



UNIVERSIDADE DA BEIRA INTERIOR  
Ciências da Saúde

# Mining the genome for ‘known unknowns’

Nicolas James Queirós da Silva

Dissertação para obtenção do Grau de Mestre em  
**Ciências Biomédicas**  
(2º ciclo de estudos)

Orientador: Prof. Doutor Nuno Pombo

Covilhã, junho de 2019



*“À minha família, à qual eu devo tudo o que sou hoje”*



# Acknowledgments

Throughout this last year, this dissertation work has counted with several collaborations and supports, directly and indirectly, from several people.

First and foremost, I want to thank my supervisor Prof. Doutor Nuno Pombo for the opportunity and confidence deposited in me, for all the support, availability and patience. I know that all that has been learnt will not be forgotten. For all of the above, I express my gratitude.

A thank you to all my friends that accompanied on this long and arduous journey. From the first year at this university, friendships that have lasted throughout the years and turned out to be fantastic, to my closest friends that have always been there when I needed them and to friendships that have lasted for years, starting even before my entry in this university. Each and every one of you contributed to the conclusion of this stage.

To my parents, whom even though have always lived far away from me during these years, have supported and loved me unconditionally, through good and bad times, whenever I needed them. Two fantastic and very persevering human beings, whom I see as my role models and have the utmost respect for. You have definitively been two huge pillars throughout these years. I cannot thank you enough.

To my brothers whom are also far but always close. Two fantastic, and crazy, boys that have always made me smile and proud of being their older brother. To you two a huge thank you.

To my grandparents who have always supported me, been available and given me a place to stay, whenever I needed one. For all their love and support throughout these years, their patience and perseverance. And a special thank you to my late grandfather, whom even though is not with us, will always be remembered for the fantastic, caring, kind and loving person he was. You will never be forgotten and will always be in our hearts.

To the rest of my family who have supported me throughout these many years, directly and indirectly. To you, a big thank you.



# Preface

My interest for biology and programming have always been present throughout my whole life. The last 4 years before starting my dissertation made me change my mind when it came to choosing a career path or even what area to work in the future. Even though I graduated in Biomedical Sciences, working in a programming related area was always a medium to long-term possibility and ambition. Despite enjoying investigation and being in a laboratory, I couldn't imagine myself doing that for the rest of my life. Along the years, it became more and more clear to me that the life of a person working in a laboratory wasn't that simple nor straightforward. It had its positives and negatives. But when I dug deeper into the whole investigation scenario, I found more negatives than positives. So, I decided I needed to change my course of action.

That brings me to my last year of my undergraduate degree. What was I going to do after I finished? So, I thought to myself "Sure, I am going to do my Masters, but in what area?". In my last year, there were two main subjects that help me make a decision as to what I might want to do in the future: Genetics and Bioinformatics. Like I mentioned before, I always liked programming, and genetics was also always an area of interest to me. Bioinformatics, since it encompassed both computer science and genetics, it was the ideal area. It combined both biology and programming, being exactly what I was looking for.

I applied for the MSc program in Bioinformatics and Computational Biology in Lisbon at the Faculdade de Ciências da Universidade de Lisboa (FCUL), but unfortunately didn't manage to enter. I applied for the MSc in Biomedical Sciences at Universidade da Beira Interior (UBI), my current university, and decided that, as my dissertation work, I would propose a theme in the bioinformatics area. And that's exactly what I did.

As mentioned before, I wanted to propose a theme related to the bioinformatics area. After extensive research, I decided on a specific area inside bioinformatics: Gene prediction tools. This area was completely new to me. I knew it would be a challenge and would require hard work, but it was going to be interesting. Many tools already exist, and are used today to aid geneticists in their investigations. But something I noticed was that, even though these tools were quite abundant, most of them are several years old. This wasn't necessarily bad, it simply meant that those tools were still good and their specificity and sensitivity was still good for modern times. The main goal for my dissertation work was to try and create something new, not necessarily from scratch, but something that could be innovative in the gene prediction tool area. Using pre-existent data and code, the goal was to try and make something a little faster and reliable. It wouldn't give us as much information as a normal gene prediction tool would nor would it be as effective, but it would point the person who used it in the right direction, when various sequences are compared, using the model in question. A code to

discover Open Reading Frames (ORF) is used, storing them in a file created by the user. Then a Profile-HMM (Profile Hidden Markov Model) is used as the main model. The Profile-HMM returns us the Logarithmic Probability (Logp) of a certain sequence matching with another. In this case, the ORFs previously discovered with the desired matching sequence that the user inputs. Logp values are negative, so the closer the number is to 0 (Zero), the more similar it is to the input sequence. Even though this is the case, we managed to make the code output the lowest positive absolute value. The respective compared ORFs are aligned and Logp values compared, manually. Comparisons with same length sequence ORFs are made, to verify that the model worked.

We can say that something new was created. It might not be the best way to predict genes or even be big or complex enough to be considered a gene prediction tool, but we think it's definitively something innovative and should be considered as a base model for future investigation and research.





# Resumo

A bioinformática é uma área multidisciplinar que combina duas áreas fundamentais: biologia e ciências da computação. É uma das áreas de investigação que mais está a crescer nos dias de hoje. É também uma área fundamental para o processamento de dados e informação na área da genética. Um ramo prominente na área da bioinformática é a predição de genes. Várias ferramentas encontram-se disponíveis para auxiliar investigadores. Estas ferramentas também se encontram disponíveis ao público em geral. Embora existam várias ferramentas, as mais utilizadas já têm muitos anos. São ferramentas fiáveis porém algumas precisam de ser otimizadas e não são muito flexíveis no que toca à sua modificação. Neste trabalho de dissertação é proposto um novo modelo. Por meio da extração de *ORFs* a partir de sequências de DNA que codificam para proteínas, inserido pelo usuário em formato *fasta*, estes são comparados com uma sequência alvo escolhida pelo utilizador. Foi utilizado um *Profile-HMM* como modelo para comparar as sequências, em que um valor de probabilidade logarítmica ( $\text{Logp}$ ) é devolvido consoante a semelhança entre as sequências comparadas: o *ORF* e a sequência alvo. Quanto mais semelhantes forem as sequências comparadas, melhor será o valor da probabilidade logarítmica. Foram criados vários cenários de modo a ver qual seria a melhor forma de implementar o *Profile-HMM*. Nestes, os estados de correspondência, inserção e deleção foram modificados, até chegar ao melhor cenário. O algoritmo de Viterbi foi utilizado para treinar o modelo, devido à sua velocidade. Os resultados obtidos pelo modelo foram concordantes com o que esperávamos: um *ORF* que está presente na sequência alvo terá um valor  $\text{Logp}$  melhor que um *ORF* que não está presente na sequência alvo.

## Palavras-chave

Algoritmo de Viterbi; Genoma humano; *Open Reading Frame*; Pomegranate; Predição de genes.



# Resumo alargado

Os genes são a base da nossa existência, a razão que somos o que somos nos dias de hoje. a acumulação e o armazenamento de informação ao longo de milhares de anos de evolução. Codificam toda a informação necessária à nossa sobrevivência como espécie. Informação que nos foi transmitida, que nos distingue entre espécies. Somente uma pequena parte do nosso genome é diferente entre nós Humanos, sendo que partilhamos a maior parte dele.

A genética é o ramo da biologia que estuda genes, variação genética e hereditariedade. Ajuda-nos a compreender a complexidade da vida em si e entre organismos, desde um nível molecular até um nível populacional.

Um gene pode ser descrito como a unidade de armazenamento da informação, sendo nos humanos o Ácido Desoxiribonucleico (ADN). Este contém certas características genéticas, que posteriormente são convertidos num produto funcional final.

A área da genética é uma das áreas que mais está a crescer nos dias de hoje, sendo que milhares de publicações são lançadas todos os dias. Porém, a história da genética começou há milhares de anos atrás, com as primeiras descobertas a começar há 10,000 a 12,000 anos atrás. Em 1866 Gregor Mendel, considerado o pai da genética, publicou um artigo sobre hereditariedade em ervilhas. Porém o seu trabalho só foi reconhecido nos anos 1900s, marcando o início de uma nova era na área da genética.

Aos longo dos anos as investigações e as publicações têm vindo a ser cada vez mais, havendo a necessidade de processar e armazenar os dados recolhidos. Um ramo associado à genética que processa e analisa esses dados é a bioinformática. A bioinformática é um ramo que advém da combinação de dois outros principais: A ciência da computação e a biologia.

Este ramo tem vindo a ser cada vez mais popular nos dias de hoje, havendo a necessidade da criação de novas ferramentas para processar dados. Um sub-ramo dentro da bioinformática é o de predição de genes. Este ramo trata da tentativa de prever zonas de genes que possam codificar para proteínas. Diversos programas já existem nos dias de hoje, sendo que todos os anos são lançados novos modelos e implementações, com algoritmos inovadores ou melhorados.

Este plano de trabalhos tem como objetivo criar um modelo de predição de genes, com base em informação disponível e já descrita na literatura. Com este modelo, pretendemos explorar a área de predição de genes e tentar implementar um novo método.

Com a ajuda de um programa que encontra *Open Reading Frames* (ORF) em sequências, que nos permite escolher o seu comprimento, o número que queremos extrair da sequência entre

outras opções, foi possível extrair ORFs de sequências de genes codificantes de proteínas. Isto permite extrair sequências mais específicas das previamente sabidas.

Com a ajuda dos pacotes de Python, Biopython e Pomegranate, foi possível criar o modelo de Markov a utilizar. O modelo de Markov escolhido para implementar foi o Profile-Hidden Markov Model (Profile-HMM), devido à sua capacidade de alinhar múltiplas sequências. Isto permite uma análise ao mesmo tempo das sequências alinhadas, apresentando um resultado de todos os alinhamentos com a sequência alvo, ao mesmo tempo.

O objetivo do alinhamento múltiplo de sequências foi encontrar, inicialmente, o valor da probabilidade logarítmica ( $\text{Logp}$ ) mais baixo. Esta representa a probabilidade de uma dada sequência, dado as probabilidades de correspondência, inserção e deleção, pertencer ou estar alinhada com outra sequência alvo. Para treinar o modelo, foi utilizado o algoritmo de Viterbi, pois este é mais rápido e adequado para o trabalho em questão.

Foram criados diversos cenários, cada um testados com diferentes probabilidades de correspondência, inserção e deleção, sendo que, assim que achado o melhor cenário, o utilizamos para comparar ORFs de sequências que pertenciam à sequência alvo e ORFs de sequências escolhidas ao acaso.

Os resultados mostram que sequências pertencentes à sequência alvo, com o modelo e o cenário utilizados, apresentam uma melhor probabilidade de aparecerem do que sequências escolhidas ao acaso, o que seria esperado.



# Abstract

Bioinformatics is a multidisciplinary area that combines two major areas: Biology and Computer Science. It's one of the fastest rising areas of investigation nowadays. It's also a fundamental area for the processing of data and information from discoveries in the genetics area. One area that is prominent in the bioinformatics area is gene prediction, where various tools are available to aid researchers. Even though there are several gene prediction tools available, the most used are from several years back. They are reliable tools, but need optimization and some are not so flexible for modification. Tools created in the past years base their model on previous tools. In this dissertation work, a new model is proposed. Through ORF extraction from protein-coding sequences of a fasta-formatted file that the user inputs, these are compared to a target sequence of the user's choice. A profile-HMM is used as the model to compare the sequences, returning a  $\text{Logp}$  value for each ORF compared with the target sequence. Match, insert and delete state probabilities were modified, to find the best scenario. The Viterbi algorithm was used to train the model, due to its speed. The results obtained were concordant with what we expected: That an ORF, which would be in the target sequence, presented a better  $\text{Logp}$  value than an ORF from a randomly selected sequence.

# Keywords

Gene prediction; Human genome; Open Reading Frame; Pomegranate; Viterbi algorithm.





# Table of contents

1. Introduction .....	2
1.1. Genetics.....	2
1.1.1. Brief history of genetics.....	3
1.1.2. Genes and the Genome.....	6
1.2. Bioinformatics .....	8
1.2.1. Bioinformatics: A definition .....	8
1.3. Brief History of Bioinformatics.....	8
1.4. Genomic databases .....	11
1.5. Gene prediction tools: Types and examples.....	12
1.5.1. AUGUSTUS .....	13
1.5.2. TaF .....	15
1.5.3. BRAKER1.....	16
1.6. Hidden Markov Models.....	17
1.6.1. Profile-HMM .....	18
1.7. Aims .....	19
2. Materials and Methods .....	21
2.1. Implementation.....	21
2.2. The program and code .....	21
2.2.1. Pomegranate .....	25
2.2.2. Biopython .....	25
2.3. The ORF Finder .....	26
2.4. The Profile-HMM.....	27
3. Results and Discussion.....	29

3.1. Scenario 1 .....	32
3.2. Scenario 2 .....	36
3.3. Scenario 3 .....	39
3.4. Scenario 4 (Insert state scenario) .....	43
3.5. Scenario 5 (Delete state scenario) .....	47
3.6. Final comparison .....	50
4. Conclusion.....	53
5. Future Perspectives .....	55
6. References .....	57



# List of Figures

- Figure 1 - Gregor Mendel and Mendel’s experiments crossing monohybrid peas .....4
- Figure 2 - Timeline of some genomic analysis, over the years .....5
- Figure 3 - Dolly and her predecessor and Carbon Cat .....6
- Figure 4 - Table of all possible 20 amino acids .....7
- Figure 5 - The Human karyotype .....7
- Figure 6 - Needleman-Wunsch pairwise sequence alignment .....9
- Figure 7 - AUGUSTUS states and respective transition possibilities. .... 15
- Figure 8 - Representative scheme of TaF’s pipeline. .... 15
- Figure 9 - Scheme representing the BRAKER1 pipeline ..... 16
- Figure 10 - An example of a small Profile-HMM, with three columns ..... 18
- Figure 11 - Decision tree demonstrating possible scenarios for each state. .... 31
- Figure 12 - Logp values for Scenario 1 ..... 35
- Figure 13 - Logp values for Scenario 2 ..... 38
- Figure 14 - Logp values for Scenario 3 ..... 41
- Figure 15 - Logp values for Scenario 4. .... 45
- Figure 16 - Graphs representing the number of ORFs and Logp values ..... 50



# List of Tables

- Table 1 - Gene prediction tools ..... 13
- Table 2 - Packages/Modules used and their respective functionality. .... 22
- Table 3 - Model start transition values (Scenario 1)..... 33
- Table 4 - Match state transition values (Scenario 1) ..... 33
- Table 5 - Insert state transition values (Scenario 1) ..... 34
- Table 6 - Delete state transition values (Scenario 1) ..... 34
- Table 7 - Model start transition values (Scenario 2)..... 36
- Table 8 - Match state transition values (Scenario 2) ..... 36
- Table 9 - Insert state transition values (Scenario 2) ..... 37
- Table 10 - Delete state transition values (Scenario 2)..... 38
- Table 11 - Model start transition values (Scenario 3) ..... 39
- Table 12 - Match state transition values (Scenario 3)..... 39
- Table 13 - Insert state transition values (Scenario 3) ..... 40
- Table 14 - Delete state transition values (Scenario 3)..... 41
- Table 15 - Model start transition values (Scenario 4) ..... 43
- Table 16 - Match state transition values (Scenario 4)..... 43
- Table 17 - Insert state transition values (Scenario 4) ..... 44
- Table 18 - Delete state transition values (Scenario 4)..... 45
- Table 19 - Model start transition values (Scenario 5) ..... 47
- Table 20 - Match state transition values (Scenario 5)..... 47

Table 21 - Insert state transition values (Scenario 5) ..... 48

Table 22 - Delete state transition values (Scenario 5) ..... 49





# List of acronyms

A	Adenosine
BN	Bayesian Networks
C	Cytosine
CABIOS	<i>Computer Applications in the Biosciences</i>
CC	Carbon Copy
csHMM	Context-Sensitive Hidden Markov Models
d1	Delete State 1
d2	Delete State 2
d3	Delete State 3
d4	Delete State 4
DDBJ	DNA Data Bank of Japan
DNA	Deoxyribonucleic Acid
EM	Expected Maximization
EMBL	European Molecular Biology Laboratory
EST	Expressed Sequence Tag
FCUL	Faculdade de Ciências da Universidade de Lisboa
G	Guanine
GHMM	Generalized Hidden Markov Models
GMM	Gaussian Mixture Models
HMM	Hidden Markov Model
i1	Insert State 1
i2	Insert State 2
i3	Insert state 3
i4	Insert State 4
i5	Insert State 5
Logp	Logarithmic Probability
m1	Match State 1
m2	Match State 2
m3	Match State 3
m4	Match State 4
MLE	Maximum Likelihood Estimation
NGS	Next-generation Sequencing
ORF	Open Reading Frame
Pair-HMM	Pair Hidden Markov Models
PCR	Polymerase Chain Reaction
Perl	Practical Extraction and Reporting Language
Profile-HMM	Profile Hidden Markov Models
SMRT	Single-Molecule Real-Time
SNP	Single Nucleotide Polymorphism
SVM	Support Vector Machine
T	Thymine
UBI	Universidade da Beira Interior
U	Uracil
YAHMM	Yet Another Hidden Markov Model



# Chapter I

# 1. Introduction

Genes are our base, the reason we are what we are at this moment in time. Our genes are the result of thousands of years of evolution and adaptation to the outside world. The accumulation and storage of information throughout thousands of years of evolution, leading to modern times. Information that was stored years ago that is now encoded into us, moulding each specie, differentiating us Humans, for example, from other mammals or even other kingdoms. Us Humans share almost all of our genetic information with each other. Only a small percentage of our genome is different, distinguishing us phenotypically. We are very much alike, but very different at the same time. Our genes encode all information necessary for our survival as a species. But only about 1% of our genes are protein-coding genes, making the rest either “Junk” or regulatory. This second function for the rest of the genes, was only recently discovered to have regulatory functions [1].

## 1.1. Genetics

Genetics, a plural noun deriving from the singular “Genetic”, is referred to the branch of biology that studies genes, genetic variation and heredity in organisms. It allows us to understand the complexity of life itself between all organisms, going from a molecular level to a larger, populational level. The differences and diversity we observe intra and inter-species is known as genetic variation [2], [3].

As mentioned before, genetics is the study of genes, genetic variation and heredity in organisms. A gene is the basic unit for heredity. A gene can be described as a unit coded with information, for example a segment of DNA in Humans, that carries a certain genetic characteristic and produces a functional product. The product of this segment is a polypeptide, or a sequence of amino acids, that form proteins [2], [3].

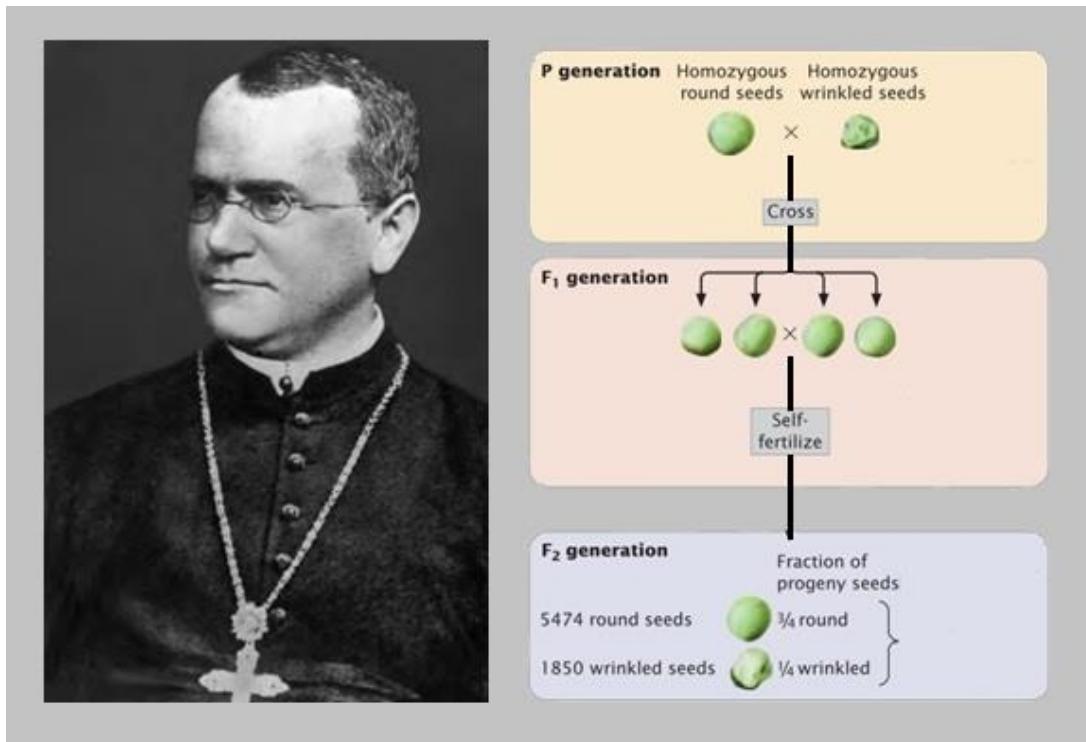
Even though “Genetic” refers to “The study of heredity and the variation of inherited characteristics” [4], Genetics (Plural) is commonly used to define the area of study. It’s a massively growing field in modern times, with new discoveries being published almost every day. Thousands of publications online are directly or indirectly related to genetics, being published at an outstanding speed. Just to get a quick view of how fast articles are published related to the topic “Genetics”, a quick search on PubMed [5] showed that, since the beginning of 2019, 38780 articles have been published with the keyword “Genetics” (as of June 2019). This staggering number of published articles just goes to show the prevalence and importance of genetics as a topic of modern-day research.

### **1.1.1. Brief history of genetics**

Even though there are a lot of articles being published nowadays, the topic of genetics has been around for thousands of years. Evidence shows that the concept of heredity was first applied approximately 10,000 to 12,000 years ago in the Middle East. Organisms like wheat, peas and dogs are some examples. The Assyrians and Babylonians experimented with date palms, planting them so they varied in size, colour, taste and time to ripen. Hindu's, around 2000 years ago, hypothesized that different traits and characteristics were transmitted from the father, and differences between siblings were attributed to the mother. The ancient Greeks developed the concept of pangenesis, which assumed that certain particles from an organism carried information to the reproductive organs. This theory persisted until the late 1800's [3].

It was not until the 1600's that the actual science of genetics was properly recognised. Nehemiah Grew, in 1676, discovered that plants reproduced using male sex cells' pollen. Several botanists started experimenting with plants and creating different hybrid species. Gregor Mendel, the father of Genetics, was one of these botanists. Although Mendel had discovered the basic principles of heredity and published his experiments on pea plants in 1866 [6], his work was not widely recognized in the scientific community until 35 years after his discovery. Darwin also scraped the topic of heredity, and knew it carried a key role in the evolutionary process, but didn't quite understand the nature of inheritance, omitting it in his 1859 publication "On the Origin of Species" [3].

Cytologists demonstrated that the nucleus played a role in fertilization. August Weissman cut off tails of mice during 22 generations, coming to the conclusion that they remained quite long in length. He proposed the germ-plasm theory, which stated that the cells in the reproductive organs carried one full set of genes, that was then passed on to the egg and sperm. Moving on to the year 1900, Mendel's work on the pea plants was recognized, giving biologists a base to work from (Figure 1). Experiments on mice, chickens and other organisms were conducted and their results showed that many traits were according to Mendel's rules. In 1902 Walter Sutton proposed that genes were located in the chromosomes. Thomas Hunt Morgan also contributed to transmission genetics, by conducting experiments on fruit flies, discovering the first generation of them in 1910. In the 1940s geneticists started using bacteria and viruses due to their rapid and simple genetic systems, and in 1953 there was a huge breakthrough in science: The discovery of Deoxyribonucleic acid's (DNA) structure, the double helix [7].



**Figure 1** - Gregor Mendel (on the left) and Mendel's experiments crossing monohybrid peas (on the right) (Adapted from [3]).

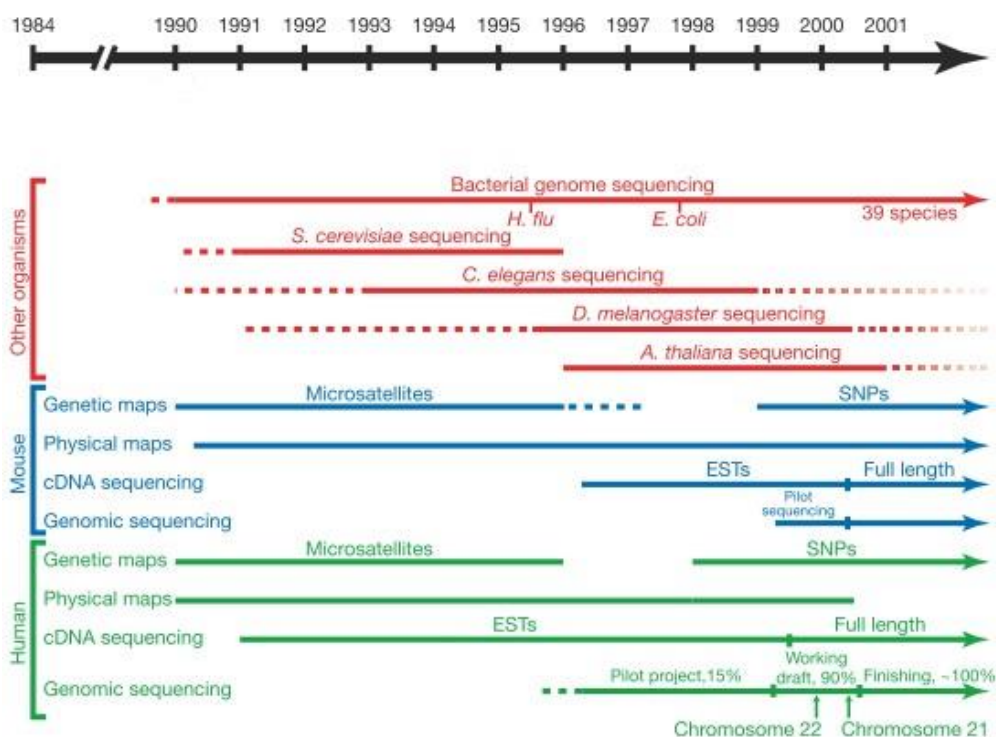
After this discovery, genetic advances took a huge leap. The first recombinant DNA experiments in 1973 provided another boost to the genetics research area. In 1977, Walter Gilbert and Frederick Sanger developed DNA sequencing methods. Kary Mullis and other scientists developed the Polymerase Chain Reaction (PCR) in 1983 [3].

In 1990 the human genome project was launched and everything changed. It was a huge mark in the history of not only genetics but for every scientific area. The project, funded by the National Institutes of Health and the U.S. Department of Energy, with a \$3 billion-dollar funding for the completion of the genomic sequence. Its two main goals, that appeared in the 1980s, were as follow: to accelerate biomedical research and to be a project that would require worldwide efforts from several biomedical communities, infrastructure-wise, on a level never seen before. In 1998 it was announced that the intention was to build a facility to complete the project over 3 years. This huge collaboration of 20 groups aimed to produce a draft genome sequence, initially. The Human Genome Project was divided into three main phases: a preliminary phase, where the aim was to develop and refine key approaches for the next phases; A draft phase, where about 90% of the information for building the sequence was available; a finishing phase that provided researchers with approximately 99% of the euchromatin form of the genome complete. The draft genome sequence was generated from a physical map that covered 96% of the euchromatic (form of chromatin rich in genes) part of the human genome and 94% with additional sequences available in public databases. This sequence

only took around 15 weeks to complete, from 10% to 90%, in this early phase. Despite much work needing to be done, several factors could be determined such as:

- There were several variations when it came to the distribution of several features such as: genes, GC content, CpG islands, transposable elements and recombination rate;
- At first glance, there were about 30,000 to 40,000 protein-coding genes in the human genome;
- Recombination rates tended to be much higher in distal regions (20 megabases (Mb));
- Identification of more than 1.4 million Single nucleotide Polymorphisms (SNP);
- Amongst others.

The Human Genome Project ended and was given as completed in 2003, with the sequencing of a near-complete sequence, going from around 90% in the draft phase, to roughly 99% of the chromatin region of the genome discovered in the finishing phase. Still, the euchromatin portion of the genome was not complete, with 1% left to determine. This was due to the lack of methods to understand these segments in this percentage of euchromatin. Even though almost all the human genome was sequenced, resources to complete the sequencing of the genome should be used in the future [3], [8]-[10].



**Figure 2** - Timeline of some genomic analysis, over the years. ESTs - Expressed sequence tags; SNPs - Single Nucleotide Polymorphisms (Adapted from [9]).

In 1995, the first fully sequenced DNA genome of an organism was determined: The bacteria *Haemophilus influenzae*. A year later, in 1996, the yeast sequence was also determined [11]. In 1997, the first cloned animal was created: Dolly. Ian Wilmut and his colleagues used mammary cells from an adult sheep (Finn Dorset, also known as white-faced sheep). In posterior years, cloning with animals such as mice and cows was done. In the year 2002, a cat by the name of Carbon Copy (CC) was cloned, being the first cloned pet (Figure 3) [2], [3].



**Figure 3** - Dolly and her predecessor (on the left) and Carbon Cat (CC) (on the right) (adapted from [2]).

In 2008 the *1000 Genomes project* was launched, in order to gather more information on human genetic variation. 1000 anonymous people from around the globe were involved, as researchers tried to determine their DNA sequences. Over 2500 sequenced genomes were described in the scientific journal *Nature* after this project concluded [2].

### **1.1.2. Genes and the Genome**

Our genome is composed of 2 sets of 22 chromosomes, plus the sexual chromosomes, which are XX for a woman and XY for a man. In total, we have 2 sets of 23 chromosomes, excluding mutations (People who have Down syndrome, for example, carry an extra copy of the 21<sup>st</sup> chromosome). These 23 chromosomes form the Human karyotype, as can be seen in Figure 5. Our genes are located in these chromosomes, consisting of DNA and proteins. Our genetic information is encoded in a molecular structure denominated deoxyribonucleic acid (DNA) and for some species, for example bacteria, have DNA and ribonucleic acid (RNA). These nucleic acids consist of repeating units of nucleotides, which consists of a sugar, a phosphate and a nitrogenous base. These bases can differ from DNA to RNA. In DNA we have four different bases: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). In RNA we have Uracil (U) instead of T. The order of bases matters for each protein-coding set of genes. Each amino acid is composed of 3 bases, forming a codon that can be translated later. Below is a list of 20 possible amino acids (Figure 4).



	T	C	A	G			
T	TTT } phe	TCT } ser	TAT } tyr	TGT } cys	T		
	TTC } phe		TCC } ser	TAC } tyr	TGC } cys	C	
	TTA } leu		TCA } ser	TAA } stop	TGA } stop	A	
	TTG } leu		TCG } ser	TAG } stop	TGG } trp	G	
C	CTT } leu	CCT } pro	CAT } his	CGT } arg	T		
	CTC } leu		CCC } pro		CAC } his	CGC } arg	C
	CTA } leu		CCA } pro		CAA } gln	CGA } arg	A
	CTG } leu		CCG } pro		CAG } gln	CGG } arg	G
A	ATT } ile	ACT } thr	AAT } asn	AGT } ser	T		
	ATC } ile		ACC } thr	AAC } asn	AGC } ser	C	
	ATA } ile		ACA } thr	AAA } lys	AGA } arg	A	
	ATG } met		ACG } thr	AAG } lys	AGG } arg	G	
G	GTT } val	GCT } ala	GAT } asp	GGT } gly	T		
	GTC } val		GCC } ala		GAC } asp	GGC } gly	C
	GTA } val		GCA } ala		GAA } glu	GGA } gly	A
	GTG } val		GCG } ala		GAG } glu	GGG } gly	G

Figure 4 - Table of all possible 20 amino acids (Adapted from [12])

Genes are also what distinguish us phenotypically. These phenotypes, or traits, are directly inherited and also affected by environmental factors. These can be psychological or morphological, like personality or eye colour, respectively. Traits can also be behavioural, affecting the way an organism interacts with a certain environment [2], [3].



Figure 5 - The Human karyotype (Adapted from [13]).

## 1.2. Bioinformatics

Bioinformatics is an interdisciplinary field that merges two major areas: Computer Science and Biology. Since its first usage in the 1960's, pioneered by Margaret Dayhoff (Dayhoff, 1968), and its coining in the 1970's [14], it is, and has been, an ever-growing area that is crucial nowadays for development and research, since its applications are used in the most diverse areas of study. Some of these areas are Molecular Biology, Mathematics, Genetics, Computer Science, Biotechnology and Biomedicine. Bioinformatics is an area that deals with big sets of biological data. With the advances in medicine and biology of modern times, the need to process this data is becoming more and more demanding. Processing and storing this data is essential for scientific advancement, leading to the creation of various public databases for a variety of data, going from DNA and RNA databases to protein databases. Some examples are UniProt [15], Ensembl [16] and NCBI [17]. Chen and Coppola summarised some of these databases, reviewed them in their article and placed them in a table presented in their publication [18]. The Israel Science and Technology Database also provides us with a plethora of databases in the biomedical area, with a much wider range [19]. A more detailed topic on Genomic databases will be presented later.

### 1.2.1. Bioinformatics: A definition

According to Encyclopaedia Britannica, Bioinformatics is the scientific area that links two main areas: biology and information technology. Biological data is used for storing information, distribution and analysis. Biomedicine is also an area included in this junction [20].

Luscombe et al. also proposed a definition for Bioinformatics, later submitted to the Oxford English Dictionary: *“bioinformatics is conceptualising biology in terms of molecules (in the sense of Physical chemistry) and applying “informatics techniques” (derived from disciplines such as applied maths, computer science and statistics) to understand and organise the information associated with these molecules, on a large scale. In short, bioinformatics is a management information system for molecular biology and has many practical applications”* [21].

## 1.3. Brief History of Bioinformatics

Unlike people may think, and even though it is fairly recent compared to other scientific areas, Bioinformatics isn't as recent as it seems. Even though the term bioinformatics was coined after it was discovered, it started to be used before both. We can argue that with the discovery of the DNA's double helix structure by Watson and Crick in 1953 [7] was a huge breakthrough in science and an important mark for the bioinformatics scene, opening the path for future research in many areas. As mentioned before, Margaret Dayhoff was an American chemist that

pioneered computational methods in the biochemistry area. She created the *Atlas of Protein Sequence and Structure*, which was an annual journal that aimed to gather all known amino-acid sequences, therefore becoming the first molecular biology database for researchers [22], (Dayhoff, 1968). FORTRAN [23] was used as the coding language for COMPROTEIN, designed for determining protein primary structure using Edman peptide sequencing data [24], [25].

The first solution to the problem between aligning distant homological sequences was first presented by Needleman and Wunsch (Figure 6). This dynamic programming algorithm was used to solve the issue for pairwise protein sequence alignment, being coined the Needleman-Wunsch algorithm [26]. In the early 80's, the first multiple sequence alignment algorithms started to emerge, ranging from a generalization of the algorithm to the multiple sequence alignment software CLUSTAL, still used in modern times [25].

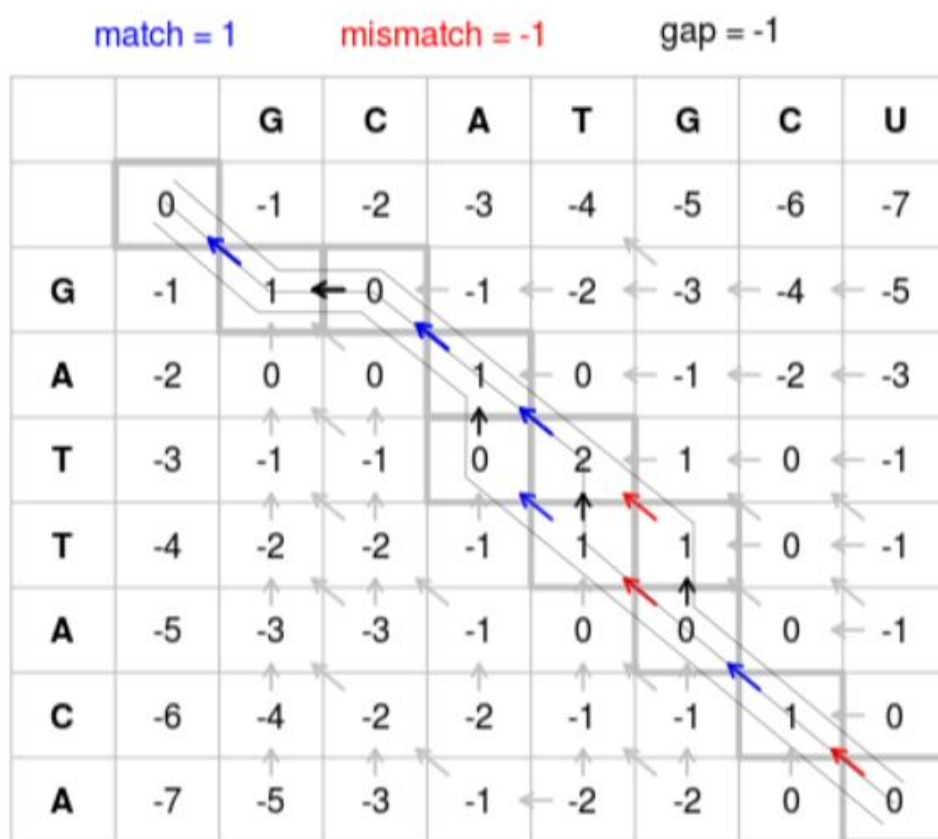


Figure 6 - Needleman-Wunsch pairwise sequence alignment (adapted from [27]).

In 1968, all 64 possible codons were discovered and this was a major breakthrough, making the attainment of DNA sequences more simple and affordable. In 1977, Sanger's team created the "plus and minus" DNA sequencing method. This method, also the first of its kind, relied on the synthesis with DNA polymerase. Some technical modifications were made, and that led to the Sanger chain termination method that is still used in modern times [28]. In 1979, Roger Staden published the first software for analysing Sanger sequencing reads. These computer

programs had several functions, such as: Searching for overlaps in Sanger gel readings; Verifying, editing and joining sequence reads into contigs, annotating and manipulating sequence files. This package included additional characters to record uncertainties in a sequence read. In 1981, Joseph Felsenstein was the first to create a Maximum Likelihood (ML) method for inferring phylogenetic trees from DNA sequences. Using nucleic acid sequences in phylogenetics added important information that we could not obtain only with amino acid sequences. This model inspired Bayesian statistics in molecular phylogeny, still used nowadays [25], [29], [30].

In 1984, the eponymous 'CGC' software was published by the University of Wisconsin Genetics Computer Group. This package consisted of 33 command-line tools designed to manipulate DNA, RNA or protein sequences, being the first software collection designed for sequence analysis [31]. In the same year, slightly after the creation of 'CGC', DNASTAR was developed. It could be run on a CP/M personal computer [25].

Richard Stallman's GNU manifesto was published in 1985, which promoted the creation of an open-source software. This software was at the core of several initiatives in the bioinformatics area, such as the European Molecular Biology Open Software Suite later developed in 1996. During the same period of time, the European Molecular Biology Laboratory (EMBL), GenBank and DNA Data Bank of Japan (DDBJ) united, to standardize data formatting, facilitating data sharing between these databases. In 1985, the journal *Computer Applications in the Biosciences* (CABIOS) was created. Nowadays, this journal goes by the name of *Bioinformatics*. Several dedicated journals started to emerge. Small-scale mainframe computers were used for large datasets instead of microcomputers [25].

Created in 1987 by Larry Wall, Perl (Practical Extraction and Reporting Language) is a high-level scripting language, to facilitate parsing and text data in GNU. Until the late 2000s, Perl was the main language for bioinformatics, due to its flexibility. In 1989, Python was created, by Guido Van Rossum. It was designed to be more simplistic and easier to understand and interpret, but it was not until the late 2000s that Python became a major programming language [25].

With the creation of the world wide web in the early 1990s, many bioinformatics resources became available throughout the world. The EMBL Nucleotide Sequence Data Library, the first nucleotide sequence database, was made publicly available in 1993. Just a year before, the GenBank database was under the responsibility of NCBI. BLAST, the pairwise alignment tool, was made available in 1994. In upcoming years, several well known databases such as Genomes (1995), PubMed (1997) and Human Genome (1999) were created [25].

Bioinformatics projects really grew in number, and there was a need to adapt to the change and gather all the data discovered. Several government-sponsored organisations appeared, such

as: Compute Canada; New York State's High Performance Computing Program; The European Technology Platform for High Performance Computing and China's National Center for High-Performance Computing [25].

## **1.4. Genomic databases**

As genome sequencing and discovery was becoming a more and more common theme as the years went by, several genomic databases started to emerge.

The Israel science and technology directory presents us with a list of available databases in varied fields, as mentioned before, that range from Biomedical databases to virus databases. Inside this list of available databases, genome databases are divided into genomic prokaryotic and plant genomes, separately, making it easier for a day-to-day user to browse. The "Genome Databases" tab, where a list of all available databases for all types of genomes can be found, has some recognizable databases such as the *NCBI Genome Database* and the *Ensemble Genome Browser*. Very specific genomes such as the silkworm and the Zebrafish genome databases can also be found (*Israel Science and Technology Directory, 2019*).

Chen and Coppola also present us with a series of databases, some included in the Israel Science and Technology directory. These are all summarised in a table, that range from DNA to RNA and Protein databases, as mentioned before in chapter. They mention that the first sequence of the human genome project, published in 2001, was created from eight bacterial artificial chromosome libraries and 16 additional ones from five donors [18]. This information on its own could not suffice to describe genetic variations across and within several populations. Allele frequencies of rare variants could only be found in large populations, something that was impossible in a small population, like the one presented. Therefore, samples from various individuals, from different populations, were used, from a project by the name of HapMap project [32]. This project consisted of determining common patterns of DNA sequence variations and make them publicly available. An international consortium developed a map of these common patterns according to the genotype of one million sequence variants from different ancestries. This would allow a much faster diagnosis of diseases, facilitate the development of diagnostic tools and enhance target choice for therapeutic intervention [18], [32].

## 1.5. Gene prediction tools: Types and examples

Gene prediction tools have been around for some years now. Since GeneMark [33], GeneMark.hmm [34], Augustus [35], GENSCAN [36] and Fgenesh [37] to CESAR 2.0 [38], BRAKER1 [39], CodingQuarry [40], Seqping [41] and TaF [42]. Some of these are presented below, in table 1. All these pipelines/programs have their differences, ranging from the type of cells they focus on: Eukaryotic or Prokaryotic. Each program can also be used for different kingdoms in the taxonomic ranks. Some are used for the fungi kingdom, such as TaF, others for the plantae kingdom, such as Seqping, others for the animalia kingdom, such as Fgenesh and others for the bacterial kingdom, such as GeneMark.hmm. This doesn't mean that each tool is restricted to a certain kingdom or cell. AUGUSTUS allows gene prediction with eukaryotic cells in the Plantae and Animalia kingdom, being *Arabidopsis thaliana* and *Drosophila melanogaster* some examples, respectively [43].

Gene prediction programs can be divided into two big groups: *Ab initio* (Intrinsic) and evidence/similarity-based (Extrinsic) [35], [44], [45]. *Ab initio* programs use solely the target sequence for gene prediction, with a known gene structure as the training set for training the parameters of the models, dealing with statistical approaches to conduct coding and non-coding region searches and find gene signals. *Ab initio* methods are quite sensitive, and their accuracy depends on the set they are given as the training set. They are faster than evidence-based methods but tend to inaccurately predict protein coding regions, misplacing start and stop codons. With no proper training set, they do not perform well [44], [45].

Evidence-based methods, on the other hand, are used to find genes based on homology, using similarity search procedures. They compare a certain sequence of interest to already available data or use external data to help the prediction, such as EST libraries or RNA-Seq information. However, evidence-based methods only predict if there is evidence of a transcription. They tend to miss genes that *ab initio* methods can encounter based on their statistical models [45], [46].

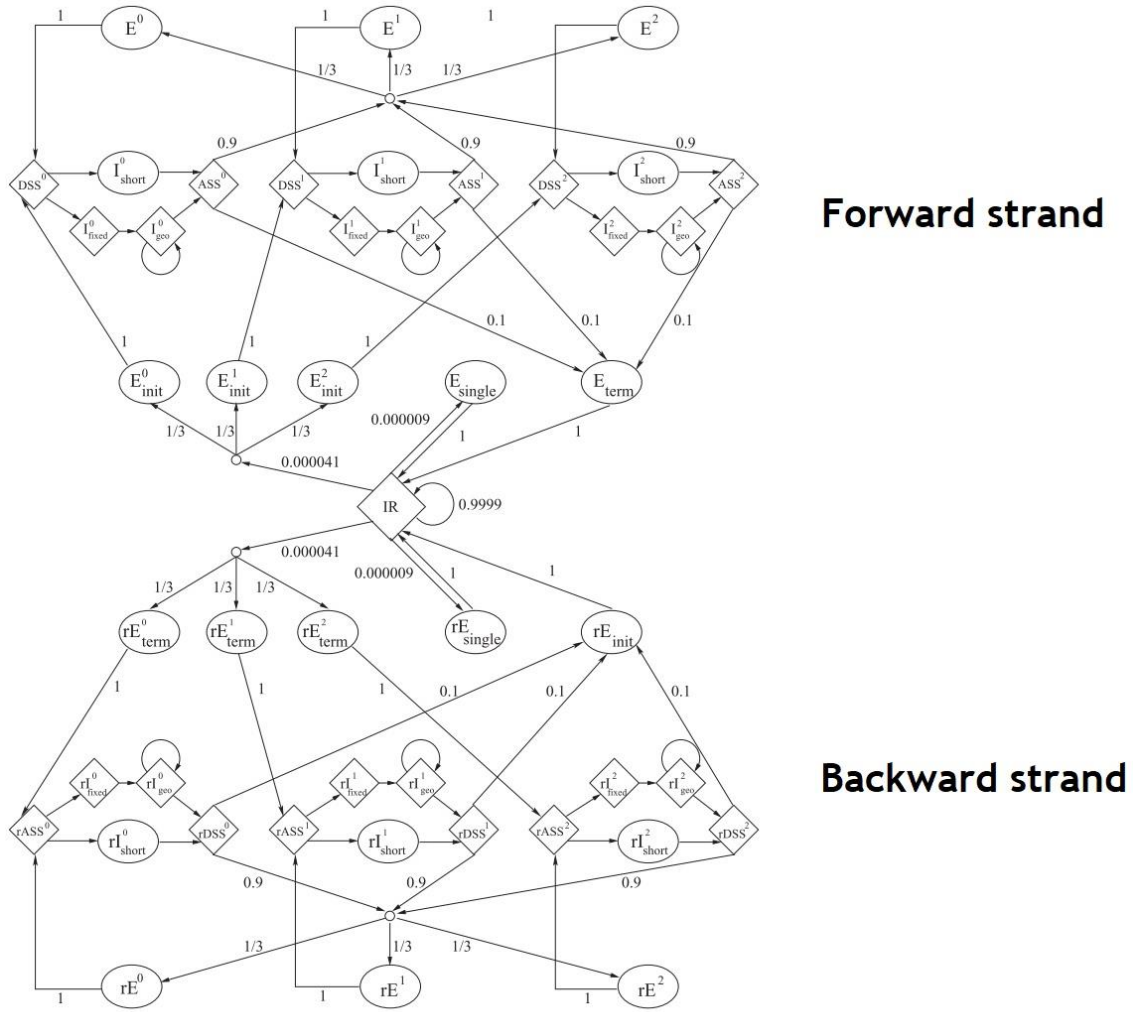
Table 1 - Gene prediction tools

Gene prediction tool	Implementation Language(s)	Model/Algorithm
AUGUSTUS	N/A	Generalized Hidden Markov Model (GHMM)
TaF	Python and Apache, PHP and MySQL (APM) for web server	AUGUSTUS
BRAKER1	Perl	GeneMart-ET and AUGUSTUS
Seqping	Bash and Perl	BLAST+, CD-HIT, Splign, GlimmerHMM, AUGUSTUS, SNAP, MAKER and EMBOSS
GENSCAN	N/A	General Probabilistic Model
Fgenesh	N/A	Hidden Markov Model
GeneMark.hmm	N/A	Hidden Markov Model
inGAP-CGD	C++	Supervised Support Vector Machine (SVM) with a codon-based de Bruijn graph

Before moving on to the next chapter in this introduction, some gene prediction tools that have been mentioned before will be briefly analysed. This analysis won't be too extensive or detailed, and will consist on mainly explaining how each pipeline or gene prediction program works and how they are structured.

### 1.5.1. AUGUSTUS

AUGUSTUS is one of the main gene prediction tools used in modern times, to either predict genes or be used as a model for other gene prediction tools [39], [41], [42]. At the time of publication, it was based on a new HMM, implemented in the actual program AUGUSTUS. It finds the optimal parse of a genomic sequence. The program uses a more accurate method of prediction for modelling intron lengths. Short introns are modelled via the length distribution, due to them clustering around a certain length. Long introns are modelled via a geometric distribution. The Viterbi algorithm is used for training [35], [47].



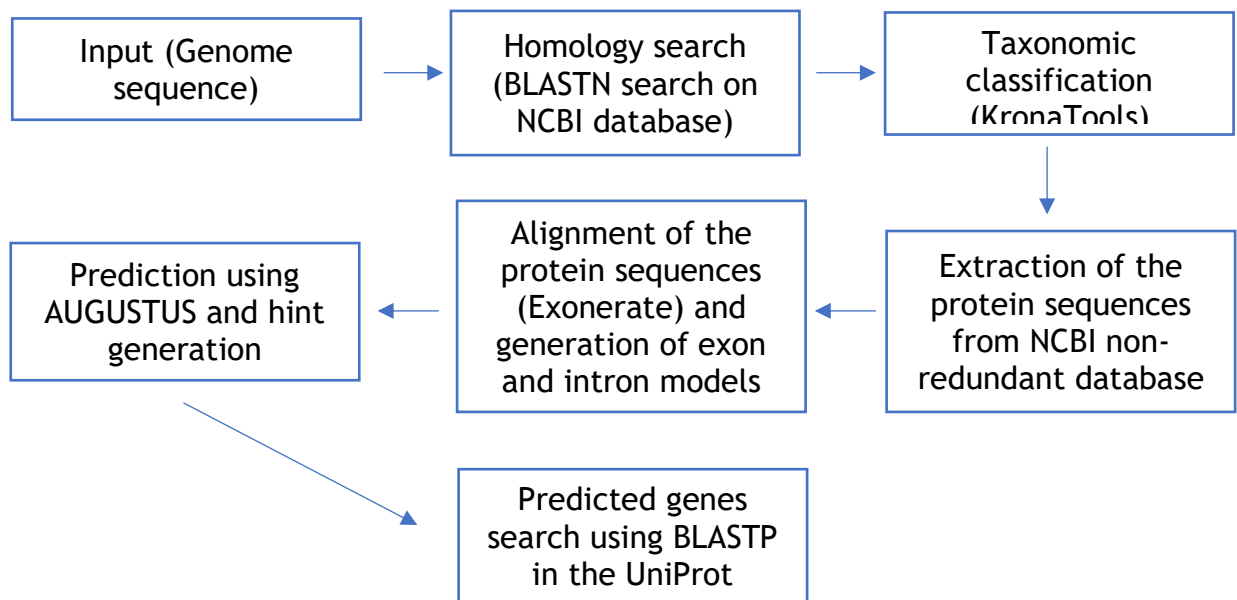
**Figure 7** - AUGUSTUS states and respective transition possibilities. The r stands for reverse, so there can be a distinction between the forward and the reverse strand modelling.  $E_{single}$ : single exon gene.  $E_{init}$ : First coding exon of a multi exon gene.  $DSS$ : donor (5') splice site.  $I_{short}$ : intron at most  $d$  nucleotides long.  $I_{fixed}$ : first  $d$  nucleotides of a longer intron.  $I_{geo}$ : individual nucleotides after the first  $d$  nucleotides of a longer intron.  $ASS$ : acceptor (3') splice site, with branch point.  $E$ : internal exon.  $E_{term}$ : last exon of a multi exon gene.  $IR$ : 'intergenic region'. States which emit fixed length strings and explicit length distribution are represented by diamonds and ovals, respectively (Adapted from [35]).



### 1.5.2. TaF

TaF is an *ab initio*, homology-based gene prediction pipeline (see Figure 7 for further details). It's a very recent pipeline and web prediction server for filamentous fungal gene prediction. Its predictions accrue from homology (Shared ancestry between a pair of genes, in different taxa) and taxonomy (Description, identification, nomenclature and classification of organisms), based on queries for close relatives. It's fundamentally divided into 4 steps:

1. Searching for taxonomy based on close relatives;
2. Gathering information on exon-intron boundaries from protein sequence data of orthologs;
3. Gene prediction via *ab initio* and evidence-based methods;
4. Gene models homology search.



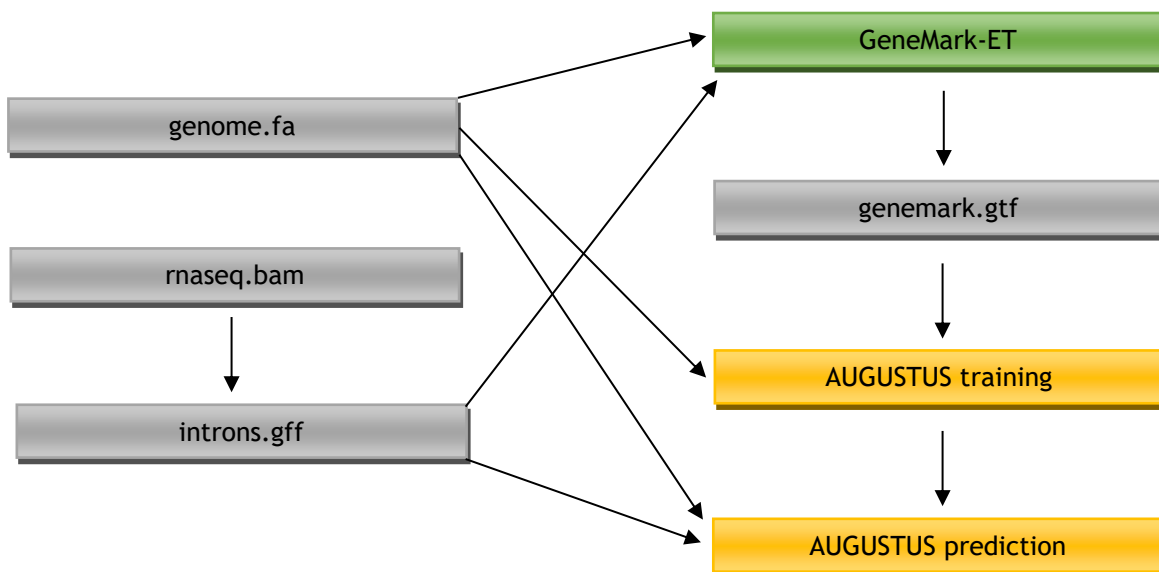
**Figure 8** - Representative scheme of TaF's pipeline.

TaF was created to fill a gap in the fungal genome area, due to the lack of experimental verification in fungal species. Most of the recently sequenced fungal genomes have been put together via Next-generation Sequencing (NGS), mostly via Single-Molecule Real-Time (SMRT) sequencing. These manage to improve assembly errors, but lack experimental verification, as mentioned before. Even though there are existing fungal prediction and annotation tools such as SnowyOwl [48], ABFGP [49] and OrthoFiller [50], TaF is based on a new approach, being useful for the prediction of gene models from fungal draft genomes [42].

### 1.5.3. BRAKER1

BRAKER1 is a pipeline that results of the combination of two gene prediction tools: GeneMark-ET [51] and AUGUSTUS [35]. Its pipeline is divided essentially into two parts, due to the two different gene prediction tools used. GeneMark-ET takes RNA-Seq data and generates *ab initio* predictions through unsupervised training. These predictions are then used as input to AUGUSTUS as training data, which also incorporates information from previously mapped unassembled RNA-seq reads [39].

BRAKER1 is coded in Perl. It requires an RNA-seq file alignment file, where the spliced alignment information is extracted and stored, and an analogous genome file. Firstly, GeneMark-ET uses the input file to produce a set of *ab initio* predictions. Those that have introns corresponding to the RNA-Seq alignments are selected for training in AUGUSTUS [39].



**Figure 9** - Scheme representing the BRAKER1 pipeline. Grey boxes represent files, green boxes represent GeneMark-ET and orange represent AUGUSTUS (Adapted from [39]).

## 1.6. Hidden Markov Models

Hidden Markov Models have been used for several years, especially in the speech recognition field. These are used to represent the sound of phone-like units and words. It's commonly modelled to the Maximum Likelihood Estimation (MLE) criterion, which is based on finding the most probable sequence of events, maximizing the probability of the training samples. This is based on the Expectation-Maximization (EM) algorithm, a variant of the Baum-Welch algorithm, presented later in this sub-chapter, relying on maximizing the log-likelihood of incomplete data and maximizing the log-likelihood of complete data [52], [53].

Hidden Markov Models (HMMs) are the most commonly used algorithms in gene prediction tools. Having said that, and coming as something quite obvious, it's also the algorithm used in this project. HMM's for biological data analysis can be subdivided into various types, such as Profile-HMM, Pair Hidden Markov Models (Pair-HMM), Context-Sensitive Hidden Markov Models (csHMM) and Profile-csHMMs [54]. In this project, a Profile-HMM was used as the model, for several reasons presented later in a sub-chapter embedded in this chapter.

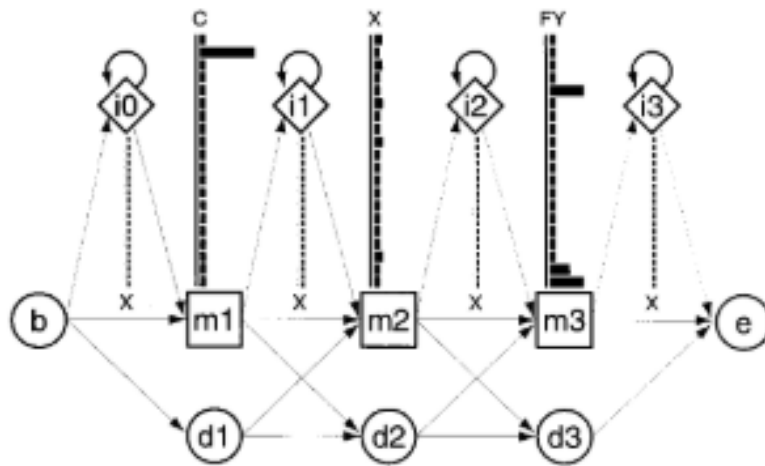
An HMM is composed of two stochastic processes: One that is visible, serving as input, and another hidden stochastic process in a layer that's invisible and can only be observed via the output, which is a sequence of symbols. The events observed, from the layer that is visible, are called "Symbols" and the underlying, invisible factors, are called "States". The probability distribution of the symbol is dependant of the underlying state [54]-[56].

There are several training algorithms for an HMM. Two of the most common are the Viterbi [57] and Baum-Welch [58]. The Viterbi algorithm is an algorithm used to obtain the most likely state sequence inside the HMM training sequence. Also called forced alignment, it's based on maximizing the likelihood of a state, given an observational sequence. It uses the most likely state sequence to estimate the parameters for the HMM, so each individual observation only contributes to the most likely state at a given time. However, the Viterbi algorithm requires some reasonable initialization and limits the data given by the training set, since only observations inside a certain HMM state are used for the parameters of that specific state, giving us a less robust model [53].

Baum-Welch algorithms are based on two computational functions: the forward and backward probabilities. These probabilities allow the model to weight each contribution of each observation, once computed. This allows for a more complex and robust model, but also takes longer to process [53].

### 1.6.1. Profile-HMM

A profile-HMM is a type of HMM that bases itself on multiple sequence alignment. There are three states present in a profile-HMM: A “Match” state, which represents columns with matching elements; An “Insert” state that represents a possible insertion of a base, in this case, between one column and another and a “Delete” state, that represents a deletion. Profile-HMMs are linear, going from left to right, analysing the sequences and comparing them. The result is then returned as a sum of the probability parameters, a Logp value. These are comparable to how BLAST and FASTA scoring work: taking the probability of the match state  $x$  emitting a residue as  $p_x$ , and the background frequency as  $f_x$ , the score for the residue at the match state in question is  $\log(p_x/f_x)$ (Figure 10) [56].



**Figure 10** - An example of a small Profile-HMM, with three columns. Five different sequences were aligned. There are three match states (m1, m2 and m3), four possible insert states (i0, i1, i2, i3) and three possible delete states (d1, d2, d3). For each insert and delete state, there are 20 possible residue emission states. Delete states are also considered mute states, where there are no emission probabilities. There is also a beginning and an end state (b, e). As observed, the letter “C” was the most probable emission state in the first match state, due to the length of the black bar compared to the rest of the bars. At match state 2, the emission state for several residues were the same, meaning that the emission state was the same for those letters. In match state 3 the most probable letters to be emitted were “F” and “Y” (Adapted from [56]).

## 1.7. Aims

The main aim of this dissertation work is to create a new gene prediction model, that has the ability to be faster and more flexible.

The specific aims for this dissertation work include:

- Implementation of an ORF finder to find shorter, more specific sequences;
- A Profile-HMM that aims to perform a multiple sequence alignment, reducing the probability of there being mistakes in the sequences;
- Creating several scenarios for finding the ideal match, insert and delete state values for the Profile-HMM;
- The target input sequence that is know having a better Logp value that a random unknown sequence the user inputs;
- Reducing training time by using the Viterbi algorithm to train the model, since it's faster and there is no need for a forward-backward algorithm, such as Baum-Welch;
- Plotting the Logp values that are returned after the Profile-HMM is run;
- Proving that some values, when input in HMM, don't return a realistic Logp value or any at all, and therefore can't be used to create a model.

## Chapter II



## 2. Materials and Methods

### 2.1. Implementation

The programming language used for implementation was Python, a high-level programming language created in the late 1980's by Guido van Rossum. Named after the British Comedy group Monty Python [59], it was based on a pre-existing programming language, ABC, that was designed to be taught to computer users with some knowledge, but were not computer programmers or software developers (Rossum, 2003). But why choose Python to implement the code? To answer the question very briefly, Python presents us with a variety of nifty features:

- It's easy to learn and interpret, making it a great language for people with less experience in the programming area, just like its predecessor ABC;
- It's syntax, dynamic typing and interpreted nature make it great for scripting and rapid development, without any major run-arounds, in a plethora of areas in an extensive number of platforms;
- Modules that are installable and callable via imports, facilitating programming and cross-language extensibility;
- Cross platform compatibility with other languages, usable as an extension for new functions and data types in C or C++;
- No need to compile the code like in C or C++;
- Amongst other functionalities.

In short, Python, just like its predecessor ABC, is a programming language ideal for computer users outside the programming area, like scientists. Its simplistic code writing and one-liners make coding easier. In Python we are able to condense several lines of code into one or just two, simplifying the whole "Coding" process by quite a fair margin [59], [60].

### 2.2. The program and code

The program that created is a mix of various sources, including original code, therefore being very diverse. With the help of some pre-existing code and information, and the junction of both, below is presented what is thought to be the start of an experimental code for gene prediction or gene finding in specific target sequences. It still is a gene prediction tool, but not as complex as others present in modern times. With the help of some Python modules, the code was condensed and simplified, making it easier to interpret and to read. A table of the modules/packages that were used are presented below (Table 2). Most modules are part of the main packages used in this project, meaning that their purpose is more of a "support" module for a specific package than an actual main module used in the project. This code is purely

experimental and an attempt to create something new without having the need to create something entirely from scratch. Due to Python's versatility, this was possible. With the help of documentation available for each package, the best parsers and workarounds were found and used to facilitate the merging of the code, making it work and giving an output that was what was expected.

The program isn't very extensive, most of it being occupied by the ORF finder [61], that suffered a few modifications. Visual Studio was used to implement the code, on an 8 GB RAM Laptop, with an AMD Ryzen 7 Processor. The program runs relatively fast and there is no need for a long waiting time.

**Table 2** - Packages/Modules used and their respective functionality.

Package/Modules	Version (As of May 2019)	Description
attrs	19.1.0	Helps writing concise and correct software, without slowing down code writing
backcall	0.1.0	Specifies for call-back functions passed int to an API
biopython	1.73	Python tools for computational molecular biology
bleach	3.1.0	HTML sanitizing library that escapes or strips mark-up attributes
colorama	0.4.1	Makes ANSI escape character sequences work for MS Windows
cycler	0.10.0	Creates "base" cycler objects and a class with composition and iteration logic
decorator	4.4.0	Facilitate the definition of signature preserving function decorators and decorator factories
defusedxml	0.6.0	Python-only workarounds and fixes of service and other vulnerabilities in Python's XML libraries
entrypoints	0.3	Helps packages advertise objects with some common interface
graphviz	0.10.1	Facilitates the creation and rendering of graph descriptions in the DOT language of Graphviz



ipykernel	5.1.1	IPython kernel for Jupyter
ipython	7.5.0	Helps use Python interactively
ipywidgets	7.4.2	HTML widgets for Jupyter notebooks and the IPython kernel
jedi	0.13.3	Static analysis tool for using in IDEs/editors
jinja2	2.10.1	Template engine written in pure Python
joblib	0.13.2	Lightweight pipelining in Python
jsonschema	3.0.1	Implementation of JSON Schema
jupyter-client	5.2.4	Jupyter protocol implementation and client libraries
jupyter-console	6.0.0	IPython-like terminal for Jupyter kernels
jupyter-core	4.4.0	Jupyter core package
jupyter	1.0.0	Jupyter metapackage
kiwisolver	1.1.0	C++ implementation of the Cassowary constraint solving algorithm
MarkupSafe	1.1.1	Safely add untrusted strings to HTML/XML markup
matplotlib	3.1.0	Python plotting package
mistune	0.8.4	Markdown parser in pure Python
nbconvert	5.5.0	Converts Jupyter Notebooks
nbformat	4.4.0	Jupyter Notebook Format
networkx	2.3	Creating and manipulating graphs and networks
notebook	5.7.8	Web-based notebook environment for interactive computing
numpy	1.16.3	Fundamental package for array computing with Python
pandocfilters	1.4.2	Writing of pandoc filters in Python
parso	0.4.0	Python parser
pickleshare	0.7.5	Small database with concurrency support
pip	19.1.1	Tool for installing Python packages
pomegranate	0.11.0	Graphical models library for Python
prometheus-client	0.6.0	Python client for the Prometheus monitoring system

prompt-toolkit	2.0.9	Building interactive command lines in Python
Pygments	2.4.0	Syntax highlighting package written in Python
pyparsing	2.4.0	Python parsing module
pyrsistent	0.15.2	Persistent/Functional/Immutable data structures
python-dateutil	2.8.0	Extensions for the Python datetime module
pywinpty	0.5.5	Python bindings for the winpty library
PyYAML	5.1	YAML parser and emitter for Python
pyzmq	18.0.1	Python bindings for 0MQ
qtconsole	4.4.4	Jupyter Qt console
scipy	1.3.0	SciPy: Scientific Library for Python
Send2Trash	1.5.0	Send file to trash natively
setuptools	41.0.1	Download, build, install, upgrade and uninstall Python packages
six	1.12.0	Compatibility library
terminado	0.8.2	Terminals served to xterm.js using Tornado websockets
testpath	0.4.2	Test utilities for code working with files and commands
tornado	6.0.2	Python web framework and asynchronous networking library
traitlets	4.3.2	Traitlets Python config system
wcwidth	0.1.7	Measures number of Terminal column cells of wide-character codes
webencodings	0.5.1	Character encoding aliases for legacy web content
widgetsnbextension	3.4.2	IPython HTML widgets for Jupyter

Most of the modules are support modules for Jupyter, Pomegranate or Biopython, the most used packages in this work. From all the modules present, the two that were used and are more visible, visually, throughout the project, are Pomegranate and Biopython, a package for modelling probabilistic data and another for processing data on a molecular level, respectively. After explaining what these packages do and why they are so important, it's crucial to explain the main part of the code: The Profile-HMM. It's the base of this project. Without it the project would have almost no meaning and reaching the final goal wouldn't be possible.

### **2.2.1. Pomegranate**

Pomegranate is a Python package, created by Jacob Schreiber from the University of Washington, aimed at implementing probabilistic modelling in a fast and flexible manner, ranging from Bayesian Networks (BN) to HMM. Based on its predecessor Yet Another Hidden Markov Model (YAHMM), it was created to fill a gap in the Python ecosystem, using maximum likelihood estimates to update the parameters in probabilistic machine learning models. Even though there are existing packages that implement certain probabilistic models individually, for example hmmlearn for HMMs, libpgm for BNs and scikit-learn for Gaussian Mixture Models (GMM), pomegranate implements several more probabilistic models in a modular type of programming. This has two main effects:

- Adding a new distribution makes it so that all models are built using that distribution;
- When one improvement is made to one area of pomegranate, it's automatically improved to all models that could benefit from that improvement.

Pomegranate's design allows it to be easy to use and efficient without sacrificing computational efficiency. A model can be written from scratch with each component if previous data is present, or learnt directly from the data that is used as input, if previous data isn't present [62].

Until this date, and of the data available to our knowledge, no article has been published using this package. Due to its speed because of its implementation in Cython, pomegranate is definitively a package that makes this stand out from other HMM packages or even traditional HMMs [62].

### **2.2.2. Biopython**

Biopython is a Python package and provides tools for computational molecular biology. It has a lot of functionalities, some of which were used in this project. The main goal is to provide tools for Python related to Bioinformatics, allowing the user to create high-quality, reusable modules and classes. Biopython has a variety of parsers for several Bioinformatics file formats, such as BLAST, Clustalw, GenBank, UniGene, FASTA, etc. The last format, FASTA, is what's used as input for the program. This parser was largely used throughout this dissertation work, being one of the crucial parts of the same, allowing fasta-formatted files to be read and the data used in the Profile-HMM [63].

The Biopython package offers several other functionalities, including:

- Bioinformatics file parsing into Python data structures, supporting formats such as those mentioned before;
- The files can be iterated over record by record or indexed via a Dictionary interface;
- Code to interpret online bioinformatics websites such as NCBI and ExPASy;
- Interfaces for Bioinformatics programs like Blast from NCBI, Clustalw alignment program and EMBOSS command line tools;
- Sequence class that deals with sequence, their id's and features;
- Amongst others [63].

### **2.3. The ORF Finder**

This part is very important to the project. It facilitates the HMMs work and is what allows distinction between sequences. But first, what is an ORF? An ORF is a possible protein-coding sequence, located between a start and a stop codon. It contains a continuous set of codons, where each codon specifies one amino acid and is the simplest way to find a certain DNA sequence that might encode for a protein or various proteins. In prokaryotes, the DNA sequences are transcribed into mRNA and then translated into proteins, with no major modifications. In eukaryotes the process is slightly different. Transcription of protein-encoding regions is followed by removal of introns from the mRNA via splicing, leaving us with only the protein-encoding regions, the exons. Once this process is over, and other necessary modifications have been made, the resulting mRNA sequence can be translated [64].

The ORF finder used is publicly available. It was created to identify ORFs in a study regarding wound healing and regeneration in cnidarian *Calliactis polypus* [61].

The ORF finder is very easy to use. Each step has instructions to aid the user. These were slightly modified in this dissertation work, to fit the main purpose and find smaller sequences, since protein-coding gene sequences are already being used to test. It has 8 main input steps, these being:

1. Specifying the name of the fasta file from where to extract the ORFs;
2. Specifying the name of the output file where these same ORFs will be stored;
3. The minimum length of any ORF to search for;
4. The maximum length of any ORF to search for (for bigger ORFs, we can input a larger number as to search for more);
5. The number of ORFs the user wants to extract from each nucleotide sequence;

6. Determining the weighting of non-coding start sequences for the ORFs;
7. What ORF formatted file types should be saved (protein translated, nucleotide CDS or both);
8. Ability to replace alternative start codons with M (only if the protein translated file is chosen).

## **2.4. The Profile-HMM**

The HMM model, from the pomegranate package, was used as our base model, more specifically the Profile-HMM model. And why? Due to its ability to compare various sequences at the same time, in other words, perform a multiple sequence alignment. With this program, several sequences can be compared at the same time with the target sequence that the user inputs. This gives us a more specific and accurate answer, narrowing down the margin for error, due to its parallel sequence comparison. In the end, the program is told to print out the lowest Logp value and all Logp values, each one associated to its respective ORF.

The user inputs the full directory of the ORF file that was previously saved using the ORF finder. With the help of Biopython, the data was parsed. Each ORF is compared separately with the target sequence, input by the user. Since a target file may contain several sequences, for every new comparison, the user has to input the file containing the ORFs.

Testing the model to obtain the best possible results was needed. As a method to obtain these same results, various scenarios were created, based on various possibilities. Some were more unrealistic than others, but confirmation that they didn't work was needed. In each scenario, at least one value was modified. The most important values to find, in other words ideal values, were of the match states. And why? Because the match states are what give us the confirmation if there is really a match between an ORF and the target sequence.

In each scenario, there are match states, insert states and delete states. The match states were the first to be changed. To see which match state values were ideal, various scenarios were created. Once the ideal match state was found, insert states came next. These insert states had to be managed carefully, as giving an insert state too much probability of appearing could lead to false positives, something not wanted. The last types of states to be modified were the delete states. Just like the insert states, not giving too much probability to these states was crucial, as it could give us false positives. Unrealistic scenarios such as not using any insert state or any delete states were excluded, primarily because it made no sense as it would defeat the purpose of creating a Profile-HMM in the first place and also due to the unrealistic results these would produce.

To train the model, the Viterbi algorithm was used, due to its speed and the fact that the sequences used were not that big and needed not a complex algorithm for analysis. In the next section, the most relevant scenarios will be presented.

## Chapter III

### 3. Results and Discussion

The code was run countless times, testing for the ideal amount of ORFs to compare and the ideal Profile-HMM scenario. For the ORF finder, the same settings were used for all the input files. Based on the steps previously presented before in the ORF finder sub-chapter, the ideal values and settings used for the final Profile-HMM were (in order):

1. Choosing the name of the fasta-formatted file to extract the ORFs from. The file was always in the same directory of the python file for the project, facilitating extraction;
2. Giving a name to the file to store the ORFs in. For testing purposes, naming the files was always done randomly;
3. Entering the minimum amino acid length to accept as a valid ORF. Since the program searches for codons, if the input were 100 base pairs for example, it would search 300 amino acids. 50 was used, as to find smaller and possible ORFs;
4. Entering the maximum length for the ORFs. No maximum was set, as to search for longer sequences and as many as it possible could;
5. Entering the number of ORFs to extract. The recommended is normally 1-3. But since the files are already protein-coding sequences, searching for smaller, more specific ORFs made more sense, to make it more accurate. 50 was chosen as the maximum number of ORFs to extract;
6. Choosing if the default settings of the code were to be used or not. The default settings were used;
7. Choosing if only a nucleotide ORF file was to be produced or if a translated protein file was to be produced too. Both were produced for all cases;
8. Choosing to replace the alternative start positions with an M for Methionine. These were replaced;
9. Wanting to discover ORFs with unresolved regions. Since this wasn't pertinent to the rest of the program, no unresolved region was discovered;
10. Appending the ORF numbers to the transcript ID's. These were appended.

The ORF finder also has the functionality to search for alternative start codons. Some ORFs that were retrieved don't start with the typical start codon, ATG. So, when comparing sequences, some start with an alternative start codon.

Finding the best Profile-HMM wasn't an easy task. Through trial and error, reach an ideal model was reached, with specific values for each state (Initial, match, insert and delete). The code was tested various times with different insert states, delete state and initial state values. Even if the values were different, every single attempt had at least one initial, match, insert and

delete state. What varied were the values and the number of states, depending on which attempt was currently being tested. These attempts can be divided into various scenarios, that differ either a lot or slightly from each other, which will be presented next. In each scenario presented, the values for each state are explained, the reason why those specific values were used and, for the scenarios not used for the project, the reason why they weren't feasible. Before explaining each scenario, a decision tree will be presented. This will serve as a guide, for better understanding of each scenario.

In each scenario, the Logp values are plotted and the best value is printed as a positive absolute value. This allows us to compare with scenarios. For example, if we wanted to discover if a sequence is more probable to be the one compared to another or other sequences, we simply use the program. It gives us the ORFs of the input sequence, storing them in a certain directory. Then they are used with the second part of the script, the Profile-HMM, which will run through all the sequences and gather the best Logp value. The process is repeated with the different scenarios to analyse and also gather their Logp value. The Logp value that is the lowest, or closest to zero, represents the scenario with the ORF that has the highest probability of being found in the target sequence. Only one input sequence can be analysed at a time. To obtain results regarding other sequences for analysis, re-running the program is necessary.

The final results are presented as a comparison between the Logp values of the ORFs, compared from different input files with the same target exit sequence. Sequences from the ORF files of the same size are used, taken from the same region of the Profile-HMM prediction. This allows for a more accurate analysis of the veracity of the program.

The test file used for the project was a fasta-formatted file containing protein-coding genes for JAK2 Janus kinase 2 [Homo sapiens (human)]. This specific gene, a protein tyrosine kinase involved in a specific subset of cytokine receptor signalling pathways, was chosen at random. Upon a search for the word *Haemoglobin* in the Human Genome database in NCBI's database [65].



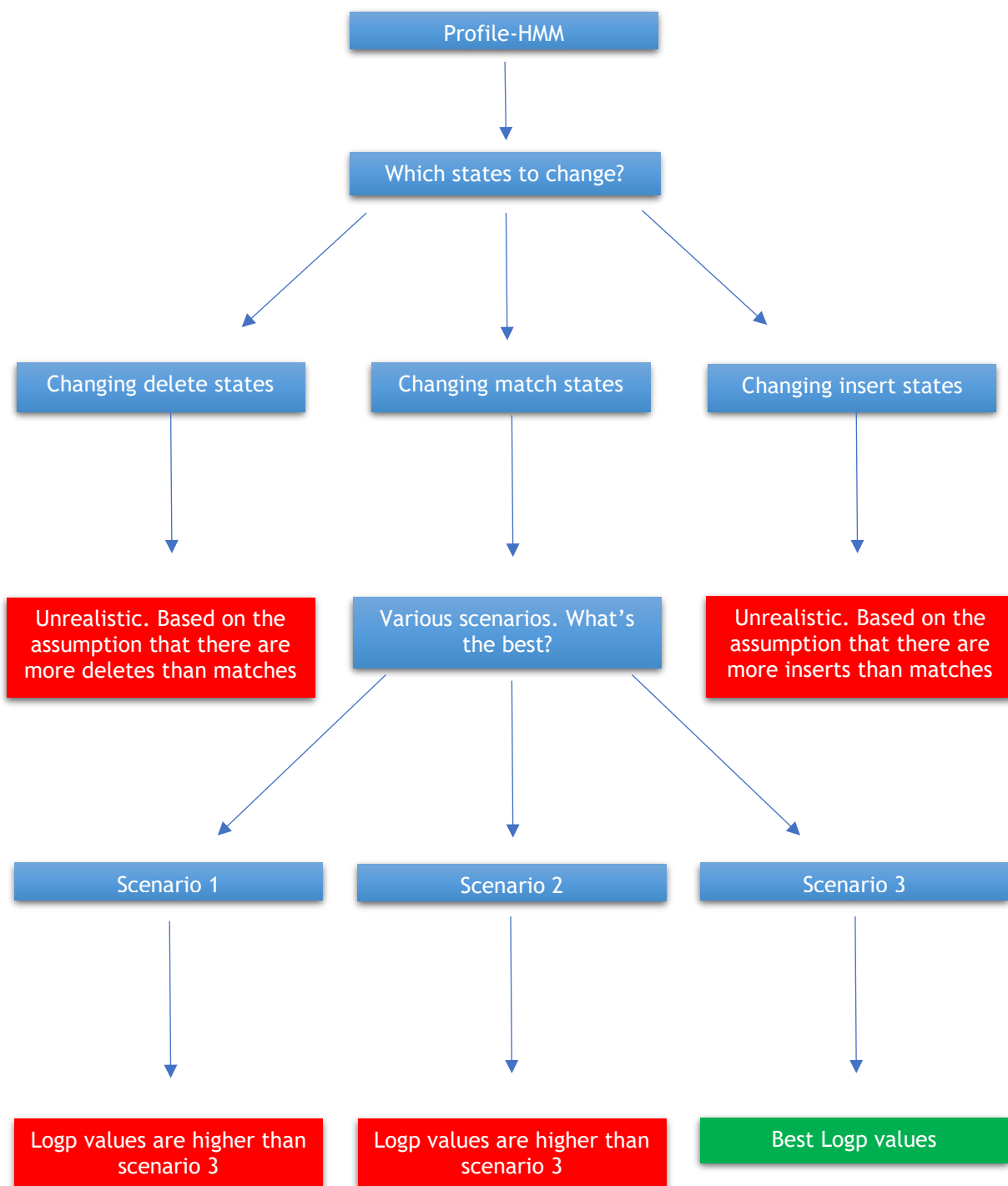


Figure 11 - Decision tree demonstrating possible scenarios for each state.

### 3.1. Scenario 1

The first scenario that's presented is based on the hypothesis that the probability of the match state is higher than the probability of there being an insert or delete state. We define four match states, in which each represents a 100% probability of encountering a specific base. When setting the distribution parameters, the assumption was made on the basis that there was no prior information, to make the program as unbiased as possible so that it could fully give us feedback on any sequence, independent of where it came from. An initial state was created, also known as an initial distribution ( $i\_d$ ), where initial probabilities were defined for each base. These were calculated from the input sequence that the user wants to use as comparison with the ORFs previously stored. It is pertinent to mention that the percentages of each base were converted to integers then multiplied by 0.001, to give us a number between zero and one, due to the fact that each overall probability adding up to one (1) in each distribution. We created four match states, each with different probabilities. These were put into four different discrete distributions, where each one had a one hundred percent chance of encountering one of the four DNA bases. For example, Discrete Distribution 1 ( $d1$ ) gave the probability of encountering the base A, a one hundred percent (1 on a scale from 0 to 1), and the other bases a zero percent chance of being found in that match state. Discrete Distribution 2 ( $d2$ ) was given a one hundred percent chance of finding a C, and so on. What changed were the transition states. Each base appears in a different amount when it comes to DNA. Chargaff's rule was applied, where the amount of the base A is the same as the amount of T, and the amount of the base G is the same as the amount of C [3], so  $A = 0.30$ ,  $C = 0.20$ ,  $G = 0.20$  and  $T = 0.30$ . Since the bases only represent 0.8 of the total amount of the distribution, converting the probabilities to fit the model had to be done. To better show what values were used, below are several with the respective values and variables: match state transition values, insert state transition values and delete state transition values. It's pertinent to mention that the insert state transition values and delete state transition values were tested and are ideal for this and several other scenarios. In other scenarios, as will be presented ahead, these were changed for testing purposes. In the table below,  $a\_int$ ,  $c\_int$ ,  $g\_int$  and  $t\_int$  represent the integral values for each base, a, c, g and t, calculated from the target sequence, respectively. Below, three different tables with each transition values will be presented (Tables 3 to 6).

**Table 3** - Model start transition values (Scenario 1)

Transition State	Value probability (0-1)
Model start - Match state 1 (m1)	$a\_int * 0.8$
Model start - Match state 2 (m2)	$c\_int * 0.8$
Model start - Match state 3 (m3)	$g\_int * 0.8$
Model start - Match state 4 (m4)	$t\_int * 0.8$
Model start - Insert state 1 (i1)	0.20

**Table 4** - Match state transition values (Scenario 1)

Transition State	Value probability (0-1)
m1 - m1	$a\_int * 0.8$
m1 - m2	$c\_int * 0.8$
m1 - m3	$g\_int * 0.8$
m1 - m4	$t\_int * 0.8$
m1 - Insert state 2 (i2)	0.10
m1 - Delete state 1 (d1)	0.10
m2 - m1	$a\_int * 0.8$
m2 - m2	$c\_int * 0.8$
m2 - m3	$g\_int * 0.8$
m2 - m4	$t\_int * 0.8$
m1 - Insert state 2 (i3)	0.10
m1 - Delete state 2 (d2)	0.10
m3 - m1	$a\_int * 0.8$
m3 - m2	$c\_int * 0.8$
m3 - m3	$g\_int * 0.8$
m3 - m4	$t\_int * 0.8$
m1 - Insert state 2 (i4)	0.10
m1 - Delete state 3 (d3)	0.10
m4 - m1	$a\_int * 0.8$
m4 - m2	$c\_int * 0.8$
m4 - m3	$g\_int * 0.8$
m4 - m4	$t\_int * 0.8$
m1 - Insert state 5 (i5)	0.10
m1 - Delete state 4 (d4)	0.10
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25

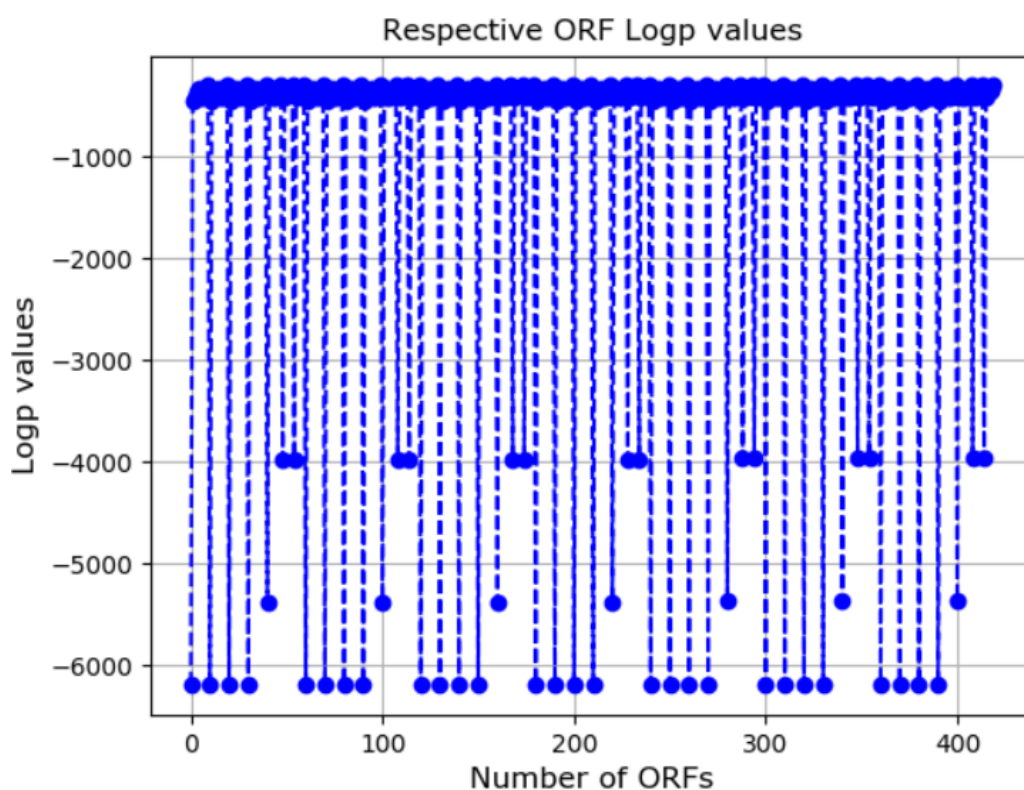
**Table 5** - Insert state transition values (Scenario 1)

Transition State	Value probability (0-1)
i1 - i1	0.10
i1 - d1	0.10
i1 - m1	0.80
i2 - i2	0.10
i2 - d2	0.10
i2 - m2	0.80
i3 - i3	0.10
i3 - d3	0.10
i3 - m3	0.80
i4 - i4	0.10
i4 - d4	0.10
i4 - m4	0.80
i5 - i5	0.10
i5 - Model end	0.90

**Table 6** - Delete state transition values (Scenario 1)

Transition State	Value probability (0-1)
d1 - d2	0.15
d1 - i1	0.15
d1 - m2	0.70
d2 - d3	0.15
d2 - i2	0.15
d2 - m3	0.70
d3 - d4	0.15
d3 - i3	0.15
d3 - m4	0.70
d4 - i4	0.30
d4 - Model end	0.70

Logp values were plotted using matplotlib (Figure 11). A graph with all Logp values of the ORFs is shown below. Dots represent the ORFs and the lines are merely connectors and don't have any significant value.



**Figure 12** - Logp values for Scenario 1. Stored ORFs were used to compare with the target sequence. The bigger oscillations seen represent the 7 sequences that the ORFs were compared to from the target file. The best score, represented as a Logp value, is -305.68. In total, 420 alignments were made.

As can be observed, the ORFs stored from the ORF finder were all compared to the sequences present in the target fasta file. Longer sequences have a larger Logp value (negative in this case) and shorter sequences have a smaller Logp value. This only means that a longer sequence may be less likely to find than a shorter one, not excluding the possibility that it doesn't exist in the target sequence.

### 3.2. Scenario 2

In this scenario all match states have the same probability, not taking into consideration Chargaff's rule. We give each match state transition a value of 0.20, making it evenly distributed. The insert states and delete states have the same values as the previous scenario, for testing purposes. This scenario is based on the premise that each base is evenly distributed in DNA and the probability of encountering any one of the four bases is the same. The initial distributions, just like the previous scenario, are integers calculated from the input sequence that the user wants to use as comparison with the ORFs previously stored. Insert state transition values and delete state transition values are the same as the previous scenario. The tables below, 7 to 10, represent the values used for each state. Logp values were plotted using matplotlib (Figure 12).

**Table 7** - Model start transition values (Scenario 2)

Transition State	Value probability (0-1)
Model start - m1	0.20
Model start - m2	0.20
Model start - m3	0.20
Model start - m4	0.20
Model start - i1	0.20

**Table 8** - Match state transition values (Scenario 2)

Transition State	Value probability (0-1)
m1 - m1	0.20
m1 - m2	0.20
m1 - m3	0.20
m1 - m4	0.20
m1 - i2	0.10
m1 - d1	0.10
m2 - m1	0.20
m2 - m2	0.20
m2 - m3	0.20
m2 - m4	0.20
m1 - i3	0.10
m1 - d2	0.10
m3 - m1	0.20
m3 - m2	0.20
m3 - m3	0.20
m3 - m4	0.20

m1 - i4	0.10
m1 - d3	0.10
m4 - m1	0.20
m4 - m2	0.20
m4 - m3	0.20
m4 - m4	0.20
m1 - i5	0.20
m1 - d4	0.20
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25

**Table 9** - Insert state transition values (Scenario 2)

Transition State	Value probability (0-1)
i1 - i1	0.10
i1 - d1	0.10
i1 - m1	0.80
i2 - i2	0.10
i2 - d2	0.10
i2 - m2	0.80
i3 - i3	0.10
i3 - d3	0.10
i3 - m3	0.80
i4 - i4	0.10
i4 - d4	0.10
i4 - m4	0.80
i5 - i5	0.10
i5 - Model end	0.90

Table 10 - Delete state transition values (Scenario 2)

Transition State	Value probability (0-1)
d1 - d2	0.15
d1 - i1	0.15
d1 - m2	0.70
d2 - d3	0.15
d2 - i2	0.15
d2 - m3	0.70
d3 - d4	0.15
d3 - i3	0.15
d3 - m4	0.70
d4 - i4	0.30
d4 - Model end	0.70

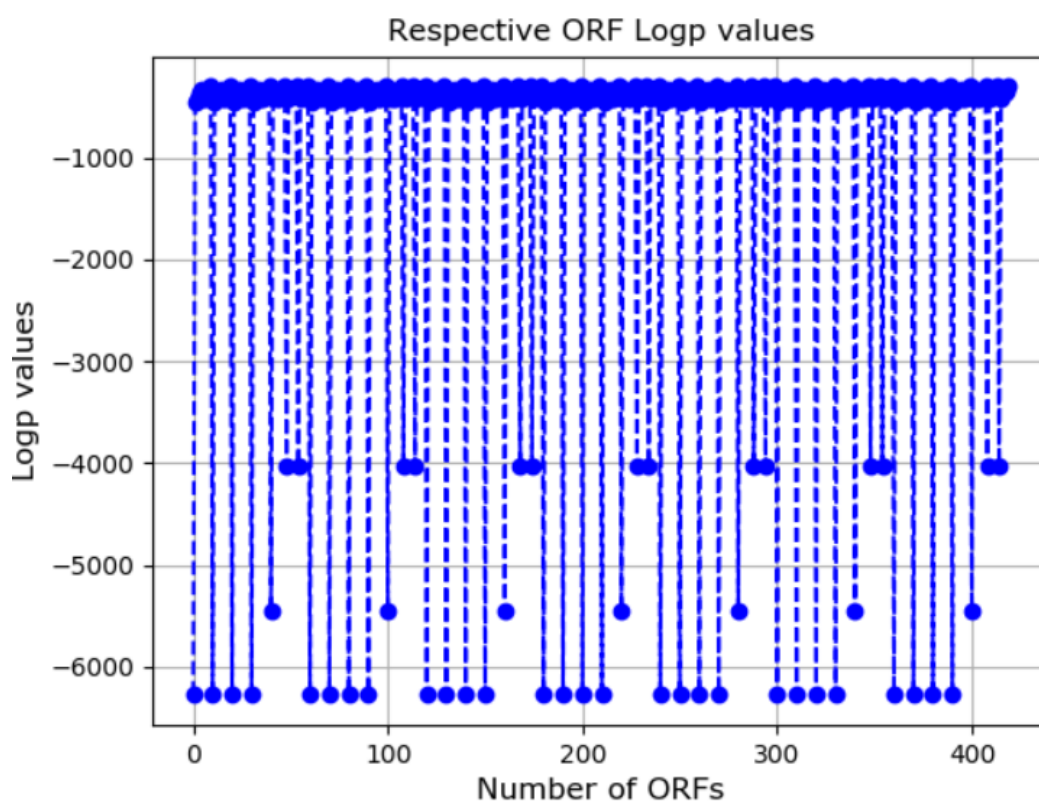


Figure 13 - Logp values for Scenario 2. The best score, represented as a Logp value, is -305.26. In total, 420 alignments were made, the same number as the previous scenario.



### 3.3. Scenario 3

This scenario is slightly similar to the first, but with some attempts to tweak the model. Chargaff's rule is applied, just like in the first scenario, but this time without the percentages being taken from the target sequence. The percentages used are the base probabilities of encountering any base with their normal probability, without any previous analysis of which part of the genome the sequence is. The percentages of the bases are also multiplied by 0.8 due to them only representing 80% of the probability, the rest going to the probability of encountering an insert state right after starting the model. In tables 11 to 14, the transition values for each state are shown. Once again, matplotlib was used to plot the Logp values (Figure 13).

**Table 11** - Model start transition values (Scenario 3)

Transition State	Value probability (0-1)
Model start - m1	0.24
Model start - m2	0.16
Model start - m3	0.16
Model start - m4	0.24
Model start - i1	0.20

**Table 12** - Match state transition values (Scenario 3)

Transition State	Value probability (0-1)
m1 - m1	0.24
m1 - m2	0.16
m1 - m3	0.16
m1 - m4	0.24
m1 - i2	0.10
m1 - d1	0.10
m2 - m1	0.24
m2 - m2	0.16
m2 - m3	0.16
m2 - m4	0.24
m1 - i3	0.10
m1 - d2	0.10
m3 - m1	0.24
m3 - m2	0.16
m3 - m3	0.16
m3 - m4	0.24
m1 - i4	0.10

m1 - d3	0.10
m4 - m1	0.24
m4 - m2	0.16
m4 - m3	0.16
m4 - m4	0.24
m1 - i5	0.10
m1 - d4	0.10
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25

**Table 13** - Insert state transition values (Scenario 3)

Transition State	Value probability (0-1)
i1 - i1	0.10
i1 - d1	0.10
i1 - m1	0.80
i2 - i2	0.10
i2 - d2	0.10
i2 - m2	0.80
i3 - i3	0.10
i3- d3	0.10
i3 - m3	0.80
i4 - i4	0.10
i4 - d4	0.10
i4 - m4	0.80
i5 - i5	0.10
i5 - Model end	0.90

Table 14 - Delete state transition values (Scenario 3)

Transition State	Value probability (0-1)
d1 - d2	0.15
d1 - i1	0.15
d1 - m2	0.70
d2 - d3	0.15
d2 - i2	0.15
d2 - m3	0.70
d3 - d4	0.15
d3 - i3	0.15
d3 - m4	0.70
d4 - i4	0.30
d4 - Model end	0.70

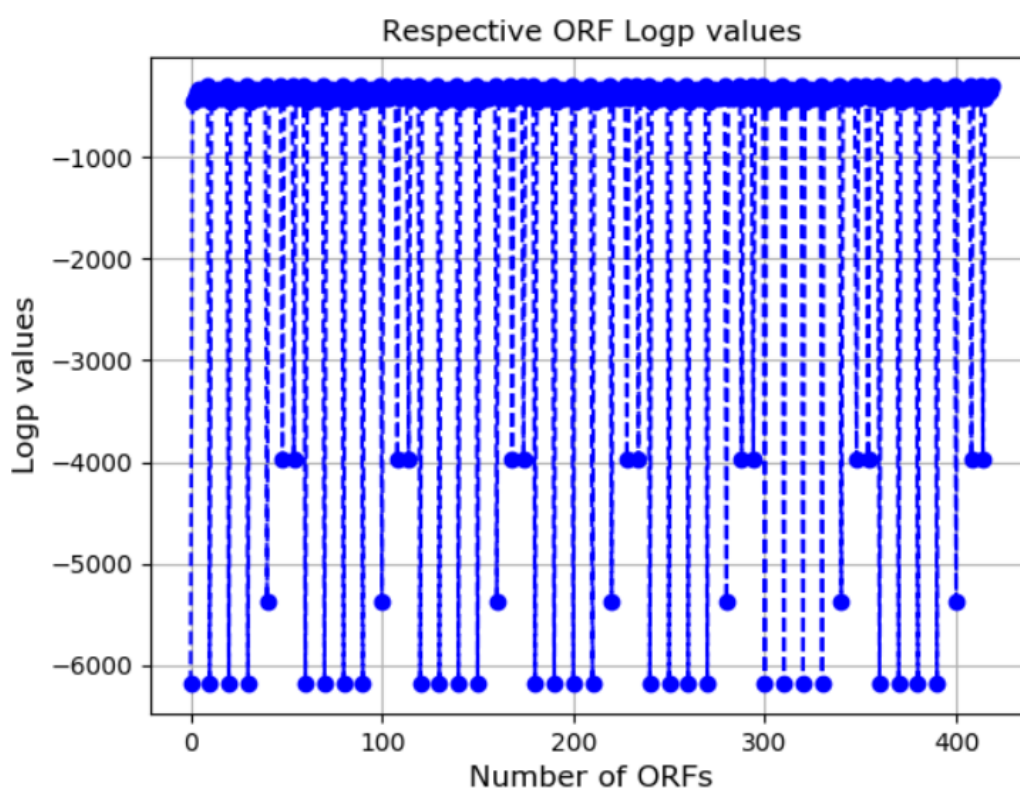


Figure 14 - Logp values for Scenario 3. The best score, represented as a Logp value, is -301.43. In total, 420 alignments were made, the same number as the previous scenarios.

Surprisingly, and comparing all three previous scenarios, this was the one that returned the lowest Logp value out of all of them. Based on that, the testing for the target sequence and then comparison with another random sequence from NCBI's genome database, was done using these values.

### 3.4. Scenario 4 (Insert state scenario)

Having discovered the best match state, insert state and delete state insert and transition values for the Profile-HMM, a scenario where the insert state transition values are modified was next. This scenario is purely experimental. We are guessing, from the get go, that this scenario is not a viable one. And why? Due to the fact that an insert states is returned only if there are is no matching base in the sequence at a given position. The aim of this program is to find sequences that are present in the target one, and not sequences that don't belong. Even though these insert states are necessary for other sequences not present in the target sequence in order to distinguish them, it isn't expected for a given sequence to only have insert states and no match states or even delete states. Insert state values are substituted so that, and purely in an experimental matter, the probability of there being an insert state is higher than there being a match state or a delete state. Insert state transition values and probabilities were given matching probabilistic values as the match states combined in the previous scenarios. Tables 15 to 18 contain the values used for this scenario. Figure 14 represents the plot of the ORF logp values, compared to the target sequence.

**Table 15** - Model start transition values (Scenario 4)

Transition State	Value probability (0-1)
Model start - m1	0.10
Model start - m2	0.10
Model start - m3	0.10
Model start - m4	0.10
Model start - i1	0.50

**Table 16** - Match state transition values (Scenario 4)

Transition State	Value probability (0-1)
m1 - m1	0.10
m1 - m2	0.10
m1 - m3	0.10
m1 - m4	0.10
m1 - i2	0.50
m1 - d1	0.10
m2 - m1	0.10
m2 - m2	0.10
m2 - m3	0.10
m2 - m4	0.10
m1 - i3	0.50
m1 - d2	0.10

m3 - m1	0.10
m3 - m2	0.10
m3 - m3	0.10
m3 - m4	0.10
m1 - i4	0.50
m1 - d3	0.10
m4 - m1	0.10
m4 - m2	0.10
m4 - m3	0.10
m4 - m4	0.10
m1 - i5	0.50
m1 - d4	0.10
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25

**Table 17** - Insert state transition values (Scenario 4)

Transition State	Value probability (0-1)
i1 - i1	0.80
i1 - d1	0.10
i1 - m1	0.10
i2 - i2	0.80
i2 - d2	0.10
i2 - m2	0.10
i3 - i3	0.80
i3- d3	0.10
i3 - m3	0.10
i4 - i4	0.80
i4 - d4	0.10
i4 - m4	0.10
i5 - i5	0.90
i5 - Model end	0.10

Table 18 - Delete state transition values (Scenario 4)

Transition State	Value probability (0-1)
d1 - d2	0.10
d1 - i1	0.10
d1 - m2	0.80
d2 - d3	0.10
d2 - i2	0.10
d2 - m3	0.80
d3 - d4	0.10
d3 - i3	0.10
d3 - m4	0.80
d4 - i4	0.90
d4 - Model end	0.10

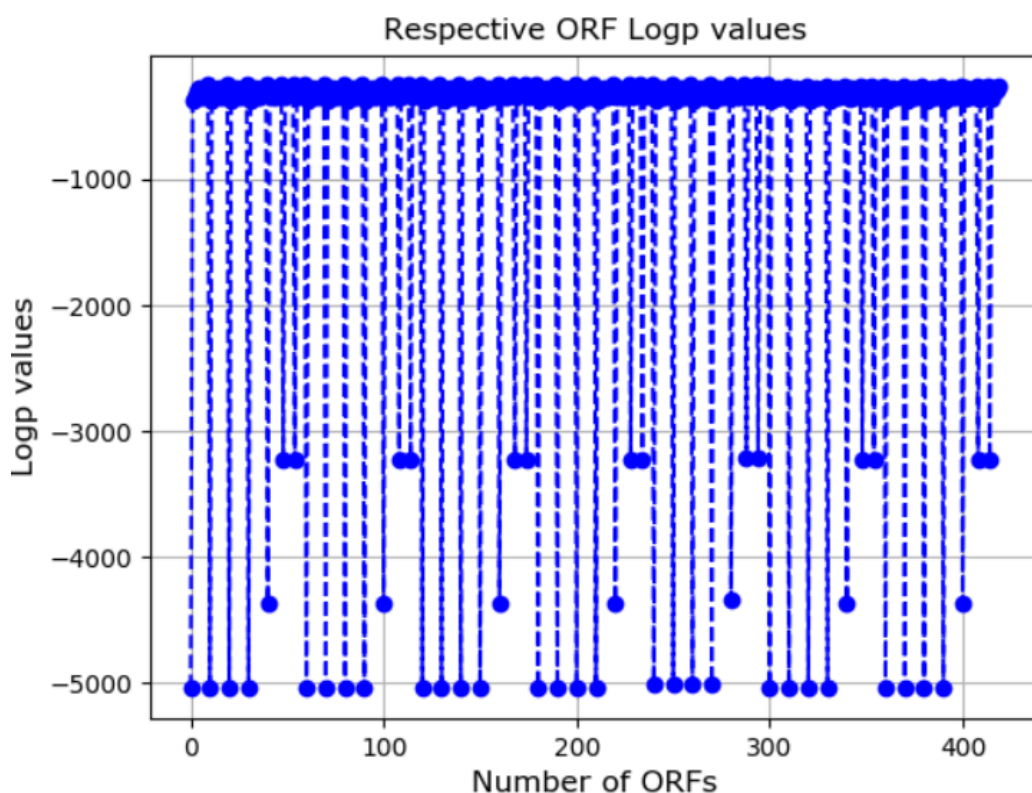


Figure 15 - Logp values for Scenario 4. The best score, represented as a Logp value, is -251.56. In total, 420 alignments were made, the same number as the previous scenarios.

Even though the Logp probability is lower than the best match state scenario, it's completely unrealistic. The HMM assumes that there are no match states and the only state present is i5,

or insert state 5. The sequences match at the first one to three stages, matching with random initial bases, but immediately jump to i5 until the end of the model.



### 3.5. Scenario 5 (Delete state scenario)

Having proven that insert state transition values higher than match state transition values is not a viable scenario, the same was proven with delete states. In this scenario, the delete states are modified. Just like when the insert state transition values were modified, this design is purely experimental, due to the fact that in this scenario, with the delete states higher than the match states or even the insert states, the assumption is made that, a sequence aligned with another will present more deletions than matches. Tables 19 to 22 represent the values used for the model in this specific scenario.

**Table 19** - Model start transition values (Scenario 5)

Transition State	Value probability (0-1)
Model start - m1	0.20
Model start - m2	0.20
Model start - m3	0.20
Model start - m4	0.20
Model start - i1	0.20

**Table 20** - Match state transition values (Scenario 5)

Transition State	Value probability (0-1)
m1 - m1	0.10
m1 - m2	0.10
m1 - m3	0.10
m1 - m4	0.10
m1 - i2	0.10
m1 - d1	0.50
m2 - m1	0.10
m2 - m2	0.10
m2 - m3	0.10
m2 - m4	0.10
m1 - i3	0.10
m1 - d2	0.50
m3 - m1	0.10
m3 - m2	0.10
m3 - m3	0.10
m3 - m4	0.10
m1 - i4	0.10
m1 - d3	0.50
m4 - m1	0.10

m4 - m2	0.10
m4 - m3	0.10
m4 - m4	0.10
m1 - i5	0.10
m1 - d4	0.50
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25
m1 - Model end	0.25

**Table 21** - Insert state transition values (Scenario 5)

Transition State	Value probability (0-1)
i1 - i1	0.10
i1 - d1	0.80
i1 - m1	0.10
i2 - i2	0.10
i2 - d2	0.80
i2 - m2	0.10
i3 - i3	0.10
i3- d3	0.80
i3 - m3	0.10
i4 - i4	0.10
i4 - d4	0.80
i4 - m4	0.10
i5 - i5	0.50
i5 - Model end	0.50

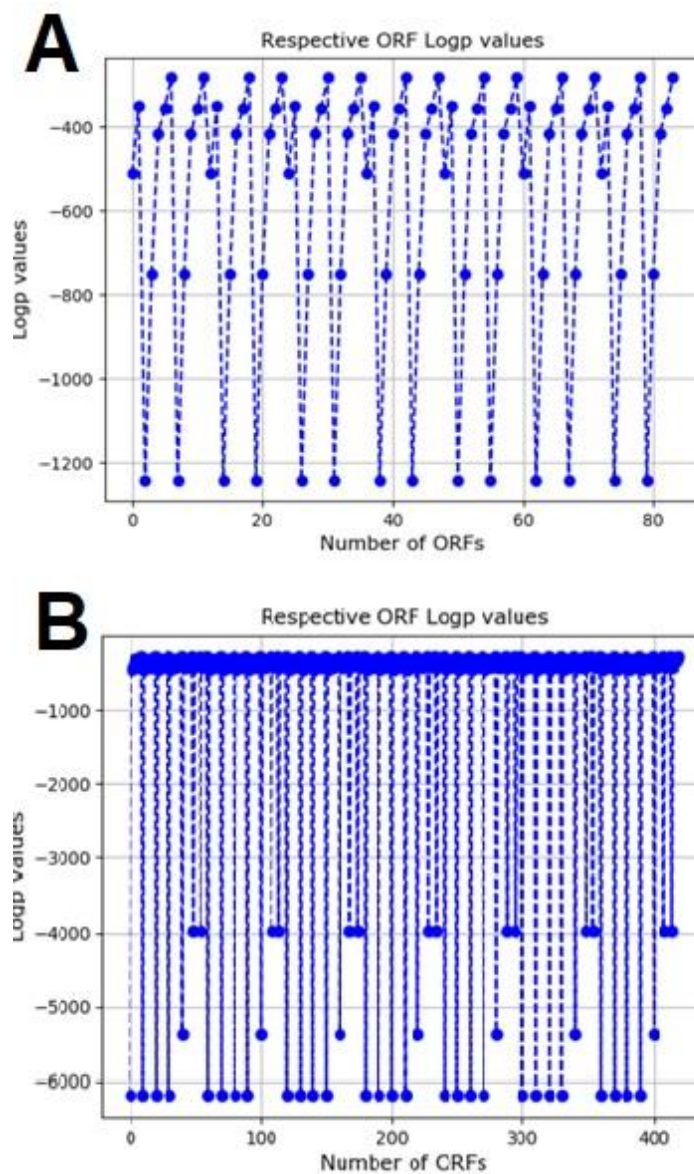
Table 22 - Delete state transition values (Scenario 5)

Transition State	Value probability (0-1)
d1 - d2	0.15
d1 - i1	0.15
d1 - m2	0.70
d2 - d3	0.15
d2 - i2	0.15
d2 - m3	0.70
d3 - d4	0.15
d3 - i3	0.15
d3 - m4	0.70
d4 - i4	0.30
d4 - Model end	0.70

The program was run but didn't manage to end due to an error. We assume this occurs due to how an HMM works, and since having solely delete states tampers with the model itself. It could also create an infinite loop of deletions, never being able to find neither a match or insert state.

### 3.6. Final comparison

To test the veracity of the program, the Logp values of two different sequences were compared. One was the test sequence, the JAK2 Janus kinase 2 [Homo sapiens (human)] genome protein-coding fasta formatted file. The other file was chosen at random. The CRP C-reactive protein [Homo sapiens (human)] genome protein-coding fasta formatted file was chosen. Both were run with the program and the results were as follow. Below are the resulting Logp graphs, plotted using matplotlib.



**Figure 16** - Graphs representing the number of ORFs and Logp values. A) CRP C-reactive protein [Homo sapiens (human)] ORFs, B) JAK2 Janus kinase 2 [Homo sapiens (human)] ORFs.

Even though the Logp value for the shortest ORF in the CRP C-reactive protein [Homo sapiens (human)] ORF file was lower, the comparison can't be made that way, due to sequence length. So, two equally sized sequences were taken from each Profile-HMM comparison run, and then compared Logp values.

Sequence taken from the CRP C-reactive protein [Homo sapiens (human)] ORF file:

```
'CAGCACAAAGTCCCACATGTTACATTTCCAATGTCTCCCACCAGGGACTGGCTTCCTTCAAAGTTCCC  
ACCGAAGGAATCCTGCTCCTGCCCAAGATGATGCTTGCTTCTGCCCCACAGTGTATCCCTTCTTCAGA  
CTCTTCCTCACCTGGGCTTCCCATCTACCCAGAACTCCACGATCCCTGA'
```

Logp value: -367.78

Sequence taken from JAK2 Janus kinase 2 [Homo sapiens (human)] ORF file:

```
'GTGTTGTCCTGTAGAGGGTCATACCGGCACATCTCCACACTCCCAAAATTACCCTTGCCAAGTTGCTGT  
AGAAATTTCAAATGTCTCTCTTCAAAGTGTGTAGGATCCCGGTCTTCAAAGGCACCAGAAAACCCAGGG  
CACCTATCCTCATATTTGGTAACATGTCATTTTCTGTTAATAGTTCATAA'
```

Logp value: -354.41

As can be observed, the Logp values for the target sequence are better than the Logp values for the sequence that is unknown. The ORF finder didn't find as many ORFs for the unknown sequence as it did for the unknown one, due to smaller protein-coding sequence sizes and therefore smaller ORFs and in less quantity. Even though this was the case, a same-sized sequence was compared from both ORF files, with the respective Logp values, resulting in the previously made analysis.

## **Chapter IV**



## 4. Conclusion

The method for gene prediction presented in this project is different. Even though the possible ORFs are taken from a previously known coding sequence, it makes it even more specific. Since a Profile-HMM is used, a comparison can be made between several files, using discovered ORFs with the same size and their respective Logp values. As mentioned, and presented above, lower Logp values mean better probability of the input file, with several sequences, belonging to the target sequence. Even though in theory this is true, sequences from different ORF files may have different sizes and therefore a smaller sequence will, in theory, already have a small Logp value than a longer one, due to the probability distribution. Hence the need to compare Logp values between equal sized sequences. As presented in the results, a comparison with a random fasta formatted file containing protein-coding gene sequences, from the CRP C-reactive protein [Homo sapiens (human)] gene, show that the program, when comparing sequences, that belong to the target sequence, with random sequences, results in a better Logp value for the sequence that is present in the target one.

With all the information present in the results and discussion, it's safe to assume that models with either insert transition states and values or delete transition states and values, are completely unrealistic, due to the fact that they bypass and defeat the purpose of having different base percentages and even that bases exist; and only delete states are present, meaning that the sequences hardly ever match or never match (respectively).

## Chapter V





## 5. Future Perspectives

Gene prediction tools have been created along the years, either from scratch or making use of previous algorithms. It would be interesting to see how the implementation of different types of models, apart from HMMs, would benefit the gene prediction tool scenario. For example, inGAP-CDG, mentioned before, uses a completely new approach to gene prediction as a whole, also taking advantage of machine learning [66]. Using a different HMM could also lead to different results and open up new insights to models in gene prediction.

In this project the VA was used, due to its speed and highest probability function value. Substituting the VA algorithm with the Baum-Welch algorithm or another Expected Maximization (EM) algorithm, could present us with quite different results. It wouldn't be as fast as the VA algorithm but its accuracy would most certainly go up, due to the forward-backward method the Baum-Welch algorithm implements. Even though training would take longer, the results would be more viable and trustworthy due to the nature of the algorithm and how it works.

The method presented in this dissertation work needs more research, but can be a starting point to the creation of a more advanced gene prediction tool in the future. Using a Profile-HMM to compare several ORFs at the same time, with a target sequence, and then comparing Logp results between same-sized sequences. Other types of HMMs could be tested or even a completely new model, such as a BN or GHMM.

## **Chapter VI**



## 6. References

- [1] “<https://ghr.nlm.nih.gov/primer/basics/noncodingdna>.”, (Last accessed on 16/6/2019).
- [2] R. J. Brooker, *Genetics: Analysis & Principles*, 6th ed. 2014.
- [3] B. A. Pierce, *Genetics: A Conceptual Approach*, 4th ed. 2012.
- [4] “<https://www.lexico.com/en/definition/genetics>.”, (Last accessed on 18/6/2019) .
- [5] “<https://www.ncbi.nlm.nih.gov/pubmed/>.”, (Last accessed on 17/6/2019) .
- [6] Gregor Mendel, “Experiments in Plant Hybridization,” in *Meetings of the Brünn Natural History Society*, 1865, no. 1, pp. 1-41.
- [7] W. F. H C Crick, “Molecular Structure of Deoxypentose Nucleic Acids,” *Symposia of the Society for Experimental Biology*, vol. 171, no. 4356, p. 192, 1953.
- [8] et al. F.S. Collins, E.S. Lander, J. Rogers, “Finishing the euchromatic sequence of the human genome,” *Nature*, vol. 431, no. 7011, pp. 931-945, 2004.
- [9] E. S. . L. Lander *et al.*, “Initial sequencing and analysis of the human genome,” *Nature*, vol. 409, no. February, pp. 860-921, 2001.
- [10] J. C. Venter *et al.*, “The Sequence of the Human Genome,” *Science*, vol. 291, no. 5507, pp. 1304-1351, 2001.
- [11] A. Goffeau *et al.*, “Life with 6000 Genes conveniently among the different interna- Old Questions and New Answers The genome . At the beginning of the se- of its more complex relatives in the eukary- cerevisiae has been completely sequenced Schizosaccharomyces pombe indicate,” *Science*, vol. 274, no. October, pp. 546-567, 1996.
- [12] “[www.chegg.com/homework-help/questions-and-answers/codon-chart--set-primers-designed-code-series-amino-acids-review-codon-amino-acid-pairs-da-q13230503](http://www.chegg.com/homework-help/questions-and-answers/codon-chart--set-primers-designed-code-series-amino-acids-review-codon-amino-acid-pairs-da-q13230503).”, (Last accessed on 13/6/2019) .
- [13] “<https://en.wikipedia.org/wiki/Karyotype>.”, (Last accessed on 13/6/2019).
- [14] P. Hogeweg, “The roots of bioinformatics in theoretical biology,” *PLoS Computational Biology*, vol. 7, no. 3, pp. 1-5, 2011.

- [15] “[www.uniprot.org](http://www.uniprot.org).”, (Last accessed on 26/5/2019).
- [16] “[www.ensembl.org/index.html](http://www.ensembl.org/index.html).”, (Last accessed on 26/5/2019).
- [17] “[www.ncbi.nlm.nih.gov/gene/](http://www.ncbi.nlm.nih.gov/gene/).”, (Last accessed on 26/5/2019).
- [18] J. Chen and G. Coppola, *Bioinformatics and genomic databases*, 1st ed., vol. 147. Elsevier B.V., 2018.
- [19] “[www.science.co.il/biomedical/databases/](http://www.science.co.il/biomedical/databases/).”, (Last accessed on 20/5/2019).
- [20] “<https://www.britannica.com/science/bioinformatics>.” (Last accessed on 29/5/2019).
- [21] N. M. Luscombe, D. Greenbaum, and M. Gerstein, “What is bioinformatics? An introduction and overview,” *Yearbook of Medical Informatics*, vol. 10, no. 01, pp. 83-100, Aug. 2001.
- [22] UNESCO, “The origins of bioinformatics.,” *Nature Reviews Genetics*, vol. 1, no. 3, pp. 231-236, 2000.
- [23] J. Backus, “The History of FORTRAN I, II and III,” *IEEE Annals of the History of Computing*, vol. 1, no. 1, pp. 25-45, Jan. 1979.
- [24] M. O. Dayhoff and R. S. Ledley, “Comproteins: a computer program to aid primary protein structure determination,” *Proceedings December 4-6, 1962, fall Jt. Computing Conference*, pp. 262-274, 1962.
- [25] J. Gauthier, A. T. Vincent, S. J. Charette, and N. Derome, “A brief history of bioinformatics,” *Briefings in Bioinformatics*, no. 1, February, pp. 1-16, 2018.
- [26] N. B. Saul and W. Christian, “A General Method Applicable To Search for Similarities in Amino Acid Sequence of two Proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443-453, 1970.
- [27] “<https://en.wikipedia.org>.”, (Last accessed on 15/6/2019).
- [28] D. G. Hert, C. P. Fredlake, and A. E. Barron, “Advantages and limitations of next-generation sequencing technologies: A Comparison of electrophoresis and non-electrophoresis methods,” *Electrophoresis*, vol. 29, no. 23, pp. 4618-4626, 2008.
- [29] J. Felsenstein, “Evolutionary trees from DNA sequences: A maximum likelihood approach,” *Journal of Molecular Evolution*, vol. 17, no. 6, pp. 368-376, Nov. 1981.

- [30] F. F. Nascimento, M. Reis, and Z. Yang, "Europe PMC Funders Group Europe PMC Funders Author Manuscripts A biologist 's guide to Bayesian phylogenetic analysis," *Nature Ecology & Evolution*, vol. 2017, no. 10, pp. 1446-1454, 2018.
- [31] J. Devereux, P. Haeblerli, and O. Smithies, "A comprehensive set of sequence analysis programs for the VAX," *Nucleic Acids Research*, vol. 12, no. 1, Part1, pp. 387-395, 1984.
- [32] T. Tanaka, "[International HapMap project]," *Nihon Rinsho.*, vol. 63 Suppl 1, no. 3, pp. 29-34, Dec. 2005.
- [33] M. Borodovsky and J. McIninch, "GENMARK: Parallel gene recognition for both DNA strands," *Journal of Computational Chemistry*, vol. 17, no. 2, pp. 123-133, 1993.
- [34] L. A and B. M, "GeneMark.hmm: new solutions for gene finding," *Nucleic Acids Research*, vol. 26, no. 4, pp. 1107-15, 1998.
- [35] M. Stanke and S. Waack, "Gene prediction with a hidden Markov model and a new intron submodel," *Bioinformatics*, vol. 19, no. Suppl. 2, pp. ii215-ii225, Sep. 2003.
- [36] C. Burge and S. Karlin, "Prediction of Complete Gene Structures in Human Genomic DNA," *Journal of Molecular Biology*, vol. 268, no. 1, pp. 78-94, 1997.
- [37] A. A. Salamov and V. V Solovyev, "Ab initio gene finding in Drosophila genomic DNA.," *Genome Research*, vol. 10, no. 4, pp. 516-522, Apr. 2000.
- [38] V. Sharma, P. Schwede, and M. Hiller, "CESAR 2.0 substantially improves speed and accuracy of comparative gene annotation," *Bioinformatics*, vol. 33, no. 24, pp. 3985-3987, 2017.
- [39] K. J. Hoff, S. Lange, A. Lomsadze, M. Borodovsky, and M. Stanke, "BRAKER1: Unsupervised RNA-Seq-based genome annotation with GeneMark-ET and AUGUSTUS," *Bioinformatics*, vol. 32, no. 5, pp. 767-769, 2015.
- [40] A. C. Testa, J. K. Hane, S. R. Ellwood, and R. P. Oliver, "CodingQuarry: highly accurate hidden Markov model gene prediction in fungal genomes using RNA-seq transcripts," *BMC Genomics*, vol. 16, no. 1, pp. 1-12, 2015.
- [41] K. L. Chan, R. Rosli, T. V. Tatarinova, M. Hogan, M. Firdaus-Raih, and E. T. L. Low, "Seqping: Gene prediction pipeline for plant genomes using self-training gene models and transcriptomic data," *BMC Bioinformatics*, vol. 18, no. Suppl. 1, pp. 1-7, 2017.

- [42] S.-G. Park *et al.*, “TaF: a web platform for taxonomic profile-based fungal gene prediction,” *Genes Genomics*, vol. 41, no. 3, pp. 337-342, Nov. 2019.
- [43] M. Stanke and B. Morgenstern, “AUGUSTUS: A web server for gene prediction in eukaryotes that allows user-defined constraints,” *Nucleic Acids Research*, vol. 33, no. SUPPL. 2, pp. 465-467, 2005.
- [44] E. Picardi and G. Pesole, “Computational Methods for Ab Initio and Comparative Gene Finding,” in *Japanese Journal of Anesthesiology*, vol. 609, no. 11, O. Carugo and F. Eisenhaber, Eds. Totowa, NJ: Humana Press, 2010, pp. 269-284.
- [45] F. Zickmann and B. Y. Renard, “IPred - integrating ab initio and evidence based gene predictions to improve prediction accuracy,” *BMC Genomics*, vol. 16, no. 1, p. 134, 2015.
- [46] C. Liang, L. Mao, D. Ware, and L. Stein, “Evidence-based gene predictions in plant genomes,” *Genome Research*, vol. 19, no. 10, pp. 1912-1923, Oct. 2009.
- [47] M. Stanke, R. Steinkamp, S. Waack, and B. Morgenstern, “AUGUSTUS: a web server for gene finding in eukaryotes,” *Nucleic Acids Research*, vol. 32, no. Web Server, pp. W309-W312, Jul. 2004.
- [48] I. Reid *et al.*, “SnowyOwl: Accurate prediction of fungal genes by using RNA-Seq and homology information to select among ab initio models,” *BMC Bioinformatics*, vol. 15, no. 1, 2014.
- [49] A. van der Burgt, E. Severing, J. Collemare, and P. J. G. M. de Wit, “Automated alignment-based curation of gene models in filamentous fungi,” *BMC Bioinformatics*, vol. 15, no. 1, pp. 1-13, 2014.
- [50] M. P. Dunne and S. Kelly, “OrthoFiller: Utilising data from multiple species to improve the completeness of genome annotations,” *BMC Genomics*, vol. 18, no. 1, pp. 1-16, 2017.
- [51] A. Lomsadze, P. D. Burns, and M. Borodovsky, “Integration of mapped RNA-Seq reads into automatic training of eukaryotic gene finding algorithm,” *Nucleic Acids Research*, vol. 42, no. 15, pp. 1-8, 2014.
- [52] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings IEEE*, vol. 77, no. 2, pp. 257-286, 1989.
- [53] L. J. Rodríguez and I. Torres, “Comparative Study of the Baum-Welch and Viterbi Training Algorithms Applied to Read and Spontaneous Speech Recognition,” in *Lecture Notes in*

*Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, no. June 2003, 2003, pp. 847-857.

[54] B.-J. Yoon, "Hidden Markov Models and their Applications in Biological Sequence Analysis," *Current Genomics*, vol. 10, no. 6, pp. 402-415, 2009.

[55] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4-16, Jun. 1986.

[56] S. R. Eddy, "Profile hidden Markov models," *Bioinformatics*, vol. 14, no. 9, pp. 755-763, 1998.

[57] V. Andrew, J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260-269, 1967.

[58] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164-171, Feb. 1970.

[59] L. Mark, *Learning Python*, 5th ed. 2013.

[60] G. van Rossum, "Python Tutorial," *Python*, no. 3.7.0. pp. 1-149, 2018.

[61] Z. K. Stewart, A. Pavasovic, D. H. Hock, and P. J. Prentis, "Transcriptomic investigation of wound healing and regeneration in the cnidarian *Calliactis polypus*," *Scientific Reports*, vol. 7, no. February, pp. 1-11, 2017.

[62] J. Schreiber, "Pomegranate: fast and flexible probabilistic modeling in python," *J. Machine Learning Research*, vol. 18, no. 18, pp. 1-6, 2018.

[63] B. W. Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich and Nski, "Biopython Tutorial and Cookbook," no. 1.73. pp. 1-343, 2018.

[64] D. W. Mount, *Bioinformatics: Sequence and Genome Analysis*, vol. 1, no. 1. 2001.

[65] "<https://www.ncbi.nlm.nih.gov/gene?Db=gene&Cmd=DetailsSearch&Term=3717>.", (Last accessed on 15/6/2019)

[66] G. Peng, P. Ji, and F. Zhao, "A novel codon-based de Bruijn graph algorithm for gene construction from unassembled transcriptomes," *Genome Biology*, vol. 17, no. 1, pp. 1-12, 2016.

