found even by using the full strength of HMMER. Lengths are in nucleotide coordination. As the table suggests, these exons range from one amino acid to 14 amino acids.

HMMER has already given us.

Another issue that needs to be mentioned here is the fact that HMMER cannot identify all the exons, even, by turning all the filters off, and setting $e$-values to the highest. This inevitable problem is a consequence of using HMMER for this task. However, Table 5.3 shows that those missed exons fall into the problem of finding small exons, which we have mentioned as one of the annotation problems that many programs suffer from. The total number of exons is 447, therefore, on average HMMER misses about 9% of the exons of our dataset.

In the next sections, we will be reporting two different performance measures regardless of the performance measuring techniques that are used. It means that if, for example, we choose TPR as our measurement, we are going to report two types of performance measures. One type is relative, which shows the performance of solely that specific step, and the second one is absolute, where the overall performance of the process up until that step is being represented and the performance of prior steps are taken into account. The relative TPR of thresholding step is 96%. It means that it loses 4% of the true hits that are given to it. However, as Table 5.1 depicts, HMMER is unable to find 9% of the whole true hits of the dataset because of how small they are. The absolute TPR for $e$-value thresholding is 87%. This means that Exon Hunter, which is the next step, is going to get

only 87% of the true hits in the dataset.

## 5.2 Parameter Optimization

As mentioned in Section 3.2, optimization of the coefficients of the objective function is a crucial step in our methodology. Each term of the objective function has an impact on whether or not a hit forms a part of the exon. Optimization of these parameters is the answer to the question of how much each term of the objective function should contribute in the decision of including or excluding a hit from the final hit set. The term that might have a more influence should have a bigger coefficient, and if it has a less effect, smaller coefficient will be optimal. In other words, optimizing these parameters will result in assigning higher scores to true hits, and lower scores to false hits.

In order to optimize the parameters, we used a parallel Grid Search. The grid search is called parallel because we changed the coefficients at the same time. The number and the values of the steps in a grid search are of importance. Exhaustively, one can set a large number of values with small steps in order to reach the most optimal solution. However, doing so can be very time consuming and would result in coefficient values that are biased to the used dataset. Since we know that the values of the two terms (length of hits and $e$-values) range approximately from 30 to 300, 10-fold difference between the coefficients gives the terms the chance of dominating the other one. Therefore, it is easy to infer that more than 10-fold difference between the minimum and maximum step is unnecessary. We chose to have steps sequence as following: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]. For each setting, we consider Accuracy of exon prediction as performance measure, which is the

summation of the number of found exons (TP) and the number of discarded false positives (TN) over the total number of exons (P+N). It is necessary to note that nucleotide-level Accuracy is not a fair metric for finding exons, since the task of scoring hits is the inclusion or exclusion of hits which does not necessarily result in better or worse nucleotide-level accuracy. Exon-level statistics are helpful to find exons of the genes. However nucleotide-level statistics can be more useful in extending or shrinking already found exons. The optimal set of coefficients needs to be specified and also validated. Although, using all the data to specify the set produces the best result in future uses, it leaves us no data to validate the quality of the parameters. To avoid this problem, we used a regular 10-fold cross validation for setting these parameters. For each data point in our dataset (each gene in each genome), we have a vector of exon-level accuracy that corresponds to each coefficient setting. For every coefficient setting, we calculate the average accuracy throughout all the genes and species, to transform the matrix ($coefficient\_setting \times data\_point$) into a vector. Obviously, each of the training folds and the testing fold have one separate vector. Finally, we choose the setting that leads to the highest average accuracy, and report the corresponding average test accuracy for each fold. The statistics of the cross validation is presented in the Table 5.4. The average accuracy that is shown in the table is the average accuracy of each test fold. As we have mentioned before, the optimal coefficient set is: $\alpha = 0.6$ , $\beta = 0.8$. The equation 5.1 shows how the objective function looks like with the optimal coefficients. The optimal coefficient set was chosen out of a $10 \times 10$ (100 coefficient sets) vector of accuracies. In contrast, one might do the optimization in a series manner where one coefficient is optimized and set, and then the other parameter undergoes the optimization.

| | Min | Max | Average | Standard Deviation |
|---|---|---|---|---|
| Accuracy | 0.86 | 0.99 | 0.93 | 0.04 |

Table 5.4: Statistics of 10-fold cross validation on the optimal coefficient set.

| | Rel. TPR | Abs. TPR | Precision (PRC) | Abs. Accuracy |
|---|---|---|---|---|
| Exon Hunter | 0.93 | 0.82 | 0.97 | 0.96 |

Table 5.5: Exon-level statistics of Exon Hunter (EH) on the dataset. 94% Relative TPR means that Exon Hunter only misses 6% of the exons. 82% Absolute TPR means that from the first step of our methodology, 82% of the exons are found. 97% Precision means that a remarkable number of resulted hits are exons. The Absolute Accuracy is very high as well as Precision because Exon Hunter is very successful in removing false positives.

$$\text{Score} = 0.6 \times \text{Coverage} - 0.4 \times \text{Overlap} - 0.8 \times \sum_{i \in S} \log e\text{-}value_i \tag{5.1}$$

## 5.3 Exon-level and Nucleotide-level Performance

As we saw in section 2.4.2, presenting the performance of a gene prediction program is being done at two separate levels: Exon-level, and Nucleotide-level.

In the previous section we talked about short exons, and their impact on the overall nucleotide-level performance of EH. The basis of EH is searching for similar sequences. If the query sequence is long, the probability of finding this sequence by chance in the genome is lower compared to a shorter query sequence. This probability is measured in terms of $e$-value. Since we used an $e$-value thresholding technique in order to eliminate false positive hits, we might have unintentionally also removed the short exons from our pool.

Figure 5.1 shows the frequency of exons for different lengths of exons, and Figure 5.2 shows the absolute exon-level TPR. As it can be seen in Figure 5.2, TPR is low for short exons. One reason is the fact that HMMER misses a huge number of short exons. And since exons of length 0-60nt account for the highest frequency (Figure 5.1), TPR level is very low. Another part of these exons are missed during thresholding $e$-values. That is why the TPR is dropped for the first range of length of exons. TPR grows immediately after that and reaches 0.90%. We also see another TPR drop in the last bar. Looking at the frequency histogram, we can see that the number of very long exons are very low. It means that the denominator of TPR formula which is the total number of exons in that range is small, and missing one exon contributes to relatively bigger drop in TPR. It is also worth mentioning that, since the number of short exons are significantly more than long exons, the impact of this low exon discovery rate in short exons will not result in catastrophic nucleotide-level TPR.

| | Sensitivity (Sn) or TPR | Precision (PRC) |
|---|---|---|
| Exon Hunter | 0.84 | 0.90 |

Table 5.6: The Nucleotide-level statistics of Exon Hunter on the dataset. It is worth mentioning that the nucleotide-level performance measures are absolute values. Relative nucleotide-level TPR would be to subtract the length of the exons that have been removed by the two previous steps from the denominator of TPR formula.

Table 5.6 shows the nucleotide-level performance of EH. Missing exons and wrong starting and ending junctions of the exons contribute to the imperfect nucleotide-level statistics. As we mentioned before, Sensitivity (TPR) is higher than expectation, since the low sensitivity in exon-level is because of missing many short exons. The absolute exon-level TPR is lower than absolute nucleotide-level TPR which is a justification for this

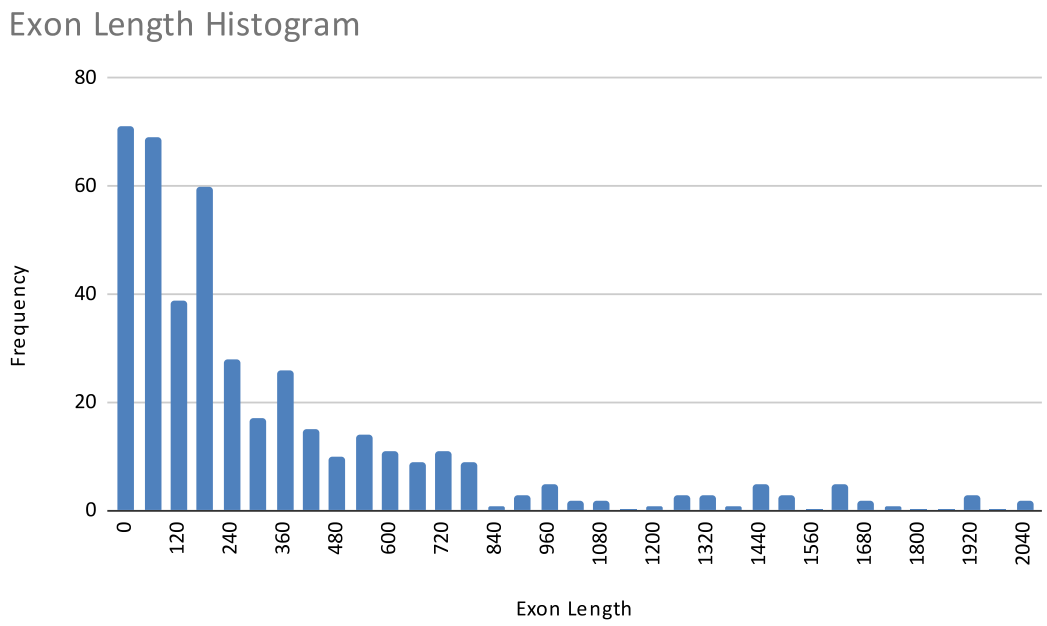


Figure 5.1: The frequency of exons in different lengths.

claim. To be more accurate in this analysis, let's take a look at Figure 5.2 that shows the TPR of Exon Hunter for different exons length. It is clear that it is easier for Exon Hunter to detect exons that are longer than 50 nucleotides. Since longer exons have a bigger impact on nucleotide-level TPR, exon-level TPR can be lower than nucleotide-level TPR.

HMMER reports the hits in two different ranges: alignment range and envelope range. The alignment range is a more conservative result of the search, and is the one that is being used by EH. On the other hand, the envelope range is a wider window of the same hit. We did an experiment where instead of the alignment range, we chose the envelope range. Our expectation was that in some cases that the hit is smaller than its corresponding exon, the

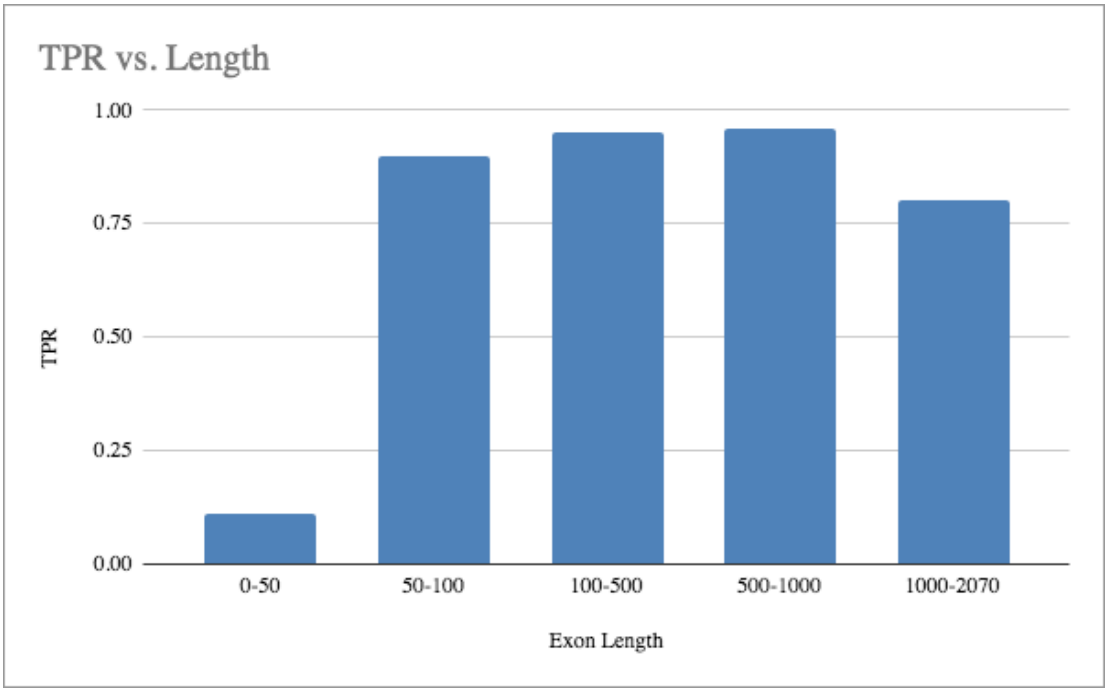


Figure 5.2: Absolute Exon-level TPR of Exon Hunter in 5 different exon length ranges. The percentages of chunks are 11%, 90%, 95%, 96%, and 0.80%.

envelope range would perform better by increasing TPR, but PRC would be lower. Our expectation turned out to be right. In case of using the envelope range, TPR increases by 2%. However PRC drops by 6%. For these calculations, the same $e$-value thresholds were used, however, recalculating $e$-value thresholds might have a slight impact on these performance measures. This result is not preferable; therefore alignment range would be our choice.

Figure 5.3 shows how Exon Hunter is performing in different areas of the exons. Each exon is divided by 3 chunks, that each chunk make up to 33% of the exon. As we talked about the performance of annotation programs prediction of splice sites are one of the main challenges that these programs have to face. The middle area of the exons seems to be

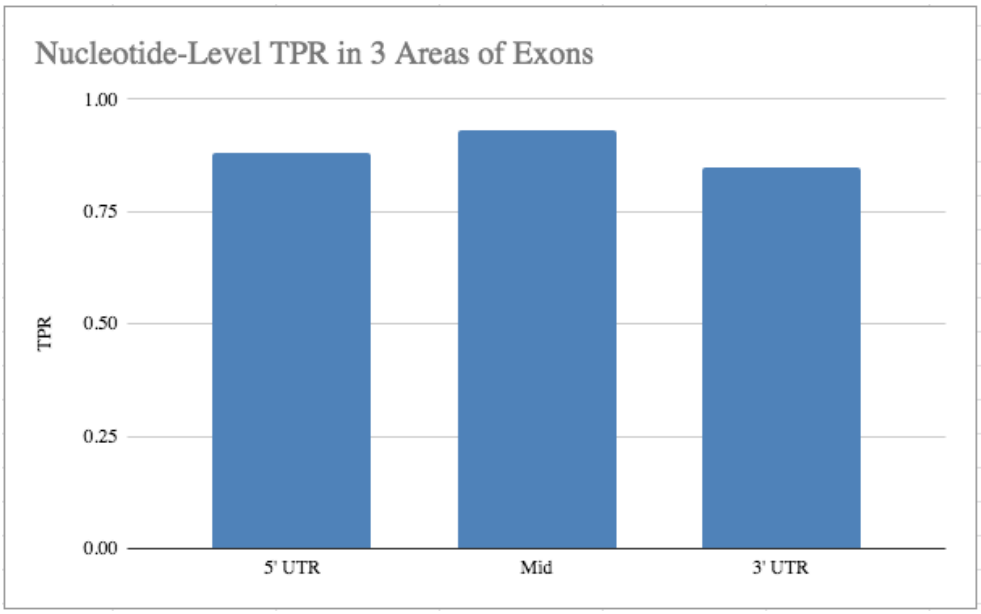


Figure 5.3: Nucleotide-Level TPR of 3 Areas of Exons. Each column of the graph corresponds to 33% of each exon. From Left the TPRs are 0.88%, 0.93%, and 0.85%.

predicted easier than the 3' and 5' end of an exon. We will talk about how we can tackle this problem later in the thesis.

Figure 5.4 shows how Exon Hunter has predicted splice sites. We divided the location of predicted splice sites into 10 distance categories, compared to the location of the actual splice site. Then, we calculated what percentage of splice sites falls in each distance categories. The acceptor splice site seems to be easier to predict, compared to donor splice site. Almost half of all the acceptor splice sites have been predicted only by 5 nucleotide distance. Since HMM is responsible for nucleotide-level performance, the HMM protein profile should be quite conserved in acceptor splice sites. The figure shows two peaks. One in the negative side for acceptor splice site, and another in the positive side for donor splice site. Both of these peaks tells us that typically, Exon Hunter tends to overextend.

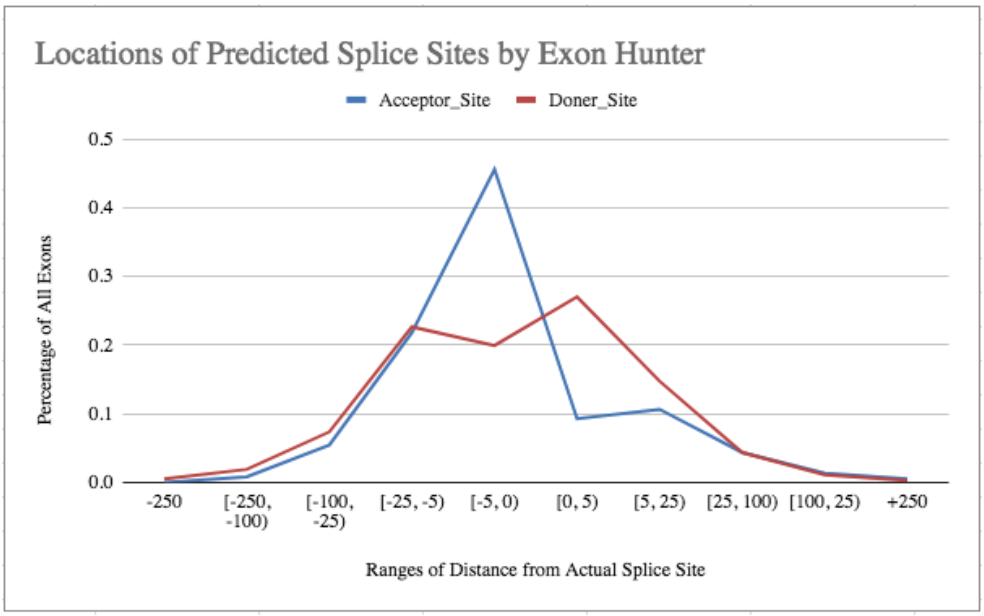


Figure 5.4: Locations of Predicted Splice Sites by Exon Hunter. Blue line is for acceptor splice site, and red line corresponds to donor splice site. Negative values for acceptor splice site mean that the hit is longer, and positive values mean hit was shorter. In contrast, negative values for donor splice site means the hit does not cover the actual splice site, and positive values mean that hit has overextended. 7% of the donor, and only 3% of acceptor splice site has been perfectly predicted.

Therefore the splice site is covered by our program.

## 5.4 Discovery Limit

Certainly, any methodology like EH has some limits. A limitation is sometimes addressed and resolved; otherwise one should be aware of them when using it.

The input data of a process is one of the factors in an experiment that typically decides the main limitation of a procedure. Here, the organisms, genes, the sequences, alignments,

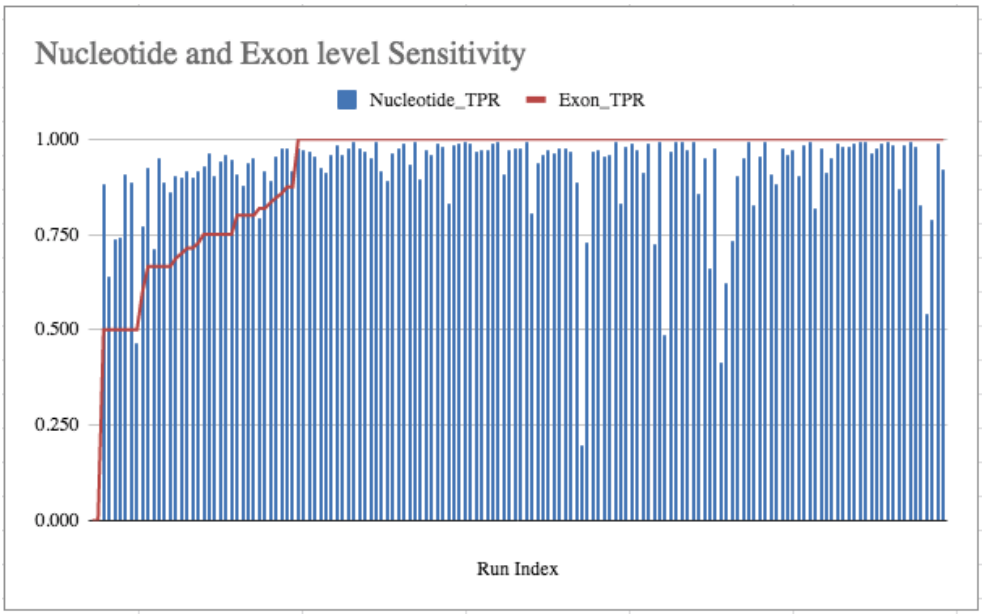


Figure 5.5: Comparing the nucleotide-level and exon-level sensitivity. Each column in this figure corresponds to a run of EH ($genomes \times genes$). For each run the exon-level (red) and nucleotide-level (blue) sensitivity is shown. When the $Exon\_TPR$ is low, which means a small number of exons are discovered (in a genome), and $Nucleotide\_TPR$ is high, it means that the missing exons are short exons. When all the exons are discovered, TPR is mostly close to 1. Run indices are sorted by exon-level TPR.

and HMMs have such an impact on the outcome of the program. Some of these limitations will be mentioned in the discussion.

Since HMMs are the queries that are being looked for in a genomic sequence, they are of great importance. The organisms that have been used to form these HMMs, and their alignment play a significant role in the quality of the results of EH. Obviously, if the target organism is phylogenetically far from the organisms that HMMs were built from, the outcome would not be as mentioned here, or as is desired. But one of these limitations that is not easily seen is the length of the HMMs. Of course, the length of the same gene in

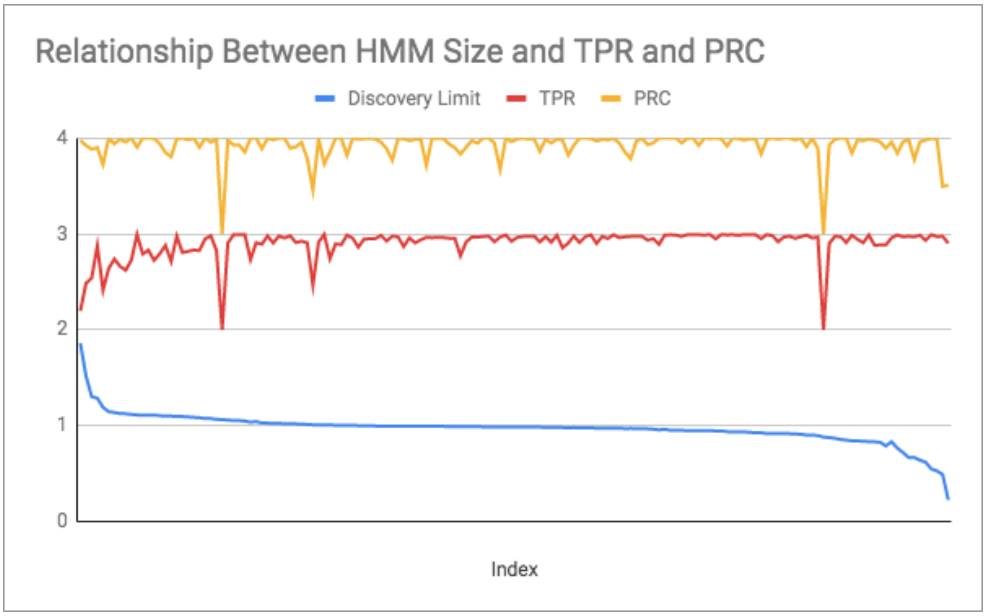


Figure 5.6: The relationship between the size of different HMMs and the outcome statistics. Index (the horizontal axis) shows different runs for each gene and each genome. The blue line is the size of HMMs compared to their corresponding gene length (Gene Size / HMM size). In order to see this relationship better, the TPR and PRC graphs are just shifted up and are shown separately. The left side of the graph shows the cases where the HMM is much smaller (almost half the size) than the gene. And the right side is where the Discovery Limit is low, which means that HMMs are longer than the target genes. The graph is sorted in ascending order according to Discovery Limit.

different organisms (even the closely related ones) can be different. This difference would, eventually, contribute to a consensus length of an HMM that models all of these sequences. In our study, we noticed a clear relationship that the length of the query HMM and the length of the actual gene in the target organism have in the accuracy of the results. Figure 5.6 shows this obvious relationship. The figure shows that when the length of query HMM is much longer or much shorter than the query gene, TPR and PRC are worse. As can be seen, when there is not enough information to make a decision and specify the boundaries

(left side of the figure), as it is expected, TPR is low, which means a lower portion of the gene is detected. On the other hand, when there's more than enough information (right side of the figure), TPR is high, showing that EH works well when the right information is provided. However, the trend is exactly the other way around for PRC. When HMM only covers a little part of the gene, the prediction does not include that many false positive sequences. But when the data is more than what is expected, EH might miss-predict intronic sequences as coding segments, which is reasonable. However, the PRC graph shows that other than a couple of cases, Exon Hunter does not suffer at all when HMM is longer than the gene. There are two cases where the TPR and PRC are dropped to 0. In each case, there is a hit in both the genomes that is even more similar to the HMM than the gene itself. These discrepancies can be a sign of an incorrect annotation which requires further experiments.

## 5.5 Comparison of EH and GeneWise

As we mentioned earlier, in this section we will compare the result of Exon Hunter with GeneWise [10] on the same dataset. Since both of these programs fall into similarity-based approaches.

The computation time of both algorithms are insignificant. The online version of GeneWise reports a time that is consistently less than 4 seconds for each run. Exon Hunter also finishes each task in less than 2 seconds. However the machines that run each method is not the same. Running "hmmsearch" from HMMER is the most time-consuming part of EH. Creating the search tree of all the hits in each run can be intensely time consuming,

| | GeneWise | | ExonHunter | |
|---|---|---|---|---|
| | TPR | PRC | TPR | PRC |
| *atp*6 | 0.24 | 0.33 | 0.87 | 0.91 |
| *cob* | 0.01 | 0.08 | 0.95 | 0.97 |
| *nad*_2 | 0.17 | 0.83 | 0.74 | 0.75 |
| *nad*_3 | 0.15 | 0.16 | 0.80 | 0.94 |
| *nad*_4 | 0.40 | 0.69 | 0.89 | 0.95 |

Table 5.7: The average nucleotide-level statistics of GeneWise and ExonHunter for 5 of the genes.

however, for cases that HMMER results in fewer than 12 hits, this time is negligible. The machine that we used our experiments on had 2.66 GHz Intel Core 2 Duo CPU, and 8 GB DDR3 RAM.

The implementation that we used for running GeneWise does not take an HMM model. We could use a randomly selected sequence representative, and give the algorithm the gene sequence of that specific genome, but the distance between the target genome, and this gene query could make it hard for GeneWise. Therefore, we used "hmmemit" of HMMER in order to produce a consensus sequence of the HMM protein profile. Such sequence is expected to minimize the average distance to any sequence of the set. In other words, it would mimic a sequence that belongs to a common ancestor of the species that the HMM was built from. This way, instead of aligning the sequence of a randomly selected species, an approximation of the sequence of a common ancestor will be used.

Unfortunately, GeneWise did not performed well compared to EH. Therefore, we used a more conservative setting so that the task of gene identification becomes easier. Previously, all the genes in each genome were considered in calculating the overall statistics of ExonHunter in both exon-level, and nucleotide-level. But in this setting, we only ran

GeneWise on the genes that are present in all the species. Let's suppose that a gene exists in multiple species. It is likely to find that gene in some of the genomes close to a reference (an HMM or a consensus sequence) gene. On the other hand, if only one genome has a specific gene, it is far less likely for the sequence of the gene to be close to a reference gene. In the latter case, typically, this significant distance would account for a remarkable decrease in the performance of the gene finding methods. In addition, since we are feeding GeneWise a consensus sequence rather than an HMM, and obviously HMM is more general and a consensus sequence (although it was produced from the same HMM) is more specific, it is expected that the reference gene sequence is more different from the genes in species. This way, at least in a fraction of those species that the consensus is phylogenetically closer to them, GeneWise would perform relatively better, and the result would show this. This selection actually worsened the performance of EH. However, as Table 5.7 shows, EH remained significantly better than GeneWise in this study. GeneWise has some settings that can be changed if needed. We checked if adjusting these options changed the outcome so that it would be a fair comparison. But unfortunately, tweaking the parameters of GeneWise did not result in better performance of the program. GeneWise has two models for finding splice sites, the flat mode is the typical mode where it uses GT/AG di-nucleotide for these junctions. The implementation also comes with two different algorithms: 6:23, and 21:93, where the latter uses a probabilistic model, but the former lacks one. Neither of these parameters, in addition to local, and global search options, made a remarkable impact. Below, the performance of the two programs is shown for each gene. According to the Table 5.7, ExonHunter outperforms GeneWise, by a large margin. Even if we were to show the same information for each organism, the difference would be mostly drastic.

There is a special case in the performance of GeneWise in predicting the genes of Allomyces Macrogynus. In GeneWise, the average TPR of the 5 genes for this species is 0.61, and the PRC is 0.99. These statistics for EH is 0.78 for TPR, and 0.71 for PRC. Specifically, EH in this organism performs poorly because of the fact that gene nad2 is not predicted. The reason is that a false positive hit is selected instead of the real single exon by mistake. If we were to remove nad2 from the calculation of the performance of EH, TPR and PRC for EH would be 0.97, and 0.88, respectively.

## 5.6 Output of ExonHunter

It is necessary to mention that what is reported by ExonHunter, and how it should be interpreted. ExonHunter performs in two main ways with two different sets of parameters. When the true annotation of the genome is not available, ExonHunter takes a FASTA file that contains the genome sequence of the target organism, an HMM protein profile of the target gene, and the name of the target gene. When the annotation is given, the only difference is that this first parameter is the annotation in MasterFile format, and the last parameter is the output format option. The rest of the parameters are the same as in the previous case. The output format option can take only two possible values of "i", if the indices of the exons are desired, and "s", if the sequences of the exons are wanted as well. Figure 5.7 shows a dummy example of a possible output of ExonHunter. Some of the details (except the performance measures) are modified in order to show all the possible scenarios of the output.

The output starts with mentioning the name of the target gene and organism. The

next line includes the labels of each column in the next rows. Each line only corresponds to an exon if the outcome value is FN, which means that it has not been identified by the ExonHunter. If the outcome is TP, the line includes an exon and its corresponding hit, which means that the exon is found. Every row only corresponds to a hit if the outcome is FP, and it is when the hit does not have any corresponding exon. Of course, all the other ranges of the genome sequence that is not mentioned in the table is considered TN. When the outcome is not TP, exon, or the hit information is filled with NA values. The left side of the table indicates the information about exons. Exon number, the reading frame of the exon, starting and ending position of the exon, and the length of the exon are the first 5 columns of the table. Hit number, reading frame of the hit, the i-evalue of the hit, and starting and ending position of the hit, its length, and its overlap with its corresponding exon are the next columns that are describing the information of the hit. TPR, and PRC indicate how much of the exon has been found, and how much of the hit belongs to the exon. Total number of exons is reported after the table. Number of optimal hits is the number of the hits that have been selected to form the gene from the pool of hits that HMMER outputs. Number of covered exons is the number of exons in the gene that have been found. It means that they have at least one corresponding hit. Total exon size is the summation of the length of all the exons. Total overlap size equals to the summation of all the overlapping positions between exons and their corresponding hits. The discovery score, and coding percentage are overlapping over total exons size, and overlap over total hits size, respectively.

As the figure shows the exon frame and the hit frame do not necessarily match, and that is because of the existence of split codon that we have described before. All the positions

Annotation Information and Statistics for gene: 'cox1' in species: '0302gsw589'

| Exon | E_Frame | Exon_Start | Exon_End | E-Length | Hit | H_Frame | i-Evalue | Hit_Start | Hit_End | H-Length | H-Overlap | TPR | Prec | Outcome |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # 1 | 1 | 44428 | 45043 | 615 | # 1 | 3 | 1.8e-107 | 44427 | 45046 | 621 | 615 | 1.00 | 0.99 | TP |
| # 2 | 1 | 46252 | 46346 | 94 | # 2 | 3 | 1.6e-15 | 46245 | 46348 | 105 | 94 | 1.00 | 0.90 | TP |
| # 3 | 3 | 47562 | 47573 | 11 | NA | NA | NA | NA | NA | NA | NA | 0.00 | 0.00 | FN |
| # 4 | 3 | 48558 | 48569 | 11 | NA | NA | NA | NA | NA | NA | NA | 0.00 | 0.00 | FN |
| # 5 | 3 | 49734 | 49870 | 136 | # 3 | 3 | 3.2e-21 | 49734 | 49870 | 138 | 136 | 1.00 | 0.99 | TP |
| # 6 | 1 | 50983 | 51016 | 33 | # 4 | 3 | 0.03 | 50973 | 51016 | 45 | 33 | 1.00 | 0.73 | TP |
| # 7 | 3 | 52032 | 52239 | 207 | # 5 | 2 | 1.8e-32 | 52031 | 52245 | 216 | 207 | 1.00 | 0.96 | TP |
| # 8 | 3 | 53328 | 53484 | 156 | # 6 | 2 | 1.6e-24 | 53327 | 53688 | 363 | 156 | 1.00 | 0.43 | TP |
| # 9 | 3 | 54690 | 54732 | 42 | NA | NA | NA | NA | NA | NA | NA | 0.00 | 0.00 | FN |
| # 10 | 3 | 56154 | 56457 | 303 | NA | NA | NA | NA | NA | NA | NA | 0.00 | 0.00 | FN |

Number of exons: 10
Number of optimal hits: 6 out of 12
Number of covered exons: 6 out of 10
Total exons size: 1608 nucleotides
Total overlap size: 1488 nucleotides
Discovery score (Overall_TPR): 0.77
Coding Percentage (Overall_PRC): 0.83

Figure 5.7: The output of Exon Hunter Program when an annotation is provided.

and lengths are in nucleotide coordination.

## 5.7 Discussion

In this chapter, we saw that ExonHunter performs well on the 12 species dataset, and also the 5 selected genes. We showed different statistics and performance measures to indicate that with further improvements, and adjustments, Exon Hunter might be able to approach a perfect gene annotation. We will see in next chapters that codon usage, splice site prediction are beneficial for the improvement of gene prediction.

Identification of short exons (typically exons that have $< 50$ nt are considered short) has always been challenging. These exons can be as short as a few nucleotides, making it very difficult for any gene prediction algorithm to detect them [59]. There are some possible approaches to address this issue. One way can be to locate different classes of introns, and then consider what is left to be exons. Classes of introns have different signatures that

can be identified. These signatures can help programs to have an approximation of the location of various parts of a sequence, so that after the assessment of different scenarios, a final decision becomes available. Another way of finding short exons is to adjust the donor and acceptor splice site of each exon, using a regular expression, PSSM (Position-Specific Scoring Matrix) or an HMM.

One of the advantages of Exon Hunter is that it uses the simple idea of maximizing the hits that cover the query sequence. Therefore, it can be applied to any dataset. The Exon Hunter program takes the corresponding annotation as an argument, and after extracting the annotation information, it will report the details about the query gene, and exon locations, and how they match with what Exon Hunter predicts.

With the use of Codon Usage Bias, and also Splice-Site Prediction programs, this annotation can be improved. In the next chapters, we will see how these programs can help the annotation process.

# Chapter 6

# Codon Usage Bias

## 6.1 Results and Discussion

The first steps of our codon bias analysis should be to know how biased our data is in terms of codon usage. If codon usage is able to discriminate coding regions from non-coding regions, there has to be a bias in the codon usage of those regions. Therefore, we need to first show that the codon usage of species and the genes are biased enough to distinguish true hits from false hits. No matter what the underlying process is (mutational or selective), if a codon bias is present, this information can possibly be used to filter false positive hits found by HMMER. For calculating the codon usage bias of the genes, and species we used the average codon usage entropy that we mentioned above.

For each gene in each genome, we calculated the average entropy of codon usage vectors. Low average entropy means that the sequence is more biased in terms of codon usage.

Figure 6.1 is showing the codon usage bias of 5 of the genes across 11 organisms. As the figure shows, nad2, nad4, and atp6 have similar CUB values. However, the graph displays a small difference in the CUB of gene cob and nad3. The overall codon usage bias of each genome is also predictable from this graph. It seems that Brettanomyces and Coccidioides are notably biased in all of their genes compared to Microboctrum, and Smitium.

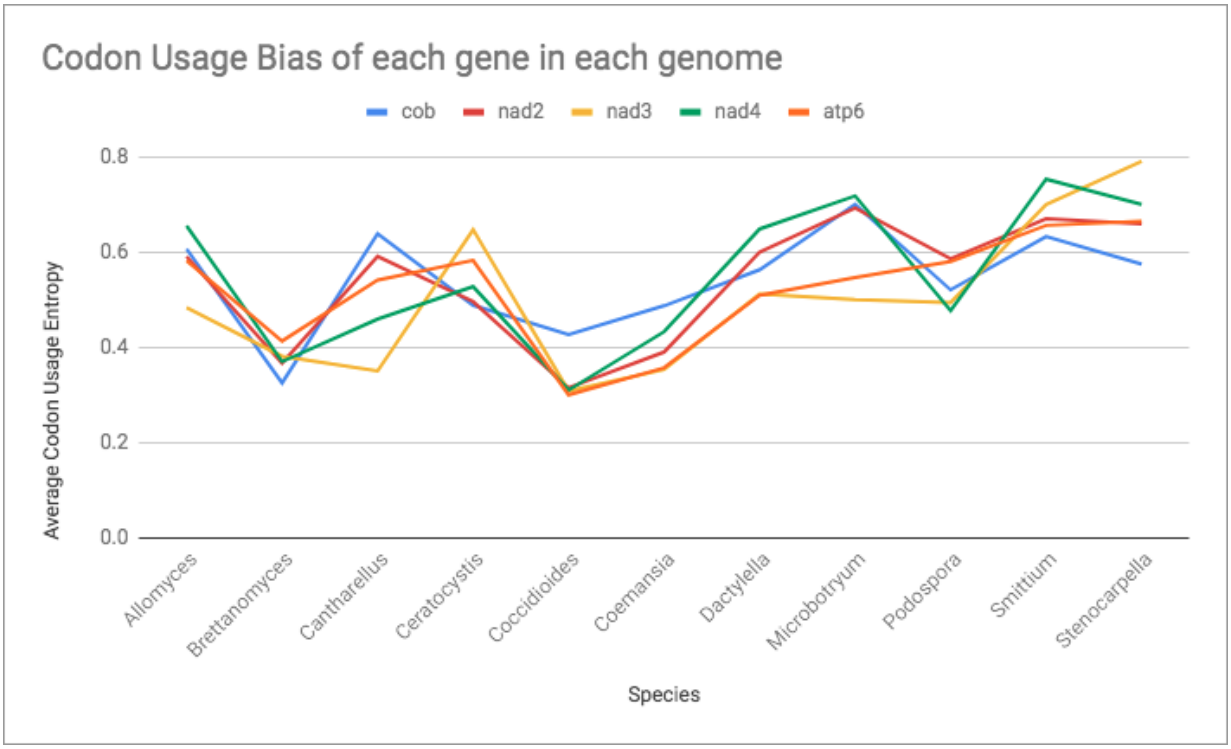


Figure 6.1: The Codon Usage Bias of 5 genes in each genome. Average entropy is used for calculating CUB. Figure shows Brettanomyces and Coccidioides are more biased in their codon usage than Microboctrum, and Smitium. Since the CUB of genes of organisms are relatively close, we can already predict that CUB of an organism is consistent throughout the genome. Figure 5.2 shows how this assumption is relevant.

Next, we examined how genes in a mitochondrial genomes are biased, or, in other words, how a genome is biased. As well as before, we used the average entropy to calculate the

CUB of genomes. Plotkin et al. [54] suggests that different segments of a gene are different in how biased they are. Codons that are closer to the splice sites are more biased than the ones in the middle of the gene. This is probably because splice sites need to be more conserved for the crucial process of initiation. The study proposes that the first 60, and 150 nucleotides of a gene tend to be more biased compared to the rest. Knowing this, for this study, we used different settings in order to show how these claims apply to our data. As Figure 6.2 depicts, the first 60 nucleotides are seemed to be more biased compared to the longer sequences, with some exceptions that seem to happen for the whole genome sequence.

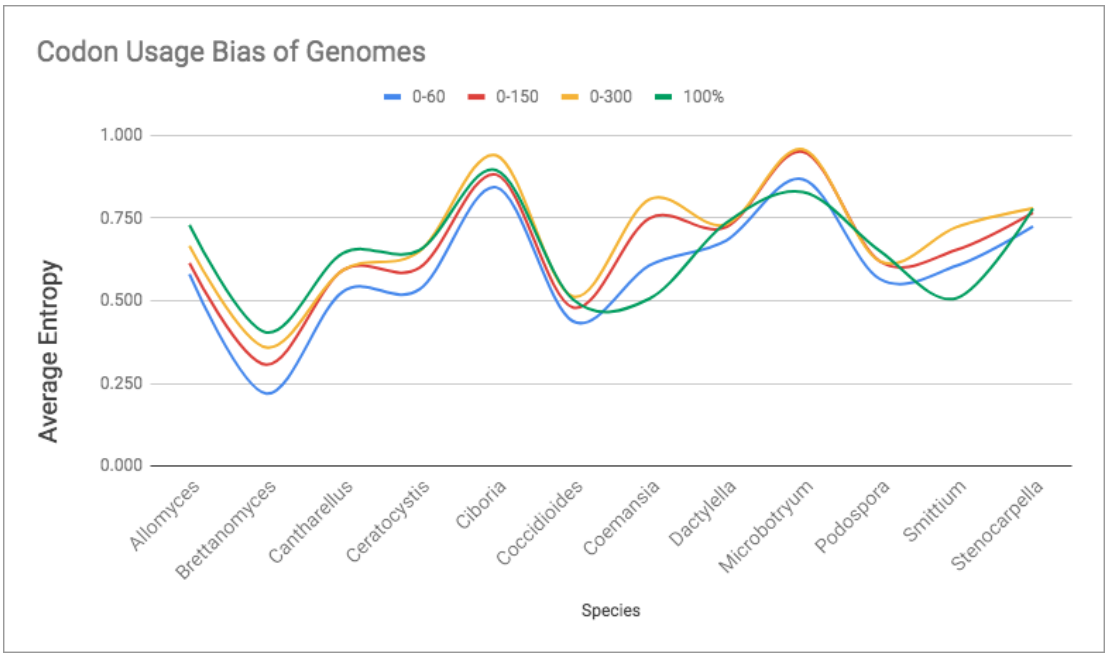


Figure 6.2: Codon Usage Bias of genomes of the dataset. The values are the average of CUB of all the genes that each genome has. Blue: CUB of the first 60 nucleotides of all the genes in each genome. Red: CUB of the first 150 nucleotides of all the genes in each genome. Yellow: CUB of the first 300 nucleotides of all the genes in each genome. Green: CUB of all the nucleotides of all the genes in each genome.

These two studies show how much codon usage can be effective as a feature to decide if a hit belongs to a gene or not. Clearly, if the entropy of codon usage of a gene is low, it means that the CUB of that gene is high, and it is easier to discriminate the hits using this information. If CUB is low, the information would not help us in deciding which hit is true, and which is not. It should be mentioned that it is necessary for the gene to be biased in codon usage, but it is not sufficient. This means that for example for a reliable and satisfactory result, CUB of the reference genome as well as the query gene should be high and at the same time similar. This is because genes of a genome might be highly biased, but have different codon usages.

There are two different ways that we could take advantage of this information. Obviously using the codon usage of a gene, and then comparing the hits with it is not an option since we are not provided by the true annotation before the prediction. One way of utilizing this information is to use the codon usage of other genes in the same genome to compare it with the codon usage of the candidate exons. If the genes of a genome have similar codon usages, and the codon usage bias is relevantly high, we could distinguish the true hits from the false hits using this information. If any of these two conditions are not satisfied, then it would be more difficult to classify the hits using this information. The other way of employing this knowledge is to look for codon usage of the same gene in different genomes. This task would be successful only if the query gene has similar codon usages over the species of our dataset. If the intensity of the CUB is low, or the codon usages are distant from one another, this strategy would not be effective. It should be mentioned that both of these strategies require additional data. For the first way, the genome of the query gene should have already a reasonable number of annotated genes. On the other hand, to

employ the second mechanism, the annotation of other instances of the same gene should be available in multiple different species. Clearly, in both cases, the references (the genes of the same genome, and genes in different species) that will be used in this distinction have a decisive factor. If these references are more relevant, the produced results would be more reliable and acceptable. But if they are from distant species, or the genes have different codon usage profiles, the outcome can be underwhelming.

$$\begin{aligned} \text{Score} = 0.6 \times \text{Coverage} - 0.4 \times \text{Overlap} - 0.8 \times \sum_{i \in S} \log \textit{e-value}_i \\ -\theta \times \sum_{i \in S} \textit{intra-genomic\_d}_i - \gamma \times \sum_{i \in S} \textit{inter-genomic\_d}_i \end{aligned} \tag{5.1}$$

To show that the two strategies that we introduced are feasible solution to filter false positives, we employed them in our objective function. As equation 5.1 shows, we added two terms to our objective function. The first term is called intra-genomic_distance and is the first way of utilizing codon usage information as a factor distinguishing true hits from false hits. Intra-genomic_distance refers to the employment of codon usages of the genes in the same genome as a reference. Inter-genomic_distance, on the other hand, is the case where codon usage of the query gene in different species is considered the reference. Intra-genomic and inter-genomic means within genome and between genome, respectively. We showed how $\alpha$ and $\beta$ were trained. Here, we have $\theta$ (Theta) and $\gamma$ (Gamma) to be trained as coefficient of the codon usage terms. $\alpha$ and $\beta$ were not trained together with the new terms since they have already been trained. However one might argue that training all these parameters simultaneously can result in better overall performance. Although this is a fair argument, the result that we will be presenting below is reasonably good. By

adding these two terms to our objective function, hits that have similar codon usages to their reference codon usage would be favourably chosen.

As we talked about the optimization of parameters in section 4.2, we will use the same grid search strategy for finding the best coefficient sets with some small differences. This time the grid search is series. It means that we are not changing all the coefficients at once. Coefficient of intra-genomic_distance is optimized first, and then with that optimal coefficient, we optimize inter-genomic coefficient. However, optimizing the coefficients in series might not result in a global optimal solution, it is considerably less time consuming. If the number of steps is $n$, a series grid search needs $n$ iteration for each term ($n+n = 2n$). But in parallel grid search, for each $n$ value of the coefficient of the first term, the second term gets $n$ possible values ($n \times n = n^2$). If we were to optimize all four coefficients in parallel, it would require ($n \times n \times n \times n = n^4$) iterations. If we set 10 steps for the search, and since there are 155 data point in our dataset, $10^4 \times 155 = 1,550,000$ iteration of Exon Hunter is needed. But if we optimize them in series, only $4 \times 10 \times 155 = 6200$ iteration is needed. Another advantage of running the optimization in series is that it can be done in four separate runs.

In order to optimize these parameters, we took the following steps. For each gene prediction, we calculate the codon usage of every gene of the same genome except the query gene itself. We then normalize the codon usage, and for every hit we calculate the kl-divergence distance between the codon usage of the hit and the reference codon usage of the genome. As the Equation 5.2 shows, since if the distance is larger it is unlikelier for the hit to belong to the genome, the term has a negative coefficient. In addition, for every gene we calculate the codon usage of the same gene in different organisms. After

| | Rel. TPR | Abs. TPR | Abs. Accuracy |
|---|---|---|---|
| Exon Hunter | 0.93 | 0.82 | 0.96 |
| Exon Hunter* | 0.94 | 0.82 | 0.96 |
| Exon Hunter*_60nt | 0.96 | 0.84 | 0.96 |

Table 6.1: The Exon-level performance of Exon Hunter and Exon Hunter with Codon Usage in two different settings on the dataset. Exon Hunter* considers the whole sequence of genes in order to calculate the codon usage. However, Exon Hunter*_60nt uses only the first 60 nucleotides of each gene in order to make this reference. Exon Hunter* has very little improvement over the Exon Hunter.

normalizing it, we compare the codon usage of the hits with that reference codon usage using kl-divergence. Likewise, closer codon usages are signals that the hit might belong to the query gene that we are trying to predict. It needs to be said that the distance that is calculated using kl-divergence is not between codon usage vectors, but rather between amino acids of each codon usage, as we talked about it in previous sections. Optimizing the coefficients lead to the following equation where $\theta = 0.2$, and $\gamma = 0.2$. Since the coefficients of these terms are non-zero, we can conclude that they have a positive impact on the improvement of the results.

$$\begin{aligned} \text{Score} = 0.6 \times \text{Coverage} - 0.4 \times \text{Overlap} - 0.8 \times \sum_{i \in S} \log \textit{e-value}_i \\ -0.2 \times \sum_{i \in S} \textit{intra-genomic\_d}_i - 0.2 \times \sum_{i \in S} \textit{inter-genomic\_d}_i \end{aligned} \tag{5.2}$$

Since we are optimizing the coefficients and we need to test how they might work in other datasets and to measure their performance, we also used regular 10-fold cross validation for each of the parameters. Table 6.1 shows the performance measures of Exon Hunter and Exon Hunter with Codon Usage.

Table 6.1 shows that using codon usage is helpful in prediction of true hits, since TPR, which is the number of predicted true hits over the total number of true hits, increases. We saw in Figure 6.2 that the first nucleotides of a gene are more biased in their codon usage profiles. Therefore, we implemented two different settings where in one setting we are taking the whole sequence of a gene to calculate the reference codon usage, and in the next setting only the first 60 nucleotides of the gene is taken in making the reference codon usage profile. The results show that the Exon Hunter*_60nt is more successful in the improvements of the outcome. However, accuracy stays the same. This is because true positives contribute very little compared to true negatives in the calculation of accuracy. Therefore, it might not be the best idea to use accuracy as the comparing performance measure. Exon Hunter*_60nt, by using codon usage information achieves a 3% improvement on relative TPR, but 2% increase in absolute TPR. This is because the denominator of relative TPR is smaller than absolute TPR. As we see, absolute accuracy is not a good performance measure in this study since we are dealing with a huge number of false positives.

The employment of codon usage has no effect on TPR of different locations of the gene. Figure 5.3 shows the nucleotide-level TPR of 3 different areas of exons. There's no noticeable difference in the results that are produced by either of new Exon Hunter versions. Although one might expect that because the codon usage bias is high in 5' UTR, improvements are expected to some degree. But unfortunately, that is not the case.

However, as we expected, in long hits, Exon Hunter*_60nt outperforms Exon Hunter. Figure 6.3 shows that Exon Hunter*_60nt has better TPR in longer hits. The reason could be that longer hits possess more information, and short hits less. When a hit is too short, codon usage is sparse. And since we calculate the kl-divergence only for the amino acids

that are present in both reference and query codon usage, small codon usage deviation will lead to irrelevant distances. So as the length of a hit increases, the amount of data that can be used to make a reliable distinction goes up as well. Although the following figure shows a noticeable improvement in in TPR, the overall TPR will not be improved with the same amount since the number of long hits are quite low.


Figure 6.3: Absolute Exon-level TPR of Exon Hunter and Exon Hunter*_60nt in 5 different exon length ranges. Blue bars show TPR of regular Exon Hunter, and red bars correspond to Exon Hunter with first 60 nucleotide codon usage. As expected, in longer hits, codon usage plays an important role in improving TPR.

# Chapter 7

# Conclusion and Future Works

In this thesis, we conducted multiple studies, used different techniques, and took advantage of presenting results in terms of various performance measures in order to predict genes in mitochondrial genomes in a similarity-based approach, and to improve it. Here, in this chapter, we will revisit some of the ideas explored throughout the thesis. We will also propose potential solutions to the problems that were mentioned in the thesis.

HMMER performs well in general. There are two aspects of HMMER that needs to be talked about. The first one is the fact that HMMER produces lots of false positives. Exon Hunter with 96% accuracy shows that it is powerful in eliminating false hits. To illustrate how Exon Hunter is capable of the exclusion of false hits we can calculate the specificity of our method. Specificity is the number of correctly predicted false hits over all the false hits. For Exon Hunter this value is more than 99%. Therefore this huge number of false positive hits is not an issue. However, the second issue which is the identification of short

exons remains unsolved. It is very challenging for many annotating programs to identify short regions that have low information and signatures to detect. There are programs that solve this problem with signal processing methods [61,83]. We could also use composition-based methods to address this issue. Another way could be to locally search for the missed exons. Let's assume that there is a short exon between two relatively longer exons. If those two flanking long exons are predicted as exons of a gene, a local search between the two exons might be able to locate the short exon. Of course if the length of the missed exon is too short (let's say 3 nucleotides) it can be challenging to find the missed exon between the introns that separate the missed exon from the other two exons. Intron prediction can also be a useful information [39, 74, 80]. If we are able to precisely locate introns, we actually have found the missed exon. Even if an approximation of the location of introns are specified, the search space is reduced, and therefore it should be easier to look for the missed short exon.

The next step of our methodology was to remove hits with high $e$-values that are thought to be false positives. We mentioned before that we have this step because the number of hits overwhelms the future step which is the building of a search tree out of every hit. This elimination step was done only using the $e$-value of the hits. But other features such as codon usage and even sequence composition can be used to discriminate true hits and false hits. Using additional information might increase the accuracy of this process, and eventually lead to the exclusion of only false hits. So for example if we were to have two features ($e$-value and codon usage), we would have a classification problem that classifies hits into two classes of false hit and true hit. Still, the trade off of specificity, and sensitivity is challenging. If specificity is high, we might eliminate too many true positive

hits from the pool, and if sensitivity is high, we may end up with too many hits that Exon Hunter cannot be handled. In addition to filtering false hits from true hits, the process of choosing the best subset of hits can also be improved. The first method that might come to mind is using dynamic programming [7]. Dynamic programming can remarkably decrease the computation time, which is currently an issue of Exon Hunter. In addition to dynamic programming, branch-and-bound [38] can be used where in the search tree, the node that have worse scores are not explored. Another approach can be a pre-processing step in which hits that have no overlap with any other hits are added to the final hit set with no need of extra computation. This is because there are no other hits that can cover the positions that are being covered by those hits. Therefore they should be chosen as a candidate exon.

After $e$-value thresholding part, we used regular 10-fold cross validation to optimize the coefficient of each term in our objective function. Obviously, we used cross validation to show the reproducibility of our methodology in an iterative way. It also provides performance measures on the test set which is helpful in the sense that it tells use what we can expect from the program. Test, absolute and relative performance measures were three types of metrics that we used to describe the efficiency, strength and weaknesses of different steps of our methodology. Test performance measures refer to the results that we obtain from the test fold by running cross validation. Relative performance measures tell us about how one specific step of our method performed, while absolute performance measure shows the overall effectiveness of the processes. Stratified cross validation is used when there is a notable class imbalance problem. We used regular cross validation because we had a significant improvement in the class imbalance problem of the hits in thresholding step

by using the stratified version of cross validation. Precision of HMMER was about 20%, however, after the thresholding step it reaches 77% which justifies the use of the regular version of cross validation. In the grid search that was implemented for the optimization of the coefficients, the granularity of step set is of importance. Coarse-grained steps would speed up the optimization at the cost of falling into local optimum result. Fine-grained steps, however, will be time consuming and might guarantee a globally optimal solution. But one should be aware that choosing fine-grained steps might result in an overfitted sets of coefficients. Overfitted coefficients have good performance measures in the training folds of our cross validation, but worse efficiency in test fold. Overfitting means that the model works too well on the training set of a machine-learning process, however loses the generalization of the problem which will result in considerably worse performance on another set of data (for example test data). In Figures of Results and Discussion, we saw that Exon Hunter fails to find short exons but it is pretty successful in finding the longer exons. For increasing the number of found exons (exon-level TPR) Exon Hunter should be able to find those small exons since they are quite frequent compared to long exons. On the other hand, in order to increase the percentage of the gene that is covered (nucleotide-level TPR), one should pay attention to longer exons since they have bigger contribution in covering the query gene. Lastly, we should mention that similarity of the query HMM to the query gene impacts Exon Hunter greatly. Of course, HMM is built on a multiple sequence alignment. If the sequence of the gene of those species are close to the sequence of the query gene, as a result, HMM would be similar, and consequently it would be easier for HMMER to find the exons.

We saw in the codon usage chapter that some genes such as nad3 and atp6 have very

biased codon usages, and the bias intensity is lower for some other genes. We also showed that organisms have different levels of bias. According to this information, we intended to use this feature to distinguish the true hits from false hits. Therefore, we added two terms (intra-genomic and inter-genomic) of codon usages to our objective function, and in a series manner optimized them using a regular 10-fold cross validation. After setting the coefficient of the new terms, we implemented two versions of Exon Hunter (Exon Hunter* and Exon Hunter*_60nt) where one of them uses the whole sequence of a gene to build the codon usage profile, and the second one only uses the first 60 nucleotides of the gene, as was suggested by [54]. We then showed that the last version performs the best and is capable of finding more long exons. In Figure 4.1 we have the Taxonomy Tree of the organisms we have in our dataset. Using this tree as the distances of species in their codon usage did not help us in the production of better results. One reason might be that the distances are not quantitative. So we are not aware how close the organisms are compared to distances of other organisms in the dataset. Such information would give us the knowledge to build the expectation that we can have from the result. However building a phylogenetic tree might be computationally intensive, it might be able to give us a better information about the similarity and differences of the organisms and consequently their codon usage profiles. Using other codon usage measures might also be beneficial for achieving a superior result. As discussed, some of these codon usage bias metrics incorporate expression levels of genes which was mentioned being important in the classification of codon usage profiles. Another pre-processing step could include the calculation of the distance of all the organisms, excluding the query gene and then creating an average codon usage profile with closest organisms according to their codon

usage profile, and consider the average profile as the reference. The same strategy can be applied in building the HMM where only closest species are selected to build the query HMM.

Splice-site prediction is also a typical solution for achieving a better nucleotide-level result. The easiest way to located splice site in the query gene is to build a regular expression. A regular expression is a sequence of characters that specifies a pattern [40]. The canonical splice site for donor site is [AG—GTRAGT] and for acceptor is [YYTTYYYYYYNCAG—G]. These two representations are regular expressions. Regular expressions are built from a specific matrix. A position weight matrix (PWM) or a position-specific scoring matrix (PSSM) is a matrix in which each position of a sequence is represented by a likelihood vector where each element of the vector corresponds to the alphabet of the sequence [71]. The main difference between regular expressions and PSSM is that the former is not probabilistic, and multi-character positions (positions that are represented by a combination of nucleotides) can only be representing uniform distribution. PSSM is calculated by a set of aligned sequences where the frequency of each character in each position divided by the number of sequences in MSA is the value of the corresponding position and character in PSSM. As shown in 7.1, a sequence logo is a visual representation of a PSSM.

In order to show the potential effect of splice sites in helping genome annotation process, we extracted the sequence of the flanking regions of the exons. We then implemented a 10-fold cross validation on both acceptor and donor site of exons. In each iteration, we build a PSSM using only 20 nucleotides (10 nucleotides from the intronic region, and 10 nucleotides from exonic region) of the training folds. We then calculate a similarity measure

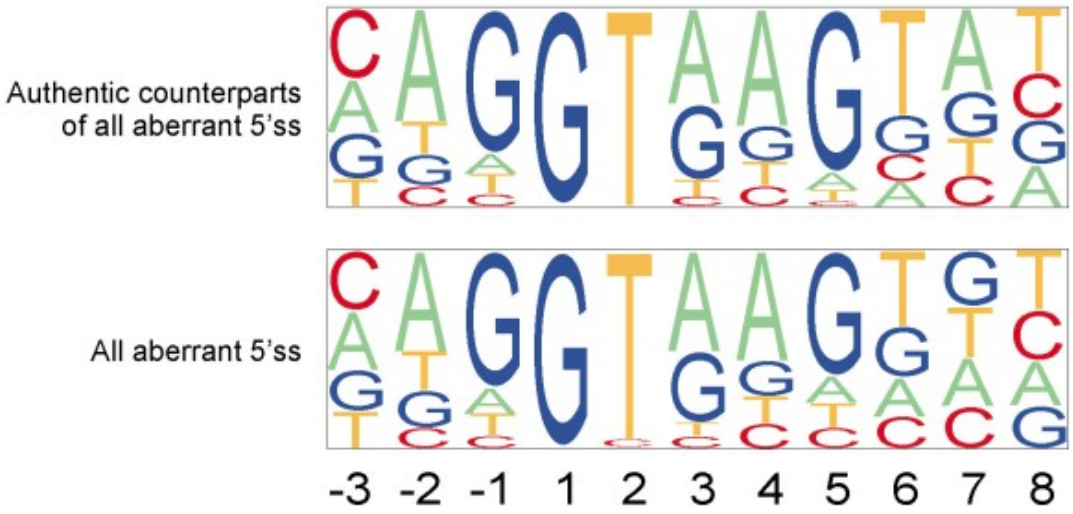


Figure 7.1: The Sequence Logo of 5' splice-site of human genome [12][1]. In each sequence logo, top letters are more frequent, and their size simply is proportional to their domination in that position. Sequence logo is built from a PSSM.

where the summation of the probabilities in the PSSM for each position of the test fold is calculated. Test fold is 10 times the training sequence length (200 nucleotides). Since test sequences are longer than PSSM, we move the PSSM like a sliding window and calculate the similarity for each possible position. Therefore we have a vector of similarity measures for each test sequence. The true splice site is 21, and 25 percent of the times is in top 5 percent of the similarity vectors, for donor, and acceptor splice sites, respectively. However this result is not considered a quality outcome, with some improvements might play a very important factor in annotation. We will point out some of the ideas that can improve the results. We saw previously that the canonical splice site sequences have a specific length in each region of the site. However in this simple and naive approach we have used an equal number of nucleotides of both regions. As we mentioned before, intron classes can be predicted. With such information, it is possible to cluster splice sites that have similar

surrounding introns, and compute PSSM for each cluster separately. Finally, an HMM can be built from an MSA of splice sites, and then one can search for that HMM profile using HMMER. Obviously, if we were to use HMM to find splice sites, we could still make use of features that can separate the sites, so that HMM has a better model of the regions, and consequently, will result in better performance. Other than the splice site, one can firstly aim for detecting the promoter of the genes, and locate the splice site afterwards.

# References


[1] Josep F. Abril and Sergi Castellano. Genome Annotation. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 195–209. Academic Press, Oxford, 2019.

[2] Bruce Alberts, Dennis Bray, Karen Hopkin, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. Essential cell biology. *Control of Gene Expression. 4th ed. Garland Science*, 2013.

[3] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[4] Siv GE Andersson and Paul M Sharp. Codon usage and base composition in Rickettsia prowazekii. *Journal of molecular evolution*, 42(5):525–536, 1996.

[5] RK Atig, S Hsouna, E Beraud-Colomb, and S Abdelhak. Mitochondrial dna: properties and applications. *Archives de L'institut Pasteur de Tunis*, 86(1-4):3–14, 2009.

[6] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

[7] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[8] Jeffrey L Bennetzen and Benjamin D Hall. Codon selection in yeast. *Journal of Biological Chemistry*, 257(6):3026–3031, 1982.

[9] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, James Ostell, Kim D Pruitt, and Eric W Sayers. GenBank. *Nucleic acids research*, 46(D1):D41–D47, 2018.

[10] Ewan Birney, Michele Clamp, and Richard Durbin. GeneWise and genomewise. *Genome research*, 14(5):988–995, 2004.

[11] Terence A Brown. Genome anatomies. In *Genomes. 2nd edition.* Wiley-Liss, 2002.

[12] E. Buratti, M. Chivers, J. Kralovicova, M. Romano, M. Baralle, A. Krainer, and I. Vorechovsky. Aberrant 5' splice sites in human disease genes: mutation pattern, nucleotide structure and comparison of computational tools that predict their utilization. *Nucleic Acids Research*, 35:4250 – 4263, 2007.

[13] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of molecular biology*, 268(1):78–94, 1997.

[14] Gina M Cannarozzi and Adrian Schneider. *Codon Evolution: Mechanisms and Models.* Oxford University Press, Oxford, UK, 2012.

[15] Ruaidhri Cappa, Cassio de Campos, Alexander P Maxwell, and Amy J McKnight. "Mitochondrial Toolbox"–A Review of Online Resources to Explore Mitochondrial Genomics. *Frontiers in Genetics*, 11, 2020.

[16] Trees-Juen Chuang, Wen-Chang Lin, Hurng-Chun Lee, Chi-Wei Wang, Keh-Lin Hsiao, Zi-Hao Wang, Danny Shieh, Simon C Lin, and Lan-Yang Ch'ang. A complexity reduction algorithm for analysis and annotation of large genomic sequences. *Genome research*, 13(2):313–322, 2003.

[17] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[18] Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[19] Francis Crick, Leslie Barnett, Sydney Brenner, and Richard J Watts-Tobin. General nature of the genetic code for proteins. *Nature*, 1961.

[20] Arthur L Delcher, Douglas Harmon, Simon Kasif, Owen White, and Steven L Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic acids research*, 27(23):4636–4641, 1999.

[21] Jin Hwan Do and Dong-Kug Choi. Computational approaches to gene prediction. *The Journal of Microbiology*, 44(2):137–144, 2006.

[22] Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 1998.

[23] Sean R Eddy. Accelerated profile HMM searches. *PLoS Comput Biol*, 7(10):e1002195, 2011.

[24] Robert C Edgar and Serafim Batzoglou. Multiple sequence alignment. *Current opinion in structural biology*, 16(3):368–373, 2006.

[25] Monica Franzese and Antonella Iuliano. Hidden Markov Models. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology - Volume 1*, pages 753–762. Elsevier, 2019.

[26] Mikhail S Gelfand, Andrey A Mironov, and Pavel A Pevzner. Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences*, 93(17):9061–9066, 1996.

[27] Mark B Gerstein, Can Bruce, Joel S Rozowsky, Deyou Zheng, Jiang Du, Jan O Korbel, Olof Emanuelsson, Zhengdong D Zhang, Sherman Weissman, and Michael Snyder. What is a gene, post-ENCODE? history and updated definition. *Genome research*, 17(6):669–681, 2007.

[28] Walter Gilbert. Why genes in pieces? *Nature*, 271(5645):501–501, 1978.

[29] Aaron David Goldman and Laura F Landweber. What is a genome? *PLoS genetics*, 12(7):e1006181, 2016.

[30] Manolo Gouy and Christian Gautier. Codon usage in bacteria: correlation with gene expressivity. *Nucleic acids research*, 10(22):7055–7074, 1982.

[31] John F Griffiths, Anthony JF Griffiths, Susan R Wessler, Richard C Lewontin, William M Gelbart, David T Suzuki, Jeffrey H Miller, et al. *An introduction to genetic analysis*. Macmillan, 2005.

[32] Roderic Guigó, Steen Knudsen, Neil Drake, and Temple Smith. Prediction of gene structure. *Journal of molecular biology*, 226(1):141–157, 1992.

[33] Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC bioinformatics*, 11(1):1–11, 2010.

[34] Toshimichi Ikemura. Correlation between the abundance of Escherichia coli transfer RNAs and the occurrence of the respective codons in its protein genes: a proposal for a synonymous codon choice that is optimal for the E. coli translational system. *Journal of molecular biology*, 151(3):389–409, 1981.

[35] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[36] Eugene V Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.*, 39:309–338, 2005.

[37] Ian Korf. Gene finding in novel genomes. *BMC bioinformatics*, 5(1):1–9, 2004.

[38] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[39] Fulvio Lazzarato, Giuliana Franceschinis, Marco Botta, Francesca Cordero, and Raffaele A Calogero. RRE: a tool for the extraction of non-coding regions surrounding annotated genes from genomic datasets. *Bioinformatics*, 20(16):2848–2850, 2004.

[40] Jing-Jing Li, De-Shuang Huang, Robert M Maccallum, and Xiao-Run Wu. Characterizing human gene splice sites using evolved regular expressions. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 1, pages 493–498. IEEE, 2005.

[41] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[42] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *Journal of Biomedicine and Biotechnology*, 2012, 2012.

[43] Alexander V Lukashin and Mark Borodovsky. GeneMark. HMM: new solutions for gene finding. *Nucleic acids research*, 26(4):1107–1115, 1998.

[44] Naim Matasci, Ling-Hong Hung, Zhixiang Yan, Eric J Carpenter, Norman J Wickett, Siavash Mirarab, Nam Nguyen, Tandy Warnow, Saravanaraj Ayyampalayam, Michael Barker, et al. Data access for the 1,000 Plants (1KP) project. *Gigascience*, 3(1):2047–217X, 2014.

[45] Catherine Mathé, Marie-France Sagot, Thomas Schiex, and Pierre Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic acids research*, 30(19):4103–4117, 2002.

[46] Irmtraud M Meyer and Richard Durbin. Gene structure conservation aids similarity based gene prediction. *Nucleic acids research*, 32(2):776–783, 2004.

[47] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[48] Cédric Notredame. Recent evolutions of multiple sequence alignment algorithms. *PLoS Comput Biol*, 3(8):e123, 2007.

[49] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, 302(1):205–217, 2000.

[50] Genome 10K Community of Scientists. Genome 10K: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

[51] José L Oliver and Antonio Marín. A relationship between GC content and coding-sequence length. *Journal of Molecular Evolution*, 43(3):216–223, 1996.

[52] Clare M O'Connor, Jill U Adams, and Jennifer Fairman. *Essentials of cell biology*. NPG Education, Cambridge, MA, 2010.

[53] Graziano Pesole. What is a gene? An updated operational definition. *Gene*, 417(1-2):1–4, 2008.

[54] Joshua B Plotkin and Grzegorz Kudla. Synonymous but not the same: the causes and consequences of codon bias. *Nature Reviews Genetics*, 12(1):32–42, 2011.

[55] Joshua B Plotkin, Harlan Robins, and Arnold J Levine. Tissue-specific codon usage and the expression of human genes. *Proceedings of the National Academy of Sciences*, 101(34):12588–12591, 2004.

[56] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[57] GPS Raghava, Stephen MJ Searle, Patrick C Audley, Jonathan D Barber, and Geoffrey J Barton. OXBench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC bioinformatics*, 4(1):1–23, 2003.

[58] Steven L Salzberg, Mihaela Pertea, Arthur L Delcher, Malcolm J Gardner, and Hervé Tettelin. Interpolated Markov models for eukaryotic gene finding. *Genomics*, 59(1):24–31, 1999.

[59] Nicolas Scalzitti, Anne Jeannin-Girardon, Pierre Collet, Olivier Poch, and Julie D Thompson. A benchmark study of ab initio gene prediction methods in diverse eukaryotic organisms. *BMC genomics*, 21:1–20, 2020.

[60] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[61] Sunildatt Sharma, Sanjeev Narayan Sharma, and Rajiv Saxena. Identification of short exons disunited by a short intron in eukaryotic DNA regions. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(5):1660–1670, 2019.

[62] Paul M Sharp and Elizabeth Cowe. Synonymous codon usage in Saccharomyces cerevisiae. *Yeast*, 7(7):657–678, 1991.

[63] Paul M Sharp and Wen-Hsiung Li. An evolutionary perspective on synonymous codon usage in unicellular organisms. *Journal of molecular evolution*, 24(1-2):28–38, 1986.

[64] Paul M Sharp and Wen-Hsiung Li. The codon adaptation index-a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic acids research*, 15(3):1281–1295, 1987.

[65] Paul M Sharp, Therese MF Tuohy, and Krzysztof R Mosurski. Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes. *Nucleic acids research*, 14(13):5125–5143, 1986.

[66] Guy St C Slater and Ewan Birney. Automated generation of heuristics for biological sequence comparison. *BMC bioinformatics*, 6(1):1–11, 2005.

[67] Roy D Sleator. An overview of the current status of eukaryote gene prediction strategies. *Gene*, 461(1-2):1–4, 2010.

[68] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[69] Mario Stanke, Oliver Schöffmann, Burkhard Morgenstern, and Stephan Waack. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC bioinformatics*, 7(1):1–11, 2006.

[70] Lincoln Stein. Genome annotation: from sequence to biology. *Nature reviews genetics*, 2(7):493–503, 2001.

[71] Gary D Stormo, Thomas D Schneider, Larry Gold, and Andrzej Ehrenfeucht. Use of the 'Perceptron'algorithm to distinguish translational initiation sites in E. coli. *Nucleic acids research*, 10(9):2997–3011, 1982.

[72] Haruo Suzuki, Rintaro Saito, and Masaru Tomita. The 'weighted sum of relative entropy': a new index for synonymous codon usage bias. *Gene*, 335:19–23, 2004.

[73] Julie D Thompson, Benjamin Linard, Odile Lecompte, and Olivier Poch. A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives. *PloS one*, 6(3):e18093, 2011.

[74] Igor Titov, Nikolay Kobalo, Denis Vorobyev, and Alexander Kulikov. A Bioinformatic Method For Identifying Group II Introns In Organella Genomes. *Frontiers in genetics*, 10:1135, 2019.

[75] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[76] Xiu-Feng Wan, Dong Xu, Andris Kleinhofs, and Jizhong Zhou. Quantitative relationship between synonymous codon usage bias and GC composition across unicellular genomes. *BMC Evolutionary Biology*, 4(1):1–11, 2004.

[77] Zhuo Wang, Yazhu Chen, and Yixue Li. A brief review of computational gene prediction methods. *Genomics, proteomics & bioinformatics*, 2(4):216–221, 2004.

[78] James D Watson and Francis HC Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

[79] Ying Xu, Richard J Mural, and Edward C Uberbacher. Inferring gene structures in genomic sequences using pattern recognition and expressed sequence tags. In *Ismb*, volume 5, pages 344–353, 1997.

[80] Long Yang and Hwan-Gue Cho. Comparative Evaluation of Intron Prediction Methods and Detection of Plant Genome Annotation Using Intron Length Distributions. *Genomics & informatics*, 10(1):58, 2012.

[81] Charles Yanofsky. Establishing the triplet nature of the genetic code. *Cell*, 128(5):815–818, 2007.

[82] Ru-Fang Yeh, Lee P Lim, and Christopher B Burge. Computational inference of homologous gene structures in the human genome. *Genome research*, 11(5):803–816, 2001.

[83] Xiaolei Zhang, Zhiwei Shen, Guishan Zhang, Yuanyu Shen, Miaomiao Chen, Jiaxiang Zhao, and Renhua Wu. Short exon detection via wavelet transform modulus maxima. *PloS one*, 11(9):e0163088, 2016.

# APPENDICES

# Appendix A

# Master File Format

The file format of these annotations is MasterFile. The manual of the MasterFile format describes how the format is structured. But in order to understand some features of the species, it is essential to mention some of the rules and syntaxes.

- Header: A header line is basically a line that is not supposed to be parsed by a parsing software. They are typically some extra information such as Date of annotation, Taxonomy ID, list of genes, ect. Example:
  ;; Gene Totals: 68

- FASTA line: The sequence of every MasterFile starts with a FASTA line.

- Gene: A gene line starts with a single ";" like a feature line, but it continues with "G-[name]" that "G-" is the indicator of a gene, "[name]" is the name of the gene. After that the direction of the contig is specified by either "==>" or "<==" for 5'

to 3' and 3' to 5', respectively. At the end, the starting point or the ending point of a gene is indicated by "start" or "end". This means a gene line is a line that determines the start or the end of a gene. Each of the elements that are mentioned are separated by an arbitrary number of spaces. Example:

; G-cob $\Longrightarrow$ start

[sequence of the gene]

; G-cob $\Longrightarrow$ end

- Exon and Intron: The same structure of the gene syntax applies for exon and intron. The only difference is that at the end of the gene name "-E[EXON-NUMBER]" is added. Example:

  ; G-cob $\Longrightarrow$ start

  ; G-cob-E1 $\Longrightarrow$ start

  [sequence of the exon]

  ; G-cob-E1 $\Longrightarrow$ end

  [the rest of the sequence of the gene]

  ; G-cob $\Longrightarrow$ end

- Sequence: A typical sequence line includes a position number, some spaces, and a sequence composed of "atcgATCGnN!". "N' or "n" represent any nucleotide, and "!" exclamation mark is used to delimit anticodons of tRNA genes. The latter character has no equivalent character in GenBank records, and is completely ignored in this work.

As we mentioned before, there are some aspects of the annotation that we need to note.