# ALGORITHMS FOR LIBRARY-BASED MICROBIAL SOURCE TRACKING

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Aldrin Montana

June 2013

COMMITTEE MEMBERSHIP

| | |
|---|---|
| TITLE: | Algorithms for Library-based Microbial Source Tracking |
| AUTHOR: | Aldrin Montana |
| DATE SUBMITTED: | June 2013 |
| COMMITTEE CHAIR: | Alexander Dekhtyar, Ph.D.<br>Professor, Computer Science |
| COMMITTEE MEMBER: | Chris Lupo, Ph.D.<br>Assistant Professor, Computer Science |
| COMMITTEE MEMBER: | Tim Kearns, Ph.D.<br>Associate Professor, Computer Science |
| COMMITTEE MEMBER: | Michael Black, Ph.D.<br>Professor, Biology |

**Abstract**

Algorithms for Library-based Microbial Source Tracking

Aldrin Montana

Pyroprinting is a novel, library-based microbial source tracking method developed by the Biology department at Cal Poly, San Luis Obispo. This method consists of two parts: (1) a collection of bacterial fingerprints, called pyroprints, from known host species, and (2) a method for pyroprint comparison. Currently, Cal Poly Library of Pyroprints (CPLOP), a web-based database application, provides storage and analysis of over 10000 pyroprints. This number is quickly growing as students and researchers continue to use pyroprinting for research. Biologists conducting research using pyroprinting rely on methods for partitioning collected bacterial isolates into bacterial strains. Clustering algorithms are commonly used for bacterial strain analysis of organisms in computational biology. Unfortunately, agglomerative hierarchical clustering, a commonly used clustering algorithm, is inadequate given the nature of data collection for pyroprinting. While the clusters produced by agglomerative hierarchical clustering are acceptable, pyroprinting requires a method of analysis that is scalable and incorporates useful metadata into the clustering process. We propose ontology-based hierarchical clustering (OHClust!), a modification of agglomerative hierarchical clustering that expresses metadata-based relationships as an ontology to direct the order in which hierarchical clustering algorithms analyze the data. In this thesis, the strengths and weaknesses of OHClust! are discussed, and its performance is analyzed in comparison to agglomerative hierarchical clustering.

# Acknowledgements

This work was supported in part by grants from **The W.M. Keck Foundation** and **The National Science Foundation**.

Special thanks to:

- The department staff, especially Christy Zolla, both *the Goddess of Wisdom and Light* and my CSC mom, for making the bureaucratic process a cynch.
- My family and friends, particularly my mom, step dad, and dad, for **all** that they've done to support me.
- The great and wonderful Jen VanderKelen and Emily Neal for being awesome and patient in working with me.
- Chris Kitts and Michael Black for being fun and enjoyable to work with. The peanut gallery and poop jokes made working with biologists such a great time.
- Anya Goodman for still working with me after four years. Especially for being really great (my favorite biochemist) and always making me feel like an amazing person/wizard.
- Alex Dekhtyar for being the best adviser *ever*. I believe all of my growth can be attributed to your guidance. I cannot thank you enough for the wisdom, knowledge, time, and fun that I've had for the last four years.
- квас for always being there to soothe the soul.
- My roommates who have always provided me with a family away from home. Our unity is strong.
- Dr. Lupo's awesome equipment which made the evaluations present in this thesis attainable.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

To maintain the cleanliness of current "fishable and swimmable" bodies of water, and improve the cleanliness of other bodies of water, health and environmental protection agencies have been interested in the ability to track sources of fecal contamination at the species level [39, 42, 8]. Identifying the animal species from which a fecal contamination originates is an important first step in controlling fecal contamination and managing bacterial risks. It enables natural resource managers to address the problem of fecal contamination at its source. Investigation of fecal contamination is especially necessary in waters used for human recreation where dangerous pathogens are transferable.

Within the last decade, several techniques of identifying and descriminating between various sources of fecal bacteria in an environment have been studied and developed [37, 16, 24, 32, 7, 5]. This field of study, *Microbial source tracking* (MST) dates back to the 1960s with the use of fecal coliform-to-fecal streptococci ratios. Many MST techniques rely on the genotypic or phenotypic analyses of fecal indicator bacteria (FIB) such as total coliforms, fecal enterococci, or fecal coliforms (e.g. *Eschrichia coli*). FIB are bacteria that are found in the intesti-

nal tracts of various host animals, and can be found in the host animal's stool. When found, FIB can be used to indicate or approximate the level of fecal contamination in, or near, a body of water [42]. In 1999, the first review of MST techniques was published by Debby Sargeant [36], followed by another review in November 2011 [37]. In her 2011 review of current MST methods, she describes the MST field as a new, experimental field that has been continuously studying and developing new methods over several years [37]. Fortunately, there have been publications that provide notable overviews of MST techniques and applications [13, 10, 14] within the last couple of years. Despite the highly experimental nature of MST at this phase of its development, Sargeant claims that there is strong pressure on natural resource managers to use these techniques to identify bacterial sources of pollution [37].

This thesis addresses the comparison of DNA fingerprints of bacterial isolates for a new library-dependent MST technique developed by Dr. Black and Dr. Kitts of the Cal Poly Biology department. As described by Sargeant, a library-dependent MST technique relies on a database, called the library, which contains genotypic information of the FIB being studied [37]. Recently developed library-dependent methods have shifted significantly towards genotypic analysis of collected bacteria, away from phenotypic analysis. Genotypic analysis strives to separate observed bacterial samples by differentiating between bacteral strains based on fingerprints of the bacterial genome. When compared to phenotypic methods, genotypic methods are advantageous since they use differences in nucleic acids to distinguish between strains of *E. coli* with greater sensitivity [1, 12, 31, 39, 42]. Library-dependent MST techniques rely on three major factors: the definition of a *bacterial strain*, the relationship between bacterial strains and host species of origin, and the size of the library.

The definition of a bacterial strain is very important for MST techniques. Unfortunately, the definition of a strain varies between research groups and the research task at hand. Thus, the level of similarity required for two organisms to be considered part of the same strain is defined and established differently for each research group. The selection of an appropriate FIB is important and will affect interpretations of analyses differently. For the research group at Cal Poly, San Luis Obispo, the FIB of choice is *E. coli* and a bacterial strain is defined as a group of isolates that have similar DNA fingerprints. DNA fingerprints of *E. coli* isolates are stored in the library. When *E. coli* isolates are compared, they are considered part of the same strain in our assay if their DNA fingerprints are considered similar.

The idea behind library-dependent MST techniques is to utilize the relationship between bacterial strains and various host species. For example, it is assumed that of the many different kinds of *E. coli*, some are only present in particular species, some can only be found in particular regions of the world, while others still may be present in many different species in many regions. Therefore, it is important to understand how the relationship between bacterial strains and host species is expressed in the environment being studied. For this reason, the library is populated with bacterial isolates that come from known host species. In this way, the library provides biologists with a knowledge base of the *E. coli* populations present in various host species in the environment. Then, newly collected *E. coli* isolates of unknown origin are compared to the library to determine the *E. coli* strains to which they belong. With a larger amount of collected *E. coli* isolates, the confidence of an *E. coli* strain being correctly associated with a particular host species increases.

Library-dependent MST techniques are incredibly dependent on the size of the

library. The amount of isolates required for the library to be useful varies based on the size and diversity of the environment being studied. Sargeant et al. [37] reference Jenkins et al. [19] when describing a suggested library size. Jenkins et al. suggested a library of 900–2000 fecal isolates to represent the number of transient and resident *E. coli* ribotypes for two cattle herds. However, if multiple fecal sources are present, the library size should be in the thousands in order to obtain a good representation of the fecal isolates present in the environment being studied [37].

As an overview, the workflow for a library-dependent MST technique starts with the collection of a large number of isolates, of a particular FIB, from a variety of known host species. The collection of bacterial isolates from known host species must continue until the library is sufficiently large (900–2000 fecal isolates or more depending on environment being studied). Once the library has become sufficiently large, it is possible to meaningfully determine the strains that collected isolates belong to. As isolates are collected, genotypic analysis is applied and resulting data is stored in the library. Genotypic information provides the basis for differentiating bacterial strains present in the environment being studied. Then, isolates collected from the environment, with unknown host species origins, are compared to isolates in the library for determining the bacterial strain to which they belong. Using this information, it is possible to associate member isolates of a bacterial strain with appropraite host animal species from which they originate.

Interestingly, Sargeant claims that the MST field is moving away from library-dependent methods and towards library-independent methods [37]. While the discussion of this topic is outside the scope of this thesis, it's worthwhile to mention that the reason for this is that library-dependent methods require a significant

effort to build an appropriately large, useful library. The library also needs to include fingerprints relative to the environment being investigated. However, it is currently a more reliable, flexible MST technique that allows for the association of FIB to host species other than humans and some domestic animal species [37]. Additionally, a library is able to allow the analysis and observation of bacterial strains in various regions and over time.

Dr. Kitts and Dr. Black of the Biology department at Cal Poly, San Luis Obispo, developed a library-dependent MST method, *pyroprinting*, shortly after a project done for the city of Pismo Beach [21, 20]. Pyroprinting follows the standard framework for library-dependent MST methods (described previously). However, pyroprinting uses pyroprints to represent genotypic information of bacterial isolates in the library, described more in Section 2.3. The contributions of this thesis consist of three software artifacts for supporting the pyroprinting process and pyroprint analysis:

- A tool for analyzing the pyroprinting process *in-silico*.
- An interim tool used to analyze pyroprints prior to the creation of a pyroprint database application.
- A new clustering algorithm, ontological hierarchical clustering (OHClust!), which is based on agglomerative hierarchical clustering, that was designed to effectively use bacterial isolate and pyroprint metadata for more meaningful and scaleable analysis.

The work described in this thesis has been used by M.S. and B.S. students in Biology for various types of bacterial research (for simplicity, members of the research group using pyroprinting are generally referred to as *the biologists*).

Since the Pismo Beach MST study, the various research projects done by Cal Poly students has extended to longitudinal, population-based studies [29, 3, 30, 44] and studies of bacterial transferrance between host species [9].

The first contribution described in this thesis is a tool that assists in the selection of a dispensation sequence for pyroprinting, *pyroprint dispensation analysis tool* (PyroprintDAT). Initially, the biologists needed a way to select a dispensation sequence that maximizes the effectiveness of pyroprinting for differentiation of bacterial strains. PyroprintDAT was developed to characterize permutations of cycles of nucleotides on given *E. Coli* rRNA sequences. This tool and its impact is detailed in Chapter 3. Once a dispensation sequence had been determined, the biologists' needs shifted towards a tool that could provide analysis of pyroprint data.

As a library-dependent MST technique, pyroprinting is best supported by a web-based database application that stores, retrieves, and analyzes isolates, associated pyroprints, and all other relevant information. *Cal Poly Library of Pyroprints* (CPLOP) is a web-based application developed for the storage and retrieval of pyroprints [43]. Prior to the development of CPLOP, however, pyroprint data was maintained in spreadsheets. During this time, a tool that could take spreadsheet data as input was developed that could perform clustering. Chapter 3 only briefly describes this tool. While it was necessary at the time, it was later absorbed into the third, and final, tool, *suite of pyroprint analysis methods* (SPAM). SPAM is a tool containing an implementation of both agglomerative clustering and OHClust!. SPAM was designed to be modular and therefore easily maintainable. The implementation and design of SPAM is described in Chapter 6.

The third and primary contribution of this thesis is a new clustering algo-

rithm called OHClust!. Chapters 4, 5, 6, and 7 focus on the development and characterization, related work, implementation, and evaluation, respectively, of OHClust!.

The following chapter, Chapter 2, fully explains the context of this thesis, and the research it supports, by providing a narrative that describes pyrosequencing, a method of DNA sequencing, and pyroprinting, an adaptation of pyrosequencing for creating DNA fingerprints. Generally speaking, the pyroprinting method may be abstracted away from the clustering algorithm and implementation of SPAM included in this thesis. The knowledge is perhaps only necessary in a minimal form, as a method that takes DNA and produces a vector-based fingerprint, for understanding PyroprintDAT. Although extensive knowledge of pyroprinting is unnecessary, a great deal of this thesis involves the support provided to the biologists in the form of developed tools. As such, an understanding of pyroprinting, and therefore pyrosequencing, is necessary to understand the context and role of these tools. As an example, PyroprintDAT provides analysis of dispensation sequences, which are sequences of particular importance in the pyroprinting process. For this reason, while the approach used by PyroprintDAT is itself straightforward, the reason and motivation for the analysis that PyroprintDAT applies only makes sense in regards to the pyroprinting method. Additionally, the explanation of pyroprinting is important in understanding the relevant objects that are represented in SPAM. Chapter 2 also provides additional information for understanding the data being stored and analyzed in CPLOP and motivates the application of clustering to address the needs of the biologists.

# Chapter 2

# Background

Pyroprinting has been developed to address some of the common complaints regarding methods of strain differentiation. Current genotypic methods can be labor-intensive or financially expensive, and many have issues in reproducibility when identifying known *E. coli* strains [12, 39, 42]. These issues are especially evident in longitudinal, single-host studies of microbial strains [42, 1, 38]. In these studies, bacterial cultures are typically grown from samples collected in roughly even intervals (daily, weekly, etc.) from a single individual. These studies are concerned with the total number of strains of a particular bacterial species (e.g. *E. coli*) present in a host at a given moment in time, strain turnover over time, the presence of dominant strains, change of strain dominance over time, as well as corresponding cause-and-effect questions (i.e. what causes all these changes) [1, 4, 40, 42].

Several students in the Biology department have conducted research projects focusing on the use and applicability of pyroprinting. Among the various research projects, one student, Emily Neal, has piloted and established a novel framework for genotypic microbial analysis of bacteria (so far applied to *E. coli*) in longitu-

dinal studies [27, 28]. This framework incorporates pyroprinting and an *in-silico* strain differentiation method based on OHClust!

## 2.1 Bacterial Sampling and Data Collection

The process of sampling begins with the selection of stool, or fecal matter. Before the library of isolates and genotypic information is sufficiently large, samples must be collected from the stool of known animal species. After the library has become sufficiently large, samples may be collected from stool of unknown origin. From the stool, a number of samples are collected, isolated, and cultured in preparation for polymerase chain reaction (PCR) and pyroprinting. Each of the bacterial samples are then streaked onto MacConkey agar plates for single isolated colonies (isolate for short) and grown overnight. Bacterial isolates are confirmed for the chosen FIB at the end of the process. Any isolates that are not of the chosen FIB are left unused. This process is highlighted in a previous submission to the CSU research competition and CSUPERB [28, 29] which describe a study where *E. coli* was sampled once a month over a six month period from three individuals–two females and one male–ranging in age from 20 to 25.

The sampling process is extremely important for MST. It is during the sampling process that necessary contextual information is recorded, including the host species from which the stool originates, the location of the stool, and the time and date of sampling. This extra information conveys understanding of the bacterial strains present in the environment being studied, as well as possible sub-relationships that exist between host species or within eco-systems contained in the environment. In addition to the relevance to analysis, contextual information is important for directing future sampling efforts. To build a library suitable

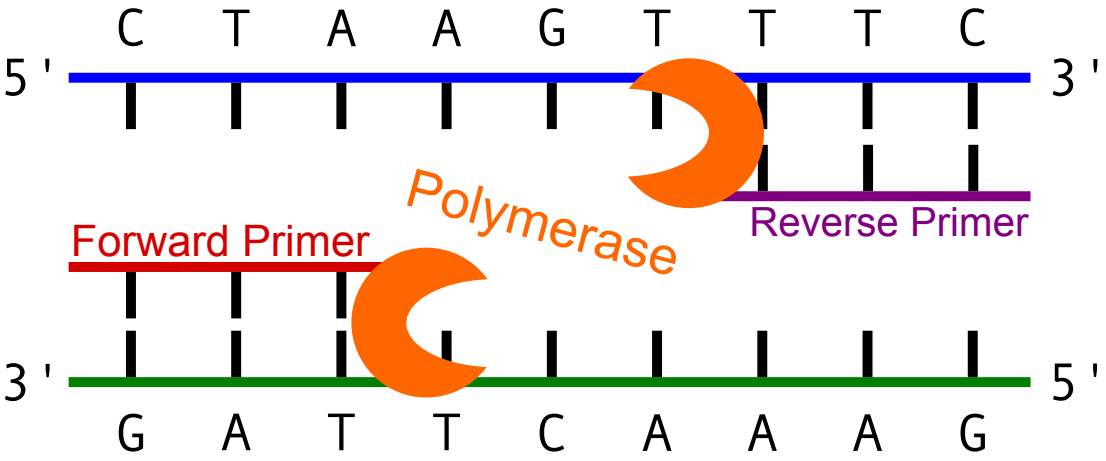



Figure 2.1: A basic depiction of PCR, with polymerase in orange, forward primer in red, and reverse primer in purple.

for pyroprinting in an environment, it is important to collect a large amount of samples from many of host species and locations contained in the environment.

## 2.2 Pyrosequencing

Polymerase chain reaction (PCR) is applied to the DNA being analyzed prior to pyrosequencing. PCR is a necessary first step for many types of DNA analysis and sequencing methods. Typically, PCR is used for DNA amplification, a process for making millions of copies of a target DNA region. While the details of the PCR process are not necessary for understanding pyrosequencing, it is necessary to know that the PCR process requires two, typically short, nucleotide sequences known as *primers*. Conceptually, a forward primer binds to the target DNA region on the bottom strand (3' end to 5' end) and a reverse primer binds to the target DNA region on the top strand (5' end to 3' end). The two primers provide a start site for the polymerase enzyme to grow a new strand based on the sequence of the other strand, called the *template strand*. This can be seen

in Figure 2.1. The importance of the primers is described in Section 2.4. After multiple rounds of replicating the DNA by PCR, the amplified product is ready for pyrosequencing.

Pyrosequencing is a DNA sequencing process where a new strand of DNA, which we will refer to as the *sequencing strand*, is built incrementally, complementary to the template strand. These components are highlighted in Figure 2.2. Ronaghi, et al. describe the pyrosequencing process in more detail [34]. In the Biology department at Cal Poly, San Luis Obispo, the PyroMark Q24 from Qiagen (see Figure 2.3) is used for the pyrosequencing process. This machine performs the sequencing reactions on a plate containing 24 wells. A pyrosequencing *run* refers to the entirety of the pyrosequencing process for a single plate. While only one plate can be processed per run, each well on the plate contains a separate pyrosequencing reaction. For the PyroMark Q24, this means that a single run can pyrosequence 24 separate reactions. During a run, nucleotides (with bases **A**denine, **T**hymine, **C**ytosine, or **G**uanine) are dispensed into each well of the plate.

The order in which nucleotides are dispensed is determined by a parameter called the *dispensation sequence*. Once a nucleotide is dispensed, it binds with the next available unbound nucleotide of the template strand if, and only if, the

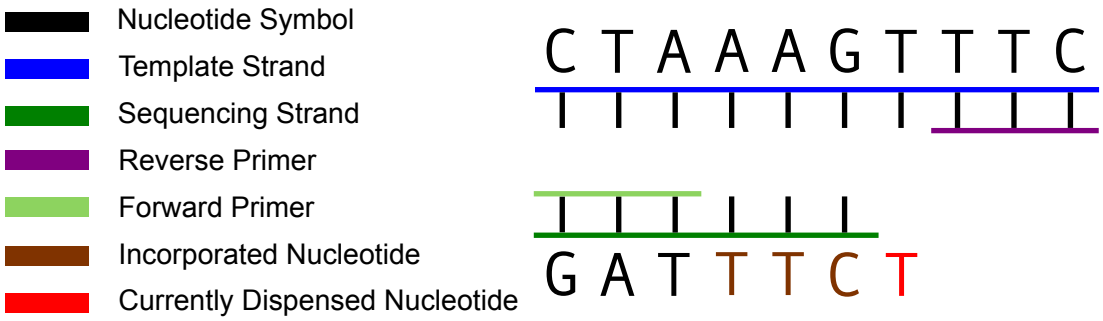



Figure 2.2: The parts of DNA during pyrosequencing.

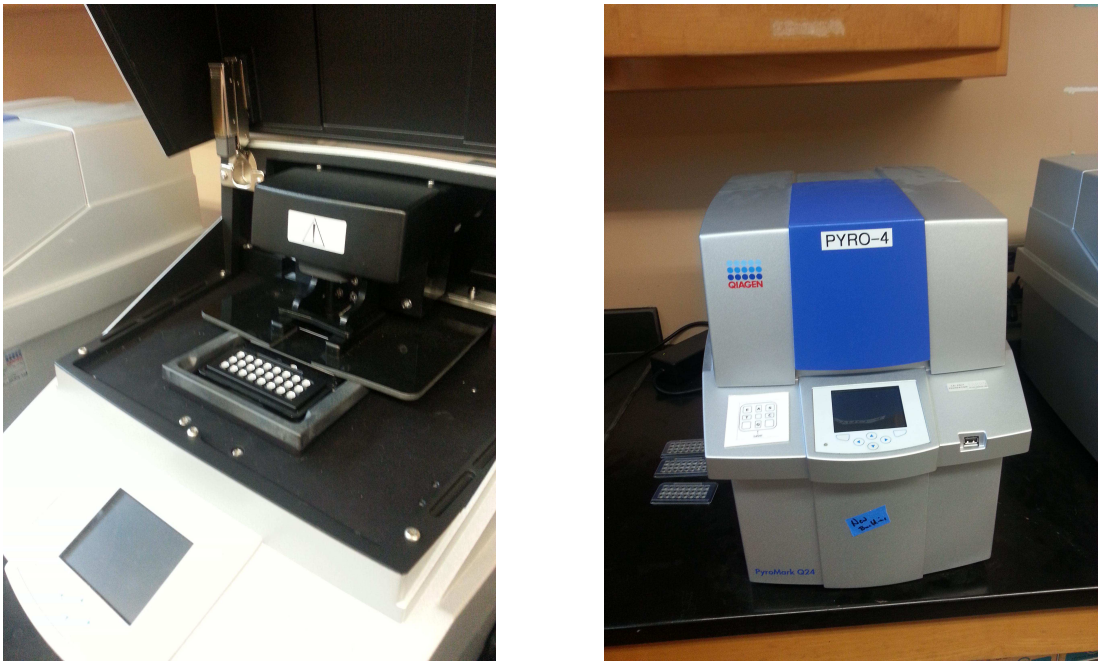
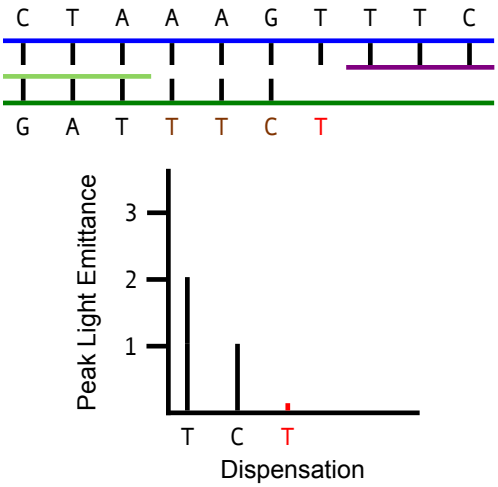
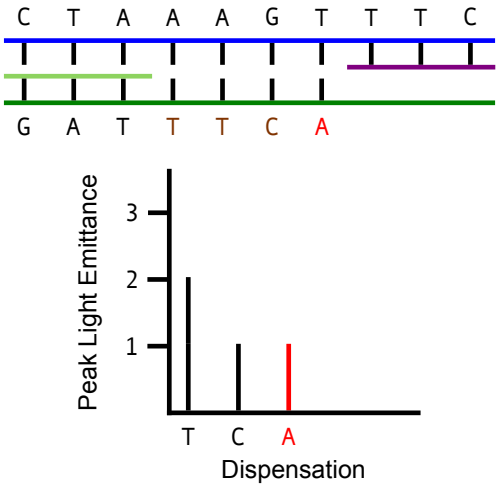



Figure 2.3: Cal Poly's Qiagen PyroMark Q24 pyrosequencer.

nucleotide bases are complementary–**A** binds with **T** and **C** binds with **G**. If the dispensed nucleotide binds to the template strand, it is incorporated into the sequencing strand. The incorporation of the dispensed nucleotide involves a series of reactions which result in the emittance of light. Any remaining nucleotide not incorporated is enzymatically destroyed and the next nucleotide in the dispensation sequence is dispensed. When the dispensed nucleotide is incorporated into the sequencing strand, the observed light emittance is proportional to the amount of nucleotides incorporated into the sequencing strand, as seen in Figure 2.4. For example, if **G** is incorporated into the sequencing strand and the light emittance is measured at 100 units and **T** is incorporated into the sequencing strand with light emittance 300 units, then the template strand has three times as many **T** nucleotides following a series of **G** nucleotides. When pyrosequencing, these proportions can be used to directly figure out the content of the template strand.




(a) No light is emitted when the dispensed nucleotide is not incorporated into the sequencing strand.




(b) The light emitted when the dispensed nucleotide is incorporated into the sequencing strand is proportional to the amount of nucleotides incorporated.

Figure 2.4: Depictions of pyrosequencing and the construction of a pyrogram.

However, as further described in Section 2.3, this is not true for pyroprinting.

The pyrosequencing machine measures the light emitted from the binding reaction of the dispensed nucleotide and the template DNA strand. The light emittance is then recorded as discrete values starting at the moment of dispensation. Light emittance is recorded until the moment when the chemical reaction between the dispensed nucleotide and the template strand completes. These light emittance values are used to construct a *pyrogram*, a graph that plots light emittance of DNA synthesis against units of time and the dispensation sequence. In the pyrogram pictured in Figure 2.5, light emittance (y-axis) is plotted against moments, which fall within a dispensation. When a nucleotide is dispensed, the light emitted increases rapidly in the initial moments, then subsides as the reac-

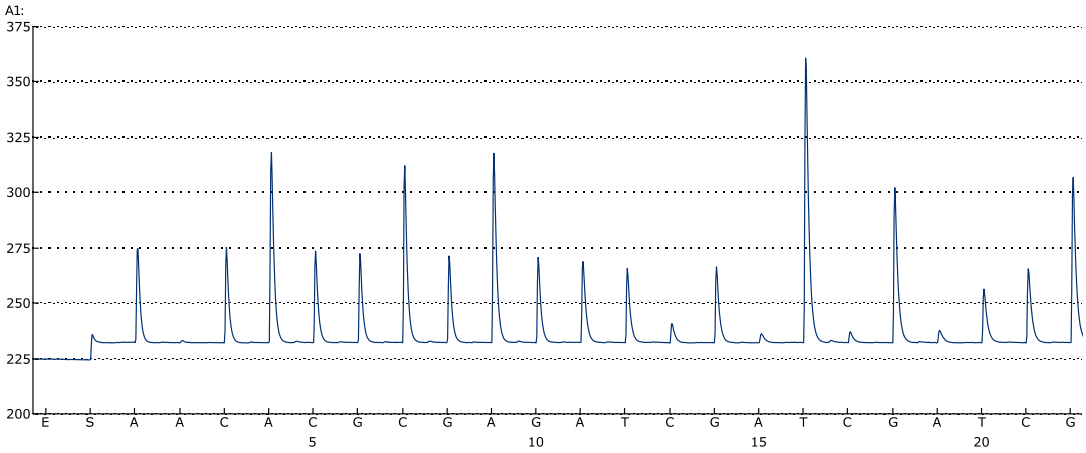



Figure 2.5: A pyrogram up to dispensation 22 (out of 104). The y-axis is light emittance, the x-axis is dispensation over time.

tion completes.

Each light emittance value recorded during the reaction corresponds to a *moment*–a unit of time that is defined internally to either the pyrosequencer software or hardware. Each moment is associated with the dispensed nucleotide actively reacting in the well. For the purposes of analysis, three scalar light emittance values are associated with each dispensation, and so a pyrogram is abstractly similar to a histogram. Figure 2.6 show three distinct properties of the light emittance values recorded during each dispensation which are used to represent pyrosequencing results:

- Peak Height – The maximum light emittance during a dispensation
- Peak Width – The amount of time light was emitted during a dispensation.
- Peak Area – The total amount of light emitted during a dispensation.

For the work in this thesis, peak heights are the only values used to represent each dispensation, or position, of a pyroprint. Based on brief studies, the biologists

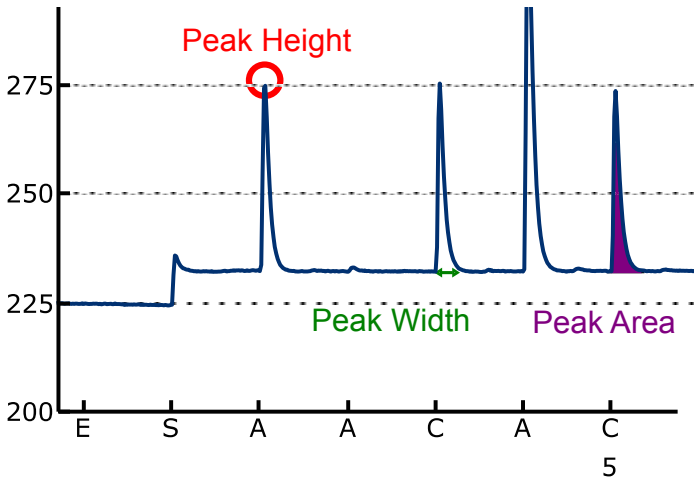



Figure 2.6: A pyrogram with peak height, peak width, and peak area annotations included.

decided that peak height is the preferred light emittance value, and so peak width and peak area, while maintained in the database, were not used for analysis.

Modern pyrosequencing equipment allows for dispensation sequences of up to 200 nucleotides in length, however, the quality of the sequencing process deteriorates beyond 100–120 dispensations. The reasons for this deterioration include build up of enzymes and chemicals in the well, hardware error such as splashes, human error in plate preparation, and others. For longer DNA sequences, overlapping reads of 100–120 nucleotides must be made and assembled together.

Although there is sequencing technology cheaper and capable of longer DNA sequence reads, known as next-generation sequencing (NGS) [25], newer sequencing technology has a high initial cost and is generally less academically accessible. Compared to other sequencing technologies, however, pyrosequencing is cheap, relatively quick, and has a lower initial cost. For this reason, the Biology department adapted pyrosequencing to be used for generating a DNA sequence based pattern that can be used as a molecular fingerprint to differentiate bacte-


Figure 2.7: A depiction of the 16S, 23S, 5S genes and their contained ITS regions.

rial strains. This process, called pyroprinting, leverages the fact that introduced nucleotides will bind with the first available nucleotide of the template strand without regard to the population of the templates in the reaction. While pyrosequencing demands that all of the templates be identical, pyroprinting is based on different templates being sequenced simultaneously.

## 2.3 Pyroprinting

Pyroprinting is an MST technique that generates pyroprints (DNA fingerprints) through the use of pyrosequencing. Constructed pyroprints are output from the pyrosequencer as pyrograms consisting of light emittances for each nucleotide dispensation based on the DNA from highly variable regions of the microbial genome. These regions, called *intergenic transcribed spacers* (ITS), are non-coding regions of the genome located between two highly conserved genes. There are two ITS regions, shown in Figure 2.7, used in the analysis of *E. coli* isolates in this thesis: 16S–23S and 23S–5S.

The 16S–23S ITS region, located between the 16S and 23S genes, is a widely used ITS region for identifying bacterial strains [2, 35, 45] in MST applications. Although less commonly used, the 23S–5S ITS region is another well-known ITS region. These ITS regions are used in pyroprinting by the Cal Poly Biology


Figure 2.8: Two sets of seven DNA sequences from unique copies of the 16S–23S (green) and 23S–5S (blue) ITS regions. Variation in the DNA sequences is shown in Purple.

department because both of these regions of DNA are non-coding. Non-coding regions of DNA accumulate more variability, or mutations, due to the absence of selective pressures that maintain the sequence in regions that encode functional RNA or proteins.

While a pyrogram is typically constructed to represent the sequence of a single template, a pyroprint is constructed to represent the mixed sequence of several, possibly different, templates. As pictured at the right in Figure 2.8, the *E. coli* genome has seven copies of both the 16S–23S and 23S–5S ITS regions, each with potentially different DNA sequences. Example DNA sequences for the 16S–23S and 23S–5S ITS regions are pictured on the left in Figure 2.8. The 16S–23S ITS template strands are colored green, whereas the 23S–5S ITS template strands are colored blue. Further, nucleotide positions that vary between template strands are colored purple to show that the variability between replicates of the 16S–23S ITS region differ from the variability between replicates of the 23S–5S ITS region.

The primary difference between pyroprinting and pyrosequencing is that pyrosequencing, using only one template strand, is used to determine the content

of a DNA sequence. Pyroprinting, using a mix of template strands, is unable to provide direct information regarding the content of each DNA sequence analyzed. However, pyroprints represent an aggregate of several sequencing strands, thus providing information about the variation within the population of different sequences when compared to other pyroprints. The variation in the ITS regions can manifest between ITS regions of different bacterial isolates and also between ITS region replicates in the same genome. This variability can be observed as polymorphisms in the sequence of the DNA and in the ratios of the different sequences represented in the bacteria. Each ITS region has varying degrees of variability depending on the choice of fecal coliform. By pyrosequencing multiple template strands in a single well, pyroprinting effectively uses the several copies of each ITS region present in the microbial genome to create useful DNA fingerprints despite the length limitation of pyrosequencing reads.

An illustration of the pyroprinting process and the use of several sequencing strands is depicted in Figure 2.9. Sequencing strands are at the top of the figure, while the pyroprint is at the bottom of the figure. Nucleotides in the sequencing strand that are not reached by the dispensation sequence are crossed out with a black line. The colored lines between nucleotides in the template sequences show what nucleotides are added as dispensations occur. The bars in the pyroprint at the bottom are colored to make it easy to relate the light emitted during a dispensation to the corresponding position in the sequencing strands.

## 2.4 Data and Digital Representation

In this thesis, the term **data point** is used to refer to a singular, usually complex, object. More generally, a data point is an individual record or object

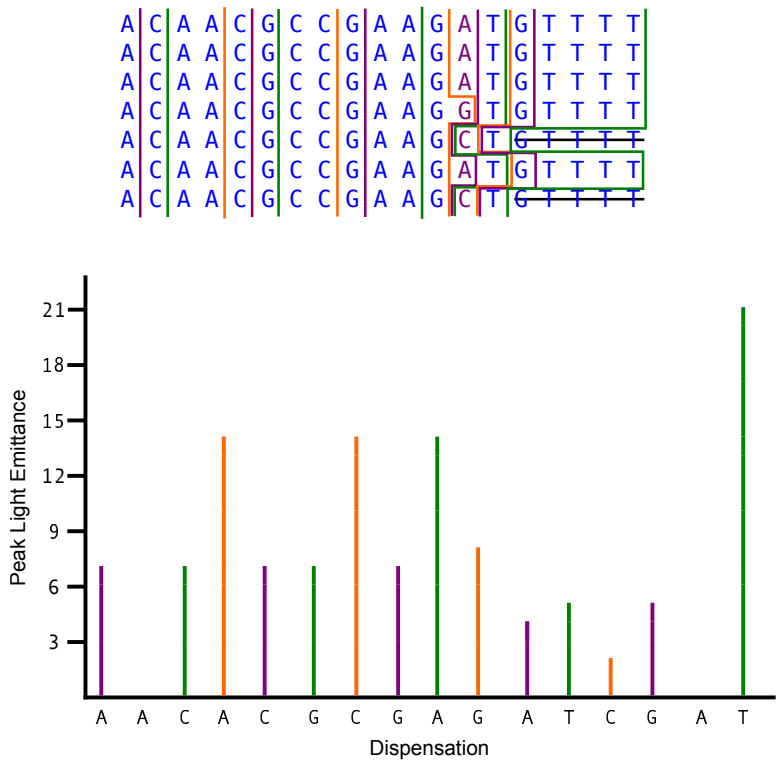



Figure 2.9: An example pyroprint constructed from seven sequencing strands in the 23S–5S ITS region.

of data that can be compared and grouped with other individual records or objects of data. Prior to this section, bacterial isolates, pyroprints, ITS regions, light emittance, and various other forms of information relevant to MST and pyroprinting were introduced and described. Typically, only bacterial isolates and pyroprints are complex entities of interest which we consider data points. In this section, the digital representations of these biological entities are described and formally defined.

There are two primary biological entities being analyzed–bacterial isolates and pyroprints. Bacterial isolates, or sometimes simply referred to as isolates, are related to pyroprints via ITS regions. Isolates have two ITS regions of interest that exist in several copies in the genome. Each set of copies of a given ITS

region, together, may be pyroprinted multiple times, and pyroprints consist of many light emittance values and a dispensation sequence. This indicates a one-to-many relationship between isolates and pyroprints.

Computationally, bacterial isolates are represented as an object containing a set of ITS regions. Here, an ITS region abstractly represents the set of copies of the ITS region in the genome. While there are only two ITS regions used in our analysis, our digital representation allows for any number of ITS regions to be used. An ITS region is represented as an object containing a set of pyroprints constructed from a mix of copies of the ITS region. A pyroprint is represented as an object containing a shorthand of the dispensation sequence used during pyroprinting, and a list of floating point values which represent the peak height light emittance values. While pyroprints serve as the basis on which isolate similarity is determined, the relationships between pyroprints and isolates is important for computing meaningful and appropriate similarities.

When comparing biological entities, it is important that the comparison only be made in a biologically meaningful context. For example, *E. coli* isolates should only be compared to other *E. coli* isolates. It is only meaningful to compare bacterial isolates of the same species. For pyroprints, there is a quintuple (5-tuple) of parameters that should be identical for the comparison to be biologically meaningful. The 5-tuple of parameters is called a *protocol* and represents the context in which pyroprints were constructed. Formally, a protocol is defined as follows:

$$\mathcal{T} = \langle R_k, d, Pr_f, Pr_r, Pr_s \rangle$$

The ITS region $R_k$ determines the actual DNA being targeted for the pyroprinting process. Ultimately, the other four variables are determined so that PCR and pyrosequencing can be applied to the target sequence contained in the ITS re-

gion. The dispensation sequence $d$ determines the order in which nucleotides are introduced in the pyrosequencing process. The significance of the dispensation sequence is detailed in Section 3.1. The forward and reverse primers $Pr_f$ and $Pr_r$, respectively, are primers chosen for PCR prior to pyroprinting. The sequencing primer $Pr_s$, is a primer used for synthesis as part of the pyrosequencing process. Each parameter that is part of the protocol $\mathcal{T}$ is very important and affects the DNA content from which a pyroprint is constructed. If the protocols for pyroprints being compared differ from each other, then the comparison between the two pyroprints is completely meaningless.

To capture the relationships and properly represent all three entities, we use the following notation:

$$\mathcal{I} = \{I_1, \ldots, I_n\}$$

$$\mathcal{R} = \{R_1, \ldots, R_k\}$$

$$\mathcal{P} = \langle \mathcal{T}, \{\langle I_1, p_1 \rangle, \ldots, \langle I_m, p_m \rangle\} \rangle,$$

where $\mathcal{I}$ represents the set of all bacterial isolates in the input dataset and $\mathcal{R}$ represents the set of all ITS regions that can be found in the genome of the FIB being studied. While we represent ITS regions in such a way that it is possible to have any number $k$, the work in this thesis only considers $k = 2$ ITS regions: $R_1 =$ "$16 - 23$" and $R_2 =$ "$23 - 5$". The set of all pyroprints, $\mathcal{P}$, is represented as a tuple consisting of:

- The protocol used for constructing the pyroprint ($\mathcal{T}$)
- A set of all pyroprints constructed using the protocol $\mathcal{T}$ tupled with the bacterial isolate $I_m$ of origin from which pyroprint $p_m$ was constructed.

Generally, an individual pyroprint $p_m$ is easily represented as a tuple of the form:

$$p = \langle d, \bar{p} \rangle$$

where $d$ and $\bar{p}$ are both represented as vectors:

$$d = (d_1, \ldots, d_n) \qquad d_i \in \{A, T, C, G\}$$

$$\bar{p} = (p_1, \ldots, p_n) \qquad p_i \in \mathbb{R}_0^+$$

Such that $d$ is a dispensation sequence and $\bar{p}$ is a vector of pyroprint peak heights. Each light emittance value $p_i$ of a pyroprint $\bar{p}$ may be any positive real number or 0. The work in this thesis only explores the use of peak light emittance values for each dispensation and does not consider peak width or peak area. For brevity, pyroprint light emittance vectors are referred to as *pyroprint vectors*.

## 2.5 Similarity Metric

For the comparison of pyroprints, assuming the protocols for each are identical, Pearson's correlation is the similarity metric of choice. If the protocol for each pyroprint differs, a default similarity of $-2$ is produced. This ensures that bacterial isolates with different protocols cannot possibly be grouped together, and the result, since it is outside the possible range of Pearson's correlation, also signifies that the comparison was not computed. Pearson's correlation is a similarity metric formulated in the following way for two pyroprints $\bar{p}$ and $\bar{q}$:

$$sim(\bar{p}, \bar{q}) = \frac{\sum_{i=1}^{n} (p_i - E(\bar{p}))\,(q_i - E(\bar{q}))}{\sqrt{\sum_{i=1}^{n} (p_i - E(\bar{p}))}\,\sqrt{\sum_{i=1}^{n} (q_i - E(\bar{q}))}}$$

For this formulation, $E(p)$ is the expected value, or mean, of the pyroprint vector $p$ such that:

$$E(\bar{p}) = \frac{\sum_{i=1}^{n} p_i}{n}$$

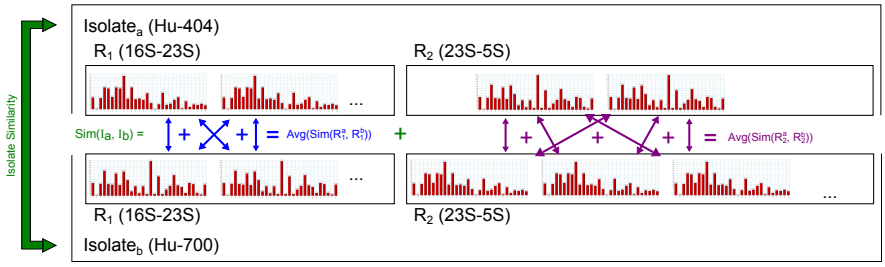



Figure 2.10: A depiction of how isolate similarity is computed. It is the average of the similarity of both regions, each of which is an average of the cross product between pyroprints.

While this is the standard formulation of Pearson's correlation, it requires two passes over the pyroprint vector and is computationally expensive. Fortunately, there is an alternative formulation used in this thesis that can be computed in a single pass over the pyroprint vector:

$$sim(\bar{p}, \bar{q}) = \frac{(N \sum_{i=1}^{n} p_i \, q_i) - (\sum_{i=1}^{n} p_i \, \sum_{i=1}^{n} q_i)}{\sqrt{N \sum_{i=1}^{n} p_i^2 - (\sum_{i=1}^{n} p_i)^2} \sqrt{N \sum_{i=1}^{n} q_i^2 - (\sum_{i=1}^{n} q_i)^2}}$$

Pearson's correlation is the preferred similarity metric because pyrosequencing machines observe fluctuations in the amplitude of light emitted during binding reactions in the sequencing process. The reasons for these fluctuations can be contributed to hardware issues or inconsistencies in the PCR process of DNA amplification. These fluctuations are accommodated by comparing the variance between the two pyroprint vectors instead of a series of direct comparison between components of each pyroprint vector.

The similarity between the same ITS region, $R_i$, of two bacterial isolates, $I_a, I_b$, is then computed as an aggregation of all pairwise Pearson's correlations computed between pyroprints in both ITS regions. The results shown in Chapter 7 are computed using an average aggregation of pairwise Pearson's correlations

as formulated here:

$$R_i^a = \{p_1^a, \ldots, p_{s_j}^a\}^{\rightarrow I_a}$$

$$R_i^b = \{p_1^b, \ldots, p_{s_k}^b\}^{\rightarrow I_b}$$

$$sim(R_i^a, R_i^b) = \underset{\substack{x \in [1,s_j] \\ y \in [1,s_k]}}{\text{average}}(sim(p_x^a, p_y^b))$$

The similarity between two bacterial isolates, $I_\alpha, I_\beta$, is then computed as the average of the similarities computed between each ITS region pair, as shown in Figure 2.10, or 0 if the similarity between any ITS region pair is too low:

$$sim(I_a, I_b) = \underset{i \in \{16-23, 23-5\}}{\text{average}}(sim(R_i^a, R_i^b))$$

## 2.6 Bacterial Strains and Clusters

A bacterial strain is a biological concept that describes a group of bacterial isolates, from a given species, that are identical based on the method used to identify them. Informally, bacterial strains are groups of bacterial isolates that have the same genotype and are distinguished from those with a different genotype. Clusters are a computational concept that describes a group of identical, or highly similar, entities. While bacterial strains have a specific biological context, clusters are very abstract and applicable to many different data. In this section, the connection between bacterial strains and clusters is explained so that the work described in this paper is clearly understood.

Formally, a bacterial strain is a set $S$ of bacterial isolates, where each bacterial isolate $I_j \in S$ has a high level of similarity to the each other bacterial isolate $I_k \in S$. Pyroprints function as DNA fingerprints, representing the sequence, or genotype, of each ITS region. This makes pyroprints an appropriate vehicle

for determining the similarity of genotypes for a pair of bacterial isolates. The definition of a bacterial strain is given by the following notation:

$$\mathcal{S} = \{S_1, \ldots, S_m\}$$

$$S = \{I_1, \ldots, I_c\}$$

Here, $\mathcal{S}$ represents the set of all determined strains $S_1$ to $S_m$. Using the similarity function $sim(I_a, I_b)$ described in Section 2.5, determining the subset of bacterial isolates, $\{I_1, \ldots, I_c\}$, from all bacterial isolates $\mathcal{I} = \{I_1, \ldots, I_n\}$ that belong to each strain, $\mathcal{S} = \{S_1, \ldots, S_m\}$, is done by incorporating bacterial isolates with a high similarity into the same strain $S_i$. It is from this definition that the problem of identifying bacterial strains can be clearly seen as a clustering problem. A direct parallel can be made between the biological concept of bacterial strains and the computational concept of clusters. In this way, the interpretation of constructed clusters is straightforward–each cluster corresponds to a *potential* bacterial strain. Although the concept of a bacterial strain is directly represented by clusters, since there is no guarantee that a cluster represents a real bacterial strain, it can only be inferred that an observed cluster *may* represent a bacterial strain, or at least closely resemble one. Described further in Chapter 4, bacterial isolates are grouped into clusters using either agglomerative hierarchical clustering or OHClust!

### 2.6.1 Thresholds

When grouping bacterial isolates into bacterial strains, it is difficult to determine when two bacterial isolates are considered the same or "similar enough." However, by definition, a bacterial strain must consist of bacterial isolates that are the same or highly similar. And, equivalently, a cluster must contain data

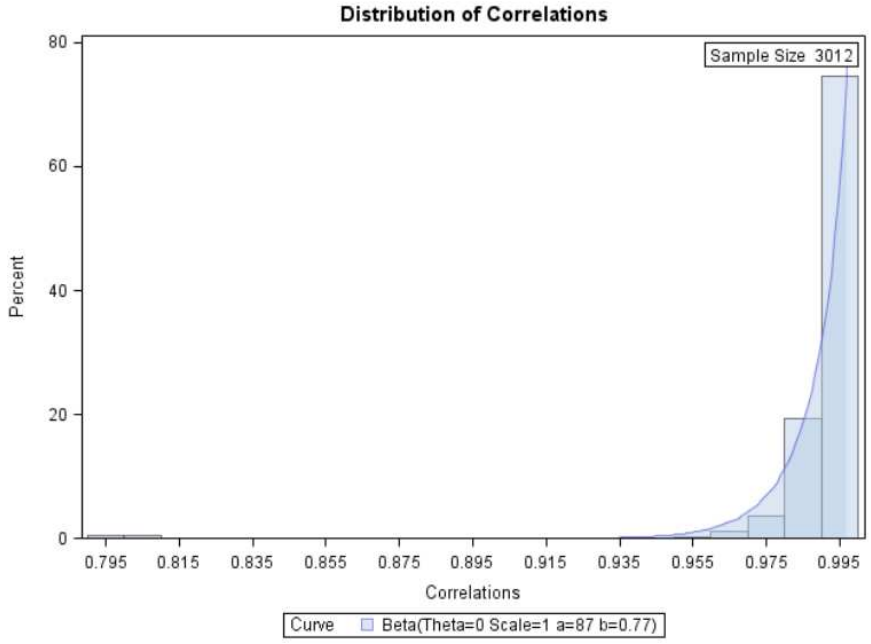



Figure 2.11: A beta distribution fit over comparisons between bacterial isolates that represent the same strain.

points that are highly similar. Therefore, it is necessary to formally define what "highly similar" means. "Highly similar" is typically defined by a threshold which is determined based on statistical or empirical properties of the data. Intuitively, we use thresholds to qualify whether the similarity between two data points is high enough to warrant grouping them together, or low enough to guarantee that they should not be grouped together. Nearly all clustering methods apply thresholds to similarity functions to ensure that only data points that are above, or meet, the threshold are clustered together. In this way, bacterial isolates will be associated with a bacterial strain if, and only if, they have genotypes at least as similar as an established threshold.

A statistics student, Diana Shealy, conducted a study [41] using a dataset of bacterial isolates that had both ITS regions pyroprinted multiple times. The study was conducted to investigate appropriate threshold values for interpreting

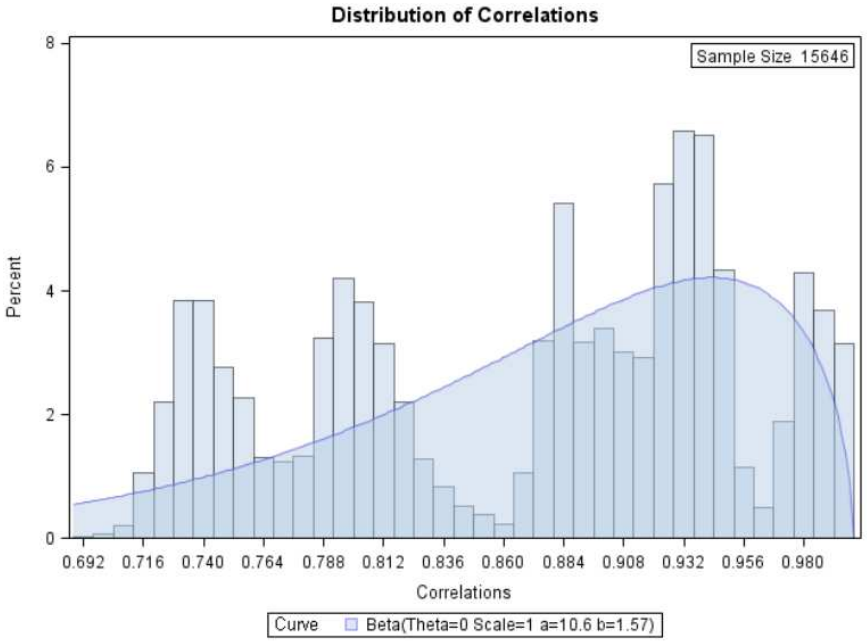



Figure 2.12: A beta distribution fit over comparisons between bacterial isolates that do not represent a single strain.

pyroprint similarities as one of three categories:

- Definitely similar
- Definitely dissimilar
- Reasonably similar but possibly a false-positive or false-negative. Colloquially referred to as "squishy."

To categorize pyroprint comparisons as one of these three categories, two thresholds were established during Diana's statistical study. The upper threshold $\alpha$ defines a minimum similarity for pyroprints to be considered definitely similar. Alternately, the lower threshold $\beta$ defines a maximum similarity for pyroprints to be considered definitely dissimilar. Between these two thresholds, pyroprint similarity seems to suggest similarity between their associated bacterial isolates,

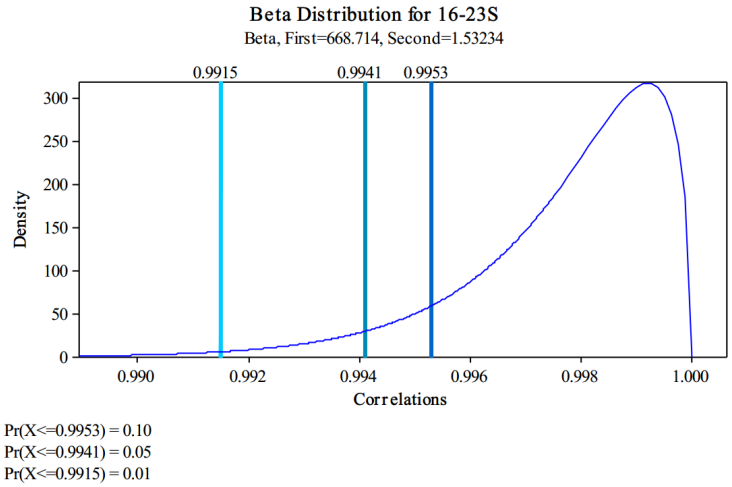



Figure 2.13: A beta distribution fitted onto comparisons between pyroprints constructed from the 16S–23S ITS regions of bacterial isolates from pidgeons for a statistical experiment.

but increases the likelihood of false positives.

For the statistical study, a likelihood ratio approach was initially taken. For this approach, pyroprint that were considered definitely similar or definitely dissimilar were separated into two groups. For each group, pictured in Figures 2.11 and 2.12, a beta distribution was fit to the data. Unfortunately, because it was determined that the distribution of dissimilar pyroprint comparisons is not constant for all combinations of different samples, Diana had to take an approach other than likelihood ratio to establish useful thresholds.

The alternate approach to determining threshold values for pyroprint similarity was to find a beta distribution that closely fit the distribution depicted in Figure 2.11. Interestingly, pyroprints that were constructed from the same bacterial strain seemed to fit the same distribution as pyroprints constructed from a different bacterial strain. Leveraging this information, Diana then determined Pearson's correlation values that resembled common rejection regions.

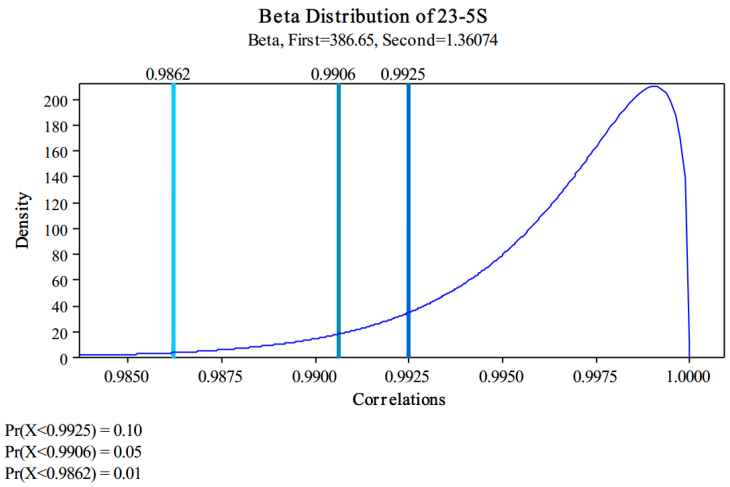



Figure 2.14: A beta distribution fitted onto comparisons between pyroprints constructed from the 23S–5S ITS regions of bacterial isolates from pidgeons for a statistical experiment.

The common rejection regions, shown in Figure 2.13 for the 16S–23S ITS region and Figure 2.14 for the 23S–5S ITS region, represent regions where the amount of false negatives are 1%, 5%, and 10%. For this analysis, a false negative refers to two bacterial isolates that are from the same bacterial strain but are incorrectly separated into distinct strains. With consideration of the results of the statistical study for both the 16S–23S and 23S–5S ITS regions, the biologists have currently decided to use two threshold values, $\alpha = 0.995$, for determining when pyroprints are definitely similar, and $\beta = 0.99$, for determining when pyroprints are definitely dissimilar. For both ITS regions, this means having a 10% false negative rate. However, the statistical study fit beta distributions to distributions of pyroprint similarities for a single ITS region. By using both the 16S–23S and 23S–5S ITS regions together for determining similarity, the false positive rate is ideally significantly lower. Unfortunately, there has not been enough time or man-power to investigate the actual false negative and false positive rates when considering

both ITS regions in conjunction.

Although only two threshold values are used in this thesis, $\alpha = 0.995$ and $\beta = 0.99$, it is important to note that thresholds will vary depending on the bacterial species being studied. Following the aforementioned statistical study, it was decided that for research conducted so far with *E. coli*, it is acceptable to use the same $\alpha$ and $\beta$ thresholds for both 16S–23S and 23S–5S ITS regions. However, thresholds are handled in a generic way and so the work described in this thesis is not restricted to using these threshold values.

## 2.7 Clustering

The analysis of data based on similarity functions determined by various relationships is known as cluster analysis, clustering, or unsupervised learning [18]. Phrased another way, clustering is the organization of a collection of patterns, or data points, based on similarity. The groups or partitions that are formed during clustering are referred to as *clusters*. Clustering is used for the analysis of many different types of data and although it can be done many ways, there are two primary families of clustering algorithms–hierarchical and partitional.

Partitional clustering algorithms iteratively assign and re-assign data points to cluster centroids or seeds. A cluster centroid is a single data point selected to represent the "center" of a cluster. Centroids are initially selected from random or maximally distant data points. Then, on each iteration of the algorithm, data points are assigned to the closest centroid. At the end of each iteration, the data point that is closest to the center of the cluster becomes the cluster centroid. The method used to assign data points to cluster centroids during each iteration differentiates many partitional algorithms. K-means clustering approaches this

by assigning data points to the cluster of the closest centroid on each iteration. For k-means clustering, clusters are mutually exclusive and when a data point is assigned to a cluster, it is removed from its previous cluster. Generally, due to the use of centroids, partitional clustering algorithms are most effective when there is a general idea of how many partitions exist in the input dataset. For the required analysis of pyroprinting, for which there is no clear idea of how many partitions exist, partitional clustering algorithms are not very useful, and therefore not explored further, except in Chapter 5 [18].

Hierarchical clustering algorithms take an approach that considers every data point a cluster and approaches data analysis in a pairwise manner. Iteratively, pairs of clusters that are maximally similar are grouped, or, alternatively, clusters that are maximally dissimilar are separated. Unlike partitional clustering algorithms, hierarchical clustering algorithms do not re-assign data points to different clusters on later iterations. Since hierarchical clustering does not use centroids when clustering, the order of clustering is deterministic for a fixed dataset.

Clustering decisions made by hierarchical clustering–which clusters should be joined to form a new cluster–are maintained in a binary tree called a *dendogram*. A dendogram represents clusters as internal nodes, or connections, and represents data points as the bottom-most points of the tree. An example of a dendogram is shown in Figure 2.15. The numbers on the left mark a point in the dendogram when two data points or clusters join to form a new cluster. Each colored part of the dendogram corresponds to a colored data point pictured at the bottom. Data points are clustered together from highest similarity (corresponding to the node marked 1) to lowest similarity (corresponding to the node marked 6). Once the dendogram is constructed and the entire dataset forms a single cluster, desirable clusters can be determined by making a "cut" in the dendogram. A cut is depicted

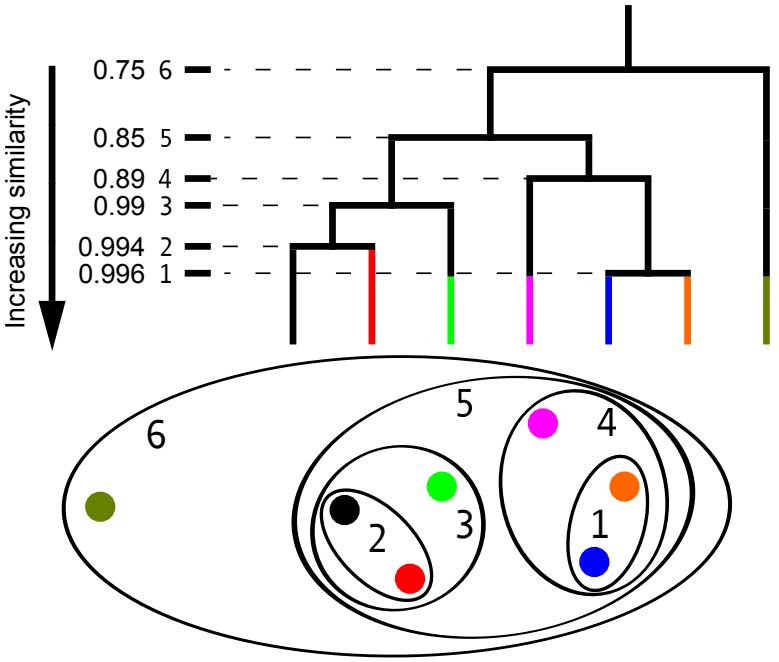



Figure 2.15: A dendogram representing clusters for a dataset of seven data points. Numbers correspond to most similar data points.

in Figure 2.16 at a 0.99 similarity. A cut represents a point at which similarities between data points or clusters must be above (below as pictured in the Figure) in order to be considered an acceptable cluster. Similarity values that are below the threshold value marked by the cut are ignored and data points or clusters are not joined. The example shown in Figure 2.16 shows three clusters were formed:

- the cluster formed with the fourth highest similarity
- the cluster formed with the third highest similarity
- the data point that has the least similarity to the other clusters.

Since hierarchical clustering algorithms do not require any knowledge of how many partitions may exist in the dataset, this is the type of clustering algorithm further explored for analysis of pyroprints [18].

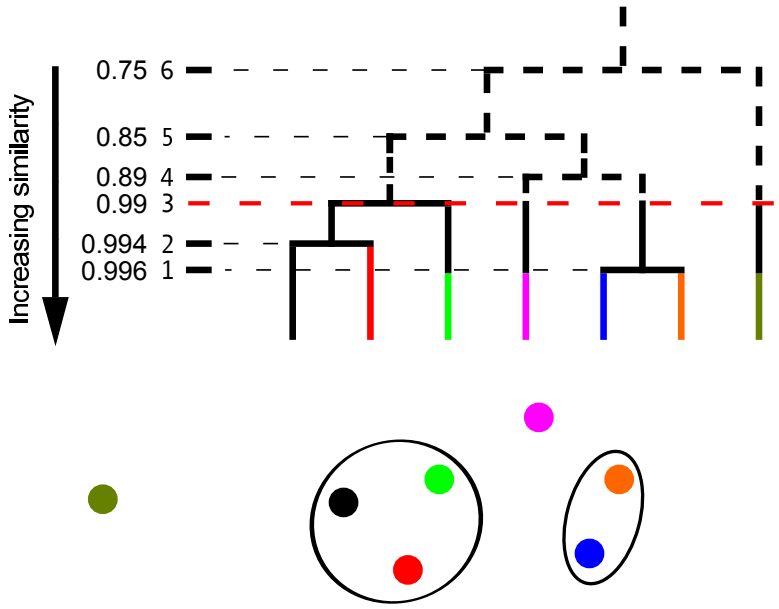



Figure 2.16: A dendogram with a cut made at the third highest similarity, forming four clusters.

### 2.7.1 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering, or agglomerative clustering for short, is the standard bottom-up hierarchical clustering approach. Agglomerative clustering initially considers each data point as a cluster and joins the pair of clusters with the maximal similarity on every iteration. This is known as a bottom-up approach because the dendogram for the input dataset is built from the leaves–each data point–to the root–all data points clustered together. There are two primary components of agglomerative clustering: the similarity metrics used between data points and clusters, and the threshold for cutting the dendogram.

The two primary similarity functions that must be computed during agglomerative clustering are: similarity between data points, and similarity between clusters. The similarity function between data points is determined independent of the clustering algorithm and is based on the data being analyzed. In contrast,

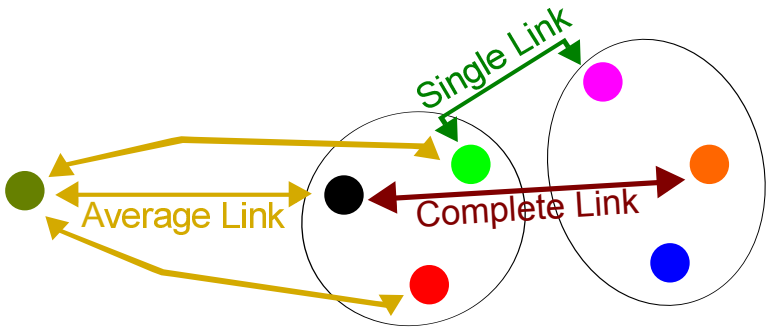
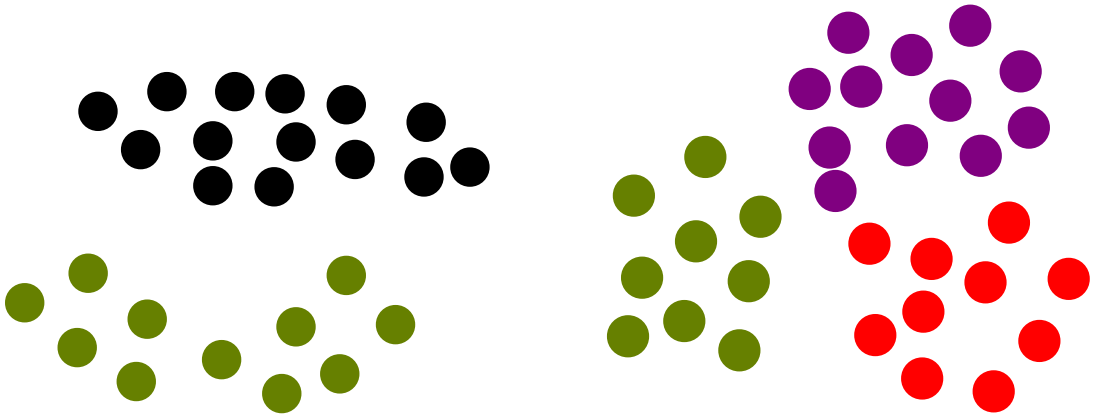



Figure 2.17: A diagram of three cluster similarity functions.

similarity functions between clusters are chosen based on the relationships in the data that are being studied. While there are several cluster similarity functions that can be used, there are three cluster similarity functions, pictured in Figure 2.17, that are typically used:

- Single-Link
- Complete-Link
- Average-Link

Each of the three listed cluster similarity functions yield clusters with very different properties. The single-link function computes similarity between clusters based on the similarity between the two closest points between the clusters. This function is preferable when data points have high similarity in a particular direction instead of equally high similarity in multiple directions. This can be seen in Figure 2.18 where the same dataset is shown to use single-link cluster similarity in Figure 2.18a and average-link in Figure 2.18b. The average-link function computes similarity based on the average of each pairwise distance between each data point in the clusters being compared. This is easily shown in Figure 2.17 where there similarities between the outlying cluster and the mid-



(a) An example of data that is better to use single-link cluster similarity with.

(b) An example of data that is better to use average-link cluster similarity with.

Figure 2.18: Clusterings using single-link or average-link cluster similarity.

dle cluster are used to calculate the average similarity between the two clusters. Referring back to Figure 2.18a, if the data points at the bottom of the black cluster were closer to the top data points in the greenish cluster, then single-link cluster similarity would be a bad choice because the whole dataset would form a single cluster. In such a case, average-link would be much better in order to observe a more wholistic relationship between clusters. In contrast to the single-link similarity function, the complete-link similarity function computes the similarity between two clusters based on the two data points that are most dissimilar, or least similar. This cluster similarity function is best used for datasets where very closely related clusters are desired. Understandably, complete-link is the most conservative cluster similarity function of the three mentioned functions. It is important to remember that "good" clusters are defined based on the dataset and the relationships being studied.

Once clustering is complete and a dendogram is constructed, the resulting set of clusters must still be determined. The dendogram is constructed only to

provide an analysis of the relationships between the data points in the input dataset. Since the root of the dendogram always represents all data points of the dataset as a single cluster, there is a "cut" that must be made in the dendogram that determines at what point joining clusters is no longer meaningful. This point is called a threshold–a value, or level, that a similarity must exceed. Intuitively, the threshold can be described as the point at which two clusters are believed to contain highly related, or even identical, data points. This will have different interpretations depending on the data being analyzed and the objective of the analysis being carried out, and so is determined based on the input dataset. The dendogram cut is made across edges between internal nodes and child nodes. Dendogram nodes above the cut are ignored, and dendogram nodes directly below the cut are considered the final set of clusters. In this way it is plain that nodes lower in the dendogram have higher similarity between sub-clusters than nodes higher in the dendogram.

# Chapter 3

# Tools for Pyroprint Analysis

As part of the contributions of this thesis, there are two tools that address specific aspects of pyroprint analysis:

- A tool for analyzing the pyroprinting process *in-silico*, specifically for determining a preferred dispensation sequence.
- An interim tool used to analyze pyroprints, using agglomerative clustering and a preliminary implementation of OHClust!.

The first tool, Pyroprint Dispensation Analysis Tool (*PyroprintDAT*), provides analysis to assist in the selection of a dispensation sequence for pyroprinting. Unlike the typical pyrosequencing process, the use of several template strands makes the choice of dispensation sequence important for pyroprinting. The second tool is a tool for analyzing isolates and pyroprints given a similarity matrix of Pearson's correlations.

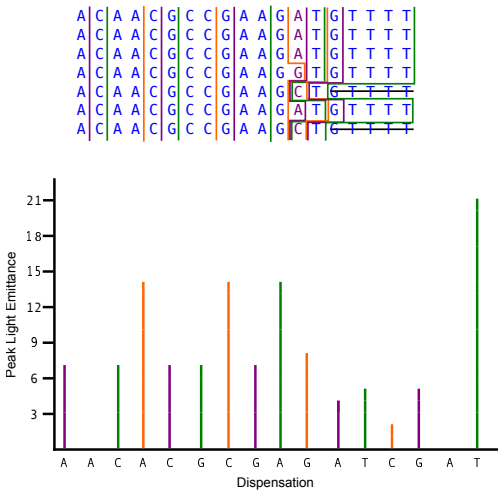
## 3.1 PyroprintDAT

PyroprintDAT provides the biologists with the ability to pyroprint known DNA sequences *in-silico* using a set of input dispensation sequences. It specifically addresses the goal of determining a "best", or most preferred, dispensation sequence for pyroprinting a population of bacterial isolates. Determining a best dispensation sequence for pyroprinting would be difficult and expensive if done *in-vivo* or *in-vitro*. By using PyroprintDAT for *in-silico* analysis, the best dispensation sequence can be determined naively.

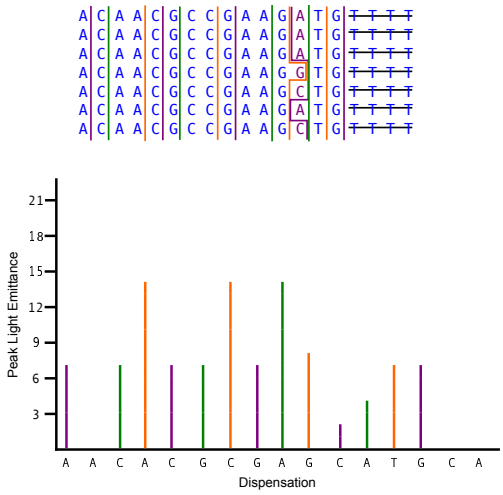
Consider that pyroprinting is a technique for creating DNA fingerprints. Informally, the best kind of fingerprint is one that is most likely to differentiate between different individuals and, at the same time, correctly match isolates from the same strain. In consideration of this, the best dispensation sequence for pyroprinting can be informally understood as a dispensation sequence that makes pyroprints of bacterial isolates from different bacterial strains very different from each other. Since pyroprinting can yield different pyroprints based on the dispensation sequence used, it is desirable to determine the dispensation sequence that produces distinct pyroprints for bacterial isolates from different bacterial strains.

Figure 3.1 shows two pyroprints produced by different dispensation sequences given the same set of sequencing strands. Looking at the twelfth nucleotide in both figures 3.1a and 3.1b, it is clear that the pyroprinting process adds to the sequencing strands differently for the two dispensation sequences AACACGCGAGATCGAT and AACACGCGAGCATGCA. In order to best understand dispensation sequence analysis, it is important to have a clear understanding of how variable regions affect the pyroprinting process.

Using Figure 3.2 as a reference, consider each sequencing strand indepen-




(a) A pyroprint with dispensation sequence AACACGCGAGATCGAT.




(b) An alternate pyroprint with dispensation sequence AACACGCGAGCATGCA.

Figure 3.1: Pyroprints using different dispensation sequences for the same set of sequencing strands.

dently. During the pyroprinting process, dispensed nucleotides react and bind with the next available nucleotide in a template strand. For multiple, independent template strands, this means that "progress" along each sequencing strand will be consistent in conserved regions, but vary in unconserved regions. Biologically, as nucleotides are added to the sequencing strand, there are fewer available unbound nucleotides in the template strand. Computationally, progress can be conceptualized as a pointer advancing along a string. As nucleotides are added to the sequencing strand, a pointer to the sequencing strand advances.

For example, a dispensation sequence that starts with **A**, will dispense a nucleotide that is added in the first position of the seuqnecing strands as a gold bracket in Figure 3.2. For this example, each of the template strands would contain a single open **T** that would be complementary to the **A** being added.

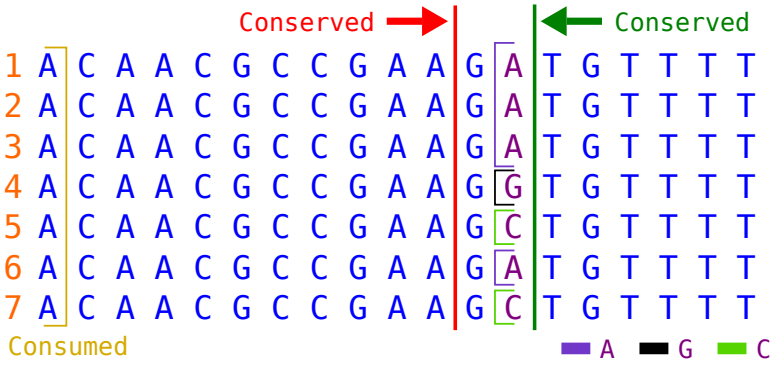



Figure 3.2: Annotated set of seven sequencing strands. The red and green lines mark the end and beginning of conserved regions, respectively. Colored brackets mark corresponding nucleotides in the variable region.

Biologically, after the **A** has been incorporated, the enzyme on each sequencing strand advances one nucleotide. Computationally, the pointer on each sequencing string is advanced one character.

Until each sequencing strand reaches the end of the conserved region, marked by a vertical red line, each dispensation is guaranteed to advance at the same rate for all seven copies. This means that for any dispensation sequence, the pyroprints for the first 11 nucleotides of the sequencing strands will always be the same. Variations in pyroprints due to dispensation sequence begin to manifest after the first conserved region. Between the two conserved regions, marked by red and vertical lines, there are two scenarios that can be considered.

In one scenario, each sequencing strand is extended equally up to the green line, then each sequencing strand is again extended at the same rate in the conserved region. This scenario occurs if **G**s, **A**s, and **C**s are added before a **T**. This scenario is shown in Figure 3.1b.

In the other scenario, a **G** is added to the sequencing strand, then a **T** is added. This scenario requires a **G** to be added first, because each template strand has

```
function CompareSequences(D, F) ▷ D: disp seqs, F: DNA in FASTA
    BestDisp ← null                          ▷ value of best dispensation sequence
    declare M[ ][ ][ ]                                  ▷ three dimensional array
    for d_n ∈ D do                          ▷ For each disp seq being considered
        P ← [ ]
        for f_m ∈ FastaFiles do                          ▷ For each FASTA DNA seq
            P[m] ← CreatePyroprint(f_m)
        end for
        for i ∈ P[ ] do                          ▷ M gets similarity matrix for d_n
            for j ∈ P[ ] do
                M[n][i][j] ← ComparePyroprints(P[i], P[j])
            end for
        end for
    end for
    BestDisp ← arg max(getAvgDissimilarity(M[n]))     ▷ Similarity matrix
with most average dissimilarity across all pyroprint similarities.
    return BestDisp
end function
```

Figure 3.3: Function pseudocode for comparing dispensation sequences.

an unbound **C** nucleotide. If **T** is added before **A**, then sequencing strands 5 and 7 will advance unevenly. This uneven advancement will persist throughout the conserved region. Alternately, if **T** is added before **C**, then sequencing strands 1 to 3 and 6 will advance unevenly. This scenario is shown in Figure 3.1a.

The best dispensation sequence maximizes both the number of pyroprints that are dissimilar to each other and the magnitude of these dissimilarities. It

should be noted that this definition of the best dispensation is **not** the same as a dispensation sequence that minimizes the similarity between pyroprints, which may lead to the selection of a dispensation sequence that is able to strongly differentiate between only a few pyroprints. By selecting a dispensation sequence for the pyroprinting process that maximizes the dissimilarity between pyroprints, it is possible to improve the sensitivity and effectiveness of pyroprinting as a bacterial strain identification method. PyroprintDAT approaches this problem as seen by the pseudocode in figures 3.3 and 3.4. The functions CreatePyroprint, ComparePyroprints, and getAvgDissimilarity are not included in pseudocode as they are relatively simple to understand and easy to describe.

CompareSequences first creates an empty list, or set, $M$, that will store a similarity matrix for each *candidate dispensation sequence*. A candidate dispensation sequence is a dispensation sequence being considered as a potential best dispensation sequence. Candidate dispensation sequences can be explicitly given as input to PyroprintDAT or special sequences can be given that will generate permutations. The generation of dispensation sequence permutations from special sequences can be seen in the function GenerateSequences. In GenerateSequences, each dispensations sequence $d_n$ is checked for the specific string (ATCG). If this string is found, then 24 new dispensation sequences are generated and added to the set of dispensation sequences, $NewDispSeqs$. Each generated dispensation sequence contains the same content as the original dispensation sequence, $d_n$, but with the occurrence of (ATCG) replaced by some permutation of ATCG. A sample of six permutations are shown in the for loop which iterates over each permutation *perm*. In addition, there is a simple notation of the form ⟨number of repeats⟩(⟨repeat sequence⟩) that allows for long DNA sequences be written more concisely. For example, ATG5(GCTA)ATATG expands

```
function GenerateSequences(DispSeqs)
    NewDispSeqs ← ∅
    for d_n ∈ DispSeqs do
        if HasSeq(d_n, (ATCG)) then
            for perm ∈ {ATCG, ATGC, ...} do
                newDisp ← ExpandSeq(ReplaceSeq(d_n, perm))
                NewDispSeqs ← NewDispSeqs ∪ newDisp
            end for
        else
            NewDispSeqs ← NewDispSeqs∪ ExpandSeq(d_n)
        end if
    end for
     return NewDispSeqs
end function
```

Figure 3.4: Function pseudocode for expanding abbreviated sequence notation (e.g. 2(ATCG)) into plain sequence notation (e.g. ATCGATCG).

into ATGGCTAGCTAGCTAGCTAGCTAATATG. Although not included in the pseudocode above, ExpandSeq expands occurrences of this notation to make *in-silico* pyroprinting simpler.

Each similarity matrix $n \in M[\,]$ is a similarity matrix where the $i$th row and the $j$th column contain the Pearson's correlation between pyroprints $P[i]$ and $P[j]$ with dispensation sequence $d_n$. This is depicted in Figure 3.5 for clarity. Each pyroprint $i \in P[\,]$ is created from the FASTA file $f_m$ where the $i$th pyroprint is constructed from the $m$th FASTA file.

A FASTA file is a file format containing one or many DNA sequences. Each

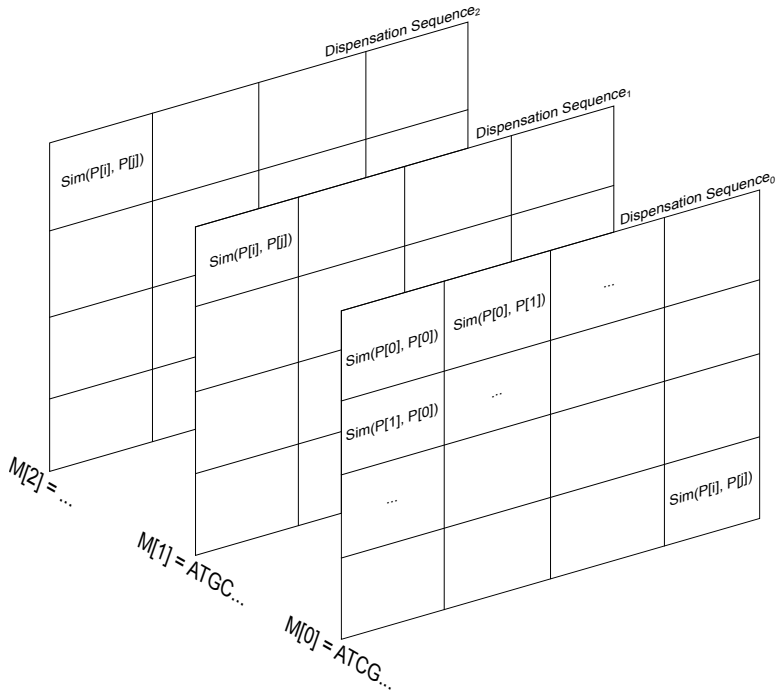



Figure 3.5: List of pyroprint similarity matrices for each candidate dispensation sequence in the `CompareSequences` function.

DNA sequence (of any length) contained in a FASTA file is preceded by a description line beginning with ">." The description line often includes metadata about the following DNA sequence. PyroprintDAT assumes input FASTA files contain DNA sequences for each replicate of a particular ITS region. For a simple example, using DNA sequences from the 16S–23S ITS region of *E. coli*, it is assumed that an input FASTA file contains seven DNA sequences as shown in Figure 3.6. Once the set of pyroprints, $P$, contains pyroprints for each FASTA file, $f_m$, a similarity matrix $M_k$ is created by comparing each pyroprint $P_i$ against each pyroprint $P_j$. The similarity matrix $M_k$ is a symmetric matrix where the each position on the diagonal is 1. Once similarity matrices are computed for all candidate sequences, each similarity matrix is assessed based on the number of similar/dissimilar pyroprints within the matrix.

```
>643441560 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (+)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAGCAGCTTGCTGCTTTGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCCGATGGAGGGGGAT
...

>643444221 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (-)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAACAGCTTGCTGTTTCGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCTGATGGAGGGGGAT
...

>643444905 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (-)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAACAGCTTGCTGTTTCGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCTGATGGAGGGGGAT
...

>643445429 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (+)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGAAAGCAGCTTGCTGCTTTGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCCGATGGAGGGGGAT
...

>643445536 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (+)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAGCAGCTTGCTGCTTTGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCCGATGGAGGGGGAT
...

>643445664 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (+)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAGCAGCTTGCTGCTTTGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCTGATGGAGGGGGAT
...

>643445704 16S ribosomal RNA [Escherichia coli O127:H6 str. E2348/69: NC_011601] (+)strand
AAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTA
ACACATGCAAGTCGAACGGTAACAGGAAACAGCTTGCTGTTTCGCTGACG
AGTGGCGGACGGGTGAGTAATGTCTGGGAAACTGCCTGATGGAGGGGGAT
...
```

Figure 3.6: An example of seven ITS region sequences in FASTA format.

The assessment of the similarity matrices can be done in many ways. One way consists of simply minimizing the number of Pearson's correlations that are 1 in the similarity matrix. Another way could be to determine the minimum Pearson's correlation values for a percentage of entries in the similarity matrix. Yet other ways for assessing the similarity matrix could be to determine the average Pearson's correlation, the median Pearson's correlation, the lowest sum of all Pearson's correlations, etc.

### 3.1.1 Dispensation Analysis Study

The pyroprintDAT tool was used by the biologists to determine an optimal dispensation sequence for the 16S–23S ITS region, and eventually the 23S–5S ITS region. A Cal Poly student, Kathryn Robb, worked on the dispensation analysis for her senior project. Kathryn was studying multiple facets of dispensations to

determine the best dispensation using pyroprintDAT [33].

**Data.**

The data used by Kathryn for investigating dispensation sequences was a set of genomes for 36 *E. coli* isolates from different bacterial strains. The *E. coli* genomes were retrieved from **Integrated Microbial Genomes and Metagenomes** (IMG), a "system for supporting the annotation, analysis and distribution of microbial genome and metagenome datasets sequenced at DOE's Joint Genome Institute (JGI)." [17]. Each replicate of the 16S gene, 1530 nucleotides in length, as well as 600 nucleotides into the 16S–23S ITS region, were retrieved from each genome and maintained in a single file in FASTA format. Each DNA sequence is from a different replicate of the 16S–23S ITS Region, thus contributing to the *in-silico* pyroprint associated with the *E. coli* isolate. Similarly, each replicate of the 23S gene, 2906 nucleotides in length, as well as 300 nucleotides into the 23S–5S ITS region was retrieved and stored in a separate FASTA formatted file. Each replicate for an *E. coli* isolate for a particular region was maintained in its own FASTA file, totally 72 FASTA files.

Unfortunately, this dataset is fairly biased and perhaps only somewhat applicable for the *E. coli* that we have studied. The *E. coli* genomes from IMG were aligned against sequences from *E. coli* isolates collected by the biologists, producing little overlap. This has two implications: biologically, the *E. coli* genomes retrieved from IMG are not representative of the *E. coli* population that has been studied in San Luis Obispo, and the dispensation sequence eventually chosen was not optimal for the pyroprinting that was done. However, since sampling of *E. coli* genomes in San Luis Obispo had not been done, a more appropriate dataset was not available.

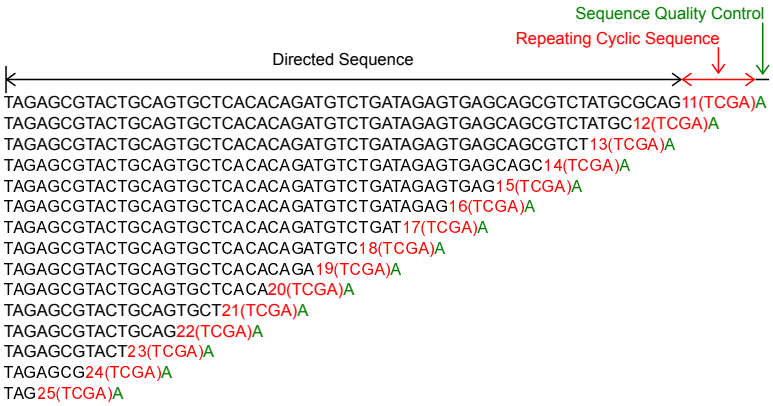



Figure 3.7: Dispensation sequences with directed, cyclic, and quality control components annotated.

### Metrics.

Pearson's correlation was used to compare pyroprints given a certain dispensation. However, to compare the pyroprint similarity matrices produced by each dispensation sequence, the average, median, and upper quartile of values in a similarity matrix were considered. Remembering that the goal is to maximize the dissimilarity between pyroprints from different strains, and that each *E. coli* genome is from a different strain, the approach taken was to minimize the average, maximize the median, and minimize the upper quartile with respect to pyroprint similarity scores. By minimizing the average pyroprint similarity we are looking to skew the distribution of pyroprint similarities lower. Maximizing the median improves the skew in our favor. Minimizing the upper quartile then widens the disparity between the $\alpha$ and $\beta$ thresholds, making false positives and false negatives less probable.

| Disp. Cycle | Avg Corr | Std Dev | Q3 | Std Dev/Q3 |
|---|---|---|---|---|
| TCGA | 54.12% | 25.51% | 72.83% | 0.3503 |
| ATCG | 54.38% | 25.40% | 73.10% | 0.3474 |
| GATC | 54.55% | 25.32% | 73.20% | 0.3458 |
| CGAT | 54.70% | 25.24% | 73.24% | 0.3445 |
| GCTA | 83.37% | 10.12% | 92.29% | 0.1106 |
| AGCT | 83.45% | 10.12% | 92.32% | 0.1097 |

Table 3.1: Statistics for the four best cyclic sequences and the two worst cyclic sequences.

**Results.**

Kathryn began her study using pyroprintDAT which showed her that **TCGA** is the cyclic sequence with the lowest average Pearson's correlation. Shown in Table 3.1 are four cyclic sequences with the lowest average Pearson's correlation, and best standard deviation to upper quartile ratio. For contrast, the two cyclic sequences with the highest average Pearson's correlation and worst standard deviation to upper quartile ratio are included. Then, using this information, Kathryn used the **TCGA** cyclic sequence to investigate fixed vs cyclic sequences as shown in Figure 3.8.

**Fixed and Repeat Sequences.**

For the dispensation analysis conducted by Kathryn, there were three components in each dispensation sequence: (1) directed sequence, (2) repeating cyclic sequence, and (3) sequence quality control. A directed sequence is a part of the dispensation sequence that is manually determined, whereas a cyclic sequence is

a sequence that uses each nucleotide in round-robin fashion before repeating the same nucleotide again. These components are highlighted in Figure 3.7. While the biologists were interested in investigating the trade-offs between directed sequences and repeating cyclic sequences, the sequence quality control component was established to always be a repeat of the dispensation before it. Because of the way pyroprinting works, the sequence quality control gives the biologists an idea of how much machine error may be attributed to a pyroprint. When a dispensation is repeated, the second dispensation of that nucleotide should produce 0, or nearly 0, light emittance. As seen in Figure 3.8, Kathryn determined that, more repeats of a cyclic sequence seemed to be preferred to a dispensation sequence with a longer directed sequence.

Although Kathryn had only a small dataset to work with, her study seems to make sense given the highly variable nature of ITS regions. Given that the DNA content of the ITS region is non-deterministic and subject to mutation, even a directed sequence may seem as arbitrary as repeating cyclic sequences. However, directed sequences are very useful for making primer design easier. Consider that when designing primers for PCR and pyrosequencing, it is important that the primers bind to the target region and that they not interfere with each other or themselves. Designing primers for a variable region, is therefore especially difficult, in which case it is uncertain whether the primer will bind to the desired region at all. One way to address this issue is to design primers that bind at the end of a conserved region, as close to the desired region of DNA as possible. And since the dispensation sequence will not improve or degrade descriminatory power of pyroprinting in conserved regions, minimizing this and increasing the number of repeats of a cyclic sequence seems reasonable.

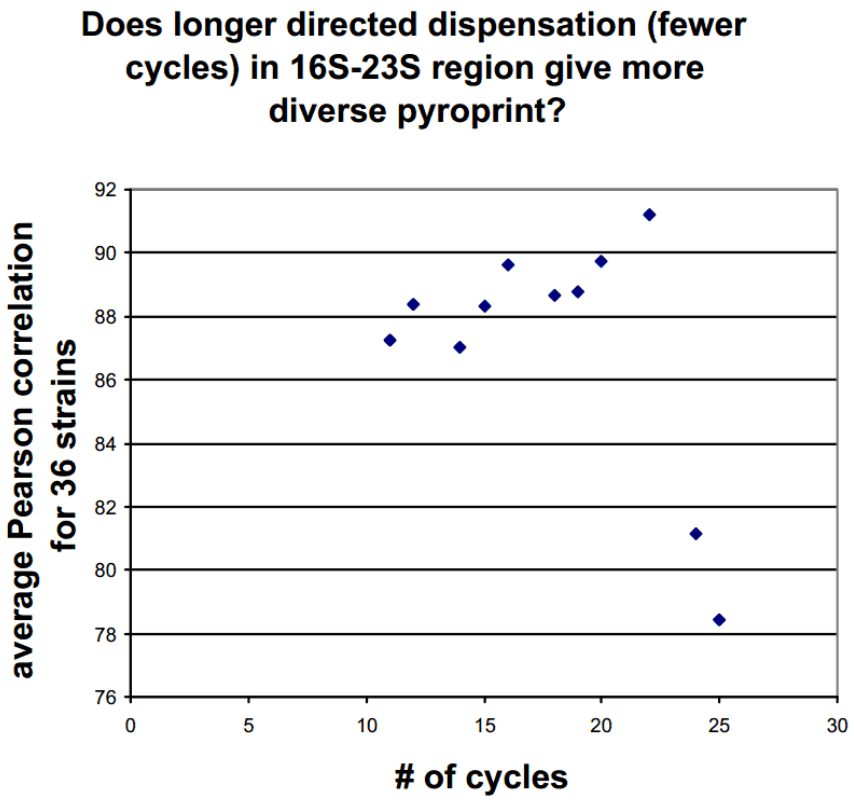



Figure 3.8: Graph comparing average Pearson's correlation to number of four nucleotide sequence cycles in dispensation sequence.

**Conclusions.**

To conclude Kathryn's study, she determined that **TAGAGCG24(TCGA)A** was the optimal dispensation order for the 16S-23S ITS region [33].

## 3.2 Interim tool

The second tool, simply called pyroAnalysis, was developed in the interim period before CPLOP when there was no data management application for pyro-

prints. Data from senior projects and master's theses were managed and maintained solely in Microsoft Excel. The goal of pyroAnalysis was to provide basic cluster analysis of bacterial isolates and pyroprints during this time. Unfortunately, pyroAnalysis had to be built quickly to address the immediate needs of Emily Neal's pilot study [27] and other biology senior projects. To speed up the development process, pyroAnalysis accepted input as a similarity matrix of Pearson's correlations between pyroprints in comma-separated value (CSV) format. Initially, pyroAnalysis only provided a proof-of-concept implementation of OHClust! using the ontology pictured in Figure 4.1 for Emily Neal's pilot study. As data from other pyroprinting senior projects required cluster analysis, and Emily Neal expanded her experiments to a six month longitudinal, population-based study, the ontology used by OHClust! was modified, agglomerative clustering was implemented, and other basic features were implemented. Many of the features provided by this tool were re-implemented as SPAM (Suite of Pyroprint Analysis Methods) and are detailed in Chapter 6.

# Chapter 4

# OHClust!

The primary contribution of this thesis is a new clustering technique, ontological hierarchical clustering (OHClust!). While agglomerative clustering produces acceptable clusters, it is unable to leverage isolate and pyroprint metadata in the clustering process. OHClust! was developed primarily to incorporate isolate and pyroprint metadata into the clustering process in a way that allows for analysis of relationships in sub-groups of the data. Isolate and pyroprint metadata includes where and when bacterial isolates were collected, whom they were collected by, when they were pyroprinted, the host animal from which they were collected, etc. Unfortunately, it is non-trivial to develop a quantitative measure to allow agglomerative clustering to accommodate this metadata. Instead, OHClust! takes a qualitative approach to incorporating metadata by expressing pyroprint and isolate metadata as an ontology. The ontology is then used to guide the order in which comparisons are made during clustering.

## 4.1 Origin and Motivation

During her pilot study of *E. coli* populations in the human gut [27], Emily Neal used agglomerative clustering to differentiate between bacterial strains. While agglomerative clustering produces acceptable results, and the dendogram provides necessary information, ideally we were interested in additional information that agglomerative clustering didn't provide. It was also desirable to observe the grouping of bacterial isolates into bacterial strains over time as the experiment progressed. These interests highlight two issues that motivated the design behind OHClust!: a lack of information regarding sub-hierarchies in each cluster, and the ability to handle incremental updates.

**Desirable Information.** For MST and population-based studies, it is desirable for the biologists to find as many relationships in sub-groups of the data as possible. This can aid in directing data collection or even understanding the influence of time or location on experimental data. Unfortunately, the clusters produced by agglomerative clustering can only provide information relevant to the similarity function used and can only (easily) incorporate quantitative measurements in the clustering process. While the similarity function is the only information that should directly influence the formation of clusters, it would be useful to leverage the knowledge of the structure of an experiment–swab techniques and date of the experiment–in order to observe sub-hierarchies within each cluster. OHClust! achieves this by using metadata that is relevant to the structure of an experiment and specified by the user. Ideally, the way this is approached in OHClust!, accuracy of the overall clustering does not diverge significantly. The extent to which clusters from OHClust! diverge from agglomerative clustering clusters is investigated in Chapter 7.

**Incremental Updates.** It is useful, and common, for the biologists to apply clustering to collected data frequently in order to always be aware of the answers to two questions: (1) what does the library convey about the environment? and (2) do there seem to be any relationships between environmental isolates and library isolates? By maintaining awareness of these two questions, it is possible for the biologists to observe trends in bacterial strains present in the library over time. By handling incremental updates, newly collected data that is collected as part of the same experiment or intent as previously collected data, we are able to provide a continuous sense of how the experiment is progressing. This is addressed by improving performance of OHClust for incremental updates over agglomerative clustering. This performance is explored in Chapter 7.

## 4.2 Metadata and the Ontological Structure

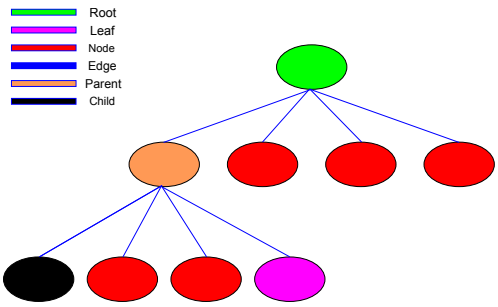
At the core of OHClust!'s design and approach to clustering is an ontology, represented as an n-ary tree, based on user-specified metadata. Informally, metadata is extra information that provides context about data (data that describes data, hence metadata). Examples of metadata include information such as the host species from which a sample is collected (samples from unknown host species are known as *environmental* samples) or the date a sample was collected, pyroprinted, or stored in the database.

Generally, isolate and pyroprint metadata are especially important for the following reasons: it determines the kinds of relationships between isolates that are of interest or it determines whether resulting analysis is biologically meaningful. While the use of an ontology is important for guiding the order in which comparisons are made during clustering, the use of metadata is important for guiding

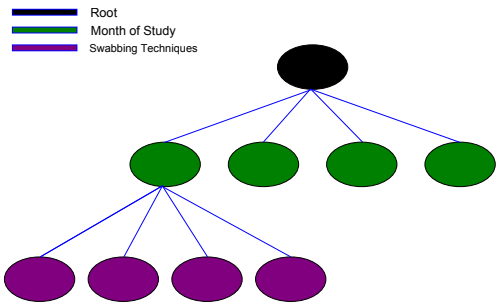
the order of comparisons in a useful way. The ontology is the only differentiating point, though it introduces significant differences, between agglomerative clustering and OHClust!.

During clustering, agglomerative clustering has a flat view of a data set, clustering data points relative to the whole data set. In contrast, OHClust! uses the ontology to construct a hierarchical view of a data set, clustering data points in descending order of data point similarity *within a node*. Resulting clusters for a node are then merged in the parent node and the process is repeated in a bottom-up fashion for the entire ontology. In this way, the ontology allows OHClust! to form clusters in the same way as agglomerative clustering for a sub-group of the data set, but that also provides insights into clusters for these sub-groups. The clusters formed within these sub-groups represent sub-structures and may convey potentially useful relationships relevant to metadata specified by the user. Interestingly, as a result, the information provided by the dendogram that agglomerative clustering produces is unnecessary. For the purposes of the research that the biologists are conducting, and any other situation in which OHClust! should be applied, it is far more interesting to maintain information about clusters formed by the sub-structures observed in sub-groups of the data.

For the data from Emily Neal's pilot study, our approach was to first group bacterial isolates collected in the same day using different sampling methods– Immediate, Fecal, and Later, etc.–then by chronology of collection days. The ontology is depicted by Figure 4.1.




(a) A diagram illustrating the parts of an n-ary tree.




(b) A general ontology for month of isolate collection and sample collection method.

Figure 4.1: Depictions of an n-ary tree and general ontology.

### 4.2.1 Ontology

Due to the flexibility desired for OHClust!, we represent the ontology as a general $n$-ary tree, as seen in Figure 4.1a. For clarity, we briefly define an $n$-ary tree. A tree is a type of structure in computer science that is used to organize information in a hierarchical manner. An $n$-ary tree is a general type of tree such that each level of the hierarchy may have any number $n$ of partitions or sub-hierarchies. The parts of a tree are described as follows:

***node:*** An item in the ontology representing an entity, depicted as circles in Figure 4.1a.

***edge:*** A relationship between two nodes represented as a connection, possibly associated with a value. An edge can only exist between nodes of different levels.

***parent:*** Of two nodes with an edge between them, the parent is the node on a higher level, closer to the root.

```
Isolates.userName():Cal Poly;
Isolates.commonName(): Cow, Pigeon, Dog;
Samples.dateCollected(TimeSensitive)::;
```

(a) Example of a three level ontology with specific and general partitions.

```
Feature   ::= Table.Column([Options]): [Values];
Options   ::= Option, Option, ...
Option    ::= LocalOpt | GlobalOpt
Values    ::= Value, Value, ...
Table     ::= Id
Column    ::= Id
Value     ::= Id
LocalOpt  ::= Time-Sensitive, SimilarCorr, SquishyCorr
GlobalOpt ::= Dynamic, Static, Transform
```

(b) The format for defining an ontology.

Figure 4.2: The format used for constructing an ontology.

***child:*** Of two nodes with an edge between them, the child node is the node on a lower level, further from the root.

***root:*** The initial node in the tree, which has no parent.

***leaf:*** A node that has no children.

***depth:*** The number of edges to get from the root node to a particular node.

***level:*** The set of nodes for a given depth, starting at zero.

Figure 4.1b shows how a general ontology with a depth of two may be represented computationally. The second level of nodes (at depth one) contains isolates separated by month of the study. The third level of nodes (at depth two) contains isolates separated by sample collection method used.

The rest of this section goes in depth in the construction and usage of the ontology.

### Ontology Construction

When constructing an ontology, we use the format specified in Figure 4.2 to establish the translation from user-specified metadata and options to an ontology with desired behavior. The contents of an ontology are user-specified, with the

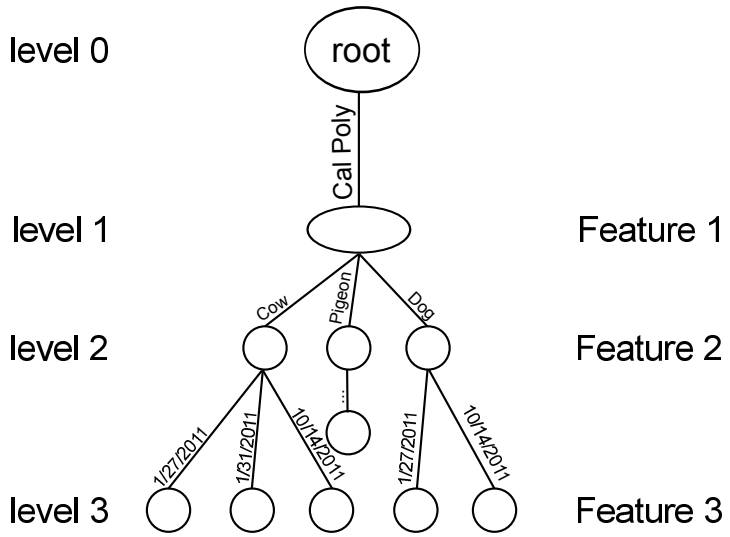



Figure 4.3: A three level ontology corresponding to the ontology format in Figure 4.2a.

layout being partially user-specified and partially controllable through the use of *options*. From the example in Figure 4.2a, it can be seen that each *feature* is listed on a new line and terminated with a semi-colon. Components of a feature that are used for determining content of an ontology are *table*, *column*, and *values*. *Options* are primarily concerned with clustering parameters using the ontology. However, both values and options of a feature can affect the layout of the nodes in the ontology. The table and column components of a feature are to ensure access to the correct data in a database. Valid feature options include:

**TimeSensitive:** Force clustering of children to be in chronological order.

**SimilarCorr:** Only cluster data points above the $\alpha$ threshold.

**SquishyCorr:** Only cluster data points above the $\beta$ threshold.

**Dynamic:** If data point labels do not match any partition for the corresponding metadata type, a new partition is created.

**Static:** If data point labels do not match any partition for the corresponding metadata type, the data point is stored at the lowest (most specific) partition it was able to reach.

**Transform:** Force transformation of definitely similar and definitely dissimilar isolates.

Feature options in bold are default options for each feature.

Figure 4.2 shows two different kinds of options: *LocalOpt* and *GlobalOpt*. LocalOpt options are particular to a level of the ontology. For example, it may be preferable to have dates in an ontology in chronological order. In this case, the feature for dates (third feature in Figure 4.2a) should have the *TimeSensitive* option set. This ensures that edges of the ontology representing *partitions*, or distinct values, of the feature column are in chronological order from left to right. In contrast, GlobalOpt options are set for every level of the ontology. If desirable, GlobalOpt options can also be treated as LocalOpt options, however, for simplicity, it seems to make more sense for these options to affect every level of the ontology if specified for any level.

The construction of an ontology is done in a top down manner such that each feature in the ontology format describes a level of the ontology, starting with the second level (depth of one). There are two components of a feature that describe the contents of each level of the ontology: columns and values. The column is a column in CPLOP that is also called a metadata *attribute*. The metadata attribute determines what values to assign to each edge between a node and its child. If no values are specified, then all possible values of the metadata attribute in the database are used as edges from each parent to a node of the appropriate level.

For example, Figure 4.2a would have edges for values of userName from the nodes on level 0 (root) to the nodes on level 1. However, because "Cal Poly" is specified, there is only one edge used, which is labelled Cal Poly. When a particular value of the metadata attribute is specified, we refer to this feature as a *specific feature*. Similarly, the second feature in Figure 4.2a means that from each node on the first level (only one for Cal Poly) there is an edge for each specified value of commonName: Cow, Pigeon, and Dog. This yields three nodes for the second level for each node present in the first level. For the third feature given in the example ontology format, the metadata attribute to be used is dateCollected. Because no values are specified, this feature is called a *generic feature* and the corresponding level of the ontology is constructed using every distinct value for dateCollected currently in the database. The ontology constructed from the example format is depicted in Figure 4.3.

The third level, in Figure 4.3, consisting of values for the dateCollected metadata attribute, only has a small subset of the values of dateCollected due to space constraints (there are 128 distinct values of dateCollected). However, notice that, going from the second level to the third level, the Cow node has an edge (1/31/2011) that the Dog node does not have. This difference represents the case that there are no data points with metadata matching Dog on 1/31/2011. The exclusion of this edge is an optimization for reducing the size of the ontology. A naive construction may construct the same number of edges and children for each node when adding a level to the ontology. Unfortunately, this leads to millions of nodes being required for general features (for the ontology used by CPLOP, discussed later, it naively takes around 200 million leaves or 400 million nodes total) where only several thousand is actually *useful*. An edge is useful if it is on a *path* that is needed by a data point in the database to reach the appropriate

leaf. A path is the set of edges from the root to a particular node. The path that we are primarily interested in when constructing the ontology is the path from the root to each leaf. If there is a path that will never be used–a case where no data point has a corresponding combination of metadata attribute values–then edges on that path are not added to the ontology, with the exception of edges that are required for another path.

As each level is constructed, LocalOpt option flags are set for the parent that each corresponding edge to the feature has a connection to. For example, the nodes on the second level have a flag set for *TimeSensitive*. This is because it is at the parent nodes that clustering behavior is relevant for partitions of a feature. Most of the options are explained here.

When *TimeSensitive* is set, then each child node, or each sub-hierarchy, is organized in chronological order. When clustering is applied, then the cluster results for each child is accumulated and clustered before the clustering is applied to the next child. This ensures that cluster results are done as if the data from each child was added as an incremental update. If this option is not specified, then a node in the ontology will accumulate clusters from each child, then cluster the merged data from each child once. Finally, once clustering at a node is complete, resulting clusters are percolated up to the parent.

When *SimilarCorr* is set for a node, then clustering is applied using the $\alpha$ threshold. Conversely, when *SquishyCorr* is set for a node, then clustering is applied using the $\beta$ threshold. Once all clusters are percolated up to the root of the ontology, OHClust! will apply clustering using $\beta$. However, if *SimilarCorr* is set then the data percolated to the root is clustered using $\alpha$ first, before clustering using $\beta$. Conceptually, using *SimilarCorr* ensures that only very tight clusters are formed before considering squishy isolate similarities. Alternatively, *Squishy-*

*Corr* is more lenient toward squishy isolate similarities and prefers that they are considered at each level of the ontology.

When *Transform* is set, OHClust! uses a thresholded version of the Pearson correlation coefficient to compare individual pyroprints to each other:

$$thr(sim(I_a, I_b)) = \begin{cases} 0 & if sim(I_a, I_b) < \beta \\ 1 & if sim(I_a, I_b) > \alpha \\ sim(I_a, I_b) & otherwise \end{cases}$$

Ideally, when comparing pyroprints, the similarity falls into three categories: definitely similar, squishy, and definitely dissimilar. For the work in this thesis, the categorization of pyroprint similarities into these three categories is controlled by two parameters, $\alpha = 0.995$, the upper threshold, and $\beta = 0.99$, the lower threshold. This transformation represents our single interpretation for isolates having similarity above $\alpha$ as well as for isolates having similarity below $\beta$. The thresholded version of Pearson's correlation takes this into account by transforming similarities above the $\alpha$ threshold to 1 and similarities below the $\beta$ threshold to 0.

### Ontology Usage

After construction, the ontology is used by OHClust! in the following manner:

1. Isolates are added.
2. Clustering takes place within each node bottom-up in the ontology.
3. Agglomerative clustering is used for clustering.
4. Clusters are percolated up the ontology.

5. Repeat until all clusters are percolated up to the root.

The placement of data points into an ontology is done on a data point by data point basis. Starting at the root, each node in the ontology has $N$ different edges leading to $N$ different children. Naively, the value associated with each edge is checked against the appropriate metadata for a data point. If the value of the edge matches the corresponding metadata for the data point, then the data point is passed to the node corresponding to the matching edge, and the process is repeated until the data point is placed in the correct node.

More formally, we use metadata attributes, representing columns of the database, and *partitions*, values of a metadata attribute on which to separate subgroups of the data set, in order to define and construct an ontology. Metadata attributes have a one-to-one correspondence with attributes or columns in the database schema. As an example, the metadata attributes that are important for the general ontology used for CPLOP are *commonName*, *hostID*, *sampleID*, and *dateCollected* [43]. Each data point is associated with a vector of matching partitions, collectively known as *labels*, in order of the features specified in the ontology format, are compared to each edge of the appropriate level of the ontology. Whenever the partition of the ontology (particular edge) is found as a matching partition for the data point, then the data point is recursively placed into the child node corresponding to the matched partition.

Once each data point in the data set being analyzed is placed appropriately into the ontology, clustering may be applied. The way in which OHClust! clusters data contained in the ontology is detailed in Section 4.3.

### 4.2.2 Metadata in CPLOP

CPLOP is the database at Cal Poly that houses pyroprint and bacterial isolate information. The CPLOP schema reflects the process used by biologists from data collection to pyroprint construction. The primary entities are:

- Host Species
- Hosts
- Samples
- Isolates
- Pyroprints
- Histograms

The histogram entities represent the vector of peak heights for a pyroprint. Pyroprint entities maintain metadata for a particular pyroprint, including the ITS region from which it was constructed, and a link to isolates. The isolate entity represents bacterial isolates collected and stored in CPLOP. Samples are entities representing a fecal sample taken by a biologist, from which many bacterial isolates may be cultured. Sample entities are particular important as they represent the provenance, or origins, of a bacterial isolate being analyzed. Hosts are entities referring to a specific host individual (i.e. a specific dog, as opposed to dogs in general). Finally, host species represent a species as a whole.

CPLOP was originally deployed by Kevin Webb, but has been continued as a computer science master's thesis at Cal Poly by Jan Soliman [43]. While isolates and pyroprints are discussed extensively for the work of this thesis, Jan's

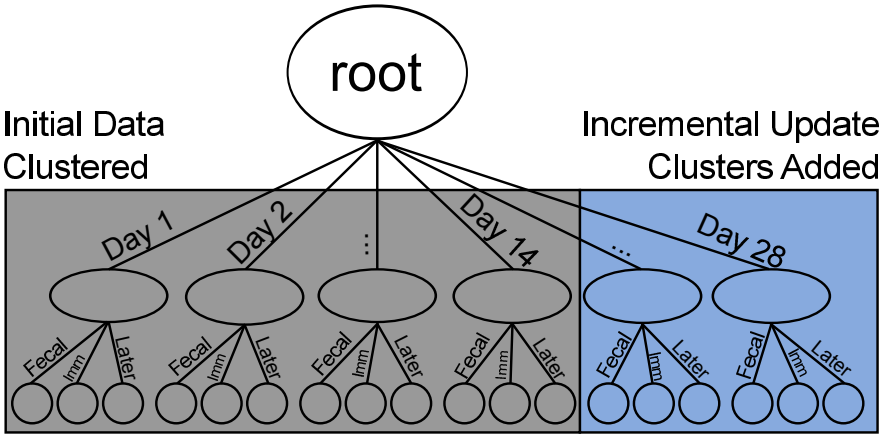



Figure 4.4: Depiction of an incremental update of 14 days to Emily Neal's initial 14 day pilot study.

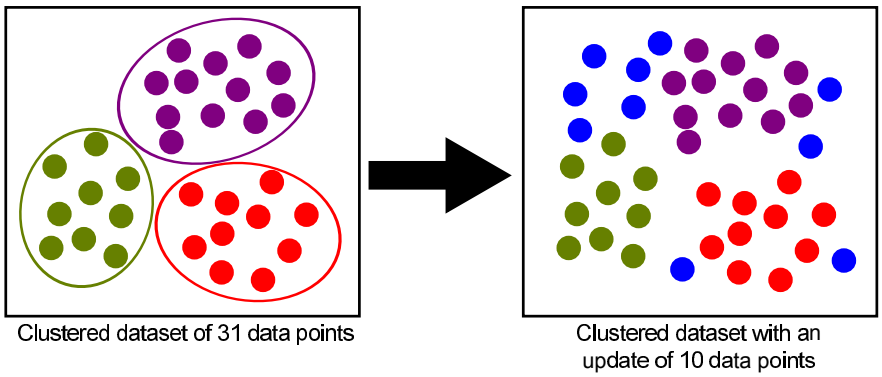



Figure 4.5: A clustered dataset receiving an incremental update.

thesis discusses the schema and usage of CPLOP for MST and related biological research in more detail. For this thesis, the metadata used for ontologies are limited to isolate metadata for simplicity. Jan's thesis also describes the ontology used for applying OHClust! to CPLOP, which consists of Isolates.commonName, Isolates.hostID, Isolates.sampleID, and Samples.dateCollected.

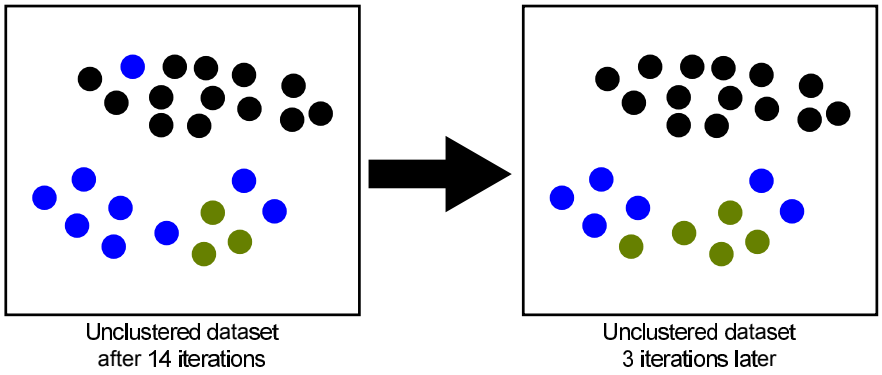



Figure 4.6: Depiction of an unclustered dataset that has gone through 14 - 17 iterations.

### 4.2.3 Incremental Updates

Incremental updates can mean several different things (as seen in related work in Chapter 5). While these many different meanings might have converging results, the approach that a clustering algorithm takes to incremental updates can vary greatly depending on the author's definition of incremental updates. For OHClust!, incremental updates are subsets of pyroprints or isolates that are added to an already analyzed, already clustered dataset. For clarity, Figure 4.5 depicts a dataset that has already been analyzed for clusters, with additional data that wants to be clustered with the already clustered dataset. In contrast, Figure 4.6 depicts an unclustered dataset that requires more iterations before having finished a clustering run. For the work described in this thesis, an incremental update is an independent request to cluster data, and not an intermediate step in the clustering process. An illustrative example is the scenario where data collected during the pilot study described above has been clustered and conclusions have been inferred. Then, following positive and meaningful results, it is decided that the student wants to extend the study and collect data for 14 more days. Instead of taking the collective data for all 28 days and clustering it all together, it may

be desirable to add clusters constructed from the newest 14 days to the clusters from the original 14 days. This scenario is depicted in Figure 4.4 and conveys the idea of a single update that doubles the size of the original dataset.

## 4.3 OHClust! Algorithm

The OHClust! Algorithm consists of three primary components:

**The ontology parser:** parses and populates an ontology

**The ontology feeder:** organizes data into an ontology

**The clusterer:** iterates over an ontology and clusters data beginning at the leaves until final clusters are percolated to the root.

The ontology parser is not described, but does a straightforward parsing of the ontology format mentioned in Section 4.2.1. The ontology feeder is discussed in detail in Section 4.2.1 and comprises a majority of the effort relating to the ontology. This section discusses the clusterer, which is a simple and straightforward application of agglomerative clustering recursively on the data contained in the ontology.

The functions computeSims and agglomerativeCluster are straightforward functions that do the major work for clustering, shown in the pseudocode in Figures 4.8 and 4.7. All similarities between clusters are calculated by computeSims and stored in the similarity matrix $D$. Average-link is used when comparing two clusters, meaning the comparison of two clusters is the average similarity of the cross-product of the two clusters. This is a straightforward computation, though not particularly efficient. However, the comparison of clusters relies on the com-

```
function AGGLOMERATIVECLUSTER(D, C, T)
    C' ← C
    (c_a, c_b) ← ∅
    D[ ] ← computeSims(D, C')
    while |C'| > 1 do
        (c_a, c_b) ← argmax(D[ ])          ▷ Determine closest clusters
        if sim(c_a, c_b) ≥ T then
            c_a ← combineClusters(c_a, c_b)          ▷ Include c_b in c_a
            C' ← C' − c_b
            D[ ] ← computeSims(D, C')
        end if
    end while
     return C'
end function
```

Figure 4.7: Pseudo-code for clustering in agglomerative clustering.

parison of data points contained in each cluster. To understand the comparison of isolates, see Section 2.5. The function agglomerativeCluster, in Figure 4.7, is the agglomerative clustering algorithm and produces clusters given a dataset. It simply compares each cluster to each other cluster then joins the two clusters with the highest similarity.

While computeSims and agglomerativeCluster are the primary working functions, ontologicalCluster handles iteration over an ontology and cluster percolation between nodes of the ontology. It is a recursive function, where the first call is passed the root of the ontology as $N$. For each child node of $N$, the function is called again, passing the child node as $N$ for the recursive function call. Once

```
function computeSims(D, C)
    for c_j, c_k ∈ C do
        D[j,k] ← sim(c_j, c_k)
        if D[j,k] ≥ α then          ▷ If similarity is above upper threshold
            D[j,k] ← 1
        else if D[j,k] < β then     ▷ If similarity is below lower threshold
            D[j,k] ← 0
        end if
    end for
     return D
end function
```

Figure 4.8: Pseudo-code for computing similarity values, with transformation.

a leaf node is reached, the data stored in the leaf node is clustered as described in agglomerativeCluster using a threshold value. The threshold value used is determined by options set for the ontology by the user. If a node in the ontology has the SquishyCorr option set, then the lower threshold, $\beta$, is used. If the SimilarCorr option is set, then the upper threshold, $\alpha$, is used. The set of clusters returned by agglomerativeCluster is then passed to the parent of $N$, the current node in the recursive ontologicalCluster function. Once all child nodes of an ontological node has been clustered, then the ontological node itself may be clustered. In this way, data is initially stored in leaf nodes. Once clustering is invoked, data is clustered at the bottom level, and results are percolated up through the cluster.

```
function ontologicalCluster(D, N, T)
    C ← ∅
    if N = null then                          ▷ If node is null; return emptyset
     return C
    end if
    if |children(N)| > 0 then
        for n_i ∈ children(N) do              ▷ cluster data in each child
            C ← C ∪ ontologicalCluster(D, n_i)
            if isTimeSensitive(N) then
                agglomerativeCluster(D, C, T)
            end if
        end for
        if !isTimeSensitive(N) then
            agglomerativeCluster(D, C, T)
        end if
    else                                      ▷ If leaf node; cluster data
        agglomerativeCluster(D, C, T)
    end if
     return C
end function
```

Figure 4.9: Pseudo-code for clustering in OHClust!.

## 4.4 Algorithmic Properties

When populating an ontology used by OHClust! there are various cases that may arise. One such degenerative case, a situation in which all isolates have been placed in the same leaf node, shows agglomerative clustering to be a special case of OHClust!. In this degenerative case, all data points are placed in the same leaf nodes, meaning that OHClust! will apply clustering to the data contained in the leaf node, and all data in the ontology will have been clustered.

## 4.5 Algorithmic Analysis

In this section, the algorithimic complexity of OHClust! is analyzed. For this analysis, only the complexity of steps directly pertinent to cluster analysis is considered. This includes:

- Placing data points appropriately in an ontology
- Traversing an ontology and applying agglomerative clustering to data in each node and percolating resulting clusters up the structure.

For our analysis we consider the following notation: Let $n$ be the number of isolates being clustered, let $p$ be the number of partitions, or children, that a node in the ontology has, and let $k$ be the depth of the ontology. Note that each level $k$ of the ontology will have the same number of partitions $p$ for each node, but the number of partitions may, and typically will, differ between levels $0, \ldots, k$. For the analysis given here, we assume a fixed $p$ as the maximum $p$ for any given $k$.

As the first step of OHClust! we place each isolate into the an ontology. The ontology is always constructed as a full, balanced tree. That is, the ontology has $O(p^k)$ leaves and $p^k - 1$ internal nodes. In the average case, $n$ isolates are evenly distributed across $p^k$ partitions. In a degenerative case, all $n$ isolates are placed in a single partition. The degenerate case may be viewed as a worst case in terms of data partitioning, but perhaps not a worst case in complexity, as only a single iteration of clustering would have to be applied. In either case, isolates are percolated down the ontology with the same time complexity. At each level $k$ there are $p$ options and isolates will percolate down $k$ levels on average, assuming a general ontology. This requires $pk$ comparisons or considerations before an isolate arrives at its appropriate location in the ontology. For $n$ isolates this yeilds a time complexity of $npk$. As $k$ is a small number on average ($k = 2$ for Emily Neal's pilot study and deep study, and $k \in \{2, 3\}$ for tests in Chapter 7), and $p$ is a small number relative to $n$, the complexity for inserting isolates into the ontology is considered $O(n)$.

The second step of OHClust! can, itself, be broken into two smaller steps:

- agglomerative clustering
- traversing and percolating clusters through the ontology.

At each node, OHClust! applies a clustering algorithm. In this thesis, agglomerative clustering is applied. Although the average case stores partitions of $\frac{n}{p^k}$ isolates per leaf, we first consider the degenerate case of $n$ isolates at a single leaf. In this case, the complexity is exactly the same as agglomerative clustering. That is, there are $n$ clusters to be considered, and there are $\frac{n\,(n-1)}{2}$ comparisons to be made before two clusters can be joined. This is approximately $O(n^2)$ for each iteration of agglomerative clustering. As two clusters are joined on each iteration

until no more clusters can be joined, this yields $n-1$ iterations. For $n$ isolates in a single leaf of the ontology, and for agglomerative clustering in general, we observe a time complexity of $O(\frac{n\,(n-1)}{2} \times (n-1))$ or, simply $O(n^3)$.

Consider the best case, which approximates the average case, such that there are $\frac{n}{p^k}$ isolates located at each leaf. Taking similar steps as before, we consider the complexity of applying agglomerative clustering at a node, except for $\frac{n}{p^k}$ isolates instead of $n$. Predictably, we observe a complexity of: $O(\frac{\frac{n}{p^k}\,(\frac{n}{p^k}-1)}{2} \times (\frac{n}{p^k}-1))$. Again, considering that $k$ is on average a small number and that $p$ is typically small in comparison to $n$, this complexity is asymptotically upper bounded by $O(n^3)$.

As OHClust! must percolate cluster results from each node up the tree, we note that $O(n^3)$ is the base case time complexity for clustering at a leaf. We then consider the following recurrence relation for OHClust!:

$$T(n) = \begin{cases} \Theta(1) & if n < c, \\ pT(\frac{n}{p}) + O(n^3) & otherwise. \end{cases}$$

For a leaf that has $n <$ some small constant $c$ isolates, the complexity is considered constant. Otherwise, consider a node $n_m$ where $m$ is the number of levels that have completed clustering. For example, $m = 0$ when clustering data located in the leaves of the ontology, and $m = 1$ when clustering takes place at the level above $k$. A given node in the ontology, during clustering, must wait for each $p_i \in p$ child to cluster a subset of the data. If considered from the perspective of $n_m$, the parent node of $p_i$, in the average case, then each child $p_i$ must cluster $\frac{1}{p}$ of the data, or $\frac{n}{p^{k-m}}$ data for the $m$th level. Once clustering has completed at the $m-1$ level, then clustering at node $n_m$ takes $O(n^3)$ where $n$ is actually $\frac{1}{p}$ of the data to be clustered for level $m+1$. Note that $T(n)$ need not consider the

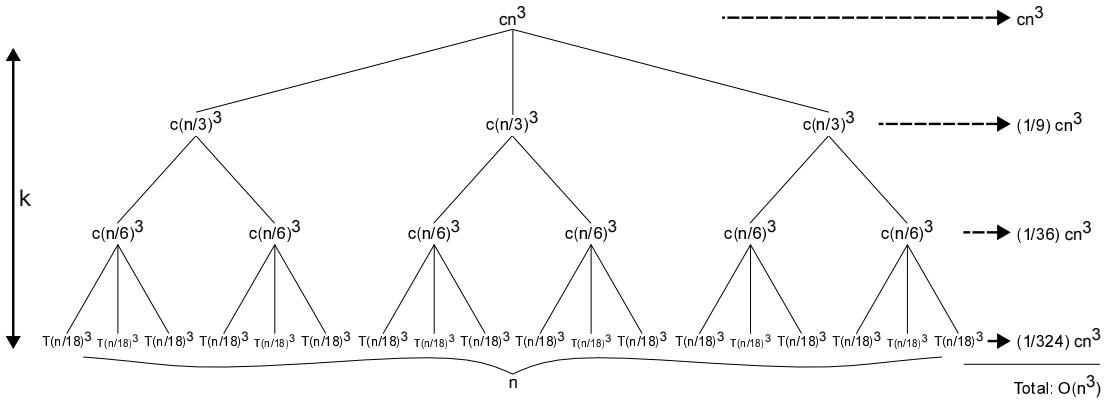



Figure 4.10: An example of the complexity for clustering in OHClust!

time to partition data into sub problems for each child $p_i$ of $n_m$, because that is accounted for during the first step of OHClust! where isolates are placed in the ontology. A recursion tree for this recurrence relation is shown in Figure 4.10. Notice that unlike typical $n$-ary trees, an ontology used by OHClust! has constant depth $k$ instead of variable depth $n \log n$.

To solve the recurrence relation we fix $p$ to $p = 3$ as an upperbound for our ontology, which has $p = 2$ when $k = 2$. Additionally, we use a sum to represent an upper bound of the $T(\frac{n}{p})$ term and solve our recurrence relation as follows:

$$T(n) = \sum_{i=0}^{k} (\frac{1}{3})^{3i} cn^3 + \Theta(n^3) < \sum_{i=0}^{\infty} (\frac{1}{3})^{3i} cn^3 + \Theta(n^3)$$

$$= \frac{1}{1 - \frac{1}{3}} cn^3 + \Theta(n^3)$$

$$= \frac{3}{2} cn^3 + \Theta(n^3)$$

$$= O(n^3)$$

# Chapter 5

# Related Work

## 5.1 Incremental Clustering

Since OHClust! is claimed to be applicable to data types other than biological, this section discusses incremental clustering algorithms developed for generic datasets. These algorithms were developed to incrementally cluster data streams or datasets too large to fit in main memory. While this definition of incremental clustering is still applicable and relevant to our notion of incremental clustering, there are some assumptions that may cause divergence between the optimal clustering of a growing dataset and proposed clustering of these algorithms. More specifically, clustering a dataset in increments, still has the assumptions of a single, whole dataset. In contrast, clustering a dataset that is updated in increments has the assumptions that the ideal number of clusters may change between updates. The subtle difference may make a difference in how quickly produced clusters diverge from optimal clusters, and thus affect how frequently the accumulated dataset must be re-clustered.

BIRCH, Balanced Iterative Reducing and Clustering using Hierarchies, is a clustering method developed by Zhang, et al. to address the problem of clustering large datasets and minimizing I/O costs [50]. BIRCH incrementally clusters numerical data by doing a scan of the target dataset until memory constraints are reached. Each chunk of data is added to an N-ary tree, called a cluster feature (CF) tree, which represents clusters as internal nodes and maintains only aggregate information for each cluster. The aggregate information, and clustering requirements, are computed based on basic algebraic functions: Each data point is represented as a triple of the form $D = (N, LS, SS)$ where a data point $D$ is assumed to be a vector, $N$ is the number of dimensions, $LS$ is the linear sum of the dimensions of $D$, and $SS$ is the squared sum of the dimensions of $D$. In this way, distances (not similarities) are additive and easily computed in a single scan of the data. This approach uses the CF Tree as a fast, guiding method for directing a data point to the appropriate cluster. However, determining an appropriate, meaningful CF representation is difficult.

Young, et al.'s work on incremental clustering takes a partitional approach to clustering [49]. The described work utilizes a competitive learning algorithm that adjusts itself based on a calculated pseudo-entropy of the clusters such that a tradeoff is made between updating centroids aggressively earlier in the clustering process (earlier iterations and/or updates) and updating centroids conservatively later to ensure cluster stability. Additionally, there is a credit-based algorithm for preventing centroid starvation which is computed based on a fixed value or the previously calculated pseudo-entropy such that centroids selected for updates lose credit and centroids that are starving accumulate credit over time. Although Young, et al. mention the importance of being mindful of the possibility that clusters may move, disappear, or reappear, the described work only discusses

moving centroids and assumes a fixed number of centroids. Since the number of clusters that may be formed from clustering pyroprints is completely unknown, and will grow unpredictably as more data is gathered from different regions, a partitional scheme with a fixed number of centroids is not very applicable.

## 5.2 Incremental Clustering for 16S rRNA Sequences

This section discusses incremental clustering algorithms that were developed specifically for clustering DNA sequence reads. These are especially relevant to the work described in this thesis as OHClust! was developed with pyroprints in mind. Incremental clustering algorithms described in this section are all derived from a greedy incremental sequence clustering algorithm developed in 1998 by Holm and Sander [15], hereafter referred to as SeqClustering for brevity. The motivation for this incremental clustering algorithm was to effectively handle continuously growing biological datasets, specifically protein sequences, and to efficiently determine and convey clusters of similar protein sequences. Due to the high volume of redundant protein sequence reads, Holm and Sander decided that each cluster should be represented by a single data point, or sequence read. By using a representative data point for each cluster, it was much more efficient to compute cluster membership for each unclustered data point and to convey meaningful, non-redundant information for understanding each cluster. Variants of this clustering algorithm refer to the representative sequence read of a cluster as a *seed*. This algorithm, as well as its variants such as CD-HIT [22], UCLUST [11], and DySC [51], cluster DNA sequences by using a single sequence as a cluster representative to which unclustered DNA sequences are aligned against. That is,

DNA sequences are data points, and each cluster is represented by a single data point, against which other data points are compared. The comparison metric used is typically a sequence alignment, which gives a similarity score based on the edit distance between the compared sequences.

One variant of Holm and Sander's SeqClustering algorithm is CD-HIT and each of its variants: CD-HIT-2D, CD-HIT-EST, CD-HIT-EST-2D [22, 23]. CD-HIT has SeqClustering as its core algorithm, but optimizes performance by using short word filtering instead of sequence alignments as its comparison metric. Short word filtering, in short, verifies that the compared sequences share a minimum number of identical short substrings, referred to as 'words', such as dipeptides, tripeptides, etc. For certain length words, it's possible to have the seed for a cluster indexed so that it is very fast to compare a sequence to the seed via short word filtering.

Another, very relevant, variant of the SeqClustering algorithm, called dynamic seed clustering (DySC), was developed by Zheng, et al. [51]. DySC is a clustering algorithm developed for reads of the 16S rRNA marker gene commonly used in microbial studies. DySC differentiates itself from other SeqClustering algorithms by using both fixed seeds and dynamic seeds. Seeds in SeqClustering correspond to a fixed seed in DySC, whereas SeqClustering has no equivalent to dynamic seeds in DySC. Using dynamic seeds, DySC constructs *pending clusters*, clusters containing reads that are not within a similarity threshold to the fixed seeds. Pending clusters are transient and will either join a fixed seed cluster or become a fixed seed cluster after reaching a specified size. Zheng, et al. claim that DySC is able to improve cluster quality while maintaining comparable runtime compared to UCLUST and CD-HIT [51].

Yooseph, et al. have done work on the incremental clustering of microbial

metagenomic sequence data [47]. Incremental clustering is a three stage clustering process based on CD-HIT. **First Stage**. patterns are combined with clusters in three steps, each with 90%, 75%, and 60% identity thresholds, respectively. Incremental clustering takes this approach of decreasing identity thresholds for efficiency and quality. For efficiency, CD-HIT-2D runs faster at high thresholds (90%) than at low thresholds (60%). For quality, the parallel implementation of CD-HIT-2D being used by Yooseph, et al. at the time (2008) could only assign patterns to the first cluster whose similarity met the threshold. **Second Stage**. Patterns from the data set not incorporated into clusters in stage one are clustered together using CD-HIT at 90%, 75%, and 60% identity. The clusters formed here are referred to as *core clusters* by Yooseph, et al. **Third Stage**. Two similarity measures are used to join *big core clusters* (cluster with cardinality $\geq 20$) with each other and small core clusters or singleton clusters (cluster with cardinality $= 1$) with final clusters, respectively FFAS and PSI-BLAST.

Each variant of the SeqClustering algorithm is highly relevant to OHClust! in their ability to cluster data incrementally and their relevance to biological data. Using a seed, or cluster representative, that is $\leq 90\%$ similar to all other seeds allows new clusters to easily form, which is important to consider for a continuously growing dataset. Interestingly, Yooseph, et al.'s work on incremental clustering and its three phase approach seems to be the most similar to OHClust!. Where Yooseph, et al. take a three phase approach, OHClust! seeks to improve clustering performance and quality in a two phase approach. The first phase creates core clusters using the $\alpha$ threshold, and the second phase incorporates pyroprints into a boundary or fuzzy cluster using the $\beta$ threshold. However, incremental clustering partitions its data in a three tier scheme by thresholds (90%, 75%, and 60%) instead of the hierarchical partitioning employed by OHClust!.

Seemingly, these algorithms are more accommodating of updates than incremental clustering in Section 5.1 and can handle situations where constructed clusters may significantly differ from initial predictions of observable patterns. In this way, these algorithms may be more amenable to incremental clustering across updates than OHClust!. However, there is a major aspect in which SeqClustering algorithms differ from OHClust!: the use of a core cluster in OHClust! instead of a representative data point. Instead of a representative data point, the use of core clusters potentially reduces clustering errors of pyroprints, especially across updates. This is especially important since pyroprints are not exactly DNA sequences and are vulnerable to machine and human error. Additionally, cluster similarity is computed using each point in the cluster (in the case of average-link or ward's method) instead of a single comparison. Due to this being computationally more expensive, the advantages and disadvantages of not using cluster representatives are explored in Chapter 7.

Additional related work include feature guided clustering, as well as the use of specific data to guide clustering of particular data sets. Clustering algorithms in these fields are very interesting as they take a mathematical approach to guiding the order of clustering through the use of subspaces [6]. Unfortunately, determining just how closely related to OHClust! subspace, or mathematically, guided clustering is is difficult to determine. It sounds incredibly similar in the fact that the clustering is being guided to group or compare certain data points prior to other data points given some useful information. However, it also seems that the work described by Chopra, et al. takes the approach of quantifying relationships between data points. Clustering that utilizes subspaces of a data set takes a more multi-dimensional quantitative approach to data point comparison whereas OHClust! is primarily applicable to single-dimensional data point comparison

with guidance being given through categorical means. However, for categorical metadata that can be easily quantifiable, I suspect that the two clustering algorithms would take a relatively similar approach.

# Chapter 6

# Suite of Pyroprint Analysis Methods

The implementation of Suite of Pyroprint Analysis Methods (*SPAM*) described in this chapter was done in Java, and compliant with Java 1.6. The majority of users have JRE 1.6 installed, but not JRE 1.7. Additionally, the code is accessible from the SPAM repository on Github[1]. This framework is written such that many components can be written consistently and interchangeably, including clustering algorithms, comparison metrics, and data types. In Section 6.1 the overall design of the framework is described, while Sections 6.3 and 6.4 discuss how data types can be written, and how comparison metrics are called and can be implemented, respectively. Finally, Section 6.5 details how analysis algorithms–particularly hierarchical clustering algorithms–can be extended and customized. This chapter also has a dual purpose of describing how the algorithm has been implemented, and providing a reference of documention for anyone desiring to use or extend the code provided.

[1] http://www.github.com/drin/spam

Although discussed further in Chapter 7, it is worthwhile to note here that this implementation was not used for larger test sets in evaluation. However, the implementation of SPAM described here is currently able to perform well enough for current dataset sizes. Unfortunately, there is still a lot of work to be done on this framework in order to achieve good performance yet maintain modularity and flexibility.

## 6.1 Design

SPAM was designed to be a maintanable, modular framework for analyzing pyroprints. The design of SPAM heavily emphasizes the use of objects and polymorphism to achieve these goals. There are two primary components of the framework that were carefully designed in a general way:

- Data types
- Data metrics
- Analysis (currently only clustering)

The data types are designed in a way that the specific data type being used does not have to be known, but clustering and analysis will still be applicable. The analysis components are designed so as to make various clustering algorithms easy to include or modify. This section lays out the relationships and entities of primary interest in SPAM in a way that is de-coupled from the implementation.

It is desirable for data types to be general so that analysis methods can be applied easily and so that the ontology can handle them appropriately. To address this, there are two general data types that we focus on: `Clusterable` and

`Cluster`. The need to use and interact with clusters is unavoidable in a clustering algorithm. However, it is possible to represent the data points that a cluster contains in a way that any type of data point may be created and stored and analyzed for clustering without modifying code for a cluster. This is the role of the `Clusterable` entity. Additionally, for various clustering algorithms, clusters have different properties or representations that are optimal or appropriate. For example, a hierarchical cluster will always be composed of two smaller hierarchical clusters, whereas a partitonal cluster is composed of a centroid and a list of data points. Without loss of generality, a `Cluster` entity may be used that specific cluster implementations may extend so that analysis methods need only be concerned with a general `Cluster` entity and not a hierarchical or partitional cluster specifically.

In order to generally design analysis methods, it is preferable to have a single interface for comparing data types. In order to make this interface general, however, we use the notion of `DataMetric` entities. These entities are used by a data type to compare to another instance of the same data type. While the high-level `DataMetric` entity provides the interface for retrieving a similarity for two instances of a particular data type, specific similarity functions can extend and override a `DataMetric` in order to provide different similarity functions. This makes modularity easy, as a configuration can be specified or modified during runtime that names a particular data metric implementation to be used without the need to modify or swap analysis methods.

The generality of cluster analysis methods was relatively easy to design. As long as strict naming and polymorphism schemes are maintained, then cluster analysis methods can be built as extensions of other methods. Given that OHClust! currently only employs agglomerative clustering we are able to utilize the

agglomerativeCluster function in OHClust!. The design of cluster analysis methods mentioned in this thesis is to have a high-level `Clusterer` interface which defines how other entities may interact with a cluster analysis method. Two entities that are direct descendents of the `Clusterer` are hierarchical cluster analysis methods and partitional cluster analysis methods. Then, every particular cluster analysis method is simply an entity that is somehow a descendent of these two entities, and subsequently the `Clusterer` interface.

## 6.2 Overview

There are four packages that are immediately important to the implementation of SPAM–`biology`, `clustering`, `analysis`, and `ontology`. The `biology` and `clustering` packages are closely related, because the `clustering` package maintains information for generally interfacing with clusters and data points stored in a cluster, but the `biology` package simply contains the specific implementations of the biological entities being analyzed–isolates, pyroprints, and ITS regions. The `analysis` package maintains all code for cluster analysis methods and, possibly in the future, classification methods. As of the current iteration of SPAM, only agglomerative clustering and OHClust! algorithms are implemented (as descendents of `HierarchicalClusterer`). Finally, the `ontology` package maintains code for creating the ontology, parsing input in ontology format, and managing the ontology and ontology nodes (called `OntologyTerm`).

With these four packages, important functionality is available, though not yet optimal. These four packages allow the representation of isolate and pyroprint data, clusters of isolates, an experimental ontology containing isolate data, and the cluster analysis to form clusters of isolates. The classes within each package

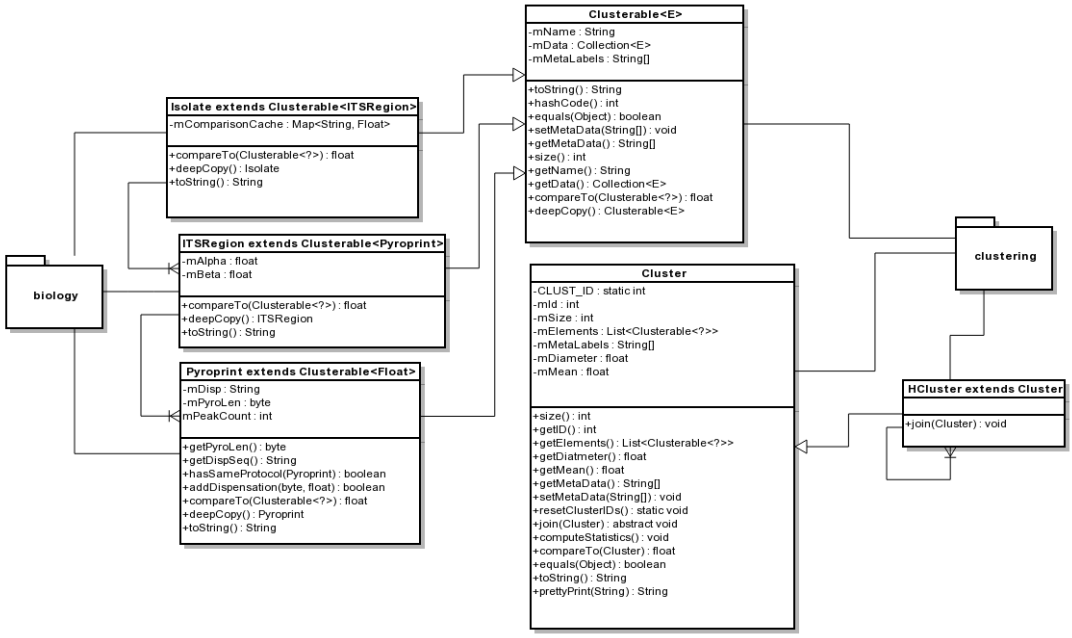



Figure 6.1: A UML diagram the biology and clustering packages of SPAM.

are explained in more detail as they pertain to data types or analysis methods.

## 6.3 Data Types

Custom data types are necessary and allow SPAM to be applied to datasets other than pyroprinting, but also allows for a great amount of flexibility in the aspects of pyroprinting that can be researched and investigated. Custom data types are expected to extend the `Clusterable` abstract class. This class is simple, and all necessary information for clustering can be accessed through this interface. In this way, clustering modules can be written and used without worrying about specifics of the data types involved in clustering. Figure 6.1 shows the currently implemented biological data types–isolates, pyroprints, ITS regions–and their relationship to the `Clusterable` abstract class.

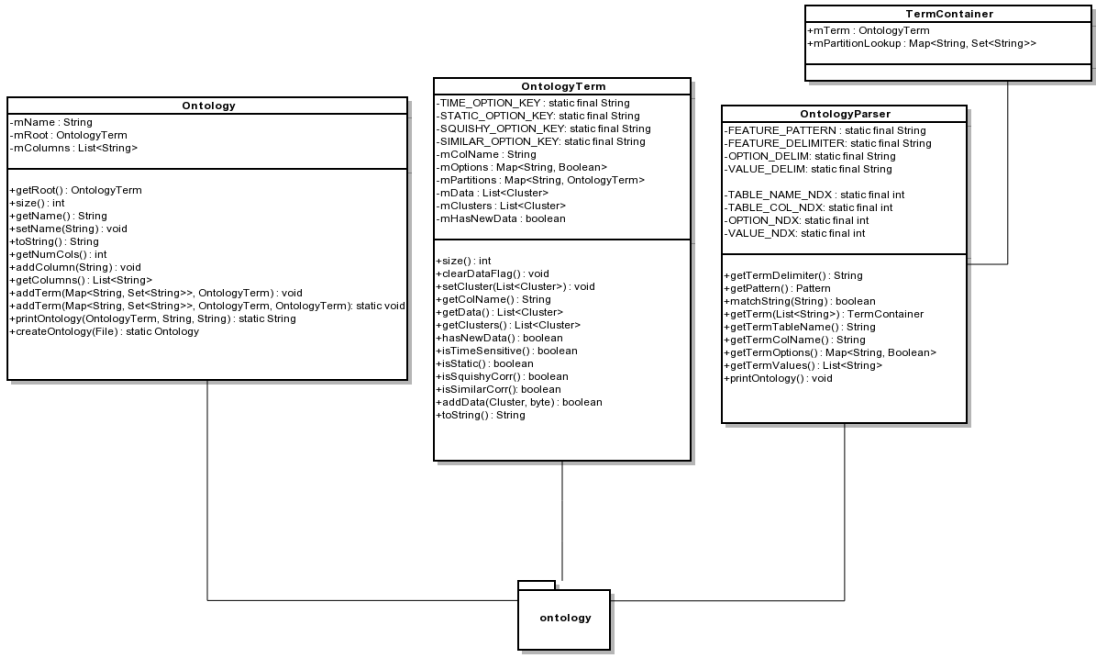



Figure 6.2: A UML diagram for the ontology package of SPAM.

The `Clusterable` class has a name attribute, `mName`, which serves as the basis of identity and equality. `Clusterable` also maintains a collection of data points, `mData`, which contains the relevant content, or data, for this data type. While it is common to apply clustering to simple data types or numerical data types, SPAM was developed around pyroprints and bacterial isolates, which are complex data types consisting of extra associated data points. Another important aspect of the `Clusterable` class is the compareTo method which should be overridden. A specific implementation of compareTo should be explicit regarding the data type it may be applied to for clarity for a developer.

An important custom data type other than `Clusterable` is the abstract class, `Cluster`, which provides the basic implementation of a Cluster. The `Cluster` class consists of metadata labels, *mLabel*, a set of elements contained in the cluster, *mElements*, and some extra information of cluster properties. Generally,

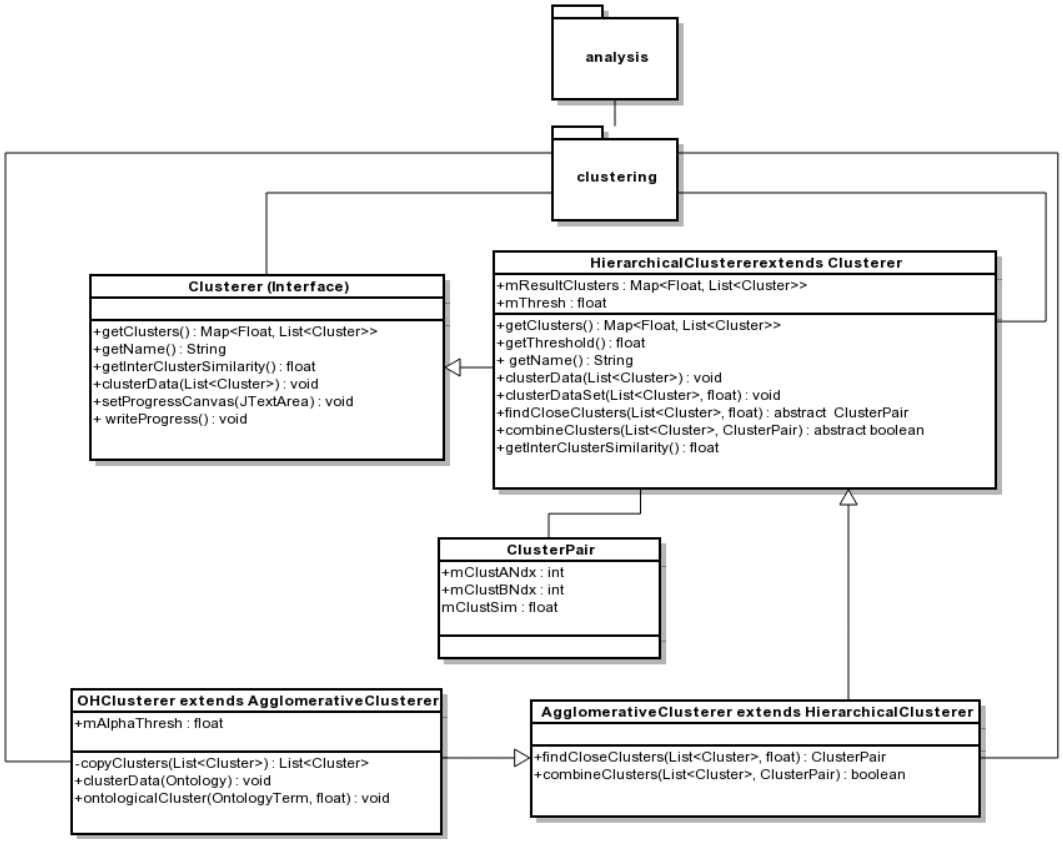



Figure 6.3: A UML diagram for the analysis package of SPAM.

a cluster is a simple data structure that contains similar data points, and can be compared with other clusters.

Important to the general analysis of pyroprinting, there are three custom data types located in the `biology` package. These data types are `Isolate`, `ITSRegion`, and `Pyroprint`, corresponding to bacterial isolates, ITS regions, and pyroprints, respectively. Each of these data types are simply implemented, primarily containing a name (the identifier of the data type, i.e. "Hu-123" for a human and a unique integer ID for pyroprints), associated data, and a comparison method. Each data type is implemented in a way that reflects the relationship between the three components. An isolate maintains in its collection, `mData`, a set of ITS

regions (`Set<ITSRegion>`). Practically speaking there are only two ITS regions in use, 16S–23S and 23S–5S, however, a set of ITS regions allows for flexibility in the future to expand to several ITS regions. An `ITSregion` maintains in its collection, `mData`, a set of pyroprints. Each pyroprint maintained in the collection of an ITS region is a pyroprint that was constructed from that ITS region of a particular isolate. For example, an isolate, $I_a$, has a set of ITS regions, $R^a = \{16-23, 23-5\}$. The region $R^a_{16-23} = \{P_1, \ldots, P_k\}$ contains $k$ pyroprints constructed from the 16S–23S ITS region of Isolate $I_a$. Unfortunately, the descriptiveness and richness of the relationships represented in this way complicate the transformation from a relational database representation of bacterial isolates and pyroprints and performance is lost. On the other hand, the custom data types accurately represent the data and relationships within the data.

## 6.4 Comparison Metrics

While comparison metrics are a useful point of design, the implementation tended to have logical errors than desired or simply introduced complexity that could not be easily optimized. For these reasons comparison metrics were not used in code for evaluation for either this thesis nor Jan's [43]. However, they will be re-introduced in an improved form and so their description in this chapter is still relevant.

Various comparison metrics have only vaguely been investigated, and alternate comparison metrics are not well understood. For future research, several comparison metrics will need to be experimented with and researched for pyroprinting. To define custom comparison metrics, the abstract class, `DataMetric`, should be extended.

`DataMetric` was designed to be relevant to any possible kind of comparison metric. `DataMetric` maintains state in a result value, *mResult*, and error state in a byte, *mErrCode*. Accessible methods are reset, which sets mResult to a default value, apply, which defines how the comparison metric is implemented, result, which simply returns mResult, and various methods for accessing or setting mErrCode. For many basic comparison metrics, it is possible to override only the apply method and the `DataMetric` class will be functional.

To help manage the various levels of comparison required when analyzing bacterial isolates, the `DataMetric` class uses java generics to define the data type it compares. For example, a `DataMetric` that compares isolates is defined as `DataMetric<Isolate>`, `DataMetric<ITSRegion>` for a `DataMetric` that compares ITS regions, and so on. This is to help the developer ensure that the correct `DataMetric` is being used for the correct data types. Additionally, comparison metrics maintain state so that it is simple to implement a similarity function like Pearson's correlation using the `DataMetric` class in a way that is flexible and robust.

## 6.5 Analysis Methods

All clustering analysis methods are expected to implement the `Clusterer` interface, making necessary clustering actions generally available: getClusters and clusterData. The relationships between `AgglomerativeClusterer`, `OHClusterer`, and `Clusterer` can be seen in Figure 6.3. Generally, cluster analysis methods should not implement `Clusterer` directly. Instead, `HierarchicalClusterer` or `PartitionalClusterer` should be extended. These classes abstractly define partitional clustering algorithms and hierarchical clustering algorithms, respec-

tively. Currently, `PartitionalClusterer` is not yet implemented, though this is on the immediate list of future work as mentioned in Chapter 8. It will be useful to compare agglomerative clustering to partitional clustering algorithms, such as *k-means*. The class inheritance hierarchy for `OHClusterer`, the class that implements *OHClust!*, currently includes:

- `Clusterer` (interface)
- `HierarchicalClusterer` (abstract class)
- `AgglomerativeClusterer` (class)
- `OHClusterer` (class)

`Clusterer` defines the method clusterData, which provides and defines a standard point of access to invoking any clustering algorithm's clustering. The clusterData method simply takes as input a list of clusters to analyze. This is so that `Clusterer` objects do not need to be constructed with a dataset of clusters, which would encourage a new object be made for each clustering run. Additionally, although it cannot be enforced by the `Clusterer` interface, cluster analysis methods should be constructed with appropriate threshold values. With this approach, the clusterData method does not require threshold values as input, but can still construct and maintain clusters appropriately. For variants of hierarchical clustering algorithms that use different threshold values (e.g. OHClust! and agglomerative clustering), this helps with modularity, so that high level logic does not need to be concerned with the use and maintenance of threshold values for different clustering algorithms on different clustering runs.

`HierarchicalClusterer` is an abstract class which defines the basic components of hierarchical clustering algorithms. For `HierarchicalClusterer`, clus-

terData is a publicly visible wrapper around clusterDataSet. The clusterDataSet method takes both a list of clusters and a threshold value as input. Additionally, `HierarchicalClusterer` defines the methods findCloseClusters and combineClusters as abstract, so that different variants of hierarchical clustering must define how they compare clusters to find the closest pair of clusters and how these clusters are supposed to be combined into one cluster. A default, standard implementation of these two methods is left to agglomerative clustering, the standard hierarchical clustering algorithm. For many variants, agglomerative clustering may be extended and the amount of code that must be re-written is minimal.

`AgglomerativeClusterer` is a class that extends `HierarchicalClusterer` and defines findCloseClusters and combineClusters, thus completing the agglomerative clustering implementation. Admittedly, the design of `HierarchicalClusterer` and `AgglomerativeClusterer` was done in consideration of the code that can be shared between `AgglomerativeClusterer` and `OHClusterer`. Due to the fact that OHClust! is a recursive application of agglomerative clustering on an ontology, it was desirable that the primary agglomerative clustering work, defined in clusterDataSet, be highly re-usable and de-coupled from as much contextual information as possible.

`OHClusterer` is a class that extends `AgglomerativeClusterer` and implements OHClust!. `OHClusterer` overrides clusterData to invoke ontologicalCluster instead of clusterDataSet. The ontologicalCluster method, luckily, is able to invoke clusterDataSet within each node of the ontology and avoid re-writing any significant clustering code.

# Chapter 7

# Evaluation

There are two aspects of OHClust! we are interested in evaluating: (1) the quality of resulting clusters, and (2) the performance for datasets which receive continuous, incremental updates. Unforunately, for evaluating the quality of resulting clusters, an optimal clustering is not known. Alternately, the biologists consider the results of agglomerative clustering to be acceptable. This motivates us to use agglomerative clustering as a baseline for the evaluation of OHClust! clusters and performance. Informally, we explore these aspects by addressing the question "how does the performance of OHClust! compare to agglomerative clustering?" while at the same time ensuring that *clusters formed by OHClust! and agglomerative clustering do not diverge significantly.* Preliminary results on the comparison between OHClust! and agglomerative clustering were reported in a previous case study [27], but, in this chapter, the comparison is investigated more thoroughly using larger datasets.

## 7.1 Notation

Throughout this chapter, we utilize the same notation described by Wagner and Wagner [46]. This notation is necessary for understanding the metrics we use for evaluation. Let $\mathcal{C}$ be the set of clusters produced by OHClust!, and let $\mathcal{C}'$ be the set of clusters produced by agglomerative clustering. When comparing the two sets of clusters, we must consider four disjoint sets of isolate pairs:

$$S_{00} = \{I_j, I_k \notin C_a \in \mathcal{C} \wedge I_j, I_k \notin C'_b \in \mathcal{C}'\}$$

$$S_{10} = \{I_j, I_k \in C_a \in \mathcal{C} \wedge I_j, I_k \notin C'_b \in \mathcal{C}'\}$$

$$S_{01} = \{I_j, I_k \notin C_a \in \mathcal{C} \wedge I_j, I_k \in C'_b \in \mathcal{C}'\}$$

$$S_{11} = \{I_j, I_k \in C_a \in \mathcal{C} \wedge I_j, I_k \in C'_b \in \mathcal{C}'\}$$

For convenience, let $n_{ab}$ be the cardinality of the set $S_{ab}$ where $a, b \in \{0, 1\}$. The cardinality of the union of all sets $S_{00}, S_{10}, S_{01}, S_{00}$ is:

$$n_{00} + n_{10} + n_{01} + n_{11} = \binom{n}{2}$$

because we are considering each pairing of isolate data points $I_j$ and $I_k$. This notation is accurate only for two clusterings $\mathcal{C}$ and $\mathcal{C}'$ of the same data set $\mathcal{I} = \{I_1, \ldots, I_n\}$.

## 7.2 Evaluation Metrics

For the comparison of clusters produced by OHClust! and agglomerative clustering, we use four comparison metrics:

- Jaccard Index

- Dice's coefficient
- Rand Index
- Variation of Information

The Jaccard Index, Rand Index, and variation of information are explained very well by Wagner and Wagner [46]. The Jaccard Index was described as being commonly used for ecology (e.g. for measuring species diversity between two different communities), which seems like a natural choice for evaluation. Dice's coefficient is slightly different from Jaccard, using the cardinality of the two sets being compared as the denominator instead of the cardinality of the union of the two sets being compared. The Rand Index is a standard metric for comparing sets of clusters and has been mentioned multiple times in literature [46, 48]. Finally, the variation of information metric was described by Wagner and Wagner as having extensive analysis, further detailed by Meilă [26].

The Jaccard Index measures the similarity between two sets as the cardinality of their intersection over the cardinality of their union:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This produces a value from 0 to 1 such that a value of 1 represents a 100% overlap of the two clusters, and 0 represents disjoint sets. We adjust the Jaccard Index for comparing two sets of clusters $\mathcal{C}$ and $\mathcal{C}'$ as follows:

$$\mathcal{J}(\mathcal{C}, \mathcal{C}') = \frac{n_{11}}{n_{10} + n_{01} + n_{11}}$$

where $n_{11}$ represent the intersection, or agreement, between $\mathcal{C}$ and $\mathcal{C}'$, and $n_{10} + n_{01} + n_{11}$ represents the union, or isolate pairs that are in the same cluster in either $\mathcal{C}$, $\mathcal{C}'$, or both. The only isolate pairs not considered in the Jaccard Index

function are pairs that are in different clusters in both $\mathcal{C}$ and $\mathcal{C}'$. However, these pairs are accounted for in Dice's coefficient.

Dice's coefficient measures the similarity between two sets as the cardinality of their intersection over the cardinality of all possible elements:

$$D(A, B) = \frac{2\,|A \cap B|}{|A| + |B|}$$

When comparing two sets, Dice's coefficient has a higher penalty for elements that are not present in both $A$ and $B$. Dice's coefficient is formulated as follows when adjusted for comparing clusters:

$$\mathcal{D}(\mathcal{C}, \mathcal{C}') = \frac{2\,|n_{11}|}{n_{00} + n_{10} + n_{01} + n_{11}}$$

Unlike the Jaccard Index, Dice's coefficient gives extra penalty for isolate pairs that are in different clusters in both $\mathcal{C}$ and $\mathcal{C}'$. This means that while the Jaccard Index does not penalize the similarity between two clusters $\mathcal{C}$ and $\mathcal{C}'$ for single isolate clusters, Dice's coefficient does. It is preferable, however, not to penalize the similarity of two sets of clusters for single isolate clusters, so we consider Jaccard Index with more weight than we do Dice's coefficient. It is useful however to have insight into the amount of cluster agreement in comparison to the number of single isolate clusters. Alternately, the Rand Index considers isolate pairs in the set $S_{00}$ as part of the similarity between two sets of clusters.

The formulation of the Rand Index we use is:

$$\mathcal{R}(\mathcal{C}, \mathcal{C}') = \frac{n_{11} + n_{00}}{\frac{n\,(n-1)}{2}} = \frac{2\,(n_{11} + n_{00})}{n\,(n-1)}$$

Notice that the denominator $\frac{n\,(n-1)}{2}$ is the number of isolate pairs being considered ($\binom{n}{2} = \frac{n\,(n-1)}{2}$). The numerator $(n_{11} + n_{00})$ represents the isolate pairs with the same cluster assignment, or classification, in both sets of clusters $\mathcal{C}$ and $\mathcal{C}'$.

The Rand Index produces a similarity between two sets of clusters from 0 to 1, where 0 means that each set of clusters disagree on every cluster assignment. A Rand Index of 1 means that each set of clusters completely agree on cluster assignment. The Rand Index is a simple comparison metric that counts the number of agreements between each set of clusters and then divides by the total number of isolate pairs considered.

Finally, the comparison metric for variation of information between two clusterings is a similarity measure based on entropy. The motivation for this comparison metric is to determine the change in information when going from one set of clusters $\mathcal{C}$ to another set of clusters $\mathcal{C}'$. To calculate the variation of information, we need to first understand entropy of an entity. Consider the entropy $H$ of a DNA sequence $d$:

$$H(d) = -\sum_{a \in \Sigma} p_i \log_2 (p_i)$$

To calculate the entropy of a DNA sequence, we consider each nucleotide $a$ in our alphabet $\Sigma = \{A, T, C, G\}$. Intuitively, we are interested in the probability that a unique value occurs in our input, where our input is the DNA sequence $d$, a unique value is a nucleotide $a$, and an occurrence of $a$ is simply the number of times $a$ is observed in $d$. The probability that a nucleotide $a$ occurs in our input, then, is simply $p_i = \frac{|a|}{|d|}$. For a DNA sequence $d$ that has an even distribution of **A**s, **T**s, **C**s, and **G**s, $H(d) = 2$. In contrast, a DNA sequence $d$ that has only **A**s has an entropy of $H(d) = 0$. The range of entropy $H$ is from 0, for homogeneous inputs, to $-\log_2 (\frac{1}{|\Sigma|})$ for perfectly heterogeneous inputs. The formulation of $H$

for comparing clusters is as follows:

$$
\begin{aligned}
P(I_j) &= \frac{|C_a|}{n} \\
\mathcal{H}(\mathcal{C}) &= -\sum_{a=1}^{j} P(I_j)\, log_2\, P(I_j) \\
P(I_j, I_k) &= \frac{|C_a \cap C'_b|}{n} \\
\mathcal{I}(\mathcal{C}, \mathcal{C}') &= \sum_{b=1}^{k} \sum_{a=1}^{j} P(I_j, I_k)\, log_2\, \frac{P(I_j, I_k)}{P(I_j)\, P(I_k)}
\end{aligned}
$$

Where $P(I_j)$ is the probability that a randomly chosen isolate $I_j$, is in cluster $C_a \in \mathcal{C}$. Using $P(I_j)$, it is now possible to calculate the entropy for a set of clusters $\mathcal{C}$ using $\mathcal{H}(\mathcal{C})$. We then extend this to account for the entropy of two sets of clusters based on isolate pairs, much as we did for previous evaluation metrics. We use $P(I_j, I_k)$ to calculate the probability that an isolate pair $I_j, I_k$ is in a cluster $C_a \in \mathcal{C}$ and in a cluster $C_b \in \mathcal{C}'$. We then use $\mathcal{I}(\mathcal{C}, \mathcal{C}')$ to calculate the *mutual information* between two clusterings $\mathcal{C}$ and $\mathcal{C}'$. Mutual information extends the idea of homogeneity and heterogeneity for a single input, and considers the homogeneity and heterogeneity of agreement for cluster assignment between two inputs. In this case, our two inputs are a set of clusters $\mathcal{C}$ produced by OHClust! and a set of clusters $\mathcal{C}'$ produced by agglomerative clustering. By summing the entropy of each set of clusters, and subtracting the mutual information from each, we are then able to compute the mutual information between two sets of clusters:

$$
\begin{aligned}
\mathcal{VI}(\mathcal{C}, \mathcal{C}') &= \mathcal{H}(\mathcal{C}) + \mathcal{H}(\mathcal{C}') - 2\,\mathcal{I}(\mathcal{C}, \mathcal{C}') \\
&= [\mathcal{H}(\mathcal{C}) - \mathcal{I}\,(\mathcal{C}, \mathcal{C}')] + [\mathcal{H}(\mathcal{C}') - \mathcal{I}\,(\mathcal{C}, \mathcal{C}')]
\end{aligned}
$$

## 7.3 Evaluation Data

Currently, CPLOP (Cal Poly Library of Pyroprints) contains about 10000 pyrorints and 4300 isolates. While CPLOP is still growing, this is not enough data to properly compare the performance of OHClust! against agglomerative clustering. To address this, the existing data was used to generate new data.

To generate isolates and pyroprints we apply genetic algorithm concepts– crossover and mutation. This approach selects two isolates at random, multiple times, from a random pool of a given size. If the two isolates are identical, then they are simply duplicated with a new ID. If they are not identical, then there is an 80% chance that a crossover will occur between the two isolates. In the event that a crossover does not occur, then mutation will occur in one of the two isolates selected at random with equal probability.

**Crossover.** Crossover is an analog to the process that happens between two chromosomes, called chromsomal crossover. This occurs between two chromosomes that are homologous (pair of chromosomes for the same traits, i.e. one chromosome from mom and one chromosome from dad) to each other, where a section of one swaps with the adjacent section of the other. At the end of the process each chromosome is partly original content, and partly content from the other chromosome. When crossover occurs during data generation, a pyroprint $p_a^i$ from one isolate $I_a$ is selected at random. Then a random pyroprint $p_b^i$ from the same ITS region $R_i$ is selected from the second isolate $I_b$. The location of the crossover that occurs between the two pyroprints is randomly chosen in the last 25% of the pyroprint peak heights. This corresponds to somewhere in the last 23 dispensations or peak heights. Once the location is chosen, then the data

from one pyroprint is swapped with the data from the other, from the chosen location to the end of the pyroprint vector. Then, there is a 25% chance that a second, or double, crossover will occur between the two pyroprints. In the case of a double crossover, the second region that is swapped is randomly chosen from the $25\% - 50\%$ of the pyroprint vector. The second crossover is always for 25% of the pyroprint vector length.

**Mutation.** Mutation is the process where nucleotide sequences, or even single nucleotides, in DNA are changed. When mutation occurs during data generation, each isolate has a 50% chance of being chosen to be mutated. Once an isolate is selected, a pyroprint is selected at random with each pyroprint having equal probability of being chosen. Once a pyroprint is chosen, each peak height in the pyroprint vector is scaled $X \in [-15, +15]\%$, with each peak height being mutated independently of others.

While many of the parameters were arbitrarily set, the main goal for this data is to investigate the performance of OHClust! and agglomerative clustering on large datasets. Additionally, because the accuracy of resulting clusters are not compared against an optimal clustering, but between OHClust! and agglomerative clustering, the results should be similar for the same pool of isolates regardless of the relationships present in the data. To provide some form of control, however, each algorithm is also run on the full CPLOP dataset.

## 7.4 Hypothesis

For clustering performance, it is expected that OHClust! will perform similarly to agglomerative clustering for a single or initial run. For smaller datasets, the

overhead associated with building and filling the an ontology used by OHClust! is likely to worsen performance when compared to agglomerative clustering. However, for large datasets, the partitioning of data points into $O(p^k)$ paths of an ontology should make OHClust! perform better. For cluster results, it is expected that OHClust! will produce clusters that are a *refinement* of clusters from agglomerative clustering. A refinement is where each cluster $C_a$ in a set of clusters $\mathcal{C}$ is a subset of a cluster $C'_b$ from a set of clusters $C'$. In this case, $\mathcal{C}$ is a set of clusters produced by OHClust! and $C'$ is a set of clusters produced by agglomerative clustering. Wagner and Wagner formalize this concept as follows [46]:

$$\forall C_a \in \mathcal{C},\ \exists C'_b \in \mathcal{C}'\ : C_a \subseteq C'_b.$$

We formalize our expectation of cluster quality as the null hypothesis that OHClust! and agglomerative clustering produce similar cluster results for the same dataset:

$$H_0 : \mathcal{VI}(\mathcal{C}, \mathcal{C}') \approx \frac{2}{n}.$$

Formally, the variation of information has a lower bound of $\frac{2}{n}$ between two clusterings $\mathcal{C}$ and $\mathcal{C}'$ where $\mathcal{C} \neq \mathcal{C}'$. If our null hypothesis is true, then the variation of information will be approximately $\frac{2}{n}$, showing that clusters produced by OHClust! and agglomerative clustering will be highly similar.

## 7.5 Evaluation Setup

This evaluation was done using java and attempted in many other combinations of python, CUDA, and java. Initially tests were run using a java implementation of the SPAM framework. After it became apparent that calculating Pearson's correlations for isolates was a bottleneck, a more optimized python/pycuda

implementation of OHClust! and agglomerative clustering for pyroprints was developed. Where the java code was developed to be maintainable and modular, the python/pycuda code took several shortcuts and utilized Nvidia GPUs so that more tests could be run in a shorter amount of time. While python and pycuda were chosen due to the speed of development and the ability to use CUDA, the python itself proved to be a significant bottleneck. Although CUDA was used to calculate the similarity matrix in as little as three seconds for 15000 isolates, the python clustering code is itself so much slower than its java counterpart that this was unusable as well. After this, a simplified java version was developed that takes the same shortcuts taken in python and pycuda, but that simply uses a threadpool for calculating the isolate similarity matrix and cluster similarities on CPU.

The evaluation was done using a machine with two Intel Xeon E5-2650 processors with 8 cores each, but hyper-threaded, making for 16 available cores on each processor, at 2GHz each. Sixty-four GB of RAM was available, though the java heap size typically never grew over 8 GB. There are also two Nvidia K20x GPUs available that were used when CUDA was being explored.

The initial java code was a direct implementation of what was described in Chapter 6. However, python and pycuda took several shortcuts, which are now present in the current multi-threaded java implementation. Generally speaking, an isolate may have any number of pyroprints $m$ associated with any number of ITS regions $n$. In order to gather evaluation data in time, only 1 pyroprint for each of 2 ITS regions was used. However, the evaluation conducted may still be considered appropriate, since there are only 2 ITS regions present in the actual data, and over 60% of isolates have only 1 pyroprint constructed for each ITS region. More formally, instead of calculating an average or median similarity

between the same ITS region $sim(R_i^a, R_i^b)$ of two isolates $I_a$ and $I_b$ (possibly having many pyroprints each), each isolate only consisted of a single pyroprint $p_x^a$ for the region $R_i$. This guaranteed that an isolate could be represented as two pyroprints concatenated together as a single vector. By convention, the 23S–5S ITS region was the first 93 indices the vector and the 16S–23S ITS region was made up the second 95 indices of the vector. By doing this, the dataset of isolates to be clustered could be represented as a single array, where the vector representing each isolate were stored adjacently. This made the CUDA code very easy to write and relatively fast in implementation. I haven't compared different shapes or layouts of the isolate data, but seemingly it has made the multi-threaded java implementation also easier and more efficient. Unfortunately, due to the comparison of clusters requiring a cross product of member isolates, the simplest approach to speeding up evaluation was to store a single similarity matrix. Being careful, this is easily addressable using K20x GPUs, each of which have about 6GB of global memory. To be more efficient in general, storage of the isolate similarity matrix was done as a single vector corresponding to a packed representation of one half of the similarity matrix–similarities are bi-directional and the diagonal does not need to be calculated. Additionally, the storage of data in MySQL was optimized for querying by the evaluation code. While typically the schema is normalized into many tables, the data is condensed into one table on which a join with another table must be made to query data. When a new set of random isolates is to be clustered, the database selects from all isolates at random and prepares a table containing isolate metadata and two pyroprint IDs to be joined with pyroprint peak height data. Insertion into this table, `isolate_selection`, is done in a manner that maintains pyroprint peak height order. In this way, when joining the table with the pyroprint peak heights, an

order by is not necessary and a query for all necessary information for 11000 isolates takes approximately 14 minutes.

For our evaluation we consider the following information:

- Performance:
  - Run time of clustering for an entire run
  - Run time of cluster after each update
  - Run time of clustering for the initial run
- Similarity of cluster results:
  - Cluster diameters
  - Cluster separation
  - Results of evaluation metrics

## 7.6 Results

This section presents evaluation of OHClust! results against agglomerative clustering based on performance and cluster quality. For all evaluations, agglomerative clustering is used as a baseline against which OHClust! is compared. In order to properly interpret the results of OHClust! clusterings, the ontologies used in evaluations are first discussed. Then the performance of OHClust! and similarity of clusters in comparison to agglomerative clustering are discussed in Sections 7.6.2 and 7.6.3, respectively.

### 7.6.1 Test Ontologies

The ontologies pictured in Figures 7.1 and 7.2 are used for all of the OHClust! tests. Where the ontology in Figure 7.1 is used for most tests of OHClust!, variants of the ontology in Figure 7.2 are used to measure the performance of ontologies with varying depths and varying widths. For clarity, the following sections refer to the ontology pictured in Figure 7.1 as the *fixed ontology*, whereas the ontology pictured in Figure 7.2 is referred to as the *dynamic ontology*.

The fixed ontology has a depth of only two, but uses every possible partition available in the database (uses every distinct value of a database column as an edge to a child node). This ontology provides a minimal ontology on the data, meaning the comparisons between OHClust! and agglomerative clustering should be quite similar. Although this ontology provides two levels of depth, the second level (commonName) has a small number of partitions by which to partition data separated in the first level (userName). The reason for this being that many of the biologists collected samples from only one to two different host species. The fixed ontology is the most deeply explored as it provides a simple application of OHClust! in comparison to agglomerative clustering.

The dynamic ontology is dynamically extended in two ways in order to obtain a more comprehensive understanding of the effects of the ontology on performance. First, the dynamic ontology is incrementally built top-down. In this case, the first level (hostID) is used for computing a clustering, then the first and second levels (hostID and commonName) are used for computing a clustering, then all three levels are used. The incremental extension of the depth of the dynamic ontology reveals the influence of ontology depth on OHClust! performance. It is important to understand the influence of ontology depth on clustering per-

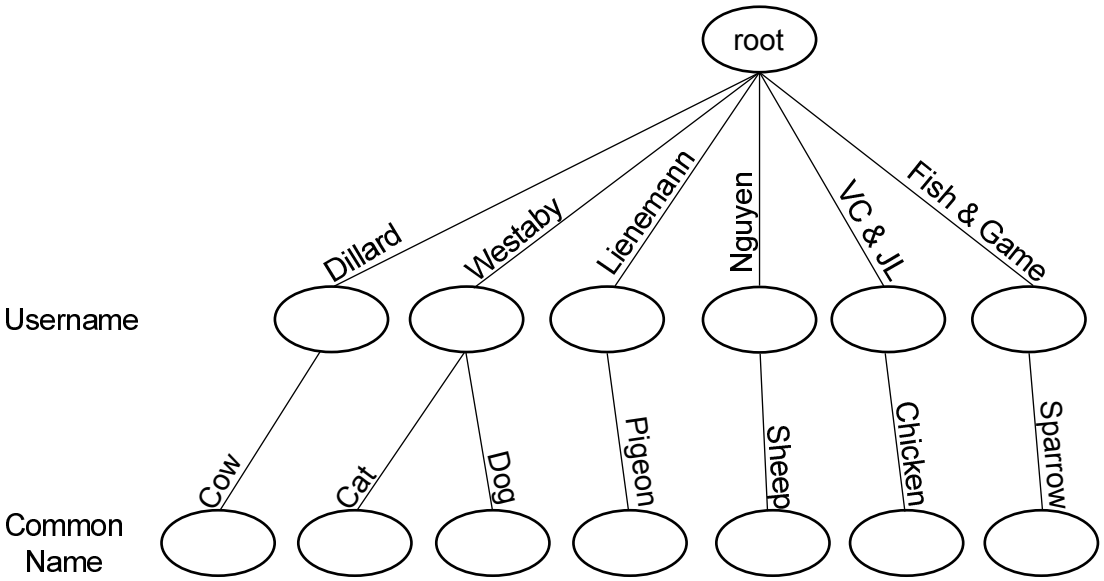



Figure 7.1: A portion of the fixed ontology used for most tests consisting of the seven largest partitions of the data set.

formance in order to determine how well OHClust! scales with experimental complexity.

After testing the top-down construction of the dynamic ontology, the number of partitions present in the dynamic ontology is varied. This provides insight into the influence of the width of the ontology on OHClust! performance. This is done by first using the dynamic ontology with only the partitions labeled:

- *unk*, *1*, and *15* on the first level.
- *Sheep*, *Pigeon*, and *Human* on the second level.
- *Westaby*, *Nguyen*, and *CA Dept Fish and Game* on the third level.

There are four clustering runs executed that use three, four, five, and six partitions on each level of the ontology, where the dynamic ontology having six partitions on each level is the one pictured in Figure 7.2. While there are some

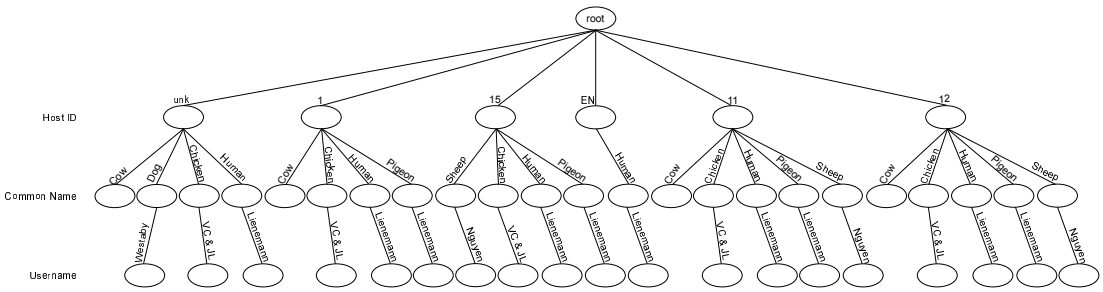



Figure 7.2: A portion of the ontologies with varying depths used for determining influence of ontology on performance. Only shows the partitions with the most data present.

nodes seemingly missing from the dynamic ontology, Figure 7.2 only shows the useful paths of the ontology. Isolates labelled with "CA Dept Fish and Game" unfortunately do not have labels matching any of the six common names. For this case, and other similar cases, isolates will not be propagated lower than the root node. While this is not a completely accurate evaluation for testing varying widths of the ontology, due to time constraints, an improved version of this evaluation will not make it into this thesis.

### 7.6.2 OHClust! Performance

For performance results presented in graphs, OHClust! results are labeled "ohclust" and agglomerative clustering results are labeled "agglom" (or "agglomerative" where "agglom" was not accepted by the spreadsheet program as a label). The label for a line on a graph is additionally suffixed with "_<number>" where appropriate, to differentiate between results for different test runs with the same parameters.

The graph pictured in Figure 7.3 shows performance times for OHClust! vs agglomerative clustering on datasets of size 500, 1000, 1500, 2000, and 2500. This

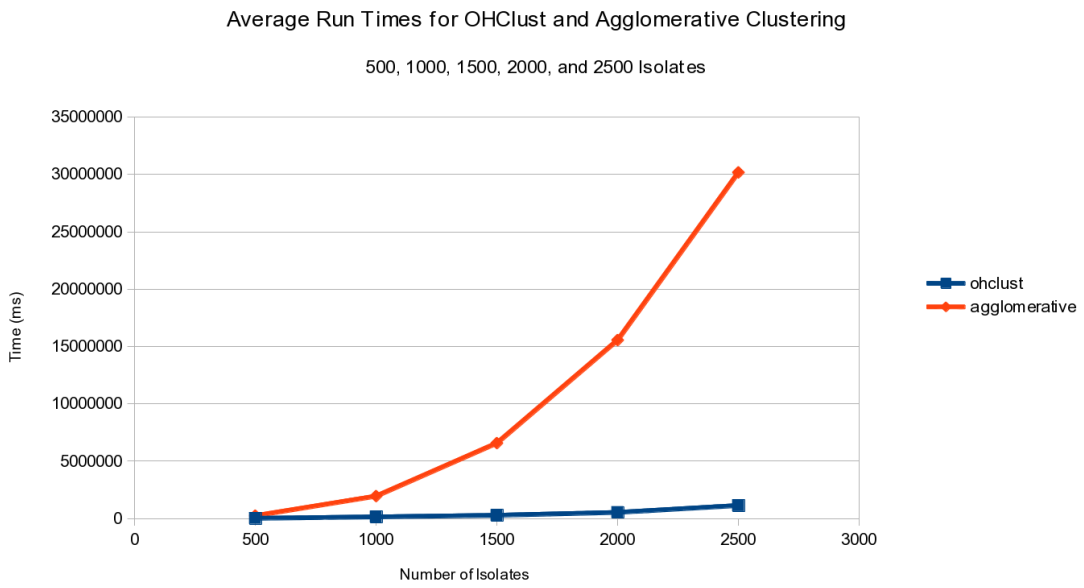



Figure 7.3: Runtime for OHClust! vs Agglomerative for fixed data set size of 500, 1000, 1500, 2000, and 2500 isolates.

graph is particularly meaningful because it is for fixed data set sizes. This provides the best "apples to apples" comparison of the performance of both OHClust! and agglomerative clustering because OHClust! was designed to be able to handle incremental updates. However, even in this case where there are no incremental updates, OHClust! clearly performs significantly better.

Figure 7.4 shows the cumulative run time for each given incremental update. As expected, OHClust! significantly outperforms agglomerative clustering when incremental updates are considered. In the future, after both clustering implementations have been improved, this evaluation will be run on larger data sets in order to ensure that this trend is consistent for cases where the overhead of the ontology or memory usage becomes a limitation.

Figure 7.5 shows a slightly different perspective on the data shown in Figure 7.4. In this graph, we see the run time for a given incremental update instead

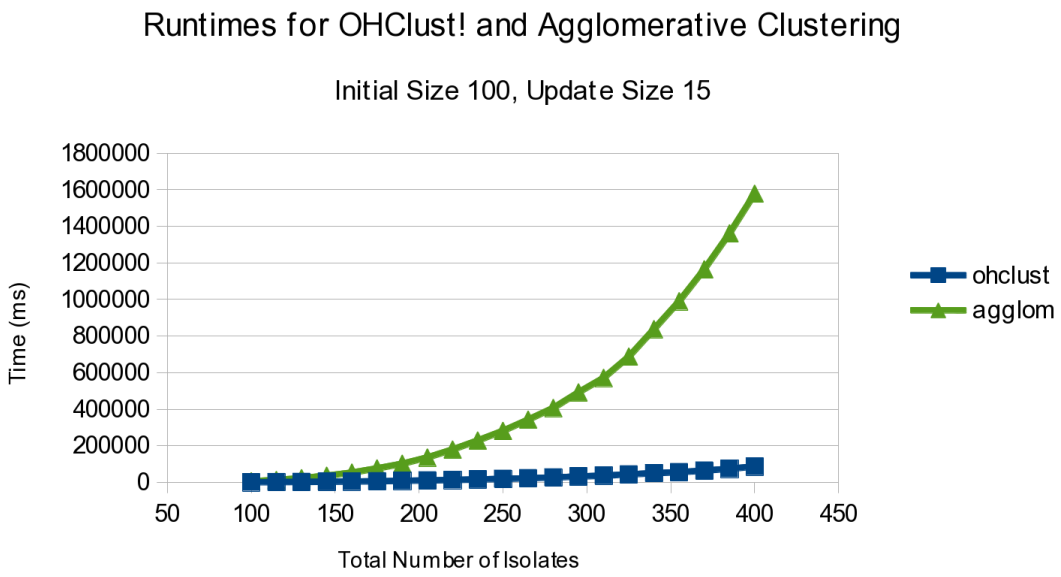



Figure 7.4: Total Runtime for OHClust! vs Agglomerative with initial size 100 and 20 updates of 15%.

of the cumulative run time for contiguous incremental updates. Given the data in Figure 7.4, this behavior is to be expected, although this also verifies that the improved performance of OHClust! over agglomerative clustering are not merely due to the behavior seen in Figure 7.3.

For the performance results presented in Table 7.1, OHClust! performance times are pictured in gray, while agglomerative clustering performance times are pictured in white. This table presents information for three important aspects:

- Initial clustering
- Clustering for updates
- Total clustering

Unfortunately,

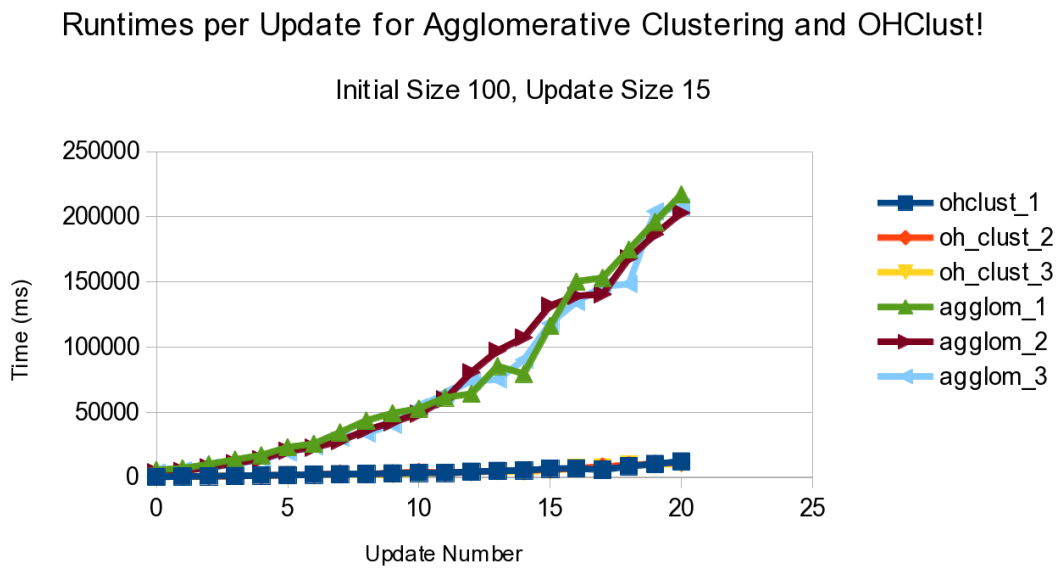



Figure 7.5: Runtime for incremental updates for OHClust! vs Agglomerative with initial size 100 and 20 updates of 15%.

The results shown in Table 7.2 are interesting in that they show that the ontology depth has minimal impact on runtime. I suspect that there was a problem with the ontology used, and that the ontology depth has no impact on performance is simply because some of the paths to leaves are useless, meaning they do not represent actual combinations of metadata seen in the database. Unfortunately, due to time constraints, a re-run of this evaluation with a more meaningful ontology will not be included in this thesis. Although, it is important to note that a very deep, or tall, ontology is only useful if the possible paths from the root to each leaf is useful. This means that if data collection is biased for some values of metadata, i.e. more *E. coli* isolates are present in the database for only a few host species, then the data contained in the ontology will not be evenly distributed, and so extra levels adds minimal or no improvement. In order to better evaluate the influence of ontology depth in the future, a synthetic data

| Algorithm | Initial | | Update | | | Total | |
|---|---|---|---|---|---|---|---|
| | Size | Time | Number | Size | Avg Time | Size | Time |
| OHClust! | 100 | 0.398 s | 20 | 15 | 4.276 s | 400 | 85.927 s |
| agglomerative | 100 | 4.345 s | 20 | 15 | 76.991 s | 400 | 1544.17 s |
| OHClust! | 500 | 27.519 s | 20 | 25 | 92.623 s | 1000 | 1883.56 s |

Table 7.1: Run times for OHClust! and agglomerative clustering.

| Ontology | Total Size | Average Total Runtime |
|---|---|---|
| **small.ont** | 500 | 452.93 s |
| **medium.ont** | 500 | 461.48 s |
| **large.ont** | 500 | 460.72 s |
| **small.ont** | 1000 | 2258.22 s |
| **medium.ont** | 1000 | 2192.47 s |
| **large.ont** | 1000 | 2292.60 s |
| **small.ont** | 1500 | 5317.02 s |
| **medium.ont** | 1500 | 5167.73 s |
| **large.ont** | 1500 | 5067.26 s |

Table 7.2: Cluster performance for varying depths of an ontology.

set containing with more diverse metadata values may be much more useful.

Unfortunately, although ontologies of various widths were evaluated, some of the tests were ran with incorrect configurations. The only tests that produced usable results were for a total data set size of 1000, shown in Table 7.3. The results in this table also show minimal impact on the performance of OHClust! for an ontology of varying width. A three level ontology was used where the first test used three nodes on each level, the second test used four nodes on each level,

| **Ontology** | Total Size | Average Total Runtime |
|---|---|---|
| **full_3_3_3.ont** | 1000 | 14663.85 s |
| **full_4_4_4.ont** | 1000 | 13171.74 s |
| **full_5_5_5.ont** | 1000 | 12436.13 s |
| **full_6_6_6.ont** | 1000 | 11851.55 s |

Table 7.3: Cluster performance for varying widths of an ontology.

up to six nodes on each level. For a random data set and a not well-balanced ontology, the average total runtime increases 0.81% from three nodes on each level to six nodes on each level. The ontology used was the same ontology as "large.ont" used for the results in Table 7.2.

### 7.6.3 Clustering Similarity

Interestingly, the cluster similarity evaluation metrics presented in Table 7.4 do not agree as much as was initially predicted. While the Rand Index agrees with the hypothesis that cluster results between OHClust! and agglomerative clustering should be very similar, Jaccard and Dice both seem to disagree. The initial thought is that since many clusters have a single isolate, the pairwise evaluation of cluster similarity used by Jaccard and Dice leads to a particularly low similarity.

In order to explore this possibility, the same evaluation metrics should be computed for all data points that are present in clusters of size $\geq 5$. These evaluations are presented in Table 7.5. However, it seems that the evaluation metrics report even lower similarity for clusters of size $\geq 5$. This definitely does not agree with our initial hypothesis, however the high Rand Index values in

| **Num Isolates** | Transform? | Jaccard | Dice | Rand | VI |
|---|---|---|---|---|---|
| **300** | Yes | 0.3486 | 0.0967 | 0.9097 | 3.43805 |
| **400** | No | 0.7934 | 0.7649 | 0.9004 | 1.92669 |
| **500** | Yes | 0.3907 | 0.0896 | 0.9302 | 4.16226 |
| **750** | Yes | 0.2503 | 0.0451 | 0.9324 | ——- |
| **1000** | No | 0.7295 | 0.4201 | 0.9221 | ——- |

Table 7.4: Cluster similarity results for various sized data sets.

| **Num Isolates** | Transform? | Jaccard | Dice | Rand |
|---|---|---|---|---|
| **300** | Yes | 0.0735 | 0.0308 | 0.8058 |
| **400** | No | 0.4118 | 0.5726 | 0.5911 |
| **500** | Yes | 0.0608 | 0.0233 | 0.82 |
| **750** | Yes | 0.0424 | 0.0121 | 0.8635 |
| **1000** | No | 0.2025 | 0.1894 | 0.6271 |

Table 7.5: Cluster similarity results for various data set sizes for clusters having more than one data point.

Table 7.4 in comparison to the Jaccard and Dice metrics seems to indicate a high similarity in some respect. While it's clear why Rand would fall–single isolate clusters can contribute to higher similarity in for this metric–the removal of single isolate clusters seems like it should greatly improve the Jaccard and Dice coefficients. Perhaps the naive removal of clusters with only a single data point introduced error into the comparison metrics, or perhaps there was some other error in the computation of the evaluation metrics.

Additionally, while the data set of size 400 has high agreement and coincides with our predictions, the tests with data set sizes 300 and 500 have very low

Jaccard and Dice similarity. Although, the rand index is still very good. The tests with 300 and 500 sized data sets used the same pool of random isolates. However, the test with 400 data points was ran with a different pool of isolates. I suspect that perhaps one data set simply had more isolates that were similar.

# Chapter 8

# Future Work and Conclusions

## 8.1 Future Work

There are many aspects of pyroprint analysis and SPAM that can be extended or improved in the future. While the application of OHClust! provides greater insight into sub-structures of clusters given metadata of interest, work on using this information to influence scientific workflows or interpretations of cluster analysis can prove to be even more useful. However, this is a very long-term possibility for future work.

Immediate future work for OHClust! includes more extensive evaluation and characterization, methods of determining optimal or preferable ontologies, and exploring other application areas. While the evaluation included in this thesis shows better performance for OHClust! more evaluation can provide insight on the use cases where OHClust! performance is best, and how different ontology shapes influence the performance. Characterization of the effects that the various feature options have on the performance and quality of clustering produced by

OHClust! would also be very useful. More interesting areas of future work include the comparison of ontologies for cluster quality or experimental relevance. Extensive work has been done for managing scientific workflows, and perhaps the application of an ontology and OHClust! can help to direct some of these scientific workflows. Finally, other applications of OHClust! that have not been explored include some natural language processing (NLP) problems, or perhaps highly-structured data sets such as traffic or automotive collision data.

Immediate future work for SPAM includes a re-implementation of comparison metrics, the implementation of more analysis methods, the implementation of other comparison metrics besides pearson's correlation, etc. These are all low-hanging fruit as there are many analysis methods or comparison metrics available in literature that could be applicable to pyroprints or even other applicable data sets. Additionally, optimization of the size, or memory requirements, of the ontology and the performance of OHClust! is always on the list of future work.

In addition to performance concerns, further integration of the OHClust! implementation with CPLOP is becoming incredibly important. The biologists are now conducting a microbial source tracking study on San Luis Obispo Creek and the ability to match and visualize collected environmental samples of *E. coli* against *E. coli* isolates from known host species is desirable. While the OHClust! implementation is able to address these issues in a basic manner, integration with CPLOP provides the ability for CPLOP to more quickly and effectively provide assistance to the biologists in interpreting the meaning and significance of isolate clusters, or strains. A lot of this integration has initially been addressed by Jan Soliman [43], however tighter coupling of SPAM and CPLOP can provide performance benefits and perhaps alleviate unnecessary resource utilization.

## 8.2 Conclusion

The work described in this thesis involves software tools for enabling a variety of biological research, namely population-based and MST studies. This research has significant importance for water quality assessment of any community. With the use of PyroprintDAT, pyroprinting protocols may be appropriately adapted for any environment being investigated. OHClust! has been shown to be an improvement over agglomerative clustering for use cases common in research involving pyroprinting which is being used by many students investigating both population-based and MST studies. While OHClust! results are not provably correct, the results and information it provides given a dataset and relevant metadata can really enhance the process through which biologists analyze and interpret their data.

Overall, the software tools developed as a part of this thesis greatly benefit the Biology Department at Cal Poly and helps to make the biological research being conducted possible.

# Bibliography

[1] Matthew A. Anderson, John E. Whitlock, and Valerie J. Harwood. Diversity and distribution of *Escherichia coli* genotypes and antibiotic resistance phenotypes in feces of humans, cattle, and horses. *Appl. Environ. Microbiol.*, 72:6914–6922, Nov 2006.

[2] S.L. Boyer, V.R. Flechtner, and J.R. Johansen. Is the 16s-23s rrna internal transcribed spacer region a good tool for use in molecular systematics and population genetics? a case study in cyanobacteria. *Molecular Biology and Evolution*, 18(6):1057–1069, June 2001.

[3] C. P. Bufano, M. W. BLack, J. VanderKelen, and C. L. Kitts. Comparing the dominant escherichia coli strains in humans, pigs, and cows using pyroprinting: A novel method for strain identification, May 2012.

[4] D.A. Caugant, B.R. Levin, and R.K. Selander. Genetic diversity and temporal variation in the e. coli population of a human host. *Genetics*, 98(3):467, 1981.

[5] E. Chase, J. Hunting, C. Staley, and V.J. Harwood. Microbial source tracking to identify human and ruminant sources of faecal pollution in an ephemeral florida river. *Journal of Applied Microbiology*, 113(6):1396–1406, 2012.

[6] Pankaj Chopra, Jaewoo Kang, Jiong Yang, HyungJun Cho, Heenam S Kim, and Min-Goo Lee. Microarray data mining using landmark gene-guided clustering. *BMC bioinformatics*, 9(1):92, 2008.

[7] Chris Cornelison, Marek Kirs, Brent Gilpin, and Paula Scholes. Microbial source tracking (mst) tools for water quality monitoring. Technical report, Cawthron Institute, Nelson, New Zealand, Jan 2012.

[8] Timothy R. Desmarais, Helena M. Solo-Gabriele, and Carol J. Palmer. Influence of soil on fecal indicator organisms in a tidally influenced subtropical environment. *Applied and Environmental Microbiology*, 68:1165–1172, Mar 2002.

[9] J. Dillard, J. Kent, D. Britton, T. Branck, A. Frey, P. McCreesh, J. VanderKelen, C. Kitts, and M. Black. E. coli strain demographics and transmission in cattle, Jan 2013.

[10] Jorge W. Santo Domingo, Regina Lamendella, and Nicholas J. Ashbolt. *Microbial Source Tracking: Current and Future Molecular Tools in Microbial Water Quality Forensics*, chapter 10. Caister Academic Press, Norfolk, U.K., 2011.

[11] Robert C. Edgar. Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461, 2010.

[12] David M. Gordon. Strain typing and the ecological structure of *Escherichia coli*. *Journal of AOAC International*, 93:974–984, May 2010.

[13] Charles Hagedorn, Anicet R. Blanch, and Valerie J. Harwood. *Microbial Source Tracking: Methods, Applications, and Case Studies*. Springer, 2011.

[14] Valerie J. Harwood, Kristin V. Gordon, and Christopher Staley. Final report: Validation of rapid methods for enumeration of markers for human sewage contamination in recreational waters. Technical Report 600-R-05-064, Water Environment Research Foundation, Alexandria, Virginia, June 2005.

[15] Liisa Holm and Chris Sander. Removing near-neighbour redundancy from large protein sequence collections. *Bioinformatics*, 14(5):423–429, 1998.

[16] Charles Hagedorn III, Brian L. Benham, and Sara C. Zeckoski. Microbial source tracking and the tmdl (total maximum daily loads) process. Technical Report 442-554, Virginia Cooperative Extension and Virginia Polytechnic Institute and Virginia State University, Virginia, May 2009.

[17] DOE Joint Genome Institute. Img data management & analysis systems. http://img.jgi.doe.gov/, Feb 2013.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sep 1999.

[19] M. B. Jenkins, P. G. Hartel, T. J. Olexa, and J. A. Stuedemann. Putative temporal variability of *Escherichia coli* ribotypes from yearling steers. *Environmental Quality*, 32:305–309, Jan 2003.

[20] Chris Kitts. Cal poly biology in action: Pismo beach source identification study, Dec 2010.

[21] Christopher L. Kitts, Michael W. Black, Mark M. Moline, Marie Yeung, Alice K. Hamrick, Ian C. Robbins, Andrew A. Schaffner, and Nathania I. Boutet. Pismo beach fecal contamination source identification study: Final report. *Biological Sciences*, Aug 2010.

[22] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[23] Weizhong Li, Lukasz Jaroszewski, and Adam Godzik. Tolerating some redundancy significantly speeds up clustering of large protein databases. *Bioinformatics*, 18(1):77–82, 2002.

[24] Julie Lowe, Debby Sargeant, and B. Kammin. Focus on microbial source tracking (mst). Technical Report 12-03-010, Department of Ecology, Washington, Olympia, Washington, Feb 2012.

[25] Elaine R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24:133–41, Mar 2008.

[26] Marina Meilă. Comparing clusterings: An axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 577–584, New York, NY, USA, 2005. ACM.

[27] Aldrin Montana, Alexander Dekhtyar, Emily Neal, Michael Black, and Chris Kitts. Chronology-sensitive hierarchical clustering of pyrosequenced dna samples of e. coli: A case study. In *2011 International Conference on Bioinformatics and Biomedicine*, pages 155–159. IEEE, 2011.

[28] Aldrin Montana, Alexander Dekhtyar, Emily Neal, Michael Black, and Chris Kitts. Investigating temporal strain diversity in human *E. coli* populations using pyroprinting: A novel strain identification method. Technical report, California Polytechnic State University, San Luis Obispo, CA, May 2012.

[29] Emily Neal, Collin Sabatini, Winnie Tang, Michael Black, and Chirs Kitts.

Demographics of e. coli strains in the human gut using pyroprints: A novel mst method, Jan 2012.

[30] J. Nguyen, J. VanderKelen, M. W. BLack, and C. L. Kitts. Investigating the dominant escherichia coli strains in lambs and ewes using pyrosequencing: A novel method for strain identification, May 2012.

[31] H. Ochman, T.S. Whittam, D.A. Caugant, and R.K. Selander. Enzyme polymorphism and genetic population structure in escherichia coli and shigella. *Microbiology*, 129(9):2715, 1983.

[32] Berenise Rivera and Channah Rock. Microbial source tracking: Watershed characterization and source identification. Technical report, University of Arizona, Tucson, Arizona, Aug 2011.

[33] Kathryn Robb. The optimal dispensation for the 16s-23s region and the optimal region that differentiates strains of escherichia coli more accurately. Technical report, California Polytechnic State University, San Luis Obispo, California, June 2011.

[34] Mostafa Ronaghi. Pyrosequencing sheds light on dna sequencing. *Genome Research*, 11:3–11, Jan 2001.

[35] A. Roth, M. Fischer, M.E. Hamid, S. Michalke, W. Ludwig, and H. Mauch. Differentiation of phylogenetically related slowly growing mycobacteria based on 16s-23s rrna gene internal transcribed spacer sequences. *Journal of Clinical Microbiology*, 36(1):139–147, Jan 1998.

[36] Debby Sargeant. Fecal contamination source identification methods in surface water. Technical Report 99-345, Department of Ecology, Washington, Olympia, Washington, Oct 1999.

[37] Debby Sargeant, William Kammin, and Scott Collyard. Review and critique of current microbial source tracking (mst) techniques. Technical Report 11-03-038, Department of Ecology, Washington, Olympia, Washington, Feb 2012.

[38] Theresa A. Schlager, J. Owen Hendley, Alison L. Bell, and Thomas S. Whittam. Clonal diversity of *Escherichia coli* colonizing stools and urinary tracts of young girls. *Infection and Immunity*, 70:1225–1229, Mar 2002.

[39] Troy M. Scott, Joan B. Rose, Tracie M. Jenkins, Samuel R. Farrah, and Jerzy Lukasik. Microbial source tracking: Current methodology and future directions. *Appl. Environ. Microbiol.*, 68:5796–5803, Dec 2002.

[40] H.J. Sears and I. Brownlee. Further observations on the persistence of individual strains of escherichia coli in the intestinal tract of man. *Journal of Bacteriology*, 63(1):47–57, Jan 1952.

[41] Diana Shealy. Exploration of pyroprinting for environmental forensics. Technical report, California Polytechnic State University, San Luis Obispo, California, June 2012.

[42] Joyce M. Simpson, Jorge W. Santo Domingo, and Donald J. Reasoner. Microbial source tracking: State of the science. *Environmental Science and Technology*, 36:5279–5288, Dec 2002.

[43] Jan Lorenz Soliman. Cplop: Cal poly library of pyroprints. Master's thesis, California Polytechnic State University, San Luis Obispo, California, 2013.

[44] W. Tang, C. J. Sabatini, E. R. Neal, A. Goodman, A. Dekhtyar, K. McGaughey, M. W. Black, and C. L. Kitts. Investigating changes in prevalent

human escherichia coli strains with different collection methods using pyroprinting: A novel mst method, May 2012.

[45] S.D. Tyler, C.A. Strathee, K.R. Rozee, and W.M. Johnson. Oligonucleotide primers designed to differentiate pathogenic pseudomonads on the basis of sequencing of genes coding for 16s-23s rrna internal transcribed spacers. *Clinical and Diagnostic Laboratory Immunology*, 2(4):448–453, July 1995.

[46] Silke Wagner and Dorothea Wagner. Comparing clusterings–an overview. Technical Report 2006-04, Universität Karlsruhe, Fakultät für Informatik, 2007.

[47] Shibu Yooseph, Weizhong Li, and Granger Sutton. Gene identification and protein classification in microbial metagenomic sequence data via incremental clustering. *BMC bioinformatics*, 9(1):182, 2008.

[48] Genane Youness and Gilbert Saporta. Comparing partitions of two sets of units based on the same variables. *Advances in Data Analysis and Classification*, 4(1):53–64, Apr 2010.

[49] Steven Young, Itamar Arel, Thomas P. Karnowski, and Derek Rose. A fast and stable incremental clustering algorithm. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 204–209, 2010.

[50] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.

[51] Zejun Zheng, Stefan Kramer, and Bertil Schmidt. Dysc: software for greedy clustering of 16s rrna reads. *Bioinformatics*, 28(16):2182–2183, 2012.