An Operon-Based Data Science Approach for the Inference of tRNA and rRNA Gene Evolution

by

Tomasz Pawliszak

A thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada

December 2019

Thesis advisors
Drs. Carson K. Leung & Olivier Tremblay-Savard

Author
Tomasz Pawliszak

An Operon-Based Data Science Approach for the Inference of tRNA and rRNA Gene Evolution

Abstract

With advancements in technology, big data can be easily generated and collected. Big data mining and analytics is in demand for discovery of important information and useful knowledge from these big data. An example of big data includes ribonucleic acid (RNA) genes in bacterial genomes in the area of bioinformatics and biological data mining. Specifically, in bacterial genomes, ribosomal ribonucleic acid (rRNA) and transfer ribonucleic acid (tRNA) genes are often organized into operons, i.e., segments of closely located genes that share a single promoter and are transcribed as a single unit. Analyzing how these genes and operons evolve can help us understand what the most common evolutionary events are affecting them and give us a better picture of ancestral codon usage and protein synthesis. We introduce a new approach for the inference of evolutionary histories of rRNA and tRNA genes in bacteria called `BOPAL` for Bacterial Operon Aligner, which is based on the identification of orthologous operons. This approach allows for a better inference of orthologous genes in genomes that have been affected by many rearrangements, which in turn helps with the inference of more realistic evolutionary scenarios and ancestors. From our comparisons of `BOPAL` with other gene order alignment programs using simulated data, we have found that `BOPAL` infers evolutionary events and ancestral gene orders more accurately than other methods based on alignments. An analysis of 12 Bacillus genomes also showed that `BOPAL` performs well in building ancestral histories in a minimal amount of events.

Table of Contents

Abstract i
List of Figures v
List of Tables vii
Acknowledgements viii

1 Introduction **1**
1.1 Thesis Statement 2
1.2 Thesis Organization 3

2 Biological Background **4**
2.1 DNA 4
2.2 Genes 7
2.3 Proteins 8
2.4 Genome 8
2.5 Operons 9
2.6 RNA 10
2.7 Non-Coding RNA (ncRNA) 10
2.8 Protein Synthesis 11
2.8.1 Transcription 11
2.8.2 Translation 12
2.9 Homologs 13
2.9.1 Orthologs 13
2.9.2 Paralogs 14
2.10 Phylogenetics 15
2.11 Phylogenetic Tree 15
2.12 Origins of Early Life 16
2.13 Evolution of Early Genomes 17
2.14 Genome Modifying Events 18
2.14.1 Genome Modifying Events at the Gene Level 18
2.14.2 Genome Modifying Events at the Operon Level 20
2.15 Transfer RNA Evolution 23
2.15.1 Evolution of Transfer RNA in *Escherichia Coli* 23

2.15.2 Evolution of Transfer RNA Genes in Drosophila 23
2.16 Summary 24

3 Related Works **27**
3.1 Pairwise Alignments 27
3.1.1 Needleman-Wunsch Algorithm 28
3.1.2 Smith-Waterman Algorithm 31
3.2 Multiple Sequence Alignment 34
3.2.1 Sequence Searching in Databases 35
3.2.2 FASTA 36
3.2.3 BLAST 41
3.3 Orthology Inference 44
3.3.1 OrthoMCL 44
3.3.2 OrthAgogue 45
3.4 Reconstruction of Ancestral Genomes 46
3.4.1 InferCARs 46
3.4.2 Anges 46
3.4.3 Gap Adjacency 47
3.4.4 ProCARS 48
3.5 Inference of Evolutionary Histories of Transfer RNA Genes 48
3.5.1 Integer Linear Programming Algorithm 48
3.5.2 OrthoAlign and MultiOrthoAlign 49
3.6 Related Studies on Bacterial Genome, Operon, and Transfer RNA Gene Evolution 50
3.7 Summary 51

4 BOPAL **53**
4.1 Evolutionary Model 53
4.2 Research Problem 54
4.3 Annotation of the Gene Orders 55
4.3.1 Location of the Origin and Terminus of Replication 55
4.3.2 Location of the Operons 55
4.4 Algorithm 56
4.4.1 Step 1: Inference of Orthologous Operons and Singletons . . . 56
4.4.2 Step 2: Inference of Duplications, Deletions, and Substitutions 58
4.4.3 Step 3: Inference of Rearrangements 59
4.4.4 Step 4: Inference of the Ancestral Gene Order 59
4.5 Runtime Complexity 61
4.6 Summary 62

5 Evaluation Results And Discussion **64**
5.1 Evaluation on Simulated Datasets 64
5.1.1 Accuracy on Cherries with Neighbor 65
5.1.1.1 Accuracy on Varying Genome Sizes 70
5.1.2 Runtime 72
5.2 Evaluation on Biological Datasets 74
5.3 Summary 76

6 Conclusions and Future Work **80**
6.1 Conclusions 80
6.2 Future Work 83

Bibliography **88**

List of Figures

2.1 Adenine 5
2.2 Guanine 5
2.3 Thymine 6
2.4 Cytosine 6
2.5 DNA Sugar Phosphate Backbone 7
2.6 Genome Modifying Events - Deletion 19
2.7 Genome Modifying Events - Duplication 19
2.8 Genome Modifying Events - Inverted Duplication 20
2.9 Genome Modifying Events - Substitution 20
2.10 Genome Modifying Events - Inversion 21
2.11 Genome Modifying Events - Transposition 22
2.12 Genome Modifying Events - Inverted Transposition 22

3.1 Global Alignment - Matrix Construction 29
3.2 Global Alignment - Matrix Initialization 29
3.3 Global Alignment - Score Computation 30
3.4 Global Alignment - Traceback 30
3.5 Global Alignment - Alignment 31
3.6 Local Alignment - Matrix Construction 32
3.7 Local Alignment - Matrix Initialization 33
3.8 Local Alignment - Score Computation 33
3.9 Local Alignment - Traceback 34
3.10 Local Alignment - Alignment 34
3.11 FASTA - Initialize the Dot Plot 37
3.12 FASTA - Score the Diagonals 38
3.13 FASTA - Re-score the Diagonals 39
3.14 FASTA - Join Diagonals Using Gaps 40
3.15 BLAST - Generate Words and Scan the Database 43
3.16 BLAST - Construct the Alignment 43

4.1 `BOPAL` Workflow 60

5.1 Total Number of Events Inferred 66
5.2 F-measure of the Reconstructed Ancestral Gene Orders 67
5.3 Strict Event Accuracy 69
5.4 Relaxed Event Accuracy 69
5.5 Average Size of the Events Inferred Correctly 70
5.6 F-measure of the Reconstructed Ancestral Gene Orders for Varying Genome Sizes 71
5.7 Strict Event Accuracy for Varying Genome Sizes 71
5.8 Relaxed Event Accuracy for Varying Genome Sizes 72
5.9 Average Runtimes 72
5.10 Runtime on Large Genomes with DupLoss 73
5.11 Runtime on Large Genomes without DupLoss 73
5.12 Tree Used for the Evaluation on Biological Datasets 75
5.13 Size Distribution of the Duplications 76
5.14 Size Distributions of the Deletions 76

List of Tables

5.1 Average Runtimes for $n = 1000$. 73
5.2 Description of the 12 *Bacillus* Genomes Studied 74
5.3 Number of Events Identified on the Dataset of 12 *Bacillus* Genomes . 75

Acknowledgements

I would like to thank my academic supervisors, Dr. Carson K. Leung and Dr. Olivier Tremblay-Savard, for their advice, wisdom and guidance during my M.Sc. thesis.

I would thank my internal examiner, Dr. Michael Domaratzki, and my external examiner, Dr. Georg Hausner from Microbiology, for their constructive comments and suggestions towards this M.Sc. thesis. Also thanks Dr. Max Turgeon for chairing my M.Sc. thesis oral defense.

I would like to thank all of the students in the Bioinformatics Lab and the Data Science, Database & Data Mining Lab for all of their advice and feedback to help me become a better public speaker. I would like to thank Meghan Chua for his help in this research project and for being a good friend. I would like to thank Vince for encouragement and support during my M.Sc. thesis. I would also like to thank my mother and sister for believing in me and being patient.

Tomasz Pawliszak
B.Sc.(Maj.) in Microbiology, The University of Manitoba, 2012
B.C.Sc.(Hons.), The University of Manitoba, 2015

The University of Manitoba
December 05, 2019

Chapter 1

Introduction

With advancements in technology in the current era of big data (including rich and complex data), high volumes of a wide variety of valuable data (which can be of different level of veracity) have been generated and collected from various data sources at high velocity. Hence, big data mining and analytics is in demand for discovery of important information and useful knowledge from these big data. An example of big data includes non-coding ribonucleic acid (ncRNA) sequences in the area of bioinformatics.

With all the advancements in sequencing and culturing methods, coupled with burgeoning interests in gut microbiomes [DGKB19, FKA+19] and transmission risks of pathogenic microbes [KAS19, PWW+19], bacterial genomes are now being sequenced at a very fast pace. This wealth of genetic information provides a great opportunity to study bacterial genome evolution, compare evolutionary rates between different genera, and study the prevalence, frequencies and average size of different evolutionary events.

An interesting aspect of bacterial genomes is the presence of operons [JPSM60] (operons have been identified more recently in eukaryotes [Blu04], but they seem to be more prevalent in prokaryotes). An operon is basically a cluster of closely located genes (also called polycistronic genes) that share a single promoter and are transcribed simultaneously into a single polycistronic messenger ribonucleic acid (mRNA). These genes can then be translated together, or separately when spliced into separate mR-

NAs (differential expression of polycistronic genes has also been observed [CCM+14]). Some of the most studied and well-defined operons in bacteria are ribosomal RNA (rRNA) and transfer RNA (tRNA) operons [KDS00, TBBT15]. There are several reasons for the interest in these operons, since those genes are fundamental for protein synthesis, and their numbers and organization can be used to better understand codon usage [DNK96]. Comparing the rRNA and tRNA gene contents and organization in different species and inferring evolutionary scenarios allows to make predictions about core sets of tRNA genes, ancestral protein synthesis, ancestral codon usage and evolutionary rates.

1.1 Thesis Statement

In this M.Sc. thesis, I propose `BOPAL` (Bacterial OPeron ALigner) [PCLT19], a new approach that is designed to consider the organization of genes into operons and to be more flexible to the relocation of operons into different regions of the genome because of rearrangements. Instead of trying to find orthologous genes, which might not be located in the same region in both gene orders being compared, our method is based on identifying orthologous operons of rRNA and tRNA genes. Indeed, these operons tend to be more conserved in general, since rearrangement events mostly change their location inside the genome, but rarely modify their composition (e.g., a rearrangement event is unlikely to split an operon into two parts) [TBLE15]. Considering operons also allows our method to infer more realistic events by not considering events that would affect blocks of genes that are not part of the same operon, so technically not close to each other on the chromosome. In this thesis we consider the two small phylogeny problem (2-SPP). Given two gene orders G_1 and G_2, and a set of evolutionary events, the 2 small phylogeny problem asks us to infer an ancestral gene order such that the number of operations required to reconstruct the ancestral gene order is minimized. Our heuristic, which considers duplications, deletions, inversions, transpositions and substitutions, can be used for pairwise comparisons (2-SPP), or it can be used to reconstruct the complete evolutionary history and ancestors of a set of gene orders on a phylogeny by solving instances of the 2-SPP in a post-order

traversal of the tree.

We validate our new approach on simulated datasets (cherries and cherries with a neighbor), and we also test it on the same *Bacillus* dataset of 12 genomes used in [ARC13, BE14]. Our results show that with our simulated data, `BOPAL` has the ability to infer events and ancestral gene orders with higher accuracy than other gene alignment algorithms. Similarly, on a biological dataset, `BOPAL` has performed equally as well as multiOrthoAlign [BE14] and DupLoCut [ARC13] at generating the ancestral tree in a minimal amount of events.

Some key questions we will answer in this M.Sc. thesis are:

1. "Can the alignment of gene orders from operons be used to infer an orthologous relationship between two operons?"

2. "Can strains on neighboring branches help infer rearrangement events on the correct branch for two siblings sharing the same parental node?"

3. "Can we prevent incorrectly labeled events from cascading up the phylogeny?"

1.2 Thesis Organization

In Chapters 2 and 3, I will present background material in terms of the biological aspect and the methodological aspect of the project. Further, I will also introduce my method `BOPAL` in Chapter 4, and discuss our evaluation results and findings in Chapter 5. Both Chapter 4 and Chapter 5 have been adapted from the publication in BMC Genomics [PCLT19]. Finally, in Chapter 6, I will conclude by discussing where `BOPAL` performed well and where there is room for improvement.

Chapter 2

Biological Background

This chapter introduces common biological concepts that will be used throughout this M.Sc. thesis. The purpose is to provide the reader with insight and background knowledge about the biological aspect of this M.Sc. thesis rather than just the computational aspect. This will hopefully provide a better understanding as to why certain design decisions were made.

2.1 DNA

Deoxyribonucleic acid (DNA) is present in all living organisms. It is the main mechanism by which living organisms pass genetic information from one generation to the next. DNA contains fundamental instructions enabling an organism to grow, develop, function, and reproduce. A DNA sequence consists of a chain of nucleotides. A nucleotide is the basic unit of DNA. It consists of three main components, a phospate group, a deoxyribose sugar, and a nitrogen-containing nucleobase. Depending on the nucleobase, a nucleotide can either be a pyrimidine or a purine. The pyrimidines consist of cytosine and thymine. The purines consist of adenine and guanine. See Figures 2.1, 2.2, 2.3, and 2.4 for the molecular structure of each nucleotide. Nucleotides are able to form base pairs with each other. Generally, the purines form hydrogen bonds with pyrimidines. Specifically, adenine forms hydrogen bonds with thymine and guanine forms hydrogen bonds with cytosine. We will discuss this in more detail

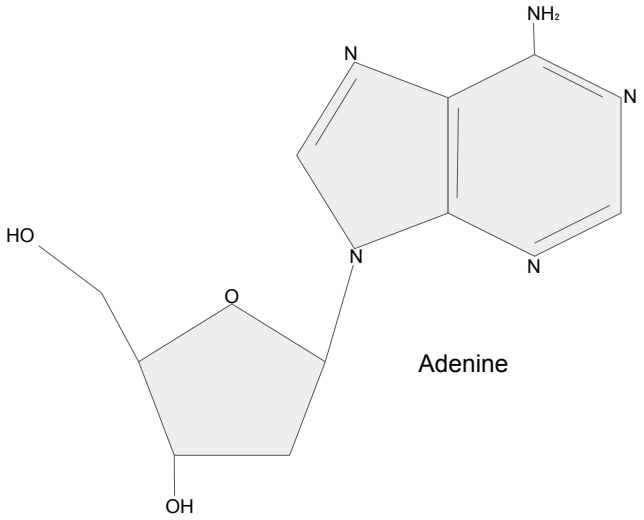


Figure 2.1: Adenine

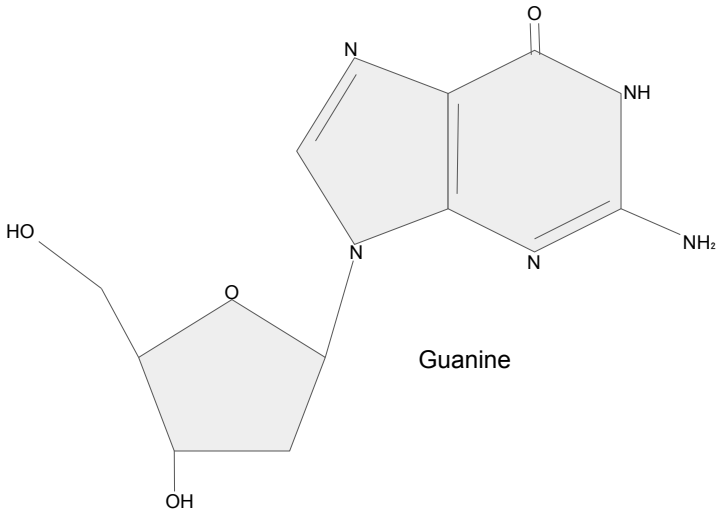


Figure 2.2: Guanine

in Section 2.4. DNA sequences are chains of nucleotides held together by covalent bonds. A covalent bond is a molecular bond where two atoms share an electron with each other. A nucleotide forms a covalent bond with a neighboring nucleotide by

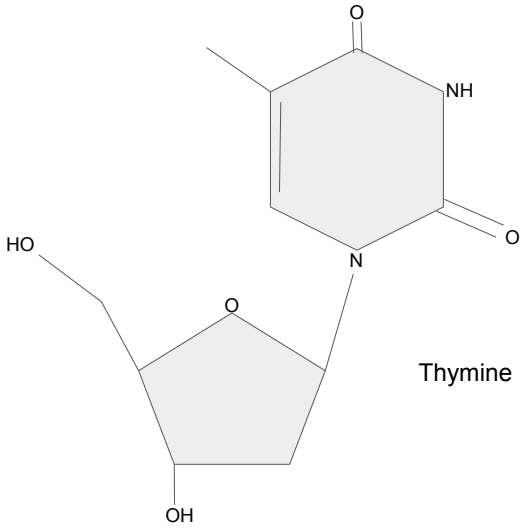


Figure 2.3: Thymine

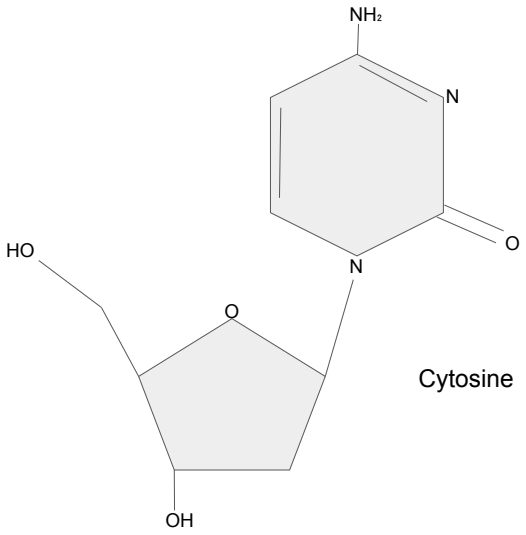


Figure 2.4: Cytosine

sharing an election with the sugar component with the neighbor's phosphate group. As a result this forms an alternating sugar phospate backbone. See Figure 2.5 for the molecule structure of a DNA segment.

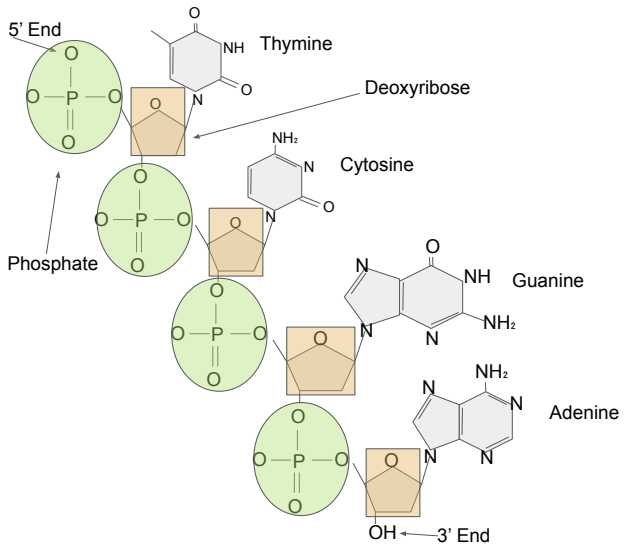


Figure 2.5: DNA Sugar Phosphate Backbone

2.2 Genes

A gene is a sequence of nucleotides within the DNA. The sequence consists of a combination of adenine (A), thymine (T), guanine (G), and cytosine (C). The length of a gene may vary from a few hundred nucleotides to several thousand nucleotides. A genome may contain many genes, therefore, a gene is a sub-component of the genome. We will discuss genomes in more detail later in Section 2.4. Genes are generally used for the purpose of synthesizing organic molecules that will be utilized by the organism for the purpose of structural support, functional support, inhibitors or facilitators of events within the cell, etc. The genome is generally divided into coding regions and non-coding regions. The non-coding regions are generally ignored during transcription, however, they are usually located between the coding regions and act as separators. The size of a non-coding region can be used to predict whether a group of coding regions are in a close proximity to be transcribed together. Genes are located in the coding regions. The coding regions in the genome are what is used during transcription in order synthesize organic molecules. These regions are coded using triplets of nucleotides to describe the components required to construct the

molecule. However, certain organic molecules may require multiple genes in order to be synthesized. Each of these genes acts as a sub-component to the molecule being synthesized. Various combinations of genes transcribed together result in the organic molecule having different properties and/or functions.

2.3 Proteins

The sequence of nucleotides within a gene are used to synthesize organic molecules called proteins. A protein is a large complex molecule that consists of one or more amino acid chains. These molecules are an essential part of all living organisms since they provide structure, regulate processes within the body, and transport various materials throughout the body. The basic unit of a protein consists of an amino acid. An amino acid is an organic compound consisting of a carboxyl group, an amino group, and a side chain that is specific to each individual amino acid. These organic molecules link together to form an amino acid chain called a polypeptide. The length of a polypeptide can vary depending on the protein but is generally between 2 and 50 amino acids long. These polypeptides join together to form a protein molecule. Hence a protein molecule consists of multiple polypeptides. Proteins fall into one of three general categories depending on their function [Pet03]. Fibrous proteins provide structural support such as bone, connective tissue and muscle fiber. Globular proteins are utilized for the purpose of regulating chemical processes within the body and transporting material throughout the body. Finally, membrane proteins are utilized for the purpose of relaying cellular signals within the body allowing cells to communicate with each other.

2.4 Genome

The genetic material of an organism is called the genome. The genome contains all of the genes required to build and maintain the organism. However, the genome may not necessarily consist of one long continuous sequence of genes. The genome may be broken down into several segments called chromosomes. All the chromosomes are present

in all different cell types. However, these chromosome might not be compressed in the same way. Some regions are going to be more accessible in some cell types, some other regions are going to be more compacted. The reason for this is differential gene expression. However, depending on the type of organism the general characteristics of the genome will be different. Organisms generally fall into one of two categories, prokaryotes or eukaryotes. Prokaryotes are single celled organisms with no nucleus. Generally, bacteria, cyanobacteria and Archaea are considered prokaryotes. Genomes in the prokaryotes are generally circular, however, they could be linear genomes and could consist of multiple chromosomes. Eukaryotes could be single celled or multi-cellular organisms, however, the key difference between prokaryotes and eukaryotes is that eukaryotes have a nucleus. In eukaryotes the nucleus is where the genetic material is stored. Generally, eukaryotic genomes are linear and consist of multiple chromosomes. Interestingly, it has been observed that genomes in eukaryotes tend to have a lot more repeated regions, more genes, more complex promoter regions, more non coding regions and spliceosomal introns than prokaryotic genomes [TR84]. For this reason, genomes in eukaryotes tend to be a lot longer compared to prokaryotes.

2.5 Operons

An operon is a cluster of genes in close proximity to each other under the control of a single promoter. The classic lac operon consists of 3 major components, the promoter, the operator, and the genes [HG02]. However, this does not describe all possible operons. Operators are not necessarily found in all operons. In addition, we can have positive or negative regulators and regulatory proteins. The promoter is located upstream of the genes being regulated and is used to initiate transcription by allowing the ribonucleic acid (RNA) polymerase to bind to the DNA. See 2.8.1 for more details about transcription. The operator is located downstream of the promoter but upstream of the genes. Proteins called repressors can bind to the operator site to prevent the RNA polymerase from transcribing the genes into a messenger RNA molecule by blocking its traversal of the DNA. Due to the genes being in close proximity to each other the RNA polymerase transcribes these genes

into a single messenger RNA strand. Depending on the organism, the messenger RNA synthesized by the RNA polymerase may be the final product or it may undergo further processing before it is translated. See Section 2.8.2 for more details about translation.

2.6 RNA

Ribonucleic acid (RNA) is a fundamental molecule that is required for synthesizing proteins. RNA is similar to DNA, however, there are several key differences between them. Firstly, the uracil (U) nucleotide replaces the thymine (T) nucleotide in RNA. Secondly, RNA is always a single stranded molecule. However, RNA may fold on itself to form a thermodynamic more favorable state to stabilize the molecule to prevent it from degrading rapidly. Thirdly, DNA is used for long term storage of genetic information whereas RNA is a molecule that will degrade over time. This means the cell must first synthesize RNA molecules in order to synthesize protein molecules. This is the so called central Dogma of molecular biology, we go from DNA to RNA to Protein. Cells use this mechanism to control which protein molecules will be synthesized and which ones will not. This prevents the cell from wasting resources and energy synthesizing components that will not be utilized.

2.7 Non-Coding RNA (ncRNA)

Non-coding RNA is a molecule that is transcribed from the DNA, however, the RNA molecule is not translated into a protein molecule [MM06]. Transcription is a process that copies a section of the DNA into a newly synthesized complementary RNA molecule. We will discuss transcription and translation in more detail in Section 2.8. These genes are often described as functional non-coding RNA genes. These genes are generally used to regulate gene expression at the transcription level. Non-coding RNAs generally fall into two main categories. The first category comprises of non-coding RNAs that are shorter than 30 nucleotides and the second category comprises of non-coding RNAs that are longer than 200 nucleotides. Two very well known classes

of non-coding RNA molecules include transfer RNAs (tRNA) and ribosomal RNAs (rRNA). Both transfer RNA (tRNA) and ribosomal RNA (rRNA) are fundamental molecules required for protein synthesis. Each transfer RNA (tRNA) consists of an anticodon sequence that can recognize a messenger RNA (mRNA) nucleotide triplet and an amino acid that is specific to the anticodon. Finally, ribosomal RNA (rRNA) is the molecular component that assembles on the messenger RNA (mRNA), traverses the strand allowing transfer RNA (tRNA) molecules to bind and transfer their amino acid to assemble a polypeptide chain that will go to form a protein molecule.

2.8 Protein Synthesis

2.8.1 Transcription

Transcription is the first step of protein synthesis [Pai96]. This process copies a segment of the DNA into a newly synthesized RNA molecule. The newly synthesized molecule is called messenger RNA. The messenger RNA is not an identical copy of the DNA from which it was transcribed (the DNA template strand), rather, it is complementary to the DNA template strand. Additionally, the messenger RNA contains uracil instead of thymine. In order to avoid wasting resources and energy synthesizing messenger RNA when it is not needed there are additional proteins that are needed to initialize transcription. These proteins will bind to the promoter and enhancer regions which are upstream of the region that needs to be transcribed in the DNA. These small protein molecules are called transcription factors and are used to allow the RNA polymerase to bind to the DNA in order for transcription to initialize. The transcription initiation complex forms once the RNA polymerase binds. As the complex traverses the DNA segment, it elongates the messenger RNA molecule by matching complementary nucleotides with the DNA template strand. Once the complex reaches the end of the segment, transcription terminates by unbinding from the DNA and the messenger RNA is released into the cytoplasm.

2.8.2 Translation

Translation is the second step of protein synthesis [Pai96]. In this process the messenger RNA is used to assemble a chain of amino acids. Amino acids are the basic building blocks of proteins and there are 20 unique amino acids. Translation proceeds in 3 phases, initiation, elongation, and termination.

1. Initiation. The ribosome attaches to the messenger RNA. The ribosome is responsible for traversing the messenger RNA and linking the amino acids together to form a chain. The ribosome consists of two major components, a small subunit and large subunit. The small subunit is responsible for reading the messenger RNA strand and the large subunit is responsible for linking the amino acids. Initially, the small subunit attaches to the ribosome binding site that includes the start codon in the messenger RNA. A codon is a sequence of 3 nucleotides which is used to specify one of the 20 amino acids. Since there are 64 possible codons and only 20 amino acids some codons specify the same amino acid. The start codon consists of the nucleotides, adenine, uracil, and guanine or AUG. AUG represents the amino acid methionine. A transfer RNA carrying a methionine amino acid binds to the messenger RNA's start codon by forming hydrogen bonds with its anticodon. Transfer RNAs are molecules used to move amino acids from the cytoplasm to the ribosome for protein synthesis. The anticodon of the transfer RNA is complementary to the codon of the messenger RNA. At this point the large ribosome subunit is able to attach to the small subunit, messenger RNA, and transfer RNA completing the ribosome complex.

2. Elongation. The ribosome traverses to the next codon in the messenger RNA. The transfer RNA with the appropriate anticodon then binds and transfers the amino acid to the ribosome. The ribosome then attaches the amino acid to the growing amino acid chain. The process is mediated with the help of various elongation factors and repeated until the ribosome encounters a stop codon.

3. Termination. Begins when the ribosome reads a stop codon. A stop codon

could be either, UAG, UAA, or UGA. There are no transfer RNAs with these anticodons therefore the ribosome recognizes that theses are stop codons. The ribosome complex with the aid of termination factors releases the amino acid chain and then comes apart. The amino acid chain starts folding while being synthesized on the ribosome and it can interact with other synthesized amino acid chains

in order to form into the appropriate protein molecule. Chaperons are protein factors that can mediate protein folding to ensure the generation of functional proteins.

2.9 Homologs

In evolutionary biology, homology refers to the existence of a shared ancestry. Normally we do not define homology for organisms since it does not work for organisms as a whole. The reason for this is that all living organisms share a common ancestry. Instead, homology is normally defined in terms of sequences, genes, limbs, organs, or structure. In this M. Sc. thesis we will be be focusing on the development of bacterial gene orders that have descended from a common evolutionary ancestor. Gene orders that are a strong match is evidence that these two gene orders are related by a common ancestor. However, homology does not indicate whether the two gene orders have a shared ancestry due to a speciation event or a duplication event. Gene orders that are related due a speciation event are generally referred to as orthologous genes and gene orders that are related due to a duplication event are generally referred to as paralogous genes.

2.9.1 Orthologs

Orthologous DNA segments can be described as segments of the DNA in two different species that were present in a common ancestor but these segments evolved independently after a speciation event. This means the segment of DNA was originally present in an ancestor and the ancestor passed this genetic material to its offspring. After the offspring diverged into two separate species, any changes to the segment in one of

the species would not be reflected in the other because there is no exchange of genetic material between them. This implies that the DNA segments in the two species may not necessarily retain the same sequence originally present in the common ancestor. If the changes introduced into the segment were advantageous, then generally there were environmental pressures that favored these changes. As a result, these changes became the norm in that particular species since they probably survived longer and produced more offspring. This results in the DNA segments diverging from the original sequence in the ancestor. Over long periods of time these sequences diverge even more. However, in order for the sequence to remain functional there must be certain regions in the segment that must be retained otherwise the byproduct will be non-functional. We can use these conserved regions to identify orthologous segments of DNA. If these region were conserved in both species, then it's strong evidence that these regions were present in the ancestral genome and provides insight into the ancestral genome [JRWK02].

2.9.2 Paralogs

Paralogous DNA segments can be described as segments of the genome that have been duplicated and inserted into another region of the genome [GVSV04]. This is the key difference between paralogs and orthologs. If a region of the genome is duplicated before the speciation event, then both of the descendants will inherit the paralogous regions. If the region is duplicated after the speciation event, then only one of the species will have the paralogous segment. This makes reconstructing ancestral genomes difficult since we do not have ancestral genomes available as a reference. If the only one of the species has an additional segment, then we have to decide whether this region was duplicated before the speciation event and the region was deleted from the sibling or whether the region was duplicated after the speciation event and only one of the siblings had the region. Paralogs are advantageous if an organism is to acquire a new functional byproduct. Paralogs generally do not interfere with the original segment. This means there is less pressure to maintain the duplicate segment [GVSV04]. This allows for minor mutations to be introduced into the duplicates.

These mutations could include nucleotide deletions or insertions. This will generally result in a functional byproduct, however, due to the mutations the structure of the byproduct will change which can result in an altered function. Another possibility if there is no pressure to maintain the duplicate segment is that there could be so many mutations introduced that the byproduct is no longer functional. These are called pseudogenes. In addition to paralogs, we could potentially have xenologs. Xenologs are homologous segments that have been acquired via horizontal gene transfer. They are rare but are seen in bacterial genomes that are good at transferring exogenous DNA into their cells.

2.10 Phylogenetics

Phylogenetics is the branch of bioinformatics that studies the evolutionary history of organisms and reconstructs the evolutionary relationships between a set of sequences or species. In order to deduce which species are closely related and which species are distantly related, we need to be able to measure the amount of similarity between them. Generally, in phylogenetics we compare characteristics that are inherited from generation to generation including, DNA, morphological features such as beaks in birds, RNA, etc. [Hal13] A relatively high amount of similarity between two organisms is an indicator that these two organisms shared a recent common ancestor. The amount of similarity might be an indicator of whether they shared a recent common ancestor or whether they shared a more distant ancestor. Two organisms exhibiting a relatively low amount of similarity is an indicator that these organisms are distantly related. Using this information we can identify which organisms are similar and which are distinct and predict how these organisms diverged throughout history.

2.11 Phylogenetic Tree

A phylogenetic tree is a graph that is used to visually represented the genetic relatedness and evolutionary relationship of a group of organisms [Hal13]. There are various sophisticated techniques used to construct phylogenetic trees but the core

idea of these techniques is that organisms exhibiting a high amount of similarity will be have fewer branches separating them compared to a pair of organisms exhibiting a low amount of similarity. The leaf nodes of a phylogenetic trees generally represent extant species. Extant species refer to organisms we can observe today. The internal nodes of the tree represent ancestors that no longer exist. Generally these trees are constructed such that the number of branches required to represent the evolutionary history between these extant organisms is minimized. In some phylogenetic trees the branch lengths represent estimates as to how many years passed between each node in the tree. There are two types of phylogenetic trees, a rooted phylogenetic tree and an unrooted phylogenetic tree. A rooted phylogenetic tree has a node in the tree that is the common ancestor for all of the organisms in the tree. It is generally used to provide a time line of when and where species diverged throughout history. In an unrooted phylogenetic tree we do not have to infer a common ancestor for the organisms in the tree. The root node in the tree is omitted. The general purpose of an unrooted phylogenetic tree is to visually show how similar the organisms are to each other without indicating a root node.

2.12 Origins of Early Life

Scientists hypothesize (Oparine-Haldane theory) that the early oceans formed a primordial soup over 4 billion years ago [FEF09]. The primordial soup was rich in terms of organic compounds. Due to these complex, organic compounds being present in the environment the early ancestral species were able to survive by depending on external sources for nutrients. For this reason, there was no need for organisms to synthesize these compounds on their own. Hence, the early organisms were termed as heterotrophs. Heterotrophic organisms are unable to synthesize organic compounds. As a result, these organisms rely on the environment to provide these nutrients. Scientists suggest that these ancestral species populated the oceans and over time certain organic compounds became exhausted. Due to an exhaustion of certain organic material it became necessary for organisms to synthesize these compounds themselves. As a result, this led to the first autotroph. Autotrophs were able to use the inorganic

material in the environment to synthesize organic compounds required for molecular processes. As a result, there was strong selective pressure favoring organisms that could synthesize these organic compounds on their own. This led to formation of the first metabolic pathway. Metabolic pathways are responsible for synthesizing and catabolizing organic compounds. With the formation of metabolic pathways, organisms became less reliant on the environment to provide all of the organic compounds [FEF09]. As a result metabolic pathways became the main mechanism for providing energy to run molecular processes required for life.

2.13 Evolution of Early Genomes

Early genomes were simple, short sequences and contained few metabolic pathways. These genomes consisted of approximately of 20 to 100 genes [FEF09]. These simple, short genomes have undergone gradual changes throughout history in order to develop new metabolic pathways. In order to introduce a new metabolic pathway, the genomes had to be extended [FEF09]. Various mechanisms were responsible for extending the genome and developing new metabolic pathways including, gene duplication and exon shuffling. Gene duplication was one of the first fundamental mechanisms driving the evolution and elongation of early genomes. The primary function of gene duplication is to create new genes and new metabolic pathways from pre-existing ones. Genes that have originated as duplicates are called paralogs. We've introduced and discussed paralogs in 2.9.2. A paralogous gene may undergo successive duplications forming a paralogous gene family. Upon a close examination of eukaryotes, bacteria and Archaea genomes researchers hypothesize that the vast majority of genes within a genome are the result of tandem duplications. However, researchers are unable to explain how the first ancestral genomes originated. Additionally, exon shuffling allowed for the fusion of cistronic regions. The segment of the gene that codes for a peptide is called the cistronic region. With the fusion of multiple cistronic regions, this resulted in proteins with multiple functions [FEF09]. These types of genes commonly code for catalytic enzymes used in metabolic pathways and are generally unstable and localized only within cells.

2.14 Genome Modifying Events

In order to infer an evolutionary history of a genome we have to identify the genome content modifying events that we will be searching for. This allows us to perform an evidence based approach when inferring the evolutionary history of a genome. We will be scanning the genomes identifying regions of similarity and regions that are distinct. With differing regions, we will be searching for evidence to identify the sequence of events that must have occurred in order to explain the change. We will be searching for evidence of duplications, deletions, inversions, transpositions, and inverted transpositions.

2.14.1 Genome Modifying Events at the Gene Level

Duplication is the process of copying a region of the genome, denoted as the source, to another region of the genome, denoted as the target, resulting in two identical copies within the genome. We can have duplications of individual genes, or a group of genes clustered together, or whole operon duplications. Deletion is the process of removing genes from genomes. Once a gene is removed it is no longer transcribed since it is no longer within the genome. Deletions can remove individual genes, or a group genes in close proximity to each other, or whole operons. Interestingly, we could also have deletions and duplications within a gene's nucleotide sequence. A series of deletions and/or duplications of a gene's nucleotide sequence could potentially result in a new gene. This would result in a slightly modified amino acid sequence when the genes within an operon are transcribed and translated. However, the overall functionality of the protein generally would not change. The modified amino acid sequence would retain the original protein's functionality. Generally, in these types of mutations the amino acids being modified in the amino acid sequence are interchangeable due to them having similar chemical characteristics. In the context of transfer RNA (tRNA) coding genes we could potentially see that one amino acid was substituted for another. These mutations are called substitutions and we rarely see these types of mutations in protein coding genes. We could also have less extreme mutations of gene sequences

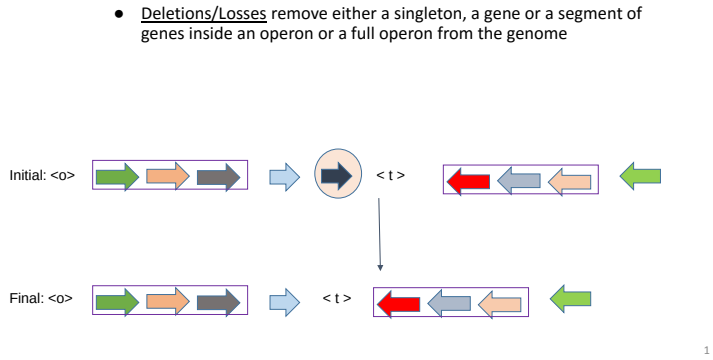


Figure 2.6: Genome Modifying Events - Deletion

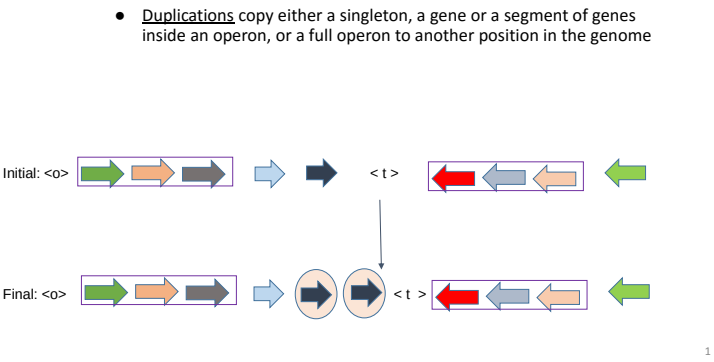


Figure 2.7: Genome Modifying Events - Duplication

where the gene is not changed but the codon of the gene changes. As a result the amino acid would match but the codons would not. These are called silent mutations. See Figures 2.6, 2.7, 2.8 and 2.9 for examples.

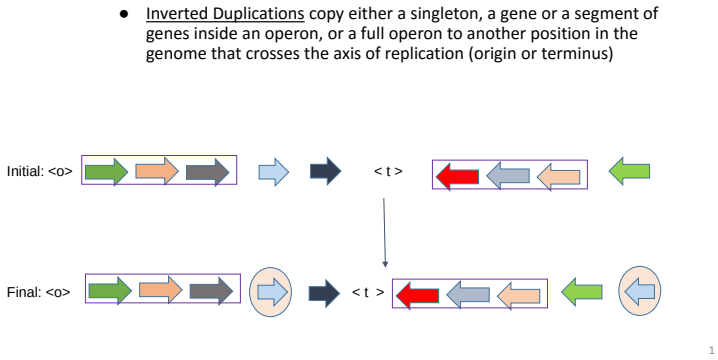


Figure 2.8: Genome Modifying Events - Inverted Duplication

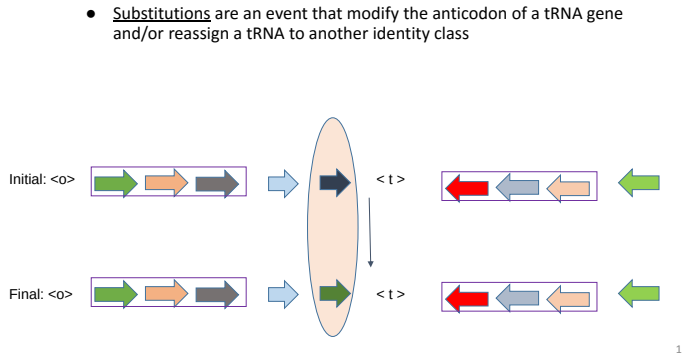


Figure 2.9: Genome Modifying Events - Substitution

2.14.2 Genome Modifying Events at the Operon Level

In bacterial genomes, genes tend to be located mostly on the leading strand of DNA (pointing away from the origin of replication or in other words pointing towards the

terminus of replication) [Roc04], so as to avoid potential head-on collisions between the RNA and DNA polymerases [Bre88]. As observed in a previous study of the *Bacillus* genus [TBLE15], inversions are mostly occurring around one of the *axes of replication* (origin or terminus) because this causes the genes to stay on the leading strand.

In circular unichromosomal genomes, an inversion moves genes across an axis of replication (origin or terminus) resulting in the sequence being in the opposite orientation. This results in the genes being reversed. Inversions generally flip entire operon sequences and do not split an operon into multiple operons. The required components to transcribe an operon are located at the 5' end of an operon and splitting the operon in half would render the other half of the operon unable to be transcribed due to missing components. Therefore inversions generally flip one or more operons in place leaving operon sequences intact. See Figure 2.10 for an example.

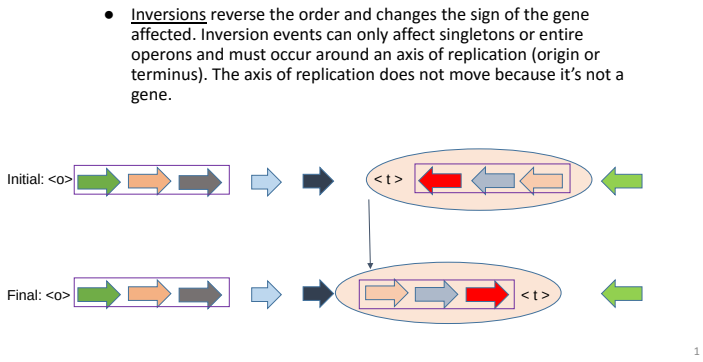


Figure 2.10: Genome Modifying Events - Inversion

Transpositions generally fall into one of two categories, replicative transpositions or conserved transpositions. Replicative transpositions make a copy of genome segment and relocate the copy into another region of the genome. Conserved trans-

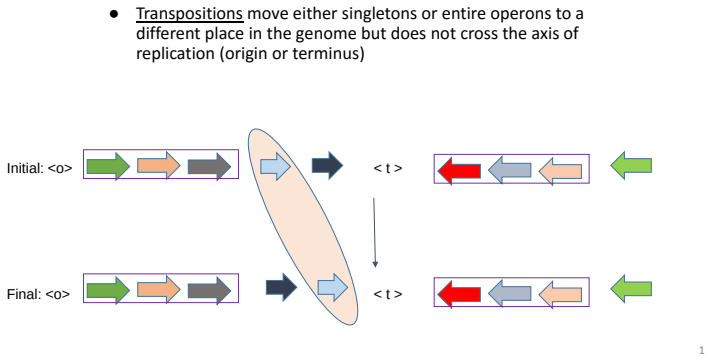


Figure 2.11: Genome Modifying Events - Transposition

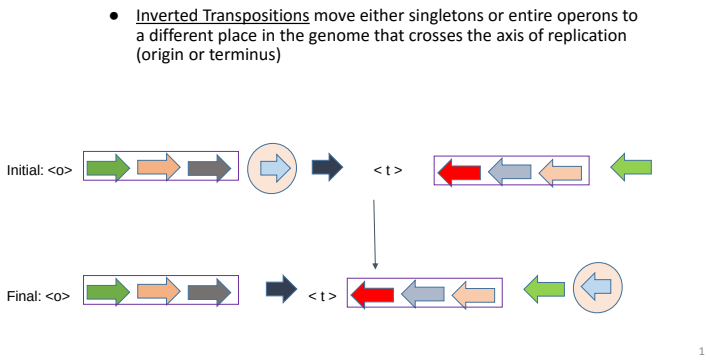


Figure 2.12: Genome Modifying Events - Inverted Transposition

positions excise a region of the genome and reinsert it into another location in the genome. In this thesis, replicative transpositions will be referred to as duplications and conserved transpositions will be referred to as transpositions. An inverted trans-

position is a combination of an inversion followed by a transposition or a transposition followed by an inversion. See Figures 2.11 and 2.12 for examples.

2.15 Transfer RNA Evolution

2.15.1 Evolution of Transfer RNA in *Escherichia Coli*

Withers *et al.* [WWD06] focused on the evolution of tRNA genes in 5 different *Escherichia coli* strains. Closely related strains were selected in order to minimize the number of evolutionary events between the genomes [WWD06]. An analysis of the phylogeny was performed by constructing an archaeological map in order to identify the evolutionary events between the genomes. In order to construct an archaeological map, an alignment was performed between *Salmonella typhimurium* and the other four *Escherichia coli* strains using MultipPipMaker [WWD06] since *Salmonella typhimurium* is the common ancestor of the other four strains. After an optimal alignment was constructed, Kimuras two-parameter model was used to compute the number of evolutionary events between the genomes [WWD06]. Using this information a tree was constructed to infer potential ancestral gene orders. As a result, it was discovered that horizontal gene transfers, duplications, and losses were the main content modifying events of tRNA gene orders [WWD06]. Interestingly, the authors discovered that the tRNA backbone, which is a set of conserved tRNA genes across multiple genomes, is a strong regulator of what genes are utilized during protein synthesis [WWD06].

2.15.2 Evolution of Transfer RNA Genes in Drosophila

In another study, Rogers *et al.* [RBG10] studied the evolution of tRNA gene orders by performing an analysis on 12 Drosophila genomes. Here the authors focused on identifying orthologous tRNA genes by performing an analysis of the flanking regions in the tRNA genes [RBG10]. The flanking regions consist of the promoter and protein binding sites required for transcription. However, flanking regions are not transcribed

into RNA. In general, mutations in these regions result in the gene no longer being functional, therefore these regions are generally conserved [RBG10]. For this reason, the authors used these flanking regions to identify orthologous tRNA genes. Two tRNA genes exhibiting similarity in the flanking region is a strong indicator of these genes being orthologous [RBG10]. Each of the flanking regions in the Drosophila genomes were mapped to flanking regions in *D. melanogaster*. A unique mapping was a strong indicator of an ortholog and multiple mappings were an indication of duplication events. Again, the authors found that 110 tRNA genes were conserved across 11 of the 12 Drosophila genomes indicating a core set of tRNA genes regulating protein synthesis [RBG10].

2.16 Summary

In this chapter we introduced several concepts to better understand the biological aspect of this M.Sc. thesis. We started out by identifying the various components of an operon. An operon is a cluster of genes in close proximity to each other under the control of a single promoter. We went into further detail by describing what a gene is. A gene is a sequence of nucleotides connected by covalent bonds. We also learned that the genome comprises all of the genetic material (coding and non-coding). Additionally, we identified the functional role of gene operons. As we have learned in this chapter, the primary purpose of an operon is to synthesize proteins. Proteins are required by all living organisms in order to function properly. We went on to describe the entire protein synthesis process. We learned that operons first have to be transcribed into a mRNA molecule and then the mRNA molecule is translated in order to construct the amino acid chain which becomes the protein molecule.

We went on to identify the concepts we will be addressing in this M.Sc. thesis. We described what are homologous sequences and further noted that homologous sequences can fall into two general categories, orthologs and paralogs. We stated that orthologous operons were present in the ancestral genome prior to the speciation event whereas paralogous operons are the result of a duplication event after the speciation event. It is important to be able to make the distinction between the two. We are

primarily interested in identifying orthologous operons. The reason being that orthologous operons were present in the original ancestral genome and allow us to infer an evolutionary history for these operons. We want to be able to apply our algorithm in a phylogenetic context and in order to do that we had to discuss what a phylogenetic tree is. We learned that a phylogenetic tree is a visual representation of the evolutionary history and relationships between various organisms. Phylogenetic trees are one of the fundamental components required by our algorithm. The reason being that a phylogenetic tree informs us in what order to compare the various genomes in. The order in which the comparisons are performed will affect the ancestral genome reconstructed.

Finally we conclude the chapter by discussing previous studies that suggest how operons and genomes gradually evolve over time. We learned that gene duplication was the primary mechanism for elongating a genome and for creating a new metabolic pathway. Additionally, we learned that changes which render proteins non-functional are not favoured and are generally avoided. Generally changes to the genomes are slow and minimal when it comes to genes required for survival. Additionally, we learned from studies researching tRNA genes in *Escherichia coli* and Drosophila that these genes are very well preserved across several species even after speciation. Interestingly, we also learned that there appears to be a core of tRNA genes that are preserved among all organisms within a given phylogeny.

In any research project having the background knowledge surrounding the field of study is always beneficial as it allows us, as a researcher, to understand the scope of the problem, whether the goals are realistic or not, and to be able to interpret the results, etc. At the same time it allows us to make decisions when designing the algorithm and the experiment. As in most cases, the data being analyzed has a set of constraints. By learning about the data's background we can identify what those constraints are. In this M.Sc. thesis we are analyzing operons containing tRNA and rRNA genes and it is important to understand what an operon is, what is its role and function in living organisms, know the difference between orthologous and paralogous operons, the scope in which we want to apply our algorithm, how these operons change over time, identify the possible events that could have contributed to

their changes over time, and what events did not.

Chapter 3

Related Works

This chapter will introduce fundamental methods and algorithms that have inspired BOPAL. Our goal is to introduce the reader to these methods so they will understand the inspiration behind BOPAL. The purpose of this chapter is to introduce and familiarize the reader with all of the required background knowledge to fully understand the method used in this thesis.

3.1 Pairwise Alignments

In comparative genomics we generally want to be able to compare two sequences to infer differences and similarities between them. These sequences could include, DNA sequences, RNA sequences, protein sequences, etc. In order to compare two sequences we need to perform an alignment. An alignment can be described as an arrangement of two sequences based on a match mismatch scheme. The comparison of two sequences is denoted as a pairwise alignment. We generally use alignments to identify structural similarities, functional similarities, and common evolutionary characteristics such as deletions or duplications between sequences. To perform a pairwise alignment we generally use one of the two following techniques, global alignment or local alignment. We will go into more detail about the global alignment and local alignment in the following sections.

3.1.1 Needleman-Wunsch Algorithm

The Needleman-Wunsch algorithm from 1970, also known as the global alignment technique, is used commonly to compare two sequences of approximately the same length [HC03]. Generally a marker can refer to either a nucleotide, amino acid, or even a gene label. The main function of the technique is to pair all of the markers to construct an alignment that is approximately just as long as the sequences being compared. Sequences with significant differences will result in large gaps in the alignment. The Needleman-Wunsch algorithm is best suited for sequences of roughly equal length with relatively low divergence. The Needle-Wunsch algorithm is a two step method.

1. We compute a score for the sequences being compared by constructing a matrix. The two sequences being compared are inserted into the matrix, one horizontally at the top row and one vertically on the far left column. We traverse the matrix row by row, from left to right, computing a score for each cell in the matrix. The markers in the sequences may either match, mismatch, or be mapped to a gap which can be interpreted as a deletion in the sequence with the gap or an insertion in the sequence with the marker. In general, matches are rewarded whereas mismatches and gaps are penalized. When computing the score of a cell there are three potential paths and we select the path with the highest score. The top and left paths represent a gap and we apply the penalty for a gap in each of these paths. The third path is the diagonal representing either a match or mismatch. If the markers match, then we add the score for a match. If the markers mismatch, then we add the score for a mismatch. We select the path with the highest score and repeat this process until the matrix is completely filled. The number in the bottom right most cell represents the score of the best alignment.

2. We construct the alignment sequence by tracing a path from the bottom right cell to the top left cell of the matrix. The path is computed by determining which operation generated the score in the current cell. A diagonal path rep-

		T	C	G	C	A
T						
C						
C						
A						

- Construct a matrix. Insert one sequence at the top most row and the other sequence in the left most column.

Figure 3.1: Global Alignment - Matrix Construction

		T	C	G	C	A
	0	-1	-2	-3	-4	-5
T	-1					
C	-2					
C	-3					
A	-4					

- Initialize the top row and left column. The scores are computed by incrementing the values by the gap penalty from left to right and top to bottom.

Figure 3.2: Global Alignment - Matrix Initialization

resents either a match or mismatch. A horizontal path represents a gap in the left sequence and a vertical path represents a gap in the top sequence. The resulting path in the matrix represents the alignment sequence. However, there

		T	C	G	C	A
	0	-1	-2	-3	-4	-5
T	-1	1	0	-1	-2	-3
C	-2	0	2	1	0	-1
C	-3	-1	1	1	2	1
A	-4	-2	0	0	1	3

- Fill in the matrix. A path from the top left represents a match or mismatch. A path from the top or left represents a gap. We select the path resulting in the highest score. As a result, the score of this alignment is 3.

Figure 3.3: Global Alignment - Score Computation

		T	C	G	C	A
	0	-1	-2	-3	-4	-5
T	-1	1	0	-1	-2	-3
C	-2	0	2	1	0	-1
C	-3	-1	1	1	2	1
A	-4	-2	0	0	1	3

- To compute the alignment we have to construct a path from the bottom right cell to the top left cell such that a diagonal arrow represents a mismatch or a match and a leftwards or upwards arrow represents a gap. We use the current cell's score and the two letters to determine which neighboring cell is the correct path.

Figure 3.4: Global Alignment - Traceback

could potentially be multiple paths in the matrix which represent equally viable sequences. Figures 3.1, 3.2, 3.3, 3.4, 3.5 demonstrate a simple global alignment example with a scoring schema of +1 for matches, -1 for mismatches or gaps.

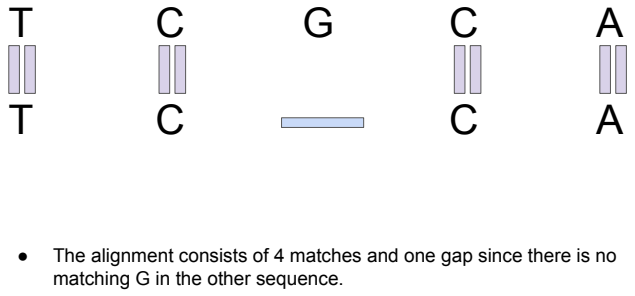


Figure 3.5: Global Alignment - Alignment

3.1.2 Smith-Waterman Algorithm

The Smith-Waterman algorithm from 1981, also known as the local alignment technique, is generally used for comparing sequences with significantly different lengths [SW92]. The main function of this technique is to identify regions of similarity and to ignore regions that are distinct. This technique is well suited for comparing sequences where one of the sequences has become more divergent than the other or when one sequence is longer than the other. This could include a number of duplications or deletions in one of the sequences. However, if there is a region that has been retained in both sequences, then the Smith-Waterman algorithm will be able to identify it. The Smith-Waterman algorithm is a two step process similar to the Needleman-Wunsch algorithm in Section 3.1.1. However, there are some key differences between the two algorithms. In the first step we construct a score matrix by inserting one of the sequences in the top row of the matrix and inserting the other sequence into the far left column of the matrix just like in the Needleman-Wunsch algorithm. Again, the markers may match, mismatch, or be mapped to a gap and we select the operation that results in the highest score. However, if the operation

results in a negative score, then we select a score of 0 instead of the negative score. This is one of the key differences between the Needleman-Wunsch algorithm and the Smith-Waterman algorithm. The Smith-Waterman algorithm does not permit negative scores in the matrix. Once again, we repeat this process until the matrix has been completely filled. In the second step we construct the alignment by tracing through the matrix, however, we do not necessary have to start at the bottom right and finish at the top left of the matrix. This is the second key difference between the Smith-Waterman algorithm and the Needleman-Wunsch algorithm. In the Smith-Waterman algorithm the starting point will be the cell with the highest score. We continue traversing the matrix by determining which operation generated the score just like in the Needleman-Wunsch algorithm, however, we stop when we encounter a 0 in the path. The resulting path represents a valid local alignment. However, there could potentially be multiple paths in the matrix which are equally viable just like in the Needlman-Wunsch algorithm. Figures 3.6, 3.7, 3.8, 3.9, 3.10 demonstrate a simple local alignment example with a scoring schema of +1 for matches, -1 for mismatches, and a -1 for gaps.

		G	G	C	A	T	G	G
C								
A								
T								

- Construct a matrix. Insert one sequence at the top most row and the other sequence in the left most column.

Figure 3.6: Local Alignment - Matrix Construction

		G	G	C	A	T	G	G
	0	0	0	0	0	0	0	0
C	0							
A	0							
T	0							

- Initialize the top row and left column. The scores are computed by incrementing the previous cell value by the gap penalty from left to right and top to bottom. However, the Smith-Waterman algorithm does not allow negative values into the matrix. Therefore, the top row and left column will be initialized to zeros.

Figure 3.7: Local Alignment - Matrix Initialization

		G	G	C	A	T	G	G
	0	0	0	0	0	0	0	0
C	0	0	0	1	0	0	0	0
A	0	0	0	0	2	1	0	0
T	0	0	0	0	1	3	2	1

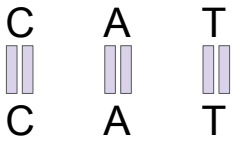
- Fill in the matrix. The current cell`s score can come from one of 4 possibilities. The score can come from the top or left which represents a gap in one of the sequences. The score can also come from the top left if the letters match or mismatch. We select the path that yields the highest score. If the highest score is a negative value, then zero is entered into the cell. Additionally, the location of the highest score must be tracked in order to construct the alignment. The highest score may not necessarily be the bottom left corner of the matrix. The highest score of this alignment is 3.

Figure 3.8: Local Alignment - Score Computation

		G	G	C	A	T	G	G
	0	0	0	0	0	0	0	0
C	0	0	0	1	0	0	0	0
A	0	0	0	0	2	1	0	0
T	0	0	0	0	1	3	2	1

- In order to construct the alignment we have to start at the cell containing the highest score. If there are multiple cells with the same high score then all of those alignments are equally valid. We traverse through the matrix constructing the alignment such that a diagonal arrow represents a match or mismatch and a leftwards or upwards arrow represents a gap in one of the sequences. We stop traversing the matrix when a zero is encountered along the pathway. We use the current cell's score and the two letters to determine which neighboring cell is the correct path.

Figure 3.9: Local Alignment - Traceback



- Based on the results of the local alignment matrix we can see that the local alignment sequence is C, A, T. Local alignments are useful for identifying regions of similarity when sequences are not necessarily similar in terms of sequence size.

Figure 3.10: Local Alignment - Alignment

3.2 Multiple Sequence Alignment

In comparative genomics occasionally it may be necessary to compare many sequences. In order to compare more than two sequences we would perform a multiple

sequence alignment. Recall that in a pairwise alignment we are computing an alignment using one comparison. In a multiple sequence alignment we are computing an alignment using $n - 1$ comparisons where n is the number of sequences. For example, say we have 10 sequences. Initially we would select two sequences and compute an alignment. Then we would select another sequence from the list of 8 remaining sequences and perform another alignment. However, we would be computing an alignment using the selected sequence and the computed alignment from the previous step. We would repeat this process 7 more times until all of the sequences have been exhausted. At the end of the last alignment computation we would have an alignment representing all 10 sequences.

3.2.1 Sequence Searching in Databases

In order to make biological data manageable researchers started storing sequence data into databases. This included DNA sequences, RNA sequences, protein sequences, etc. As the amount of sequence data accumulated analyzing the data became problematic. Researchers wanted a mechanism to compare a sequence with a large dataset of sequences. Initially, when the databases were small using the global alignment and local alignment techniques was a feasible solution. However, as the amount of sequence data increased in the databases it became a very time consuming process to compute every single alignment. Recall that the global alignment and local alignment construct a matrix for the two sequences, compute an overall score, and construct an alignment by tracing through the matrix. This is a very expensive process in terms of computing power. Therefore researchers needed an alternative solution in order to compare one sequence with an entire database of sequences.

To reduce the amount of time spent computing alignments on a large dataset, in 1988 David J. Lipman and William R. Pearson developed a software package called FASTA. FASTA was one the first algorithms to efficiently and quickly compare a sequence against an entire database of sequences. FASTA was originally designed for the purpose of comparing protein sequences, however, it was later extended to compare DNA sequences and DNA sequences with protein sequences. In 1990, Altschul

developed Basic Local Alignment Search Tool (BLAST) for a similar purpose. BLAST was an improvement over FASTA in terms of search speed, ease of use and statistical rigor. The basic idea that makes FASTA and BLAST faster compared to the global alignment and local alignment is that FASTA and BLAST identify exact matches in short stretches rather than performing an entire sequence comparison. The main idea is that good alignments contain short sequences of exact matches. Initially FASTA and BLAST identify very short exact matches between the sequences. After identifying these short stretches of exact matches, these techniques extend these short matches into longer regions of similarity. After the regions can no longer be extended they are optimized to construct an alignment.

3.2.2 FASTA

FASTA is known as the dot plot algorithm since a dot plot is constructed to identify regions of similarity between the sequences being compared [DDA14]. The two sequences being compared by FASTA are denoted as the query sequence and the test sequence. The query sequence is the sequence the user provides as input and the test sequence is the sequence from the database. FASTA identifies whether the two sequences are potentially a good match by comparing the words between the query sequence and the test sequence. A word is a substring of a given sequence. FASTA divides the query sequence and the test sequence into words. By comparing the words between the query and the test sequence, FASTA users this as an indicator whether the two sequences are potentially good matches. If there are many words with exact matches, this indicates the sequences are similar. If there are not many exact matches, then this indicates that the sequences being compared are not a good match and should not be processed any further. The length of the words is different depending on the sequence type. Generally for DNA sequences a word consists of 6 nucleotides and for protein sequences a word consists of 2 amino acids. Recall that a set of 3 nucleotides forms a codon. A codon represents an amino acid and a set of codons represent an amino acid sequence, *i.e.*, a protein sequence. In order to keep the comparisons consistent, FASTA divides the protein sequence into words which

consist of 2 amino acids which is 6 nucleotides since 1 amino acid is 1 codon which consists of 3 nucleotides.

The FASTA algorithm performs a series of 5 steps in order to compare the query sequence and the test sequence.

- In the first step FASTA identifies the similarities between the query sequence and test sequence. FASTA constructs a dot plot in order to compare the sequences. The dot plot maps the regions of similarity between the two sequences. If we have a "dot" in the cell i, j, then this means that the nucleotide at position i in the query sequence matches the nucleotide at position j in the test sequence. The entire matrix is traversed and filled in appropriately. See Figure 3.11 for an example of this step.

	C	C	A	T	C	G	C	C	A	T	C	G
G						*						*
C	*	*			*		*	*			*	
A			*						*			
T				*						*		
C	*	*			*		*	*			*	
G						*						*
G						*						*
C	*	*			*		*	*			*	

- Initialize the matrix. The query sequence is inserted into the left column and the test query is inserted into the top row. Insert a marker into the cell if the nucleotides at (i, j) match.

Figure 3.11: FASTA - Initialize the Dot Plot

- In the second step, FASTA analyzes the matrix constructed from the previous step. In this analysis FASTA identifies all of the dots in the matrix that form a diagonal. The diagonals cannot have gaps between them, therefore the dots must be consecutive when forming the diagonal. Additionally, there could be more than one diagonal. There could potentially be many diagonals in the ma-

trix. All of these diagonals may not necessarily be part of the optimal alignment. In order to filter out the diagonals that are not significant, FASTA applies a threshold where the diagonals must be a certain length otherwise they are discarded. The minimum length threshold is represented by the k value. Any diagonals shorter than k units are discarded. Generally, FASTA uses a k value of 2. In addition, FASTA also computes the offset for each of the diagonals in the matrix. See Figure 3.12 for a visual example.

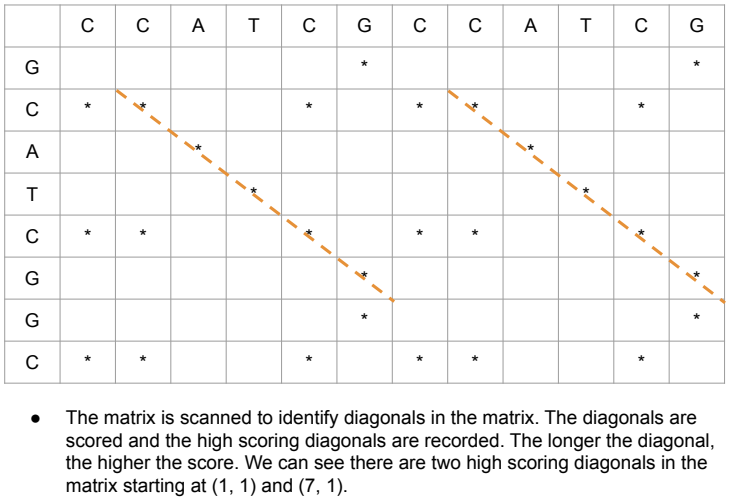
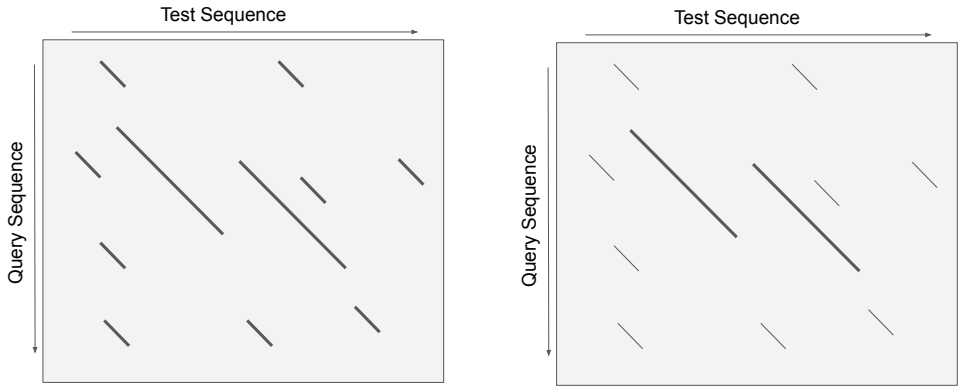


Figure 3.12: FASTA - Score the Diagonals

- In the third step FASTA analyzes the quality of each diagonal identified in the previous step. With many diagonals it is difficult to compute the optimal alignment, therefore we have to filter out the diagonals not part of the optimal alignment. FASTA does this by computing a score for each of the diagonals. Diagonals that score high are recorded and the rest are discarded. FASTA uses a substitution matrix called the PAM matrix to compute the score for each of the diagonals. See Figure 3.13 for a visual example.



- The image on the left represents all of the diagonals identified with a length greater than or equal to 2. The image on the right represents the same diagonals after re-scoring with the PAM matrix. There are two high scoring diagonals marked as thicker lines. These diagonals are recorded and the remaining diagonals are discarded.

Figure 3.13: FASTA - Re-score the Diagonals

- In the fourth step all of the remaining diagonals from the previous step are joined together using a gap. A matrix can potentially still have many diagonals. If there are many diagonals, it is possible that the diagonals can be joined in various combinations. The purpose of this step is to join as many diagonals as possible to construct a path from the top left to the bottom right of the matrix. The only requirement is that the diagonals do not overlap each other. Since the diagonals can not overlap each other, there could potentially be many combinations of diagonals leading from the top left to the bottom right of the matrix. However, we want to maximize the total diagonal length for the path and we want to minimize the amount of gaps we use to join these diagonals. In order to identify which paths are better than others, FASTA computes a score for each path traversing from the top left to the bottom right. The score for a path is computed by summing the PAM matrix scores for the diagonals used in the path and applying a penalty for each gap in the path. The score for each path is stored and we compute the score for each path in the matrix. Once we've scored every possible path traversing from the top left to the bottom right of the matrix, the path that resulted in the highest score is the path that is

selected. If there are multiple paths that result in the same score, then all of these paths are equally viable. However, FASTA will select the path with the lowest offset to be the optimal path in the matrix.

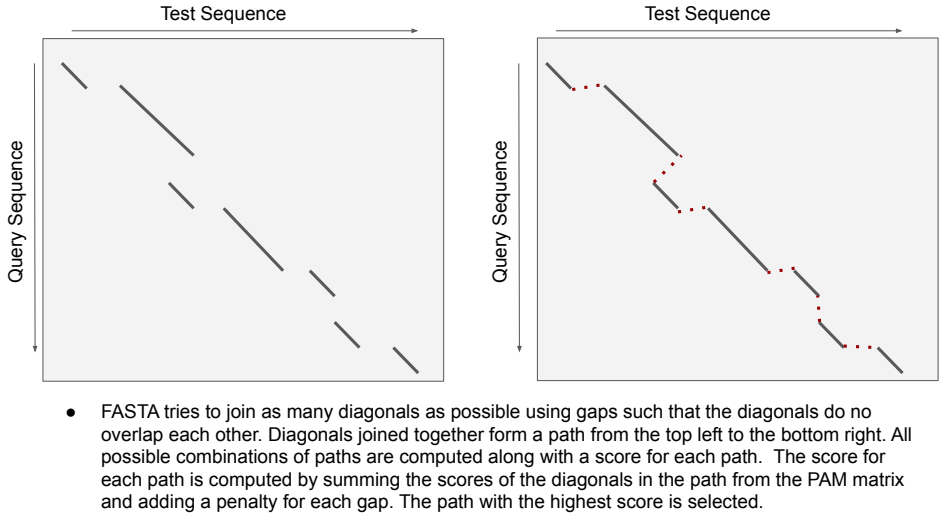


Figure 3.14: FASTA - Join Diagonals Using Gaps

- In the fifth and final step, FASTA takes the path selected from the previous step and uses this path to construct an alignment. Each path from the previous step represents one potential alignment and the scores from the previous step were used to help identify the best alignment. At this point FASTA has identified the optimal alignment.

To assess and analyze the significance of the alignment produced by FASTA we have to look at the z score and the e value returned by the algorithm [DDA14]. To compute these values, FASTA generates random alignments and calculates their scores. Those scores are used to compute a mean and a standard deviation from these random scores. The z score represents the deviation of the actual score from the mean. The z value is a combination of the mean and the standard deviation of

the random scores.

$$z = \frac{Mean}{StandardDeviation} \tag{3.1}$$

The probability of a z score is called the e value. A high deviation will result in a high z score therefore the e value will also be high. This generally indicates the alignment is not significant. A low deviation will result in a low z score therefore the e value will also be low. This generally indicates the alignment is significant. The general rule to follow is,

- e values that are below 10^{-6} are statistically significant
- e values that are above 10^{-6} but below 10^{-3} might be significant and require further analysis
- e values that are above 10^{-3} are not significant.

3.2.3 BLAST

BLAST or the Basic Local Alignment Search Tool was originally published in 1990. It is a heuristic method for the local alignment and it is specifically designed for searching through a sequence database [DDA14]. BLAST was designed on the same concept as FASTA that good alignments contain short stretches of exact matches. However, BLAST's and FASTA's algorithms and statistical approaches are different. BLAST however is advantageous over FASTA in terms of speed, providing a user friendly interface, statistical rigor, and sensitivity. In BLAST the input consists of the query sequence, a database of sequences, and an s score. BLAST uses the s score as a threshold when determining whether an alignment is significant or not. If the score for the alignment is below the s score, then BLAST will not mark the alignment as significant. The output of BLAST consists of the sequences from the database that were marked as significant, a z score, and an e value. There are some similarities between the BLAST and FASTA algorithms since BLAST was developed based on FASTA. Both BLAST and FASTA break long sequences into substrings called words.

However, the length of the words in BLAST is different compared to FASTA. In BLAST amino acid sequences are broken down into substrings of 3 amino acids and DNA sequences are broken down into substrings of 11 nucleotides whereas in FASTA each are 2 amino acids long and 6 nucleotides long respectively. Additionally, BLAST does not require word matches between the sequences to be exact matches whereas in FASTA the words have to be exact matches. However, FASTA handles gaps and short sequences a lot better compared to BLAST. There have been modifications to BLAST over the years to handle gaps and short sequences but we will be focusing on BLAST originally published in 1990.

Unlike FASTA, there are 3 main steps to the BLAST algorithm. In the first step BLAST takes the query sequence just like in FASTA and breaks the sequence into substrings called words. The words are length w depending on the sequence type. A word for amino acids is three amino acids long. A word for nucleotides is 11 nucleotides long. The number of words that can be generated from a sequence is given by the following formula: $NumberOfWords = L - w + 1$ where L is the length of the sequence in terms or nucleotides or amino acids, w is the length of of the word. For example, for a DNA sequence 130 nucleotides long, the maximum number of words that can generated from the sequence is 120. In addition, the neighborhood of these words is calculated expanding the number of words that are going to be searched in the database. So even though we look for exact matches in the database, the words we look for are not all present exactly in the query. This is an indirect way of looking for inexact matches. In the second step the list of words generated from the previous step are used to find matches in the sequence database. However, in this step the word matches between the query sequence and the database sequence must be exact. In the third step after BLAST finds all of the words with exact matches between the query sequence and database sequence, BLAST tries to extend these regions. When there are multiple words with exact matches, BLAST selects one of these words to act as an anchor to extend the region. The region extended represents one possible alignment. These alignments are scored based on the number of matches, mismatches, and gaps. If the score is above the s score, then the alignment is marked as significant. In addition to the alignment score, a z score and e value is also

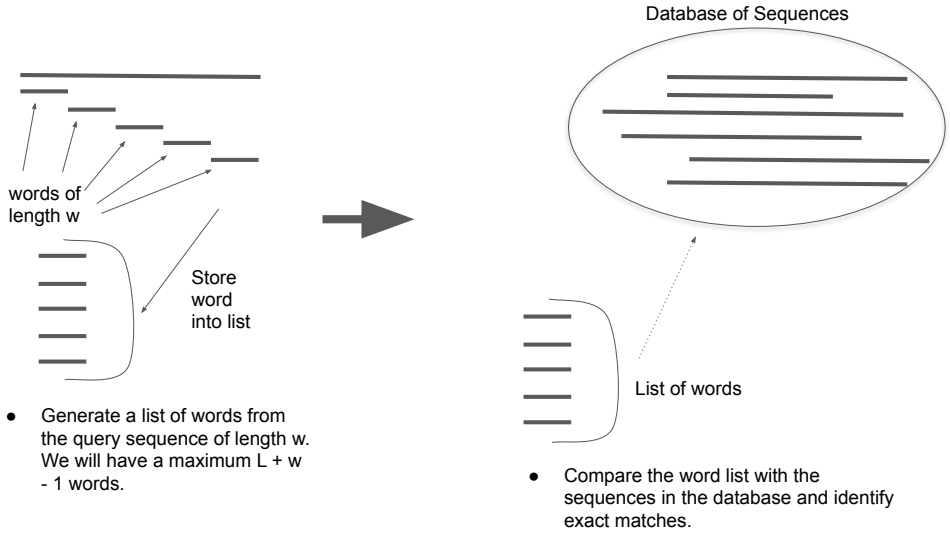


Figure 3.15: BLAST - Generate Words and Scan the Database

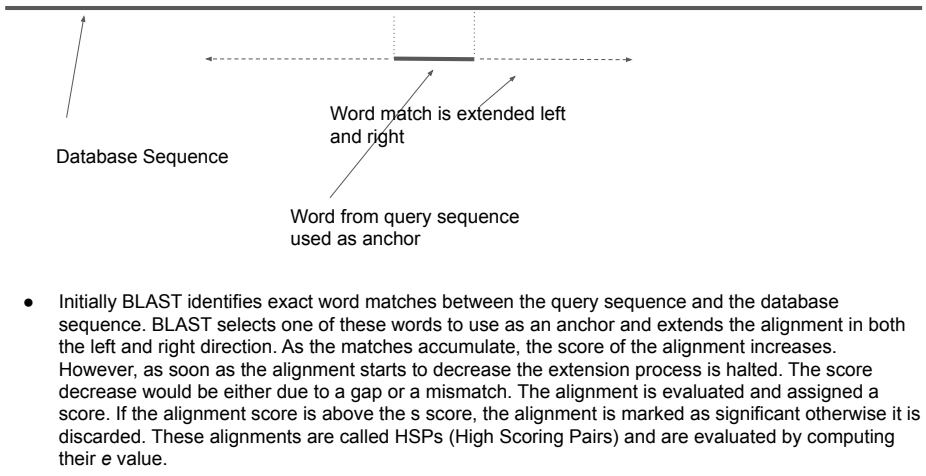


Figure 3.16: BLAST - Construct the Alignment

computed. We use the e value to assess the significance of the results. An e value of $10^{-}13$ or below indicates that the alignment is significant compared to FASTA where an e value of $10^{-}6$ indicated an alignment is significant. The regions between two

anchors that score above the s score are called High Scoring Segment Pairs or HSPs. BLAST generally computes several short HSPs rather than one long aligned region. Figures 3.15 and 3.16 briefly outline BLAST.

3.3 Orthology Inference

3.3.1 OrthoMCL

In 2003, Li *et al.* [LSR03] published orthoMCL, which is graph clustering algorithm designed to identify homologous sequences based on sequence similarity and distinguish whether these homologs are orthologs or paralogs. It was designed for the purpose to study the functional conservation of proteins between different organisms in addition to identifying ancestry and evolutionary conservation of sequences between organisms. Recall that orthologous genes are the result of a speciation event and paralogous genes are the result of a duplication event. Paralogs can be categorized into 2 different subcategories, in-paralogs and out-paralogs. An in-paralog gene is the result of a duplication event, however, the duplication event occurred after the speciation event whereas an out-paralog's duplication event occurs prior to the speciation event [EKM13].

OrthoMCL processes the data using the following sequence of steps. In the first step orthoMCL uses BLAST to compare all of the sequences together to compute a score and identify regions of similarity between each sequence. Using the comparisons from the previous step, in the second step orthoMCL implements the inparanoid algorithm to distinguish between orthologous and paralogous sequences. The inparanoid algorithm consists of constructing a graph. In the graph, vertices represent a sequence and the edge between two vertices represents a measure of similarity between the sequences. The edge's weight will be the score computed by BLAST when comparing the sequences. Recall in-paralogs are duplications within the same genome, therefore, if orthoMCL is able to find a better match for a sequence within the same genome rather than with the genome being compared, then orthoMCL will mark this sequence as an in-paralog. After the graph is constructed, in the third step orthoMCL reduces

the number of edges in the cluster graph by implementing the Markov Clustering Algorithm to eliminate edges that are not significant. The remaining edges and output after the Markov Clustering algorithm processes the graph are clustered groups of orthologs and recent paralogs [EKM13].

3.3.2 OrthAgogue

Researchers studying homologous sequences using orthoMCL to perform their analysis soon realized as the amount of data accumulated, orthoMCL was increasingly taking longer to process the data. As an example, it took orthoMCL days to process 200 proteomes [EKM13]. Ekseth *et al.* [EKM13] performed an extensive analysis of the orthoMCL algorithm. They discovered two primary bottlenecks in the algorithm. The first bottleneck was located at the point where orthoMCL uses BLAST to compute the similarities between sequences. However, Ekseth *et al.* [EKM13] proposed that this bottleneck can be mitigated by running the BLAST analysis on a larger computer cluster. The second bottleneck was located at the point where orthoMCL performs an analysis on the similarities using the inparanoid algorithm. The authors concluded that this bottleneck would have to be resolved by re-implementing this section of orthoMCL's code since this component was designed as a serial process. A serial process refers to a method that processes one item at a time.

In 2013, Ekseth *et al.* [EKM13] also published orthAgogue, which is an optimized solution that mitigated the bottleneck caused by the inparanoid implementation in orthoMCL. When designing orthAgogue, the authors considered two factors when optimizing the inparanoid algorithm. The first factor was to make the parsing of data more efficient. The second factor was to optimize memory usage. In order to achieve these goals, the authors utilized several C programming libraries to handle a lot of these optimizations internally within the code. The authors re-implemented the inparanoid algorithm utilizing the Threading Building Blocks library and the Message Passing Interface in order for data to be processed in parallel. Additionally, the authors utilized the C Minimal Perfect Hashing library in order ensure that memory usage was being optimized within the algorithm. It took orthoMCL 33.5 hours to

process 147 proteomes whereas orthAgogue processed the same data in 10 minutes. Overall, orthAgogue was 200 times faster than orthoMCL without reducing accuracy.

3.4 Reconstruction of Ancestral Genomes

In the following subsections we will briefly introduce four methods that infer ancestral gene orders using different techniques.

3.4.1 InferCARs

InferCARs [MZS+06] assigns each marker in a genome a unique identifying, signed integer. InferCARs assumes there are no duplicate markers in the genomes being compared. Markers on the genome have a predecessor and successor [MZS+06]. In the first step, InferCARs generates a predecessor graph for each node in the species tree in a bottom up fashion [MZS+06]. A path in the predecessor graph represents a chromosome from the genome. This process is repeated recursively until the predecessor graph is computed for the root node of the species tree. After the predecessor graph is computed for each of the nodes in the species tree, InferCARs computes a secondary modified predecessor graph for each node starting at the root node. The secondary modified predecessor graph is computed by taking the intersection of the predecessor graphs of the two direct descendents of the ancestor. This process is repeated until the target ancestor's secondary predecessor graph is computed. In the second step, InferCARs generates a successor graph for each node in the species tree using the same methodology as the predecessor graphs. Once the successor graph has been computed for the target ancestral node, InferCARs takes the intersection of the predecessor and successor graph such that the edges are maximized and this new graph represents the ancestral genome [MZS+06].

3.4.2 Anges

Anges [JRTC12] proceeds in a two step process where in the first step we compute the Ancestral Contiguous Sets which are sets of markers that are assumed to have been

present in the ancestor. These sets are computed from an informative pair of genomes. A pair of genomes is considered informative by Anges if the two genomes being compared both have an evolutionary history path that intersects with the ancestral node in the species tree. In the second step Anges organizes the Ancestral Contiguous Sets to form Ancestral Contiguous Regions which are organized into an ancestral genome [JRTC12]. To assemble the Ancestral Contiguous Sets into an ancestral genome, the Ancestral Contiguous Sets are organized into a binary matrix where consecutive ones in a row indicate a Contiguous Ancestral Region [JRTC12]. If the sets do not satisfy the consecutive ones property, then they are prioritized by using a computed weight and are assembled into Contiguous Ancestral Regions. If any of the sets cause a conflict during the assembly process, then they are discarded. Additionally, Anges uses PQ trees which represent a potential organization of the markers in the ancestral genome [JRTC12].

3.4.3 Gap Adjacency

GapAdj [GBE12] describes a genome as a sequence of paired markers within a genome. Additionally, GapAdj tracks the multiplicity of each gene in a genome [GBE12]. If the multiplicity of a gene increases from an ancestor to a descendant, then GapAdj assumes that a whole genome duplication event was responsible for the multiplicity increase as previous studies have shown that whole genome duplication events were responsible for multiplicity increases in eukaryotic species [GBE12]. In the first step, GapAdj computes the left and right multisets for each gene in all of the extant genomes [GBE12]. GapAdj traverses in a bottom up fashion in the species tree computing these multisets for each gene for each internal node [GBE12]. In the second step GapAdj attempts to assemble these multisets into an ancestral genome. Initially, GapAdj constructs a complete graph where each node represents a gene and the edge between the nodes represents a weight computed from the previous step [GBE12]. GapAdj then computes a Hamiltonian cycle, of maximum weight, in the graph using the Chained Lin-Khernigan heuristic [GBE12]. The Hamiltonian cycle constructed represents an ancestral genome [GBE12].

3.4.4 ProCARS

ProCARs [PVBO15] is a homology based method that works iteratively on both linear and circular genomes to detect and assemble features into Contiguous Ancestral Regions or CARs while allowing micro rearrangements at the extremities of segments [PVBO15]. In the first step ProCARs computes the divergence for each adjacency to classify them into one of the three following groups; fully conserved adjacency, partly conserved adjacency, or a non-conserved adjacency [PVBO15]. After classifying each of the adjacencies the non-conflicting adjacencies are selected and assembled into CARs. In the second step ProCARs analyzes the conflicting adjacencies and attempts to resolve them. If the conflicts for an adjacency are resolved, then it will be assembled into a CAR. If ProCARs is unable to resolve the conflict then the adjacency will be discarded. Finally, in the third step ProCARs attempts to find additional adjacencies that are not conserved but are supported due to micro-rearrangements. If the adjacency is non-conflicting then it will be assembled into a CAR otherwise it will be discarded [PVBO15]. In their analysis, the authors concluded that ProCARs detected most of the CARs detected by Anges, InferCARs, and GapAdj and it constructs a completely resolved set of CARs [PVBO15].

3.5 Inference of Evolutionary Histories of Transfer RNA Genes

3.5.1 Integer Linear Programming Algorithm

In 2012, a new method was developed to infer evolutionary histories of rRNA and tRNA genes that was based on a gene order alignment approach and considered duplication and loss events [HSAE13]. The alignment of the gene orders was used to identify orthology relationships between the genes (since there are multiple copies of each type of rRNA and tRNA genes), instead of using traditional methods for identifying gene orthologies from sequence information. The rationale for using this approach is that tRNA genes especially are very short (maximum length is around 90

nucleotides), and the sequences are highly conserved [WWD06]. As a consequence, there is simply not enough signal in the sequences themselves to identify orthology relationships. This alignment problem was then shown to be NP-hard for the duplication and losses model of evolution [ARC13, BDE13, DE13]. The exact algorithm proposed in [HSAE13], based on integer linear programming (ILP), was designed to solve the 2-Small Phylogeny Problem (2-SPP), which is to find a common ancestor A of two gene orders X and Y that minimizes the number of events on each of the two branches. Andreotti *et al.* [ARC13] then proposed a faster and more efficient linear programming algorithm for the duplication-loss model, and generalized it to the median of three genomes setting.

3.5.2 OrthoAlign and MultiOrthoAlign

Researchers studying the evolution of tRNA Repertoires in Bacillus in 2015 published OrthoAlign, which is a pairwise alignment approach that is used in a phylogenetic framework to infer the evolutionary history of transfer RNA within bacteria genomes. The evolutionary model of OrthoAlign was restricted to the following operations; duplications, losses, substitutions, inversions and inverted duplications [TBLE15]. Tremblay-Savard *et al.* [TBLE15] primarily used the algorithm to analyze bacterial genomes from the Bacillus genus. OrthoAlign uses the species tree and genomes annotated with transfer RNAs as input. The species tree is very important information since it describes the evolutionary history between the bacterial genomes. OrthoAlign uses species tree to determine which bacterial genomes are siblings, or in other words, which bacterial genomes have a recent common ancestor. Using all of this information OrthoAlign constructs the output which consists of an inferred gene order for each of the internal nodes in the species tree. This gene order is assumed to have been present in the ancestral genome. In addition to the gene order, OrthoAlign also infers a sequence of evolutionary events that must have occurred through out the course of the genome's evolutionary history in order to transform the ancestral genome into the genome at the leaf of the species tree.

Both OrthoAlign [TBLE15] and multiOrthoAlign [BE14] were developed to gen-

eralize the evolutionary model to account for rearrangements (inversions and transpositions) in addition to duplications and losses. The idea there was to use dynamic programming to align the rRNA and tRNA gene orders and identify orthologs, and then explain the mismatches and gaps in the alignment by inferring rearrangement events (inversions and transpositions) and content-modifying events (duplications and losses). While OrthoAlign was designed for pairwise comparisons between gene orders (2-SPP), multiOrthoAlign was created to compare a full set of gene orders related through a phylogenetic tree by taking initial ancestral assignments (inferred by OrthoAlign or another method) and improving them using a heuristic for the median of three problem.

OrthoAlign and multiOrthoAlign are well adapted to the study of bacterial genera which have a relatively low amount of divergence between the genomes. However, just like with traditional sequence alignment, when the gene orders being compared are not very well conserved, it quickly becomes difficult to correctly identify matches (which correspond to orthologous genes in this case). Moreover, existing methods do not consider the physical proximity of the genes in their inference of events, which might lead to evolutionary histories that are not necessarily realistic — inferring events on blocks of genes that are contiguous in terms of gene order but not necessarily close to each other on the chromosome for example.

3.6 Related Studies on Bacterial Genome, Operon, and Transfer RNA Gene Evolution

It was observed in [TBLE15] that duplications can either insert the copied genes inside or outside other operons, thus extending pre-existing operons or creating new ones. Also, rearrangements (inversions and transpositions) do not seem to break operons into separate parts. In the study of 50 *Bacillus* genomes, the inferred rearrangements always affected entire operons and not just a part of them [TBLE15]. Although these constraints on rearrangements were observed in a study of the *Bacillus* genus specifically, we assume that they can be generalized to other bacteria, since

a rearrangement affecting only part of an operon would most likely leave one part of it without a promoter. Several methodologies have been proposed to find operons in microbial genomes, which are based on several different genomic features like intergenic distances [TECM18], metabolic pathways [ZSF+02], expression profiles [PHAA05], phylogenetic information [BPHQ07], etc.

Multiple sites in tRNA sequences, extending beyond the anticodon region, are responsible for their recognition by the aminoacyl-tRNA synthetases, which charge the tRNA molecules with the appropriate amino acid [GSF98]. Mutations in these identity elements can sometimes change the identity class of tRNA genes [SSA98, LL05], which can be viewed as a substitution to a different tRNA gene.

3.7 Summary

In this chapter we presented the concept of sequence alignment. Initially we introduced two very common sequence comparison algorithms, the global alignment and the local alignment. These two algorithms compute a score for the sequences being compared and an alignment. However, as the amount of sequence data accumulated, it was discovered that running either of these algorithms on large datasets was a time consuming process. In response to optimizing searches within databases, FASTA and BLAST were both introduced. Instead of comparing whole sequences, these algorithms used the concept that sequences that are good matches contain matches in short stretches. If we do not find similarities in short stretches then we can assume that the sequences being compared are not a good match. We also introduced OrthoMCL which was designed to identify orthologous sequences. The algorithm primarily consisted of using BLAST to identify similar sequences, using the inparanoid algorithm to construct a graph and then reduce the graph using the Markov Clustering algorithm. The most important concept from OrthoMCL was that it used an existing method, in this case BLAST, to perform the sequence comparison to identify all of the potentially interesting sequences and then expanded it by using the inparanoid and Markov Clustering algorithm to filter for orthologous sequences. OrthAgogue was an optimization of OrthoMCL by using additional computing power,

processing data in parallel and optimizing memory usage.

Further, we introduced orthoAlign and multiOrthoAlign which were designed specifically to infer the evolutionary history of tRNA and rRNA gene orders in bacteria genera with a relatively low amount of divergence. These algorithms aligned tRNA and rRNA genes orders and inferred gaps and mismatches using inversions, transpositions, duplications, and deletions. We also introduce ProCARS, Anges, InferCARs, and Gap Adjacency which are each unique in terms of computing the ancestral genome. However, the underlying idea of each method is to identify regions of similarity and assembling these regions into an ancestral genome.

In comparative genomics in order to make meaningful analyses and conclusions about genomic sequences we have to be able to compare them. The method that compares these genomic sequences generally has to provide a scoring system such that we can measure how similar or distinct the sequences are to each other. We have to be able to identify which sequences are more similar to each other compared to other sequences. Additionally, when reconstructing ancestral genomes we have to be able explain the sequence of events at every step in the phylogeny. For this reason, a detailed, explicit evolutionary model is very important. An evolutionary model specifies which events we are searching to identify and what the constraints for each of these events are. Further, we have to able to justify why an algorithm made a particular selection over another such as why a deletion would be favoured in one genome rather than a duplication in another genome. At the same time when considering genome sequences, we have to take into account that there will be duplicate sequences. When there are duplicate sequences, we have to decide which pair of sequences will be selected as orthologs. We have to design a system such that if we have multiple sequences that are equally similar, we have to compute another score that reduces the number of options to one.

Chapter 4

BOPAL

In this chapter we will introduce our evolutionary model, the research problem this M.Sc. thesis will address and a step by step explanation detailing BOPAL's workflow. Additionally, this chapter has been adapted from the publication in BMC Genomics [PCLT19].

4.1 Evolutionary Model

Our evolutionary model is based on the results and observations of previous studies on bacterial genome, operon and tRNA gene evolution, as described in Section 3.6. Based on these observations, our evolutionary model aims to represent *realistic histories*. We define a *realistic history* as an evolutionary history (series of events transforming a genome into another) that considers: (1) the organization of genomes into operons, (2) that rearrangements do not split operons into separate parts, (3) events that move or copy genes across an axis of replication (origin or terminus) reverse the genes, and (4) block (or segmental) duplication/deletion events can only affect genes that are closely located (part of the same operon).

More specifically, our evolutionary model considers the following events:

- A *duplication* copies either a singleton, a gene or a segment of genes inside an operon, or a full operon to another position in the genome. If the duplicated gene(s) are copied to the other side of an axis of replication, an *inversed*

duplication occurs, which involves reversing the order and changing the signs (representing transcriptional orientation/strand) of the genes in the duplicated segment.

- A *deletion* (or a loss) removes either a singleton, a gene or a segment of genes inside an operon, or a full operon from the genome.

- An *inversion* (or reversal) reverses the order and changes the sign of the genes affected. The rRNA genes and tRNA genes are either part of an operon (polycistronic) or not (monocistronic), in which case we refer to them as *singletons* in this M. Sc. thesis. Inversion events can only affect singletons or entire operons (not breaking an operon into separate parts), and must occur around an axis of replication, *i.e.*, the segment that is reversed must be immediately next to either the origin or terminus of replication. These constraints are based on the prevalence of these types of inversions as described in [TBLE15].

- A *transposition* moves either singletons or entire operons to a different place in the genome (for the same reasons described above for inversions). Similarly to duplications, transpositions that move genes to the other side of an axis of replication will be *reversed transpositions*, also reversing the order and changing the signs of the transposed segment.

- A *substitution* is an event that modifies the anticodon of a tRNA gene and/or reassigns a tRNA gene to another identity class.

4.2 Research Problem

The algorithm we propose takes as input a phylogeny representing a bacterial genus, and annotated rRNA and tRNA gene orders, *i.e.*, circular unichromosomal gene orders in which the locations of origin and terminus of replication, the operons, the anticodons (in the case of tRNA genes), and the signs of the genes have been identified. The rRNA genes and tRNA genes are either part of an operon (polycistronic) or not (monocistronic), in which case we refer to them as *singletons* in this M. Sc.

thesis. Each gene order for each extant genome studied is associated to a leaf node. For conciseness, in this M. Sc. thesis we will not make a distinction between a node and its associated gene order.

We aim to infer a parsimonious realistic history for the annotated gene orders considering the evolutionary model described above on a full input phylogeny.

4.3 Annotation of the Gene Orders

4.3.1 Location of the Origin and Terminus of Replication

Recall from Section 3.6, there are several methodologies for finding operons in microbial genomes. To annotate the gene orders with the locations of the origin and terminus of replication, we use the SeqUtils module from the Biopython package [CAC+09]. The SeqUtils module allows us to calculate the GC skews using a sliding window in the full genome sequences, and identify the minimum and maximum values of GC skews. The extrema of the GC skew function are known to be correlated with the loci of the origin and terminus of replication [FL99].

4.3.2 Location of the Operons

Since rRNA and tRNA operons do not contain any other types of genes in the biological dataset presented below, we used a simple rule for determining operons: a maximum intergenic size of 200 bp is allowed between each consecutive rRNA or tRNA gene to consider them part of the same operon. Note that more sophisticated approaches, and/or databases of annotated bacterial operons would be necessary if one were to consider all types of operons in the genomes. An even more precise approach would be to consider experimentally identified transcriptional units, such as those integrated into the DOOR 2.0 database of prokaryotic operons [MMZ+13] (unfortunately, the DOOR 2.0 database was inaccessible at the time of experimentation).

4.4 Algorithm

The proposed approach traverses the whole input phylogeny in post-order, and compares two siblings (left and right child of an internal node, also called *cherry*) at a time to produce an evolutionary scenario and an ancestral gene order for the internal node. Once the ancestral gene order is produced, the post-order traversal continues to produce the next ancestral genomes and so on until the full evolutionary history (on all branches of the phylogeny) has been inferred. Below is a description of the four steps of the algorithm for each comparison of two child nodes (each instance of the 2-SPP), when a neighboring species is available (also see Figure 4.1 for a flowchart describing the steps on an example).

4.4.1 Step 1: Inference of Orthologous Operons and Singletons

We first use all-vs-all pairwise global alignments between the operons of the two genomes compared to identify orthologous operons. This is one of the major differences between our approach and the previous ones presented in [HSAE13, BE14, TBLE15]: instead of aligning the full gene orders to identify orthologous genes, we align only the operons, which tend to be more conserved. Moreover, the global alignments are not used to label events at this time, but only to find similar operons, which allows us to use a simpler scoring mechanism. Once pairs of orthologous operons have been identified, the matched genes contained in the paired operons are considered to be orthologous.

Let M be the dynamic programming table for the global alignment of operons X and Y, and $M[i-1, j-1]$ be the optimal score of aligning the prefix of X ending at position $i-1$ and the prefix of Y ending at position $j-1$, the score $M[i, j]$ can be

calculated using the following recursive function:

$$M[i,j] = Max \begin{cases} M[i-1,j-1]+1, & \text{full match} \\ M[i-1,j-1]+0.5, & \text{partial match} \\ M[i-1,j-1]-1, & \text{mismatch} \\ M[i,j-1]-1, & \text{gap in } Y \\ M[i-1,j]-1, & \text{gap in } X \end{cases} \tag{4.1}$$

where a full match is when both the gene and the anticodon match, a partial match is when the gene matches but with a different anticodon and a mismatch is when both the gene identity class and (necessarily) the anticodon don't match. This notion of partial match only applies to tRNA genes and not rRNA genes, which are not annotated with anticodons. Note that many different scoring schemes could be used here, as long as the score for a match is greater than the score of a partial match, which itself should be greater than the score of mismatches and gaps. The main assumption for setting the score of a partial match in between the one of a full match and the one of a mismatch is that more mutations (not just in the anticodon) would be necessary to completely change the identity class of a tRNA gene, as opposed to a change in the anticodon that preserves the identity class. We ended up using this specific scoring system because it performed well in practice.

After completing all the comparisons, we discard all pairs that have an alignment score < 0. We then label pairs of operons from the two genomes as orthologous starting from the highest alignment scores to the lowest. In case of ties (e.g., an operon from genome X aligns with two operons of genome Y with the same score), we select the pair of operons that is closest in terms of their respective indexes in the genomes.

As for singletons between the two genomes, we simply label them as orthologous if they are identical (same identity class and same anticodon, in the case of tRNA genes). When there are multiple choices, we choose the pairs that are located in the same (or most similar) position in the genome based on their respective indexes.

4.4.2 Step 2: Inference of Duplications, Deletions, and Substitutions

During this step, we first infer duplications, losses and substitutions within the orthologous operons, based on the alignments that were made in *Step 1*. Mismatches or partial matches simply correspond to substitutions. Gaps in the alignment can be labeled either as duplications in one genome, or deletions in the other genome. We follow a simple rule for determining if a gap is a duplication or a loss:

- if the gap has a size ≥ 2 and there exists an identical sequence of genes somewhere else in the same genome, we label it as a duplication;
- otherwise, we arbitrarily label the gap as a deletion.

This simple rule is prone to produce errors, especially for gaps of size one which are always considered to be deletions. The problem with gaps of size 1 is that, since there are almost always multiple copies of each rRNA and tRNA genes in each genome, we could almost always either infer a duplication (recall that to infer a duplication, we must find the same gene — same identity class and anticodon — somewhere else in the genome) or a loss. To alleviate this problem, we allow our algorithm to correct itself by changing deletions into duplications during the next comparison with the neighboring genome, *i.e.*, when we compare the produced ancestor with another sibling (see *Step 4* below for more details).

Once all the orthologous operon pairs have been resolved, we deal with the operons that have not been mapped to an orthologous one in the other genome. We must then infer if these "leftover" operons are the product of a whole operon duplication in one genome (thus being paralogous operons), or a whole operon deletion in the other genome. For each of them, we perform a global alignment with all the other operons within the same genome to find the strongest match with a score ≥ 0. If it exists, we label the whole operon as being duplicated and then we infer duplications, losses and substitutions to explain the gaps and mismatches/partial matches in the alignment in the same manner described above. This is another strength of our approach, because

it allows us to infer *overlapping*, or *non-visible* events, i.e., consecutive events on the same genes that do not directly appear on an alignment of the two genomes. This is another improvement over the previous algorithms, which were designed to consider only visible events [HSAE13, BE14, TBLE15]. If no match within the same genome is found with a score ≥ 0, we simply infer that the non-mapped operon was deleted in the other genome. We proceed in the same manner for the non-mapped singletons, except that the alignment is not required: we simply infer them as duplicated if there is an identical singleton in the same genome, and deleted otherwise.

4.4.3 Step 3: Inference of Rearrangements

Another advantage of our approach is that we infer rearrangements independently of duplications, deletions and substitutions, which once again permits the inference of overlapping events, in the sense that a gene affected by a duplication, deletion or substitution can also be affected by a rearrangement.

In this step, we produce a dot-plot representing all the orthologous operons and singletons paired in *Step 1* (each axis represents a genome and there is one dot for each pair of orthologs; see Figure 4.1 for an example). We use this dot-plot to identify conserved segments, inversed segments and transposed segments. Just like in any dot-plot, conserved segments are series of dots that are located on the main diagonal. Inversed segments can be identified on the dot-plot as a series of dots that cross the main diagonal in the opposite orientation. The other dots or series of dots which are not found on the main diagonal and not inversed are simply identified as transposed segments (either *forward* transposed or *reversed* transposed, depending on their orientation).

4.4.4 Step 4: Inference of the Ancestral Gene Order

One important detail about inversions and transpositions, as described in [TBLE15], is that they can be applied to any of the two sequences. There is simply not enough information in a pairwise comparison that can allow us to discriminate between the two equally probable scenarios. To identify the genome in which the event occurred,

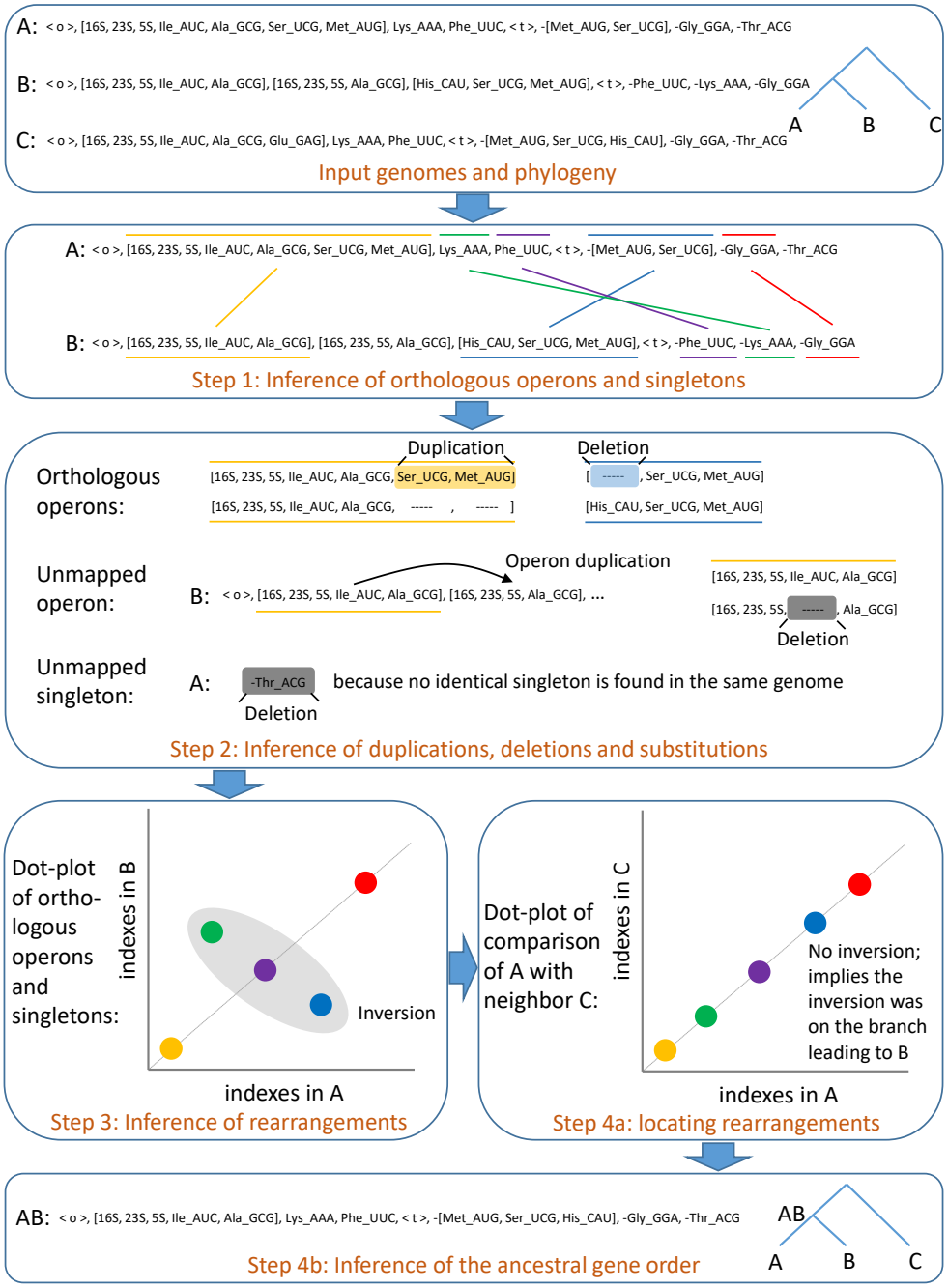


Figure 4.1: Flowchart describing the 4 main steps of BOPAL on a cherry (A, B) with a neighboring genome C. Operons are enclosed in square brackets, whereas singletons are not, and $\langle o \rangle$ and $\langle t \rangle$ represent the origin and terminus of replication respectively.

we use the same strategy proposed in [TBLE15], where we use one of the two sibling genomes X and compare it with another neighboring genome N. N is simply the first resolved genome (either a leaf of a previously built ancestor) encountered in the subtree that is the sibling of the cherry's parent. If the same segment is found to be inversed (respectively transposed) again in that other comparison, then we know that the event occurred on the branch leading to X. Otherwise, if the segment is not inversed (respectively transposed) again, then we know that the event occurred on the branch leading to the other sibling Y (see Figure 4.1 for an example).

Once all the events have been inferred on the correct branches (leading either to genome X or Y), the ancestral gene order can trivially be produced simply by "undoing" the events (e.g., a deleted gene will be placed back into the ancestor, etc.). Once the ancestor is produced, the next comparison can be made following a post-order traversal of the phylogeny. Similarly to how we deal with rearrangement events, we use the next comparison with a neighboring genome to potentially correct for errors in inferred deletions. We keep track of all the genes that were added back into the ancestor because of a deletion event, and if they cause a gap in an alignment (during Step 1), we replace the previously inferred deletion event by a duplication event and modify the ancestor accordingly.

4.5 Runtime Complexity

For each comparison of two child nodes (each cherry), suppose for simplicity that both gene orders contain n genes, distributed among c operons. On average, an operon will contain about n/c genes. *Step 1* of the algorithm requires a global alignment of all pairs of operons between the two genomes: there are c^2 such pairs, and each alignment can be done in $O((n/c)^2)$, which results in $O(n^2)$ time for *Step 1*. In *Step 2*, labeling the gaps in the selected alignments (representing orthologous operons) requires scanning the genome for potential sources of duplications: each scan takes $O(n)$ and there is a maximum of $O(n)$ gaps in total (because selected alignments must have a score ≥ 0), which results in $O(n^2)$ time. The other part of *Step 2* that identifies whole operon duplications takes $O(n^2)$, similarly to *Step 1*, for all the pairwise global

alignment of operons within the same genome. Finally, *Step 3* can be done in linear time, and *Step 4* is similar to *Step 1* but with a neighboring genome, so it takes $O(n^2)$ as well. This leads to a worst-case complexity of $O(n^2)$ for each cherry.

4.6 Summary

In this chapter we presented an operon-based approach to infer an evolutionary history of tRNA and rRNA genes. We began by identifying our evolutionary model. Our evolutionary model considers the organization genomes into operons, rearrangement evolutionary events do not split operons into separate parts, any events that move or copy genes across the origin of replication or the terminus reverses the genes, and deletions and duplications can only affect multiple genes only if they are in the same operon. The evolutionary model is a fundamental component; the reason being that it identifies the data's constraints and specifies the general rules of how the algorithm is expected to function. The input for `BOPAL` is a phylogeny representing the bacterial genera along with each of their rRNA and tRNA gene orders.

We identified the problem we intend `BOPAL` to address in this M.Sc. thesis. We want `BOPAL` to infer a parsimonious realistic history for the annotated gene orders using the evolutionary model described in the chapter along with the ancestral gene orders based on the bacterial genus's phylogeny. We also discuss how we identified the location of the origin and terminus by using SeqUtils to identify regions in the genome where the nucleotides guanine and cytosine are over abundant and under abundant. The following principle was used identify the location of operons within the bacterial genomes; the intergenic size cannot be larger than 200 base pairs. Recall that the intergenic region is the stretch of DNA located between genes.

We describe step by step `BOPAL`'s procedure for processing the bacterial genomes along with their phylogeny. In the first step `BOPAL` identifies the orthologous operons and singleton genes. `BOPAL` identifies whether a pair of operons is a good match by performing a global alignment. The global alignment outputs a score based on the number of matches, partial matches, mismatches, and gaps. After performing all of the comparisons, `BOPAL` discards all of the comparisons that have resulted in a score $<$

0. In the event there is an operon in genome X that aligns with 2 operons in genome Y with the same score, BOPAL selects the pair of operons that is closest in terms of their respective indexes in the genomes.

In the second step BOPAL infers duplications, losses and substitutions within the orthologous operons using the alignment computed from the global alignment in the previous step. Gene and codon mismatches correspond to substitutions while gaps correspond to a deletion one genome or duplication in the other genome. A gap consisting of 1 gene is labelled as a deletion by default whereas if the gap consists of multiple genes then BOPAL scans the genome to see if an identical sequence exists to label it as a duplicate otherwise it is labeled as a deletion as well. As for the remaining operons that were not paired with another operon from the previous step, BOPAL performs a global alignment for each of them with all of the other operons to check if there are any matches with a score ≥ 0. If there are matches found, then these operons will be labelled as duplicates and deletions and duplications will be inferred within the operon as well.

In the third step BOPAL infers rearrangement events for each of the operons by constructing a dot-plot aligning all of the orthologous operons identified in the first step. BOPAL performs a scan of the dot plot to identify the conserved segments, inverted segments, and the transposed segments. However, inversions and transpositions can be applied to either of the two genomes. There is not enough information to determine which genome the event should be applied to. For this reason, in the fourth step BOPAL uses one of the genomes to compare it with a neighboring genome on a different branch to determine if the same transposition/inversion exists as well. Once all of the events have been identified on the correct branch, BOPAL reconstructs the ancestral genome removing the duplicate segments, adding the deleted segments, and reverse the inversions/transpositions. Finally, we identified that BOPAL has a worst-case complexity of $O(n^2)$ for each pair of siblings compared.

Chapter 5

Evaluation Results And Discussion

We implemented our algorithm in Python 2.7 and named it `BOPAL` — Bacterial OPeron ALigner. We then evaluated it on both simulated and real biological datasets.

5.1 Evaluation on Simulated Datasets

We developed a simulated data generator that takes as input a tree topology of L leaves, an ancestral genome size denoted by a number of genes n, and the number of events to be generated on each branch of the tree E. The generator creates a random ancestral gene order (note that we do not simulate sequences, since our approach does not use sequence information, other than the tRNA anticodons), annotated with operons and anticodons, at the root of the phylogeny and randomly simulates evolution of each branch according to the selected parameters. We use a geometric distribution, with a parameter that we named p_{op}, to sample the size of the operons and then we populate them with genes. Singletons are randomly added to genomes using a probability $prob_s$, and the probability of adding an operon instead is $1-prob_s$. During the simulated evolution, when an event is chosen to be performed on a branch, a random starting point is selected and its size (number of genes or operons affected) is also sampled from a geometric distribution (we named the parameter of this geometric distribution p_{event}). In accordance with the evolutionary model described earlier, the generator will not simulate rearrangements that break operons into separate parts,

simulate inversions that are not occurring around an axis of replication, etc.

5.1.1 Accuracy on Cherries with Neighbor

We tested how our new approach compares with the 2-SPP algorithm of [HSAE13] (hereafter referred to as DupLoss) and OrthoAlign [TBLE15] on cherries, *i.e.*, two sibling leaves that share the same parental node. We also added to our simulations a third neighboring genome to test how OrthoAlign and `BOPAL` perform with the additional information coming from the neighbor. Note that we did not test the DupLoCut algorithm because the output only reports the total number of events, which would not allow us to analyze all the types of accuracy that we consider below. Also, we were not able to perform tests with multiOrthoAlign because no implementation was available online at the time of experimentation. For the analysis of the simulations an Intel Core i5 2.5 GHz with 8GB of memory was used.

For this test we used a triplet phylogeny ($L = 3$ leaves), a constant ancestral genome size (in genes) $n = 120$, $p_{op} = 0.125$ (producing an average operon size of 8.2 genes), $prob_s = 0.35$ (resulting in an average number of singletons and operons of 7.8 and 13.7 respectively), and $p_{event} = 0.7$. These probabilities and parameters were chosen to represent as closely as possible the biological dataset studied below (see Table 5.2 for more information on the biological dataset). As for the simulated events, we used one inversion randomly applied to one of the branches of the cherry, and x times a duplication, a deletion, a transposition and a substitution on each branch (so the total number of events per branch are multiples of 4, excluding the single inversion). Note that we simulated only one inversion because our model considers inversions around an axis of replication only, and multiple consecutive inversions tend to cancel each other out. Based on the previous analysis of 50 *Bacillus* genomes [TBLE15], inversions do not seem to occur very frequently (only 23 inversions were inferred in total, for an average of 0.232 inversions per branch), which makes the simulation of 1 inversion per cherry reasonable. All the results presented below are averaged over 100 replicates.

To measure the accuracy of the different approaches, we first compared the total

number of events inferred by the three different methods with the total number of events that were simulated by the data generator (see Figure 5.1). Unsurprisingly, DupLoss, which does not consider rearrangements, has to infer a lot more events to explain these evolutionary scenarios. All the other methods tend to underestimate the number of events when more events are generated, which is expected since the traces of some events can disappear after successive events, and some shortcuts can be found in the evolutionary scenarios. The use of a neighbor with `BOPAL` does not make much of a difference in the total number of events inferred, since the neighbor is used only to place rearrangements on the correct branch and potentially modify a deletion of size 1 into a duplication of size 1. In OrthoAlign however, using a neighbor increases the number of events, this probably occurs when it modifies deletions of a block of genes for more smaller duplications.

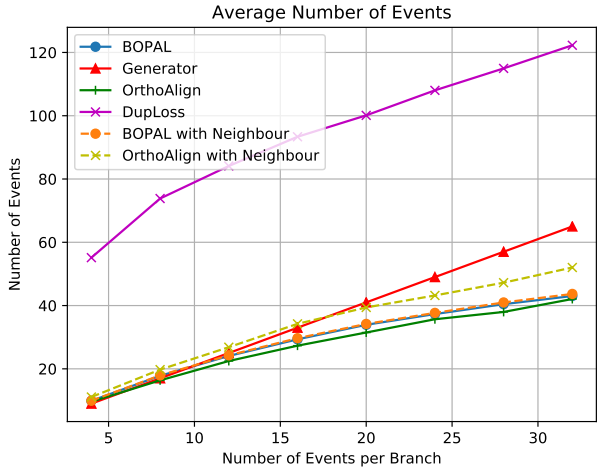


Figure 5.1: Total number of events inferred, for multiples of 4 events per branch and one inversion on one of the branches leading to the cherry.

We also measured how accurate the ancestral gene orders produced were. To do this, we used DupLoss [HSAE13] to align the inferred gene order with the simulated one and counted the gaps in this alignment (DupLoss [HSAE13] does not allow mismatches and only produces matches and gaps). Matches in this comparison of ancestral gene orders were counted as true positives (TP), gaps in the inferred ancestor, which correspond to missing genes, were counted as false negatives (FN), and

finally gaps in the simulated ancestor, which correspond to extra genes, were counted as false positives (FP). These allowed us to calculate recall and precision:

$$recall = \frac{TP}{TP + FN} \tag{5.1}$$

$$prec. = \frac{TP}{TP + FP} \tag{5.2}$$

We then combined recall and precision into one measure by calculating their harmonic mean, which is traditionally called the F-measure:

$$F = 2 * \frac{recall * prec.}{recall + prec.} \tag{5.3}$$

Results on the F-measure for the inferred ancestors are presented in Figure 5.2. In general, all methods perform similarly, except `BOPAL` with the neighbor which infers considerably more accurate ancestors. `BOPAL` without the help of the neighbor seems to perform the worst, however, this was expected, since `BOPAL` does make some arbitrary choices between deletions and duplications when there is no neighbor, and might infer rearrangements on the wrong branches. Interestingly, having a neighbor

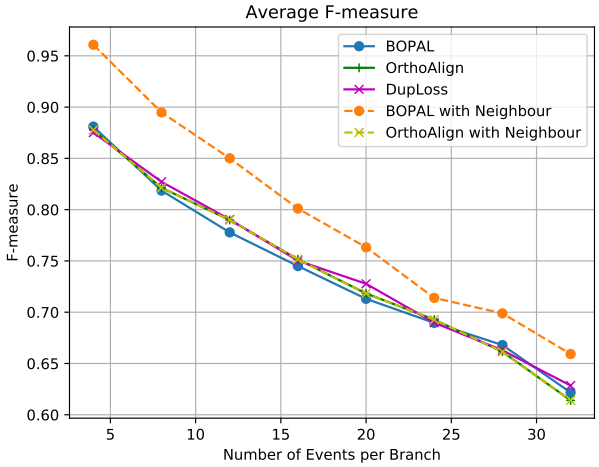


Figure 5.2: F-measure of the reconstructed ancestral gene orders.

does not seem to improve the ancestral prediction of OrthoAlign. As for DupLoss, it performs similarly to OrthoAlign for the F-measure. It is still reasonably accurate in its inference of the ancestral gene order, even if it has to use a lot more events.

Finally, we measured the accuracy of the events that were inferred on each branch of the cherry in two different ways: *strict event accuracy* and *relaxed event accuracy*. On the one hand, we define the *strict event accuracy* by the ratio of the number of events inferred completely correctly, (*i.e.*, with the exact same length and position) over the total number of events generated. On the other hand, we define the *relaxed event accuracy* as the ratio of genes labeled with the correct event over the total number of genes affected by events in the simulated data. In other words, the relaxed ratio focuses on the genes being labeled with the correct event, and not on the number or size of the events. For example, if a deletion of two consecutive genes a_1, a_2 was simulated on a branch by the data generator, and the algorithm inferred two separate deletions a_1 and a_2, the strict event accuracy would be 0%, but the relaxed event accuracy would be 100%.

The strict and relaxed event accuracy graphs are shown in Figures 5.3 and 5.4. Clearly, inferring accurate events is very difficult in general, and it becomes more difficult as the number of events per branch increases. Note that the tests went up to 32 events per branch, which is much more than what we would typically expect in a real dataset (in the study of 50 *Bacillus* genomes [TBLE15], an average of 2.525 events were inferred per branch). In terms of strict event accuracy, `BOPAL` with a neighboring genome performs the best, with values in the range of 60% to 25%. `BOPAL` without a neighbor performs similarly to OrthoAlign with a neighbor, while OrthoAlign without a neighbor and DupLoss exhibit the worst performances.

For the relaxed event accuracy, we observe a small improvement of `BOPAL` both with or without the neighbor compared with the values of strict accuracy. On the other hand, all the other methods (except OrthoAlign without the neighbor) perform worse in terms of relaxed accuracy than for the strict accuracy. To better interpret this result, we analyzed the average size (in number of genes) of all the events inferred completely correctly (the ones that were counted in the strict event accuracy), and found that `BOPAL` infers more of the longer events on average than its competitors (see

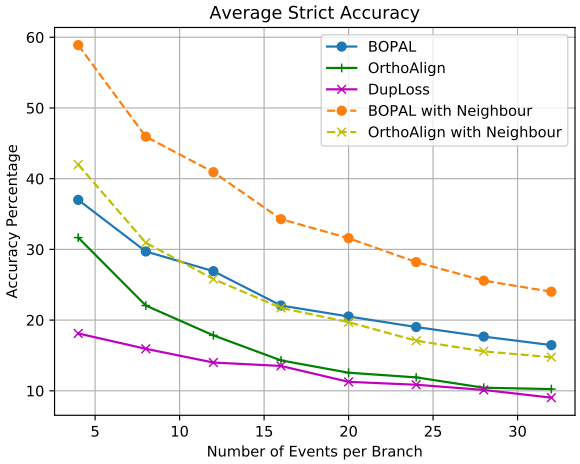


Figure 5.3: Strict event accuracy.

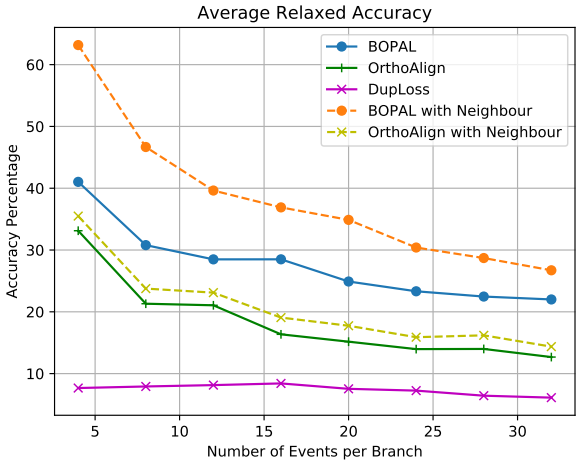


Figure 5.4: Relaxed event accuracy.

Figure 5.5). `BOPAL` with a neighbor performs the best all the time, with values ranging between 63% and 27%. Interestingly, it is followed by `BOPAL` without a neighbor, and then OrthoAlign both with and without a neighbor performing almost similarly. The curve for DupLoss is relatively flat and very low, which is a bit surprising considering that half of the events inferred on each branch are duplications and losses.

Recall that for this test we used a triplet phylogeny ($L = 3$ leaves), a constant

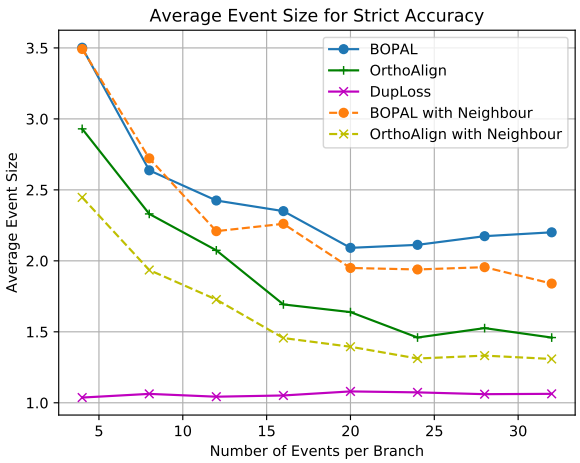


Figure 5.5: Average size of the events that were inferred completely correctly by the different methods.

ancestral genome size $n = 120$, $p_{op} = 0.125$ (producing an average operon size of 8.2), $prob_s = 0.35$ (resulting in an average number of singletons and operons of 7.8 and 13.7 respectively), $p_{event} = 0.7$. As for the simulated events, we used one inversion randomly applied to one of the branches of the cherry, and x times a duplication, a deletion, a transposition and a substitution on each branch.

Figure 5.5 presents the average size of the events that were inferred completely correctly, (*i.e.*, the events considered for calculating the strict accuracy) by the 5 different approaches.

5.1.1.1 Accuracy on Varying Genome Sizes

We also evaluated how the number of genes in the gene orders affects the accuracy of the different approaches, for a fixed number of events. Basically, we used the same parameters described above, except that x was set to 4 (resulting in 16 events per branch plus one inversion), and we used an ancestral genome size n varying from 50 to 250. The results, presented in Figures 5.6, 5.7 and 5.8, show that all the types of accuracy increase with the number of genes. These results suggest that considering more types of operons in the bacterial genomes could lead to even better inferences of evolutionary scenarios and ancestors.

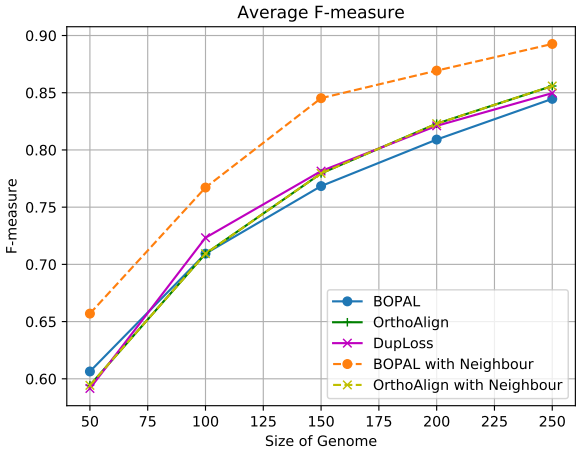


Figure 5.6: F-measure of the reconstructed ancestral gene orders.

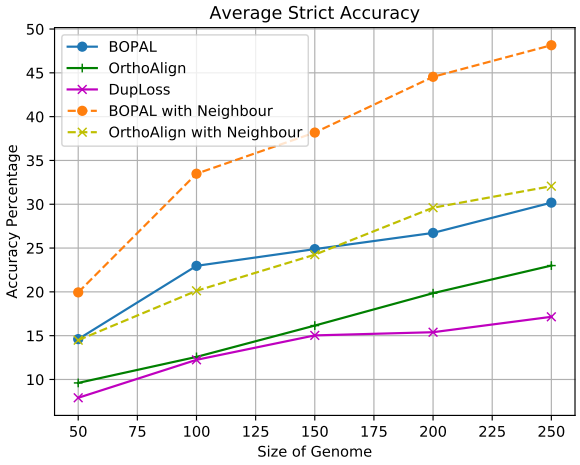


Figure 5.7: Strict event accuracy.

For this test, we used the same parameters described above (Section 5.1.1), except that x was set to 4 (resulting in 16 events per branch plus one inversion), and we used an ancestral genome size n varying from 50 to 250.

Figures 5.6, 5.7 and 5.8 are presenting respectively the F-measure of the reconstructed ancestors, the strict event accuracy and the relaxed event accuracy.

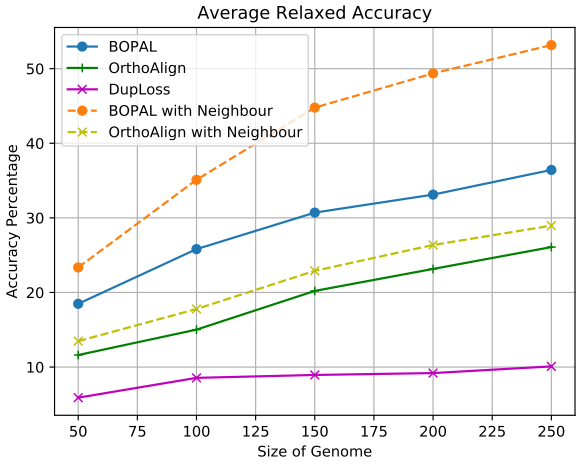


Figure 5.8: Relaxed event accuracy.

5.1.2 Runtime

We also measured the average runtimes of the 5 different methods (see Figure 5.9), using an Intel Core i5 2.5 GHz with 8GB of memory. The runtimes of OrthoAlign and `BOPAL` without the neighbor are not affected by the number of events. `BOPAL` with a neighbor is unsurprisingly slower than `BOPAL` without a neighboring genome,

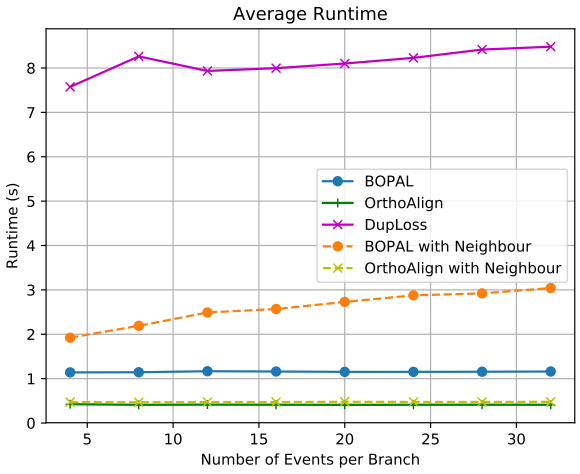


Figure 5.9: Average runtimes of the different methods compared.

Table 5.1: Average Runtimes for $n = 1000$.

Method	Runtime (s)
DupLoss	17161.50
OrthoAlign	1.14
OrthoAlign with neighbour	3.76
BOPAL	14.75
BOPAL with neighbour	130.58

and becomes a little bit slower with more events, which can be explained by the comparisons that have to be made with the neighbor for each rearrangement event to infer it on the correct branch. DupLoss, which uses ILP is unsurprisingly the slowest method of all. `BOPAL` is a little bit slower in practice than OrthoAlign, with average runtimes of just over 1 second without a neighbor, and between 2 and 3 seconds with a neighbor, in comparison with average runtimes of approximately 0.5 seconds for OrthoAlign.

To get an evaluation of the scalibility of our method, we measured the runtime of the different approaches on ancestral genome sizes n varying from 200 to 1000 genes (see Figures 5.10 and 5.11). The other parameters are the same as presented in Section 5.1.1.1. Table 5.1 shows the average runtimes of the 5 differents approaches for $n = 1000$. Due to the time required to run DupLoss, these results are averaged over only 2 replicates.

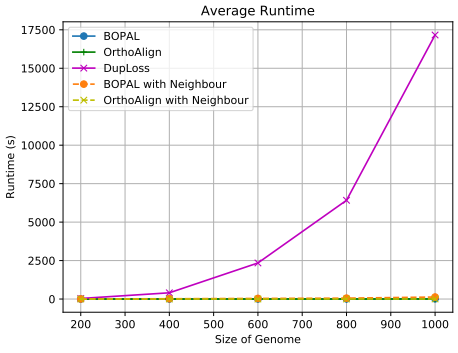


Figure 5.10: Runtime on large genomes with DupLoss.

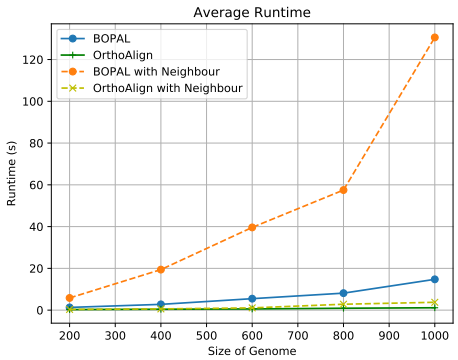


Figure 5.11: Runtime on large genomes without DupLoss (zoom-in view).

5.2 Evaluation on Biological Datasets

We compared the performance of our algorithm to multiOrthoAlign and DupLoCut on the same biological dataset of 12 *Bacillus* gene orders used in [ARC13] and [BE14], to which we added the operon annotations (see Table 5.2 for details on the genomes studied and their operon annotations, and Figure 5.12 for the phylogeny used). `BOPAL` completed the analysis of the whole tree with a runtime of 6.45 seconds (on the same Intel Core i5 2.5 GHz with 8GB of memory used for the simulations).

Table 5.2: Description of the 12 *Bacillus* genomes studied, their NCBI accession number and information about the annotated rRNA/tRNA singletons (sing.) and operons (op.). The "% of genes" column represents the proportion of all tRNA and rRNA genes over the total number of coding genes in the genome.

Genome name	Accession #	# of sing.	# of op.	Avg. op. size	% of genes
Bacillus cereus ATCC 10987	NC_003909	6	15	8.47	2.46
Bacillus cereus E33L	NC_006274	5	16	8.13	2.30
Bacillus cereus ATCC 14579	NC_004722	7	15	9.33	2.69
Bacillus thuringiensis BMB171	NC_014171	5	17	8.29	2.58
Bacillus thuringiensis serovar kurstaki str. HD73	NC_020238	6	15	8.93	2.45
Bacillus thuringiensis serovar konkukian str. 97-27	NC_005957	9	14	9.79	2.77
Bacillus subtilis subsp. spizizenii str. W23	NC_014479	9	11	8.36	2.57
Bacillus subtilis subsp. spizizenii TU-B-10	NC_016047	9	13	8.69	2.99
Bacillus subtilis subsp. subtilis str. 168	NC_000964	9	11	9.73	2.56
Bacillus amyloliquefaciens FZB42	NC_009725	9	15	7.20	3.17
Bacillus amyloliquefaciens subsp. plantarum CAU B946	NC_016784	8	15	7.80	3.30
Bacillus amyloliquefaciens DSM 7	NC_014551	9	16	7.19	3.20

Figure 5.12 presents the tree that was used for the analysis of the 12 *Bacillus* genomes. It is the same tree that was used in [ARC13] and [BE14].

`BOPAL` inferred 56 duplications, 37 deletions, 8 transpositions and 16 substitutions for a total of 117 events. Based on the results presented in [ARC13] and [BE14], multiOrthoAlign converged at 123 events and DupLoCut converged to a minimum of 120 events on this dataset (see Table 5.3 for a summary). However, multiOrthoAlign was restricted to inferring duplications and losses only, just like DupLoCut, whereas `BOPAL` was using its full evolutionary model. Interestingly, the added constraints of the operon boundaries and the fact that `BOPAL` does not calculate multiple iterations of the median problem did not result in a scenario with more events. The transpositions events inferred by `BOPAL` probably played a role in the inference of a slightly lower number of events.

Table 5.3: Number of events identified by BOPAL, multiOrthoAlign, DupLoCut on the dataset of 12 *Bacillus* genomes.

Algorithm	**Reported Events**
BOPAL	117
multiOrthoAlign	123
DupLoCut	120

87.5% of the duplications inferred by BOPAL were affecting 1, 2 or 3 genes, whereas the rest of the duplications were of size greater than 5, with the largest one being a whole operon duplication of size 25 (see Figure 5.13 for the size distribution of duplication events). Similarly, the majority of the inferred deletions were short. About 75% of the deletion events were of size 1, 2 or 3, and the rest of them had a length in the range of 5 to 17 genes. Out of the 8 transpositions inferred, three were of size 3, two of size 5, and one of each sizes 6, 12 and 15. Although we were not able to analyze the events inferred by the other methods, it is quite possible that the restrictions of

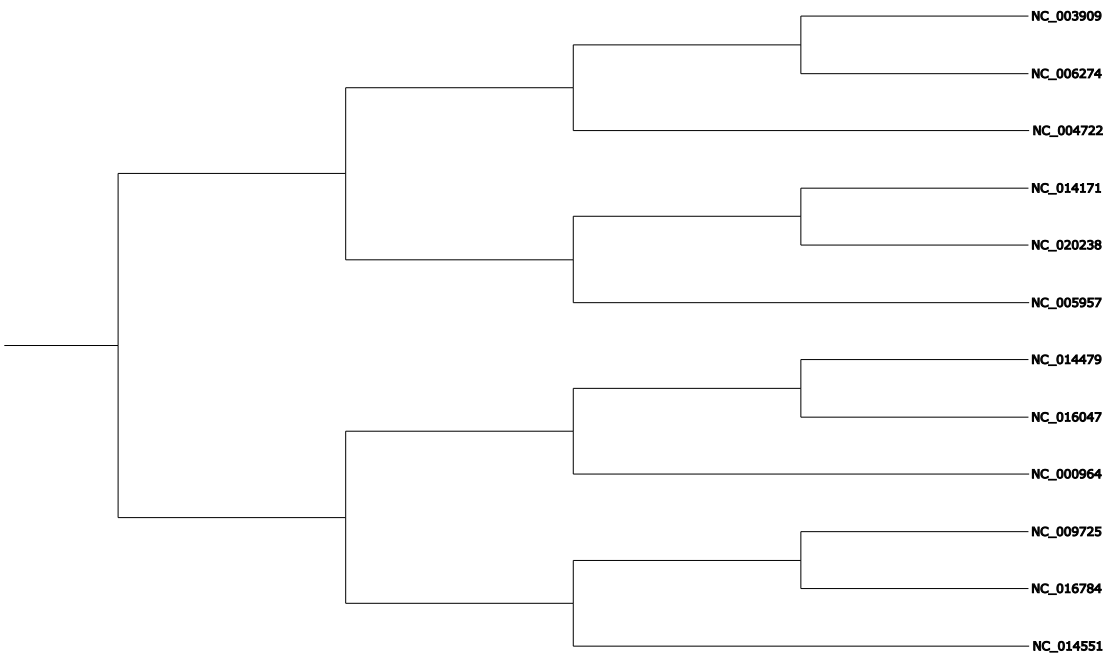


Figure 5.12: Tree used for the evaluation on biological datasets. The leaf labels represent the Genbank accession numbers of the genomes analyzed.

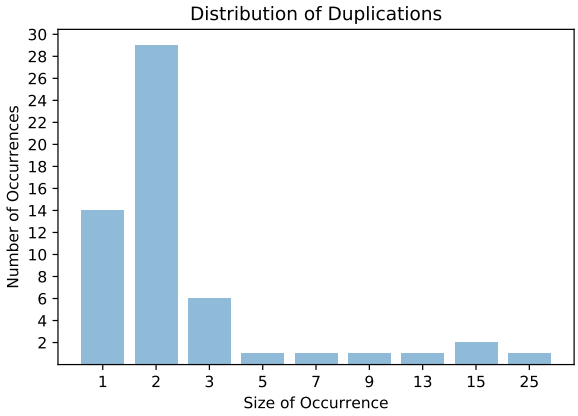


Figure 5.13: Size distribution of the duplications inferred by BOPAL on the 12 *Bacillus* genomes.

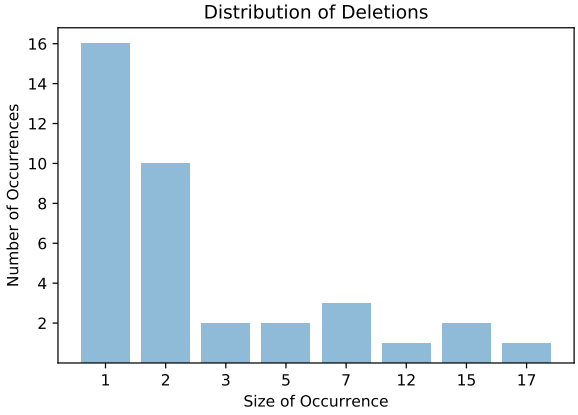


Figure 5.14: Size distribution of the deletions inferred by BOPAL on the 12 *Bacillus* genomes.

our evolutionary model have given rise to a different but equivalent (in terms of the total number of events) evolutionary history.

5.3 Summary

In order to evaluate BOPAL, we performed an analysis using simulated datasets and a biological dataset. For the simulated dataset we developed a simulator where the

input consists of the tree topology of L leaves, the ancestral genome size denoted by a number of genes n, and the number of events to be simulated on each branch of the tree E. The simulator generates a random gene order that is annotated with operons and anticodons by starting at the root of the phylogeny and simulating evolution on each branch according to the parameters specified.

The first set of experiments consists of measuring the accuracy of each method on cherries with a neighbor. In this analysis we compared `BOPAL`, DupLoss and orthoAlign. The reason for the addition of the neighbor was to test how `BOPAL` and orthoAlign perform with additional information from another genome. In the test we considered a triplet phylogeny, a constant ancestral genome size of 120 genes, an average operon size of 8.2 and an average number of 7.8 singletons and 13.7 operons. The reason we selected these parameters was that we wanted the simulated data to be similar to biological data as much as possible. In the simulated data we considered x duplications, deletions, transpositions and substitutions so that the total number of events per branch are multiples of 4. We also simulated 1 inversion around the axis of replication on one branch.

In the first experiment we measured the accuracy of the different methods by comparing the total number of events computed by each algorithm with the total number of events that were simulated by the data generator. DupLoss inferred the most events since it does not consider rearrangements and requires more events to explain the evolutionary scenarios. The other algorithms under estimated the number of events the reason being that traces of some events can disappear after successive events. Additionally, it is possible that `BOPAL` and orthoAlign found shortcuts in terms of the number of events required explain the evolutionary scenario. However, the use of a neighbor in `BOPAL` does not significantly change the number of events inferred the reason being that the neighbor is used to infer rearrangement events and switch deletions of size 1 to duplications of size 1. Interestingly we saw an increase in the number of events when orthoAlign used a neighbor. The reason behind this was probably due to orthoAlign modifying blocks of deletions to small individual duplications.

In the second experiment we measured the accuracy of the ancestral gene orders

produced by each method. Generally the accuracy of all the methods was similar, however, BOPAL with the neighbor was able to infer more accurate ancestors compared to the other methods. Interestingly BOPAL without the neighbor exhibited the worst accuracy which was expected since BOPAL arbitrarily makes decisions on deletions, duplications and rearrangements when no neighbor is present. Additionally, it was observed that orthoAlign did not exhibit a significant improvement with the addition of the neighbor. We also observed that DupLoss performed similarly to orthoAlign which was interesting the reason being that DupLoss inferred a lot more events compared to orthoAlign.

In the third experiment we inferred the accuracy of the events that were inferred on each branch of the cherry using strict event accuracy and relaxed event accuracy. Strict accuracy refers to the ratio of the number of events inferred completely correctly and relaxed accuracy refers to the ratio of the genes labeled with the correct event. With values ranging between 60% to 25%, BOPAL with the neighbor performed the best in the strict accuracy portion of the test which was expected due to the neighbor providing additional information about the ancestral genome. BOPAL without the neighbor performed similarly to orthoAlign with the neighbor. Finally, orthoAlign without a neighbor and DupLoss performed the worst in this portion of the experiment. For relaxed accuracy, there was only a small improvement for BOPAL with the neighbor and without the neighbor. The other methods all performed worse in the relaxed accuracy portion of the experiment. Interestingly we also found during our analysis that BOPAL was able to infer more of the longer events than the other methods.

In the fourth experiment we evaluated how the number of gene orders affects the accuracy of each method. In the experiment we used an ancestral genome size n varying between 50 to 250. We observed that as the number of genes increase, so does the accuracy. This suggests that considering more types of operons within the bacterial genomes leads to more accurate inferences of the evolutionary history and ancestral genomes. In addition, we measured the run time of the 5 methods. We observed that orthoAlign and BOPAL without the neighbor are not affected by the number of events. However, we observed that BOPAL with the neighbor is slower

than BOPAL without the neighbor and becomes even slower as the number of events increase. This is to be expected since there are more comparisons that need to be made. Interestingly DupLoss was the slowest method compared to the other methods. We observe that BOPAL is a bit slower compared to orthoAlign, however, BOPAL is still scalable to large genomes.

In our final analysis we compared the performance of BOPAL, multiOrthoAlign, and Duplocut on the same biological data of 12 *Bacillus* gene orders to which we added operon annotations. BOPAL identified a total of 117 events whereas multiOrthoAlign identified 123 events and Duplocut identified 120 events. The reason we see a more significant difference between BOPAL and multiOrthoAlign was due to the fact that multiOrthoAlign was restricted to a duplication and loss evolutionary model whereas BOPAL used our whole evolutionary model. Additionally transpositions inferred by BOPAL most likely resulted in a lower number of events compared to the multiOrthoAlign. We were not able to analyze the evolutionary events inferred by multiOrthoAlign and Duplocut. However it is possible that due to the restriction of our evolutionary model BOPAL has given a different but equivalent evolutionary history.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this M.Sc. thesis, I presented `BOPAL` which is an operon based approach for the inference of realistic evolutionary histories of rRNA and tRNA genes. The primary purpose of this thesis was to provide a proof of concept that we can identify orthologous operons, which ultimately helps with the identification of orthologous genes when the genomes have been transformed by many evolutionary events. Even though the analysis we presented here was focused on the evolution of rRNA and tRNA genes, our method can be adapted to the inference of evolutionary scenarios of any type of genes that are organized into operons.

Further, there were three key questions I sought to answer, as presented in 1.1.

1. "Can the alignment of gene orders from operons be used to infer an orthologous relationship between two operons?"

   The Needleman-Wunsch is more advantageous for our purposes as this algorithm constructs an alignment that is similar in length to the two operons being compared. As stated in Section 1.1, operons tend to be more conserved in general. If two operons are a good match, then the Needleman-Wunsch algorithm should be able to identify them. In order for two operons to be a good match they cannot have a score below zero. This condition does not allow for an alignment

where mismatches/gaps/substitutions are the majority but at the same time it permits for blocks of genes to be deleted/duplicated from an alignment. Based on our results, we found this approach to be an accurate method for narrowing down our options for which operons are suitable to be orthologous candidates.

2. "Can strains on neighboring branches help infer rearrangement events on the correct branch for two siblings sharing the same parental node?"

   Rearrangement events, in this case inversions and transpositions, can be equally applied to either branch of two siblings that share the same parental node. There is simply not enough information to deduce which branch the event belongs to solely on the siblings. For this reason we have to identify which arrangement of these same operons exists in other neighboring genomes located on a different branch. We randomly select one of the siblings, it does not matter which sibling is selected, and we check if the same rearrangement event is visible when comparing with the neighboring genome. If the same rearrangement event is visible, then we can assume the rearrangement event belongs on the branch of the sibling that was selected for the neighbor comparison. If the rearrangement event is not visible, then we can assume the event belongs on the branch of the other sibling that was not compared with the neighbor. As discussed in Section 5.1.1 we applied one inversion event to one of the siblings and used the neighboring genome to identify which branch to place the event on.

3. "Can I prevent incorrectly labeled events from cascading up the phylogeny?"

   Incorrectly labelling genes as deletions instead of duplications was one of the major problems that had to be addressed by `BOPAL`. If a gene is labelled as a deletion, then it will be included in the ancestral genome. However, if `BOPAL` is wrong and the gene is indeed a duplication, then the same gene will be repeatedly marked as a deletion as we traverse the phylogeny in a bottom-up fashion reconstructing ancestral genomes. Recall that we want to reconstruct ancestral gene orders with the number of events minimized and with mislabelled genes that are constantly being labeled as deletions over and over would increase

this number. For this reason BOPAL tracks which genes are labelled as deletions when reconstructing an ancestral genome. If the same gene is lost again when reconstructing the next ancestral gene order, then BOPAL will switch the gene to a duplication to avoid having the same gene deleted repeatedly. Based on our result in Section 5.1.1 we can see that BOPAL was able to minimize the number of events for reconstructing ancestral gene orders compared to the other methods based on this ability to correct itself when a gene is labelled incorrectly.

Our evaluation results have shown that BOPAL is able to infer more accurate events and ancestors compared to previous approaches with a reasonable runtime. To elaborate, major advantages of BOPAL include the following:

- BOPAL is robust in terms of identifying orthologous operons regardless of their length. The reason our approach is robust is due to the fact that the global alignment always finds an optimal alignment.

- BOPAL is able to verify whether an operon is a deletion or a duplication by identifying whether the same operon is present in a neighboring genome. This prevents BOPAL from repetitively marking the same operon as a deletion or a duplication when traversing up the phylogenetic tree reconstructing ancestral genomes.

- BOPAL is able to identify orthologous operons based on their positions within each of their respective genomes. If there are multiple operons that are equally viable, then BOPAL is able to narrow the selection down based on each of the operon's position in their respective genomes. BOPAL makes the decision by selecting the two operons that are closest to each other in terms of their genome position. This solution works when the equally viable operons are spread out in the two genomes.

- BOPAL is able to make the distinction between an operon and a singleton gene and handle each one accordingly. Singleton genes are a special case and are treated as such. A singleton gene can only be orthologous with another singleton

gene. If we fail to find a suitable match then we try to identify whether we can find another copy within the same genome within an operon. For the vast majority of the time this is an approach that leads to an optimal ancestral genome.

- `BOPAL` is able to reverse its decision if a gene is incorrectly marked as a deletion. If the same gene is deleted across multiple generations, then `BOPAL` will switch the gene to a duplication instead and adjust the ancestral genomes, accordingly.

- `BOPAL` is able to identify a rare situation that is present in the biological data where we have two operons that are a good match within the same genome but neither of these operons has a good match with any of the operons in the sibling's genome. This is a special case where one of the operons was duplicated in one of the sibling's genome resulting in the two operons being a good match and the same operon was deleted from the other sibling's genome resulting in no good matches being found in the other genome. `BOPAL` able to identify this case and selects one of these operons to insert into the ancestral gene order and the other will be labelled as a duplicate.

6.2 Future Work

In this research project, based on the results, we have shown that `BOPAL` is able to infer more accurate events and ancestors compared to previous approaches with a reasonable runtime. However, there are certain areas where `BOPAL` could have been improved upon. In the following Section we will reflect where we believe `BOPAL` performed well and some potential future work to improve `BOPAL`.

Although `BOPAL` is robust in terms of identifying orthologous operons (regardless of their length) due to the fact that the global alignment always finds an optimal alignment, a global alignment may have multiple optimal alignments and `BOPAL` only selects one of the alignments. We think this is one area where `BOPAL` could have been improved. Rather than randomly selecting one optimal alignment, `BOPAL` should maintain a list of these alignments to avoid selecting an alignment that may not

necessary result in an optimal ancestral genome. In order to resolve which alignment to select as the optimal alignment, BOPAL should use a neighboring genome. In this case BOPAL would perform an alignment of the same operon in the neighboring genome and verify whether one of the alignments from the list is favored over the other. If the alignment from the neighbor matches one of the alignments from the list, then that alignment would be selected. If none of the alignments is favored, then BOPAL should retain all of the alignments and reduce the list when computing the ancestral genomes further up the phylogenetic tree by removing alignments that result in more evolutionary events.

BOPAL is able to verify whether an operon is a deletion or a duplication by identifying whether the same operon is present in a neighboring genome. This prevents BOPAL from repetitively marking the same operon as a deletion or a duplication when traversing up the phylogenetic tree reconstructing ancestral genomes. However, we think BOPAL could have potentially been improved by performing more than one neighbor comparison. Indeed comparing with one neighboring genome significantly reduced the number of operons incorrectly marked as duplications or deletions. However, we would have liked to have seen if more accuracy could have been been attained if we compared with more than one neighboring genome. We think that BOPAL should verify with 3 neighboring genomes whether an operon should be marked as a duplicate or a deletion in order to prevent any ties and the majority would win. The reason for this additional check would be to verify whether the neighboring genome deleted the same operon. We could have a scenario where the operon was present in the ancestral genome, however, after speciation the operon was deleted from one of siblings on both neighboring branches.

Note that BOPAL is also able to identify orthologous operons based on their positions within each of their respective genomes. If there are multiple operons that are equally viable, then BOPAL is able to narrow the selection down based on each of the operon's position in their respective genomes. BOPAL makes the decision by selecting the two operons that are closest to each other in terms of their genome position. This solution works when the equally viable operons are spread out in the two genomes. However, if there are multiple identical or very similar operons in a consec-

utive sequence, then this solution does not necessarily produce the optimal mapping of orthologous operons. In this scenario, if there is a deletion or duplication of an operon or a singleton prior to these consecutive operons, then this will cause a frame shift in one of the genomes. This will result in one the genome's first operons being mapped to the second genome's last operon for these consecutive operons. In order to prevent this from happening, we propose that there could be some work done to check whether there are any consecutive operons within the genome that are identical or very similar. The process would be as follows: `BOPAL` would scan both genomes and determine if there is a sequence of consecutive operons that identical or very similar to each other that could potentially result in multiple mappings of orthologous operons. After these consecutive operons have been identified, `BOPAL` would treat these operons as one large sequence rather than as individual operons. Essentially `BOPAL` would treat this sequence of consecutive operons as one large operon which would resolve the issue of not mapping the orthologous operons optimally. The remaining operons would be handled normally.

Moreover, `BOPAL` is able to make the distinction between an operon and a singleton gene and handle each one accordingly. A singleton gene can only be orthologous with another singleton gene. If we fail to find a suitable match then we try to identify whether we can find another copy within the same genome within an operon. For the vast majority of the time this is an approach that leads to an optimal ancestral genome. However, based on our observations, we would have liked to have made a modification to this approach. We have observed in a small number of cases where we have an operon next to a singleton gene in one genome, we could potentially have the same singleton gene located within the neighboring operon in the other genome. Our current approach would be unable to find a suitable match and the singleton gene would be marked as a deletion it would be included in the ancestral genome resulting in two copies of the same gene being present in the ancestral genome. There would be a copy of the gene within the operon and another copy of the gene next to the operon. We propose an additional approach where if `BOPAL` is unable to find an identical singleton gene in the sibling genome, then `BOPAL` should perform a check to see if the same gene is located within a neighboring operon in the sibling genome.

If we are able to find this gene within an operon where the two operons between the genomes are similar, then this would indicate that we are able to find a match, however, it is within another operon. Now the tricky part would be to determine whether to keep the gene as a singleton or as part of an operon in the ancestral genome. In order to make this decision, BOPAL would have to check with the genomes on the neighboring branches to see if the same gene is present in the genome as a singleton or if it is part of an operon. Again, we believe that comparing with 3 neighbors would give a good indication how the gene should be handled.

Furthermore, BOPAL is able to reverse its decision if a gene is incorrectly marked as a deletion. If the same gene is deleted across multiple generations, then BOPAL will switch the gene to a duplication instead and adjust the ancestral genomes accordingly. However, BOPAL does not make the same check for duplicate genes. The same gene is marked as a duplicate across multiple generations then BOPAL will not switch the gene to a deletion. This is an area where BOPAL can be expanded upon to reverse decisions where genes are incorrectly marked as duplicates. Based on our results marking genes incorrectly as deletions has increased the accuracy but we believe that applying the same rule set to duplications would further increase the accuracy by reducing the number of genes marked as duplications incorrectly. Additionally, if a group of genes is incorrectly marked as a deletion and the genes are switched to duplications, BOPAL changes each gene individually into a duplicate which causes the number of events to increase. This design decision was made in the event we have a group of genes marked as a deletion, however, if only one of the genes from the group is deleted then only the one gene would be switched to a duplication. To resolve this we propose that BOPAL performs a scan of the events after the genes have been switched to a duplication and concatenate consecutive genes on the same operon and reduce the number of evolutionary events appropriately.

BOPAL is also able to use a singleton gene's position and its codon to identify an orthologous singleton in the other genome. This approach for the vast majority of the time leads to an optimal mapping. We propose an additional step that could potentially be added to BOPAL in the future. If there is a sequence of consecutive singleton genes in one of the genomes, these genes do not necessarily have to be

duplicates of each other, that `BOPAL` prioritize finding an orthologous mapping that minimizes the separation of these genes. In this case `BOPAL` would prioritize on keeping these genes together rather than their placement in the genome. The reason for this approach is that there could be a frame shift in one of the genomes due to deletions, duplications, or inversions, etc. However, the gene and codon would still have to match identically in order for the singleton genes to be marked as orthologs. We believe this would reduce the number of events inferred by `BOPAL` since it minimizes the separation of conserved singleton gene regions.

Although we did not consider xenologs in our evolutionary model, we propose a potential strategy to infer these events. Assuming that an HGT event could copy an operon from an unrelated genome, which is not necessarily present in the considered phylogeny, this operon would not be mapped to an ortholog in the sibling genome in *Step 1*. In *Step 2*, this operon would probably not be mapped to another operon in the same genome either, which would not allow the algorithm to infer a duplication of the operon. Currently, this would result in the method labeling this operon as lost in the sibling genome, and it would be placed back into the ancestor. However, we could then compare this operon with a neighboring genome N, to see if it actually matches. If it matches well with an operon in N, then we keep it as lost, otherwise, the algorithm could label it as being the result of an HGT, and similarly to a duplication, the operon would not be added to the ancestor.

In the future, a lot more work will be necessary to improve even more the accuracy of the events inferred. In order to accomplish that, more information will probably be necessary: exact position of the operons and singletons on the genome, intergenic distances between each pair of consecutive genes, and alignments of the flanking regions of each gene considered in the analysis are potential sources of additional information that could be leveraged. Also, similarly to the generalization of OrthoAlign to multi-OrthoAlign, it would be interesting to generalize the proposed algorithm to compute the median of three genomes, which could then be used iteratively on a phylogeny with initialized ancestors to further reduce the number of events inferred.

Bibliography

[ARC13] Sandro Andreotti, Knut Reinert, and Stefan Canzar. The duplication-loss small phylogeny problem: from cherries to trees. *Journal of Computational Biology*, 20(9):643–659, 2013.

[BDE13] Billel Benzaid, Riccardo Dondi, and Nadia El-Mabrouk. Duplication-loss genome alignment: Complexity and algorithm. In *International Conference on Language and Automata Theory and Applications*, pages 116–127. Springer, 2013.

[BE14] Billel Benzaid and Nadia El-Mabrouk. Gene order alignment on trees with multiOrthoAlign. *BMC Genomics*, 15(6):S5, 2014.

[Blu04] Thomas Blumenthal. Operons in eukaryotes. *Briefings in Functional Genomics*, 3(3):199–211, 2004.

[BPHQ07] Nicholas H. Bergman, Karla D. Passalacqua, Philip C. Hanna, and Zhaohui S. Qin. Operon prediction for sequenced bacterial genomes without experimental information. *Appl. Environ. Microbiol.*, 73(3):846–854, 2007.

[Bre88] Bonita J. Brewer. When polymerases collide: replication and the transcriptional organization of the E. coli chromosome. *Cell*, 53(5):679–686, 1988.

[CAC+09] Peter J.A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank

Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[CCM+14] Tyrrell Conway, James P. Creecy, Scott M. Maddox, Joe E. Grissom, Trevor L. Conkle, Tyler M. Shadid, Jun Teramoto, Phillip San Miguel, Tomohiro Shimada, Akira Ishihama, et al. Unprecedented high-resolution view of bacterial operon architecture revealed by RNA sequencing. *MBio*, 5(4):e01442–14, 2014.

[DDA14] Eric S. Donkor, Nicholas T.K.D. Dayie, and Theophilus K. Adiku. Bioinformatics with basic local alignment search tool (BLAST) and fast alignment (FASTA). *Journal of Bioinformatics and Sequence Analysis*, 6(1):1–6, 2014.

[DE13] Riccardo Dondi and Nadia El-Mabrouk. Aligning and labeling genomes under the duplication-loss model. In *Conference on Computability in Europe*, pages 97–107. Springer, 2013.

[DGKB19] Maria Gloria Dominguez-Bello, Filipa Godoy-Vitorino, Rob Knight, and Martin J. Blaser. Role of the microbiome in human development. *Gut*, 68(6):1108–1114, 2019.

[DNK96] Hengjiang Dong, Lars Nilsson, and Charles G. Kurland. Co-variation of tRNA abundance and codon usage in escherichia coli at different growth rates. *Journal of Molecular Biology*, 260(5):649–663, 1996.

[EKM13] Ole Kristian Ekseth, Martin Kuiper, and Vladimir Mironov. orthAgogue: an agile tool for the rapid prediction of orthology relations. *Bioinformatics*, 30(5):734–736, 2013.

[FEF09] Marco Fondi, Giovanni Emiliani, and Renato Fani. Origin and evolution of operons and metabolic pathways. *Research in Microbiology*, 160(7):502–512, 2009.

[FKA+19] Samuel C. Forster, Nitin Kumar, Blessing O. Anonye, Alexandre Almeida, Elisa Viciani, Mark D. Stares, Matthew Dunn, Tapoka T. Mkandawire, Ana Zhu, Yan Shao, et al. A human gut bacterial genome and culture collection for improved metagenomic analyses. *Nature Biotechnology*, 37(2):186, 2019.

[FL99] A.C. Frank and J.R. Lobry. Asymmetric substitution patterns: a review of possible underlying mutational or selective mechanisms. *Gene*, 238(1):65–77, 1999.

[GBE12] Yves Gagnon, Mathieu Blanchette, and Nadia El-Mabrouk. A flexible ancestral genome reconstruction method based on gapped adjacencies. In *BMC Bioinformatics*, volume 13, page S4. BioMed Central, 2012.

[GSF98] Richard Giegé, Marie Sissler, and Catherine Florentz. Universal rules and idiosyncratic features in tRNA identity. *Nucleic Acids Research*, 26(22):5017–5035, 1998.

[GVSV04] Dirk Gevers, Klaas Vandepoele, Cedric Simillion, and Yves Van de Peer. Gene duplication and biased functional retention of paralogs in bacterial genomes. *Trends in Microbiology*, 12(4):148–154, 2004.

[Hal13] Barry G. Hall. Building phylogenetic trees from molecular data with mega. *Molecular Biology and Evolution*, 30(5):1229–1235, 2013.

[HC03] Xiaoqiu Huang and Kun-Mao Chao. A generalized global alignment algorithm. *Bioinformatics*, 19(2):228–233, 2003.

[HG02] Gretchen Hagen and Tom Guilfoyle. Auxin-responsive gene expression: genes, promoters and regulatory factors. *Plant Molecular Biology*, 49(3-4):373–385, 2002.

[HSAE13] Patrick Holloway, Krister Swenson, David Ardell, and Nadia El-Mabrouk. Ancestral genome organization: an alignment approach. *Journal of Computational Biology*, 20(4):280–295, 2013.

[JPSM60] François Jacob, David Perrin, Carmen Sánchez, and Jacques Monod. Operon: a group of genes with the expression coordinated by an operator. *Comptes rendus hebdomadaires des seances de l'Academie des sciences*, 250:1727–1729, 1960.

[JRTC12] Bradley R. Jones, Ashok Rajaraman, Eric Tannier, and Cedric Chauve. ANGES: reconstructing ANcestral GEnomeS maps. *Bioinformatics*, 28(18):2388–2390, 2012.

[JRWK02] I. King Jordan, Igor B. Rogozin, Yuri I. Wolf, and Eugene V. Koonin. Essential genes are more evolutionarily conserved than are nonessential genes in bacteria. *Genome Research*, 12(6):962–968, 2002.

[KAS19] Maciej Kaczmarek, Simon V. Avery, and Ian Singleton. Microbes associated with fresh produce: Sources, types and methods to reduce spoilage and contamination. *Advances in Applied Microbiology*, 107:29–82, 2019.

[KDS00] Joel A. Klappenbach, John M. Dunbar, and Thomas M. Schmidt. rrna operon copy number reflects ecological strategies of bacteria. *Appl. Environ. Microbiol.*, 66(4):1328–1333, 2000.

[LL05] Dennis V. Lavrov and B. Franz Lang. Transfer RNA gene recruitment in mitochondrial DNA. *Trends in Genetics*, 21(3):129–133, 2005.

[LSR03] Li Li, Christian J. Stoeckert, and David S. Roos. Orthomcl: identification of ortholog groups for eukaryotic genomes. *Genome Research*, 13(9):2178–2189, 2003.

[MM06] John S. Mattick and Igor V. Makunin. Non-coding rna. *Human Molecular Genetics*, 15(suppl_1):R17–R29, 2006.

[MMZ+13] Xizeng Mao, Qin Ma, Chuan Zhou, Xin Chen, Hanyuan Zhang, Jincai Yang, Fenglou Mao, Wei Lai, and Ying Xu. Door 2.0: presenting operons and their functions through dynamic and integrated views. *Nucleic Acids Research*, 42(D1):D654–D659, 2013.

[MZS+06] Jian Ma, Louxin Zhang, Bernard B. Suh, Brian J. Raney, Richard C. Burhans, W. James Kent, Mathieu Blanchette, David Haussler, and Webb Miller. Reconstructing contiguous regions of an ancestral genome. *Genome Research*, 16(12):1557–1565, 2006.

[Pai96] Virginia M. Pain. Initiation of protein synthesis in eukaryotic cells. *European Journal of Biochemistry*, 236(3):747–771, 1996.

[PCLT19] Tomasz Pawliszak, Meghan Chua, Carson K. Leung, and Olivier Tremblay-Savard. Operon-based approach for the inference of rRNA and tRNA evolutionary histories in bacteria. *BMC Genomics (in press)*, 2019.

[Pet03] Howard R. Petty. Overview of the physical state of proteins within cells. *Current Protocols in Protein Science*, 31(1):1–5, 2003.

[PHAA05] Morgan N. Price, Katherine H. Huang, Eric J. Alm, and Adam P. Arkin. A novel method for accurate operon predictions in all sequenced prokaryotes. *Nucleic Acids Research*, 33(3):880–892, 2005.

[PVBO15] Amandine Perrin, Jean-Stéphane Varré, Samuel Blanquart, and Aïda Ouangraoua. ProCARs: Progressive reconstruction of ancestral gene orders. *BMC Genomics*, 16(5):S6, 2015.

[PWW+19] Anutthaman Parthasarathy, Narayan H. Wong, Amanda N. Weiss, Susan Tian, Sara E. Ali, Nicole T Cavanaugh, Tyler M. Chinsky, Chelsea E Cramer, Aditya Gupta, Rakshanda Jha, et al. SELfies and CELLfies: Whole genome sequencing and annotation of five antibiotic resistant bacteria isolated from the surfaces of smartphones, an inquiry based laboratory exercise in a genomics undergraduate course at the rochester institute of technology. *Journal of Genomics*, 7:26, 2019.

[RBG10] Hubert H. Rogers, Casey M. Bergman, and Sam Griffiths-Jones. The evolution of tRNA genes in Drosophila. *Genome Biology and Evolution*, 2:467–477, 2010.

[Roc04] Eduardo P.C. Rocha. The replication-related organization of bacterial genomes. *Microbiology*, 150(6):1609–1627, 2004.

[SSA98] Margaret E. Saks, Jeffrey R. Sampson, and John Abelson. Evolution of a transfer RNA gene through a point mutation in the anticodon. *Science*, 279(5357):1665–1670, 1998.

[SW92] Michael Schöniger and Michael S. Waterman. A local algorithm for dna sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54(4):521–536, 1992.

[TBBT15] Tam T.T. Tran, Hassiba Belahbib, Violaine Bonnefoy, and Emmanuel Talla. A comprehensive tRNA genomic survey unravels the evolutionary history of tRNA arrays in prokaryotes. *Genome Biology and Evolution*, 8(1):282–295, 2015.

[TBLE15] Olivier Tremblay-Savard, Billel Benzaid, B Franz Lang, and Nadia El-Mabrouk. Evolution of tRNA repertoires in bacillus inferred with OrthoAlign. *Molecular Biology and Evolution*, 32(6):1643–1656, 2015.

[TECM18] Blanca Taboada, Karel Estrada, Ricardo Ciria, and Enrique Merino. Operon-mapper: a web server for precise operon identification in bacterial and archaeal genomes. *Bioinformatics*, 34(23):4118–4120, 2018.

[TR84] Diethard Tautz and Manfred Renz. Simple sequences are ubiquitous repetitive components of eukaryotic genomes. *Nucleic Acids Research*, 12(10):4127–4138, 1984.

[WWD06] Mike Withers, Lorenz Wernisch, and Mario Dos Reis. Archaeology and evolution of transfer rna genes in the Escherichia coli genome. *RNA*, 12(6):933–942, 2006.

[ZSF+02] Yu Zheng, Joseph D. Szustakowski, Lance Fortnow, Richard J Roberts, and Simon Kasif. Computational identification of operons in microbial genomes. *Genome Research*, 12(8):1221–1230, 2002.