CPLOP: THE CAL POLY LIBRARY OF PYROPRINTS

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

Jan Lorenz Soliman

June 2013

COMMITTEE MEMBERSHIP

TITLE: CPLOP: The Cal Poly Library of Pyroprints

AUTHOR: Jan Lorenz Soliman

DATE SUBMITTED: June 2013

COMMITTEE CHAIR: Dr. Alexander Dekhtyar

                                Professor, Department of Computer Science

COMMITTEE MEMBER: Dr. Zoë Wood

                                Professor, Department of Computer Science

COMMITTEE MEMBER: Dr. Franz Kurfess

                                Professor, Department of Computer Science

COMMITTEE MEMBER: Dr. Christopher Kitts

                                Professor and Chair, Department of Biology

ABSTRACT

CPLOP: The Cal Poly Library of Pyroprints

by

Jan Lorenz Soliman

To date, microbial source tracking (MST), i.e. determining the source of microbial contamination based on the specific strains observed in the environment, often uses methods that are time-consuming, expensive and not always reliable. The biology department at Cal Poly, San Luis Obispo has developed a new method for MST called pyroprinting. Pyroprints can be used as DNA fingerprints for identifying sources of fecal contamination and studying bacterial populations in host animals. The MST method consists of two parts: the pyroprinting process and a database of pyroprints. The Cal Poly Library of Pyroprints (CPLOP) was developed to provide a database application for automating the storage and analysis of pyroprints. The initial version of CPLOP provided support for storing and organizing data related to microbes and their sources. This thesis describes the additional analytical tools needed to fully support pyroprinting as an MST method. This includes the ability to organize pyroprints into dataets, detect erroneous pyroprint data, run analyses to find similarities between bacterial isolates, and cluster isolates into bacterial strains.

ACKNOWLEDGMENTS

I would like to thank the following people whom I've worked with during the past two years:

- Dr. Michael Black, Dr. Anya Goodman, Dr. Chris Kitts, Emily Neal, and Dr. Jennifer VanderKelen for providing direction on this project and being patient with me as I picked up the background information. I've learned so much from you the past two years.

- Aldrin Montana for putting the effort to catch me up on previous work to get started and for providing excellent advice and help each step of the way.

- My committee members, Dr. Franz Kurfess, Dr. Zoë Wood, Dr. Chris Kitts, for their support. Thank you for taking the time to sit on my committee.

- Alex Dekhtyar, for proving the knowledge and advice to allow me to finish my work and for supporting me throughout my graduate student career.

Contents

List of Figures

Chapter 1

Introduction

Diseases caused by foodborne bacterial pathogens are a major public health issue in California and around the world. Infections associated with agricultural products can have a significant, detrimental impact on the farmers and the people that consume their products. This harmful effect continues not only during the outbreak, but for years thereafter due to loss in consumer confidence in the agricultural product.

For example, since 1993, there have been nine outbreaks of *E. coli* infections associated with vegetables from the Salinas Valley in California [1, 13, 19]. The exact sources of the contaminations have not been identified. Bacterial communities in nature are dynamic and environmental pressures cause pathogen populations to change. It is likely that the offending pathogen is gone or at undetectable levels at the time of sampling.

Environmental forensics plays a vital role in the source tracking of foodborne outbreaks. The timely and appropriate application of microbial forensic techniques and experimental design can significantly impact the success of finding the source.

Microbial Source Tracking (MST), a branch of molecular forensics, is a field of study aimed to help identify and track contaminant bacteria in foods and environmental resources to their host species [14, 18]. One common approach to MST investigations is the build-

ing of a database from the collection of fingerprinted bacterial isolates with known fecal sources. The database approach requires the isolation of fecal indicator bacteria (FIB), commonly *E. coli*, from a variety of known fecal sources, followed by the database archiving of DNA fingerprints from these isolates.

When an unknown environmental isolate is encountered, it can be isolated, fingerprinted, and compared with existing fingerprints in a database. In this way, environmental isolates can be matched to isolates from the database and a fecal source can be identified [18].

Methods commonly used in MST investigations either suffer from poor discrimination between strains or are expensive and require several weeks of processing to produce data [3]. In addition, many of these methods, such as ribotyping, are slow, technically demanding, and have limited use, as they are not open to queries outside the laboratories that house them [16]. Other methods such as phage typing, need access to large phage databases and do not guarantee that all strains are typable. The database of source *E. coli* must be continuously monitored for the appearance of transient strains that pass through multiple sources. The database must also be continuously updated with strain data for fecal sources collected from the specific time and location where source-tracking studies are conducted. Thus, to be useful, the use of database-dependent MST requires an inexpensive, rapid, reproducible, and discriminating source-typing method.

Pyroprinting is a microbial source tracking method developed by a multidisciplinary research group consisting of students and faculty in the Biology, Chemistry, and Computer Science departments. Pyroprinting is inexpensive, rapid, and, as preliminary studies have shown, reproducible and discriminating. This method is based on fingerprinting a bacterial isolate by pyrosequencing a mix of DNA material extracted from specific locations in its genome, known to be the subject of mutations between bacterial strains [9, 17]. The collected digital fingerprints, called pyroprints are stored in a database and used for actual

source tracking.

The Cal Poly Library of Pyroprints (CPLOP) is the database application component of the Pyroprinting MST method. CPLOP is built to store collected pyroprints and use them in microbial source tracking.

CPLOP started as quarter-long class project in Cal Poly's Database Modeling and Design and Implementation course, taught by Dr. Alex Dekhtyar in Spring 2011. Two Biology faculty members, Dr. Chris Kitts and Dr. Michael Black, acted as customers to four teams of students. The teams elicited requirements, created a database model, and developed a web application for storing and managing pyroprints. At the end of the course, Kevin Webb, a student participating in one of the groups, completed his team's prototype and deployed it as a senior project. In December 2011, CPLOP version 1.0 was deployed at the URL http://cplop.cosam.calpoly.edu, which serves as the production version URL for the database since then.

CPLOP 1.0 provided functionality for storing pyoprints of *E. coli* in relational tables, allowing for easy viewing and querying. However tools for efficiently processing data, discriminating between pyroprints, and building strains were missing.

This thesis details the development of CPLOP 2.0, which provides analytical tools to the CPLOP system in order to fulfill the the necessary requirements for MST. CPLOP 2.0 provides support for efficiently matching isolates, allowing researchers with an unknown isolate to identify what strain it belongs to. CPLOP 2.0 has database structures to allow for the clustering of isolates to create strains. In addition, we built manual and automated quality-control techniques for rejecting erroneous data from the pyrosequencing machine.

The main contributions of this thesis are:

1. Database design for recognizing groups of similar pyroprints as being part of a strain.

2. Algorithms for the maintenance of strains throughout the life of the database.

3. Matching functionality for quickly discriminating between pyroprints.

4. Partial automation of the flagging and removal of erroneous pyroprints.

The rest of the document is organized as follows. Chapter 2 provides details on the background and history of pyrosequencing and how it is used to discriminate strains. Chapter 3 describes the requirements for CPLOP 2.0. Chapter 4 lays out the design of the new CPLOP version and describes its implementation. Chapter 5 describes the validation and evaluation of the new CPLOP system. Chapter 6 describes future work for the next version of CPLOP and Chapter 7 concludes with a summary of the work and its value to MST researchers at Cal Poly.

Chapter 2

Background and Previous Work

The Biology department at Cal Poly San Luis Obispo has been developing pyroprinting as a low-cost MST resource for use by the county, region, state, and interested researchers across the world. This method includes the collection of samples from the environment, the building of a digital DNA fingerprint library by using pyrosequencing methods, the creation of strains in the fingerprint library, and the matching of isolates of unknown origin against strains to determine the source.

2.1 Data Collection

The development of a library-dependent MST method depends on the collection of sufficiently large bacterial collections. Samples are collected from feces found in the environment. For the purpose of building the initial library, the feces must come from known species in order to map pyroprints in the database to the host species they were isolated from. Building a database of pyroprints with known host species is integral to pyroprinting as an MST method because it provides the database with context about the each sample, such as its host species, the location where the sample was collected, and the date. This

contextual information allows researchers to perform a diverse set of studies. For example, a study was conducted by several students where *E. coli* was sampled from a human subject over 14 days to determine how strain populations change over time.

Once the fecal samples have been collected, *E. coli* are isolated, confirmed, and cultured. The cultures are then grown overnight. *Polymerase Chain Reaction* (PCR) is performed to amplify the DNA in the samples and prepare them for pyrosequencing.

While understanding PCR is not integral to understanding pyrosequencing, it should be noted that PCR requires two short nucleotide sequences called *primers* to initiate DNA synthesis. The required sequences are called the *forward primer* and *reverse primer*. The forward primer binds to the left of the target DNA on the bottom strand. The reverse primer binds to the right of the target DNA on the top strand. A third primer, called the *sequencing primer* is the primer used during pyrosequencing.

2.2   Pyrosequencing

DNA *pyrosequencing* is used as a tool to generate strain fingerprints and meet the demands of a database-dependent MST assay [12].

There is a significant body of research concerning pyrosequencing and its place in bioinformatics research. However there is little about its use in as an MST method. Pyrosequencing has been lauded as a cheap, efficient form of DNA sequencing appropriate for use in oncology, microbiology, pharmacogenetics, and other related research areas [11, 10].

The pyrosequencing process occurs on the wells of a plate in a pyrosequencing machine. The Biology department at Cal Poly uses a **Pyromark Q24** machine, manufactured by Qiagen, with 24 wells and a single plate. A pyrosequencing run is defined as the entire pyrosequencing process performed by the Pyromark machine on a single plate. Figure 2.1 shows the Pyromark Q24 and its wells.
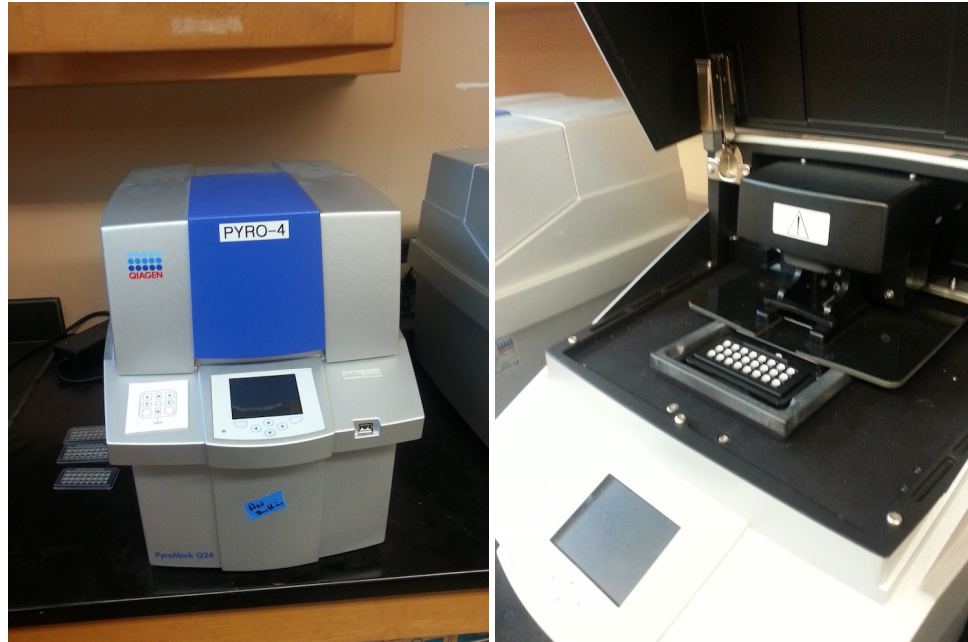
Figure 2.1: Pyromark Q24 machine with 24 wells.

Pyrosequencing builds a new strand of DNA, called a *sequencing strand*, that is complementary to the strand of DNA being sequenced. The strand being sequenced is called the *template strand*. The sequencing strand is built by incrementally synthesizing the complimentary strand of a single-stranded DNA product. This is done by repeatedly introducing nucleotide reagents into the DNA product. A nucleotide base, Adenine (A), Thymine (T), Cytosine (C), or Guanine (G) is added to the single-stranded DNA. The order of bases added in the pyrosequencing run is called the *dispensation sequence*. This order is specified by the technician or researcher for each run.

Each time the introduced reagent synthesizes one or more base pairs of the DNA strand, a binding reaction occurs. The reaction emits light, which is captured and measured by the pyrosequencing machine [11]. For example, if the complement of G is incorporated into the well causing a measured light emission of 100 units and then the complement of T is added causing a measured light emission of 50 units, then the template strand has twice as many G nucleotides as T nucleotides. The G nucleotides are also followed by a series of T

nucleotides. The pyrosequencing machine records the emission of light over time as each reagent is introduced. The amount of light emitted by the reaction is proportional to the number of nucleotides built on the sequencing strand.

Figure 2.2 describes the incremental building of a sequencing strand for the dispensation sequence **GCAGCATA**.

1. The first nucleotide in the dispensation sequence is Guanine (G). Guanine is added to measure the number of Cytosine (C) in the template strand. Since there is only one C in the template strand, only one G binds. This reaction emits light which is measured by the pyrosequencing machine as a light intensity histogram.

2. The process continues for the next two nucleotides in the dispensation sequence, C and A. Since there is only one of each of them, the histogram for each dispensation has the same height as the C histogram in step 1. When the next two nucleotides in the dispensation sequence, G and C, are added, the light emitted for each dispensation has double the height because there are two G and C nucleotides in the template strand at this point.

3. When a nucleotide not found in the template strand is dispensed, the emitted light is zero. The dispensed Adenine (A) nucleotide did not bind with anything causing no reaction.

4. The machine continues to dispense nucleotides based on the dispensation sequence. In this example, a Thymine (T) nucleotide binded with its complement, Adenine (A), three times, causing more light to be emitted. The histogram for this reaction has three times the height as a histogram for a single reaction.

The combined light emission histogram for each dispensation is recorded as a *pyrogram*. The pyrogram plots light emission against dispensation over time. A sample pyrogram is shown in Figure 2.3.
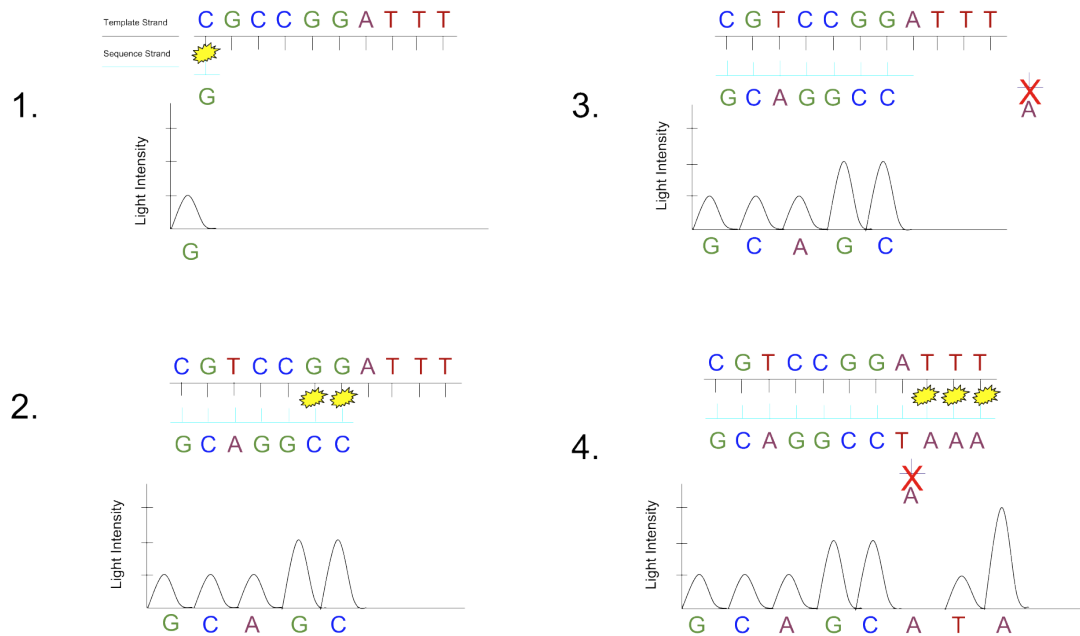
Figure 2.2: Example walkthrough of the pyroprinting process

To begin a run, first an investigator must create a run file specifying several parameters. Investigators use Pyromark Q24 software to create the run file. Pyromark Q24 software suite provides tools for setting parameters for a pyrosequencing run and viewing the results after the run has completed. This software suite runs on a desktop computer independent of the pyrosequencing machine.

Parameters including the pressure of the dispensation unit, the speed of the mixer, and the temperature of the heating block can be specified by investigators. For the purpose of our MST method, all parameters are given a default value except for dispensation sequence. Once created, the file is copied to a USB drive which is inserted into the Pyromark Q24 machine.

Once the parameters are provided to the Pyromark Q24 machine, the pyrosequencing process can start. The machine reads parameters from the Pyromark Q24. The nucleotides are predispensed into a rectangular well in the plate. Based on the dispensation sequence,
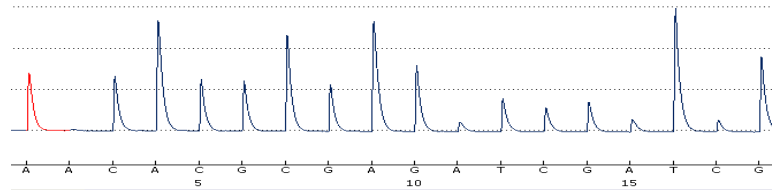
Figure 2.3: A section of a sample pyroprint graph.
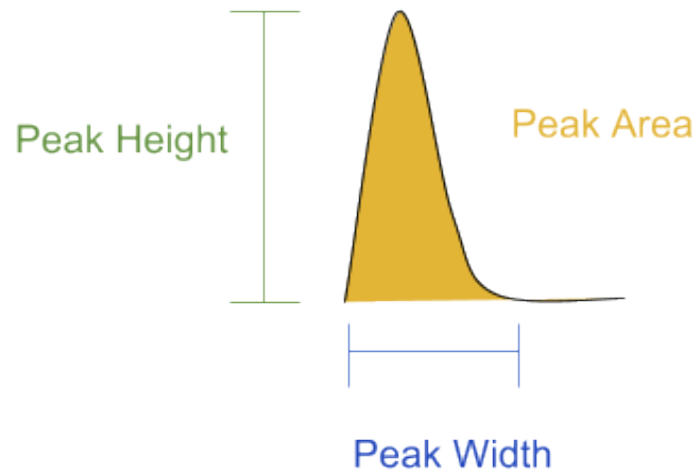
Figure 2.4: Peak area is the area under the curve. Peak width is the width of the histogram. Peak Height is the maximum value of the histogram within a dispensation.

nucleotides are dispensed into all used wells. The light values are measured by the machine and stored on the same USB drive where the run file is stored.

Once the pyrosequencing process is complete, the machine saves the results as a Pyrorun (.pyrorun) file. This file is transported from the Pyromark Q24 machine to a PC through USB drive.

The Pyrorun file contains XML data about the pyrosequencing run. The pyrogram is stored in the XML as a series of light intensity scores (e.g. total light output) corresponding to the addition of sequentially dispensed nucleotides. This series is represented as a vector of floating point values. The Pyrorun file can be viewed using Pyromark software. Investi-

10

gators can browse pyrogram data for each well using Pyromark software. Using pyrogram data, it is possible to determine the number of nucleotides complementary to the template region being sequenced. In addition, the XML contains other statistics on the pyrogram data at each dispensation including the peak height, area, and peak width. The definition of the three values are as follows:

1. *Peak Height* - The maximum value of the emittance graph for a dispensation.

2. *Peak Area* - The area under the emittance graph of a dispensation.

3. *Peak Width* - The width of the emittance graph for a dispensation.

Figure 2.4 shows a sample peak for a given dispensation and the emittance data associated with it.

2.3   Pyroprinting

DNA sequencing is commonly used in MST applications. Usually, to distinguish between two closely related strains, several genes from each strain are independently sequenced in order to find the differences. Pyrosequencing in MST takes a different approach. In addition to sequencing multiple DNA fragments separately, multiple polymorphic DNA templates are mixed in a single reaction.

While several values are provided by the pyrosequencing machine, the peak height value of the graph at each dispensation varies between strains based upon their allelic differences in the sequenced loci. The result is a unique identifier for each region, a DNA fingerprint. These pyrograms are referred to as *pyroprints*. *Pyroprinting* is the MST technique for generating pyroprints of DNA from highly variables regions in a microbial genome.

The *Intergenic Transcribed Spacer* (ITS) regions around the rRNA genes are used to generate pyroprints for *E. coli*. The ITS region is currently used to type *E. coli* strains by re-

Figure 2.5: An example of the 16S, 23S, 5S genes and the ITS regions between them.

striction fragment length polymorphism (RFLP) and sequence comparison [4]. Pyroprints are generated using short sections of each of the two ITS regions: one between 16S-23S genes and the other between 23S-5S genes. Figure 2.5 illustrates the three genes and the two ITS regions between them.

As there are typically seven copies of the rRNA operon (rDNA) in the *E. coli* genome, all seven ITS loci are amplified and sequenced together in a single reaction, maximizing the potential of discriminating between strains despite the relatively short sequence length generated by pyrosequencers. Figure 2.6 shows the ITS regions on the circular *E. coli* genome. In Figure 2.6, the seven sequences from each unique replicate of the 16S-23S region are shown in green. The seven sequences from the same replicates of the 23S-5S regions are colored blue. The nucleotides colored purple show the variability between template sequences from each replicate for each region. The variability between replicates of the 16S-23S region differs from the variability between replicates of the 23S-5S region with the 23S-5S region showing variability along a longer sequence size than the 16S-23S region.

One important detail about pyrosequencing is that the pyroprints cannot be used to determine the specific DNA sequences of the loci [6]. Rather, pyroprints provide a unique pattern of peaks for each unique sequence set analogous to a fingerprint and useful in discriminating between individual microbes. Reproducibility of the pyroprinting method has been verified in *in vitro* studies.

Figure 2.6: An example of the *E. coli* genome with the copies of the ITS regions labeled.

2.4 Pyroprinting as an MST Method

Using pyroprinting as a fingerprinting tool for Microbial Source Tracking is outlined in the following steps.

1. Collect, verify, and physically store *E. coli* isolates from a wide range of host species.

2. Culture and grow the *E. coli*.

3. Perform PCR to amplify the DNA from the samples.

4. Pyroprint the seven-locus ITS combinations for 16S-23S and 23S-5S regions. This results in two pyroprints per isolate: one for each region.

5. Create a database of collected pyroprints and store all obtained pyroprints together with identifying information and metadata.

6. Compare pyroprints in the database to determine if the *E. coli* are members of the same bacterial strain.

7. Organize individual bacterial isolates into strains based on pyroprints.

```
<Dispensations>
        <Dispensation>
          <Moment>193671</Moment>
          <Substance>A</Substance>
          <PeakValues>
            <SignalValue>17.26</SignalValue>
            <PeakArea>98.51</PeakArea>
            <PeakWidth>13</PeakWidth>
            <BaselineOffset>232.66</BaselineOffset>
            <SignalToNoise>97.76</SignalToNoise>
          </PeakValues>
        </Dispensation>
        <Dispensation>
          <Moment>258726</Moment>
          <Substance>A</Substance>
          <PeakValues>
            <SignalValue>3.98</SignalValue>
            <PeakArea>22.49</PeakArea>
            <PeakWidth>13</PeakWidth>
            <BaselineOffset>232.51</BaselineOffset>
            <SignalToNoise>22.54</SignalToNoise>
          </PeakValues>
        </Dispensation>
        ....
        ....
</Dispensations>
```

Figure 2.7: Sample XML data in a Pyrorun file. Each Dispensation tag refers to a dispensation histogram.

8. Solve MST problems involving isolate origin by comparing isolates with unknown origins to pyroprints in the database to find which strain and/or host it most likely came from.

2.5   Data Representation

As stated in Section 2.2, the Pyrosequencing machine produces a Pyrorun file containing XML data. The XML data describes information from the pyrosequencing run. Figure 2.7 shows part of the data stored in the Pyrorun XML.

Each Pyrorun file has several ways of representing the pyroprint histogram including peak height, peak width, area, and light emission scores. CPLOP stores all three values at each dispensation. The analyses built into this thesis use peak height but can be extended to use peak width or area. For the purpose of this thesis, pyroprint histograms are represented

14

using peak heights only. Based on several small, internal studies, biologists in our research group have chosen peak height as the representative way to measure light emissions for MST.

A pyroprint is represented as a vector of floating point values (light emissions) where each component corresponds to the peak height at each dispensed nucleotide. This way, the relationships between biological components are represented in an object-oriented design.

Isolates, pyroprints, and ITS regions are the biological components being examined in this MST application. These three entities are important for identifying strains. Isolates of interest may have several ITS regions. Each ITS region can be pyroprinted several times. To identify the host species of an unknown isolate, its pyroprints must be compared to pyroprints of the same ITS region for all of the isolates in the database.

The relationships between isolates, pyroprints, and ITS regions are defined using the following mathematical notations.

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

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

$$P_R = \langle R, d, P_{r_f}, P_{r_r} P_{r_s}, \{\langle I_1, p_1 \rangle, \ldots, \langle I_m, p_m \rangle\} \rangle$$

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

$$\mathcal{P} = P_{R_j} | R_j \in \mathcal{R}$$

I represents the set of all isolates in the database, $\mathcal{R}$ represents the set of all ITS regions in the genome. The database described in this thesis only considers $m = 2$ regions. d represents the dispensation sequence and consists of a vector $(d_1, \ldots, d_n)$ where $d_n \in \{A, T, C, G\}$.

$\mathcal{P}$ represents the entire set of pyroprints in the database for a given ITS region. Each pyroprint consists of a tuple containing the dispensation sequence used in pyrosequencing (d), the ITS region pyroprinted (R), and a set of all isolate-pyroprint pairs ($\{\langle I_m, p_m \rangle\}$) such

that I_m is the isolate the pyroprint originated from. P_{rf}, P_{rr}, and P_{rs} are the forward, reverse, and synthesis primers respectively.

A single pyroprint p_m is represented as a tuple. $\bar{p}$ also represents a vector $(p_1, \ldots p_n)$ where p_n is a real number representing peak light value emitted during the pyrosequencing process of ITS region r when the nucleotide d_n was introduced on dispensation n.

The forward primer P_{r_f} and reverse primer P_{r_r} are used in PCR prior to pyrosequencing. The sequencing primer P_{r_s} is the primer used for synthesis during pyrosequencing. The relationship between the dispensation sequence d and the three primers allows comparisons to correctly assume that pyroprints with the same dispensation sequences lead to valid comparisons.

2.6   Matching

As described earlier in Chapter 1, an important part of MST is determining the similarity between the microbes being tracked. In order to achieve this goal, there must be a way to compare the similarity between two pyroprints. The values represented in the pyroprints come from pyrosequencing processes. The absolute values of light intensity, such as peak heights per dispensation, depend highly on the amount of DNA material being sequenced, and can also depend on, for example, the specific batch of reagents injected. Because of this, direct vector comparison methods, such as Euclidean or Manhattan distance are not appropriate for pyroprint comparison.

Instead, we use Pearson correlation between two pyroprint vectors of the same region for the same dispensation sequence as the measure of pyroprint similarity.

2.6.1 Pearson Product-Moment Correlation Coefficient

The Pearson Correlation coefficient measures the degree and direction of the linear relationship between two variables.

The Pearson Correlation coefficient provides a method for comparing pyroprints and ascertaining a match.

Given two pyroprint light emission vectors, $\bar{p}_a$ and $\bar{p}_b$ where $\bar{p}_a, = (p_{a1}, \dots p_{an})$ and $\bar{p}_b = (p_{b1}, \dots p_{bn})$, the Pearson correlation is computed as follows:

$$pearson(\bar{p}_a, \bar{p}_b) = \frac{\sum[(\bar{p}_a - \mu_{\bar{p}_a})(\bar{p}_b - \mu_{\bar{p}_b})]}{\sigma_{\bar{p}_a} \sigma_{\bar{p}_b}},$$

$$\mu_{\bar{p}} = \frac{\sum \bar{p}}{n}$$

$$\sigma_{\bar{p}} = \sqrt{\frac{\sum(\bar{p} - \mu_{\bar{p}})}{n-1}}$$

where $\mu_{\bar{p}_a}$ and $\mu_{\bar{p}_b}$ are the mean values in vectors $\bar{p}_a$ and $\bar{p}_b$ respectively, and $\sigma_{\bar{p}_a}$ and $\sigma_{\bar{p}_b}$ are the standard deviations.

The output of this comparison is a single correlation value, which measures how similar the vector of peak heights in each pyroprint are to each other.

2.6.2 Threshold

In order to discriminate between different pyroprints, we must formally define what it means for pyroprints to be similar. We must also formally define what it means for pyroprints to be dissimilar. In order to make these definitions, several thresholds must be developed for Pearson correlation to determine similarity and dissimilarity.

The definition of thresholds is integral to our MST method. When comparing pyroprints, there must be a way to determine whether they are similar or different. We choose to use two thresholds, one for similarity and another for dissimilarity.

A statistics student at Cal Poly, Diana Shealey, conducted a study to investigate how to interpret pyroprint comparisons and group them into three categories: definitely similar, definitely dissimilar, and reasonably similar [15]. She used a dataset of bacterial isolates which were pyroprinted several times for each ITS region.

She defined two values α and β. The threshold α describes the minimum Pearson correlation score needed to consider pyroprints as being similar. This is the similarity threshold. The threshold β describes the maximum Pearson correlation score needed to consider pyroprints as being dissimilar. This is the dissimilarity threshold. A comparison score between α and β may mean that the pair of pyroprints are similar, but may also mean that a false positive or false negative has occurred [15].

In her study, Diana grouped together pyroprints in her dataset into two groups, one for the pyroprints that are definitely similar and one for the pyroprints that are definitely dissimilar. A beta distribution was fit to the data. She found that the dissimilar pyroprint similarities is not constant and was unable to obtain a β value [15]. She needed to take another approach to continue. She instead fit the beta distribution to one with high percent likelihood as the Pearson score approached 1.0. She found that pyroprints in specific region for a given isolate fit the same distribution as pyroprints from a different isolate.

Using this information, Diana was able to determine thresholds where the number of false negatives are 1%, 5%, and 10% at 0.9953, 0.9941, and 0.9915 respectively.

Biologists on the team have determined α and β thresholds using the results of Diana's study. They set thresholds for both 23-5 ITS region and 16-23 ITS region to be $\alpha = 0.995$ and $\beta = 0.99$.

2.6.3 Isolate Matching

Isolates may have several pyroprints in each ITS region. For this reason, there is a need to develop a method for comparing the pyroprints associated with each isolate.

The similarity between a pair of isolates is computed as a pairwise aggregation of pyroprints for each region. The similarity between isolates for a given region is the average of the cross product between pyroprints. The similarity between isolates in a particular region is computed as follows:

$$R_i^a = \{p_1^a, \dots 3, p_{s_j}^a\}^{\to I_a}$$

$$R_i^b = \{p_1^b, \dots 3, p_{s_j}^b\}^{\to I_b}$$

$$sim(R_i^a, R_i^b) = average(pearson(p_x^a, p_y^b)) \; x \in [1, s_j] \; y \in [1, s_k]$$

The similarity between two isolates, I_a and I_b is the average of the computed similarities for each ITS region pairs, or 0 if the similarity between any ITS region is lower than β. The similarity between isolate pairs is computed as follows:

$$sim(I_a, I_b) = average(sim(R_i^a, p_i^b))$$

2.6.4 Strain Identification

Bacterial strains describe the grouping of isolates to each other based on a similarity metric. In the pyroprinting MST method, Pearson correlation is used to find similar isolates for grouping. In order to find pyroprints that are similar to one another, Pearson correlation comparisons must be made for every pair of isolates. This solution is inefficient and will not scale when the number of pyroprints grows large. Grouping together bacterial isolates that are in the same strain is another part of our microbial source tracking method. It is possible to determine what isolates should be grouped together, thereby representing a strain, by using pyroprint comparison to facilitate isolate similarity.

By grouping together isolates we can limit the number of comparisons for finding similarity and build strains that are vital to our MST method.

A clustering method must be developed to form strains. Clusters are abstract and refer to entities in computation that describe similar entities. The definition of a strain can fit the definition of a cluster by using Pearson correlation as a similarity metric. Isolates are associated with many pyroprints. Pyroprints represent a DNA fingerprint for each ITS region. Pearson correlation provides a similarity metric. This makes isolates an appropriate entity to use for clustering, as the pyroprints of the same region in one isolate can be compared against the pyroprints in the same region of another isolates by using Pearson correlation.

2.6.5 Quality Control and Pyrosequencing

Pyroprints created from the same isolate of *E. coli* can vary though they should be identical. Several factors cause this deviation. The amount of nucleotides dispensed into each well can differ slightly. The amount of DNA in each well can vary between sequencing runs. Both can cause deviations in the light emission histogram.

Pearson correlation accounts for some deviation during comparisons by using thresholds to discriminate between vectors. However the pyrosequencing machine that measures light from the synthesis reaction is prone to some error that is not accounted for by the use of Pearson Correlation coefficient.

The pyrosequencing equipment is the culprit for these errors, but also provides insight into quality control. The raw intensity values collected from the pyroprinting machine provide clues for finding machine errors. Light peaks that occur twice in a dispensation may portend an error in the pyrosequencing process. An example of double peaks is shown in Figure 2.8. Light peaks that do not follow the curve shape of other dispensations may also represent an error. Figure 2.9 shows an example of this. The light peak has a shoulder-
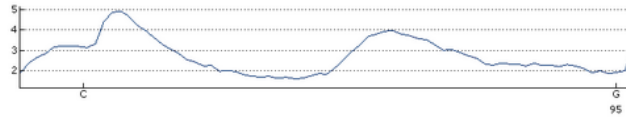
Figure 2.8: A pyroprint histogram with two peaks in a dispensation.

Figure 2.9: A pyroprint histogram with a "shoulder" peak.

like shape at the end. Lower peak values may also be representative of a mistake.

Another error comes from a tendency for the signal to fade over time as data is produced sequentially. If the rate of fade differs between two pyrosequencing runs on the same DNA then a significant difference between the pyroprints can occur.

Due to the nature of the problem, technicians have to manually inspect pyroprints to find machine errors. Once caught, the isolate must be pyroprinted again.

2.7   Cal Poly Library of Pyroprints, Version 1.0

The original version of CPLOP is a database-supported web application that allows for the storage of data from the Cal Poly Biology department's *E. coli* MST project. This database of pyroprints allowed for the storage of information about individual pyroprints as well as information about their origin.

CPLOP stores data about individual pyroprints, as well as their provenance information, broken into the following components: isolate of origin, sample, host, and host species. Each of the components is outlined below. The overall provenance structure is shown in
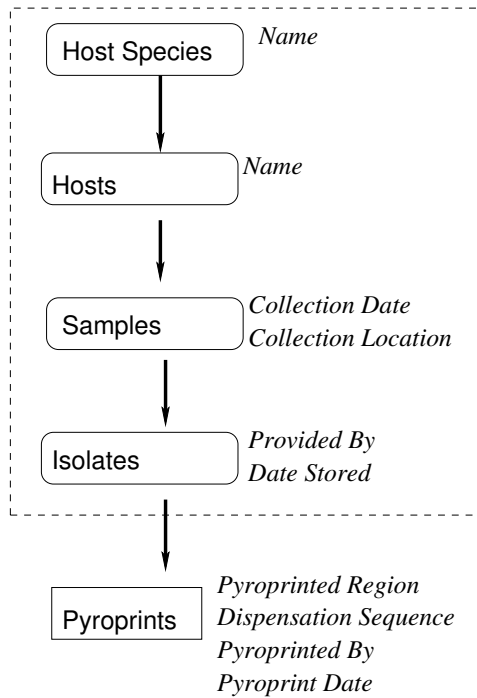
Figure 2.10: Provenance structure for isolate and pyroprint data in CPLOP.

Figure 2.10.

Pyroprints. A pyroprint is a vector of values describing light intensity at each reagent dispensation during the pyrosequencing process. Individual dispensations are represented as tuples consisting of the peak height intensity value and the corresponding dispensed nucleotide reagent. An example peak height graph of a pyroprint is shown in Figure 2.11.
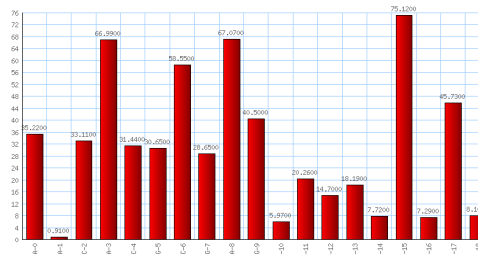

Figure 2.11: A pyroprint as a vector of peak heights.

With each pyroprint, we associate additional meta-data describing its nature: the ITS region that was pyrosequenced, the dispensation sequence used, the primer used in PCR, the date and time of the pyroprinting and the name of the person who conducted the process. As described in Section 2.6.5, different fading values must be taken into account when comparing pyroprints. This is done by associating the compensation slope values with each peak height value.

Primers are short strings of nucleotides used to prime the enzymatic reaction during PCR. Pyrosequencing requires only one primer, called the sequencing primer, to direct the DNA synthesis necessary. However since the database may include pyroprints of different segments of DNA, different primers may be used for PCR and pyrosequencing.

Isolates. Pyroprints are associated with the isolates that they were sequenced from. From each sample, individual isolates of *E. coli* are frozen and maintained by the Biology department. The number of isolates from a single sample can vary widely. Some samples may provide only one or two isolates, while others may have hundreds.

Isolates consist of a list of associated pyroprints, the people who provided the isolate, and the name of the sample it was obtained from.

Samples. Samples are gathered from the environment. Each sample is produced by a specific host or gathered from a specific location. Like the number of isolates in a sample, the number of samples from each host can vary widely.

Information about samples samples stored in the database includes a list of associated isolates, the location that the sample was taken, the id of the host, the date it was collected, and the date it was pyroprinted.

Hosts. A host is a unique individual belonging to one of the host species. CPLOP contains information about multiple hosts from which *E. coli* Isolates were drawn. For example, there may be three different horse hosts: `horse1`, `horse2` and `horse3`. With each host, we associate the following information: a name of the specific host (e.g., 'horse1' or 'Mr.Ed') and the host species.

Host species. Each hosts belong to a host species, which is the species of the animal that the fecal sample was obtained. In CPLOP, the host species has two fundamental features used to distinguish between them: a common name, such as Cat, Dog or Seagull and a Latin name, such as *Felis catus*, *Canis lupus familiaris*, *Larus californicus*.

CPLOP currently contains over 10,500 pyroprints from the DNA of *E. coli* isolates that have been isolated and pyrosequenced by students and researchers at the Cal Poly Biology department. These 10,500 pyroprints make up 4,300 isolates.

Figure 2.12 shows the data model for CPLOP 1.0. The database design shows structures for storing pyroprints and provenance information.

2.7.1 Web Application

CPLOP 1.0 contains a web interface that allows users to upload samples, browse data, and perform matching operations. By placing all of the data in a single database, researchers can compare their data with the data of others.

Users upload their samples via the web interface. CPLOP provides users with a simple way to browse sample, isolate, and pyroprint data. By providing an interface for the pyroprint data, CPLOP makes it easy for users to sort through, organize, and inspect information.

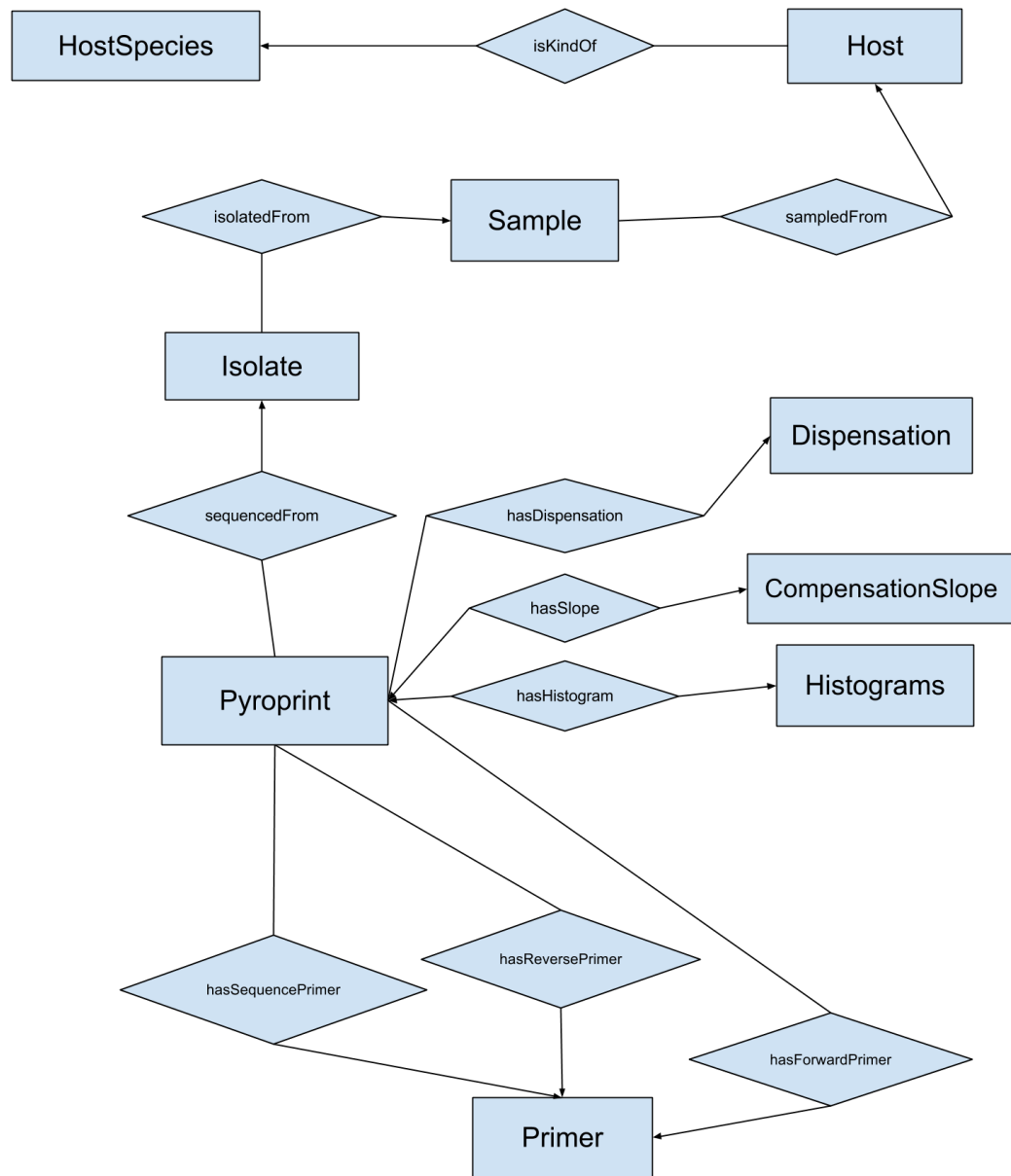As stated earlier, CPLOP extracts the peak height values during each dispensation and

Figure 2.12: CPLOP E-R Diagram

stores this as a vector of numbers in the database. We are able to visualize the peak height histogram as a bar graph for viewing on the web interface. Figure 2.11 illustrates the vector of peak heights.

2.7.2 Use Cases

The following cases describe common operations and uses for CPLOP 1.0. These were implemented by Kevin Webb as part of his Senior Project. While the only case presented for updating and deleting data is the pyroprints table, similar procedures are used to update and delete sample and isolate data. Both are accessible through the web site link bar.

Home Page
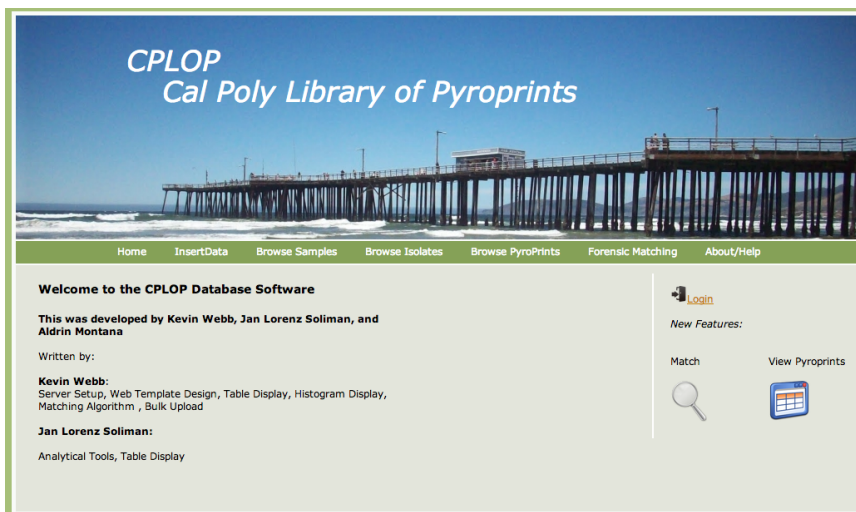
CPLOP 1.0 resides at the URL http://cplop.cosam.calpoly.edu. The home page allows all users to browse sample, isolate, and pyroprint data. Registered users can log in and upload data. Administrator users have the ability manage user accounts and view statistics on the entire database. Administrators have the ability to view pyroprints with missing isolate data and isolates with missing pyroprint data.

Inserting Data

(Step 1) Upload Isolate/FreezerStock Information (.csv)

Name

Your email address

Choose a file to upload
Choose File No file chosen

1. On the insert data page, click the first step **Part 1: Upload Isolate Data**. Choose the freezerstock .csv file to upload.

(Step 2)Upload Pyroprint Identification Information (.csv)

Name

Your email address

Choose a file to upload
Choose File No file chosen

Submit

2. Go to the second step, **Part 2: Upload Pyroprint Identification Information**. Choose the pyroprint master .csv file to upload.

(Step 1) Upload Isolate/FreezerStock Information (.csv)

Name

Your email address

Choose a file to upload
[Choose File] No file chosen

3. Go to the last step, **Part 3: Pyroprint Data (Histogram Information Upload)**. Choose the Pyromark .xml file to upload.

Browsing and Searching Data

Browse Pyroprints

Add to Experiment | Check all | Uncheck all
<< | < | Add | View | Change | Copy | Delete | > | >> | Go to | 1 ▾   Page: 1 of 91 Records: 9011

Search	Select	PyroID (Options)	IsoID	Amplified Region	Dispensation Name	FileName	WellID	Tech	PyroPrintedDate	PcrDate	ForPrimer	Rev
Clear	Sorted By: PyroID (Options) ascending											
🖼📷📋🗑	☐	1	Hu-001	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A1	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
🖼📷📋🗑	☐	2	Hu-002	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A2	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
🖼📷📋🗑	☐	3	Hu-003	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A3	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
🖼📷📋🗑	☐	4	Hu-004	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A4	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
🖼📷📋🗑	☐	5	Hu-005	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A5	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
🖼📷📋🗑	☐	6	Hu-006	23-5	AACACGCGA23(GATC)GAA	082311 23-5 hu001 to hu024	A6	VanderKelen	8/23/2011		23-5 ITS-F	23-5 bio
						082311						

1. On the home page, click the **Browse Pyroprints** link.

2. The page contains a list of uploaded pyroprints. To search, click **Search** button and put the desired word to search for in the corresponding field.

3. The results from the query will be shown on the table. The same can be done for Samples, Host Species, and Isolates.
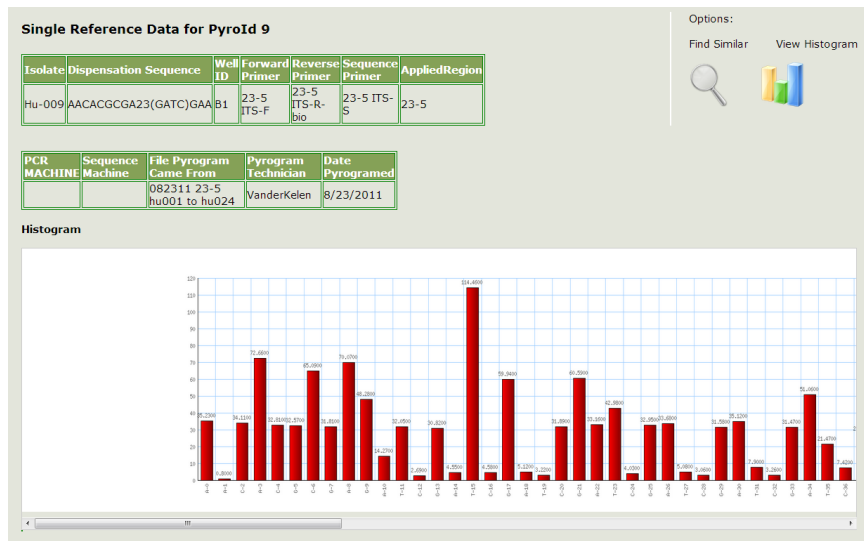
4. Clicking on the pyroprint id will lead to meta data about that entry, including the histogram of peak height values.

Updating Data

1. While browsing pyroprints, data editors and administrators can modify existing pyroprint metadata. Clicking on the **Edit** icon, brings up a form.

2. The metadata can be changed. After modifications are completed, clicking **Submit** will save the changes to the database.

Deleting Data

1. While browsing pyroprints, data editors and administrators can delete existing data. Clicking on the **Delete** icon, brings up a confirmation form.

3. Clicking the **Delete** button will remove the pyroprint and its associated histogram from the database.

Managing Users

1. User accounts can be added, edited, and deleted through the user management system. Clicking on the **Admin Portal** icon on the home page leads to the administrator portal.
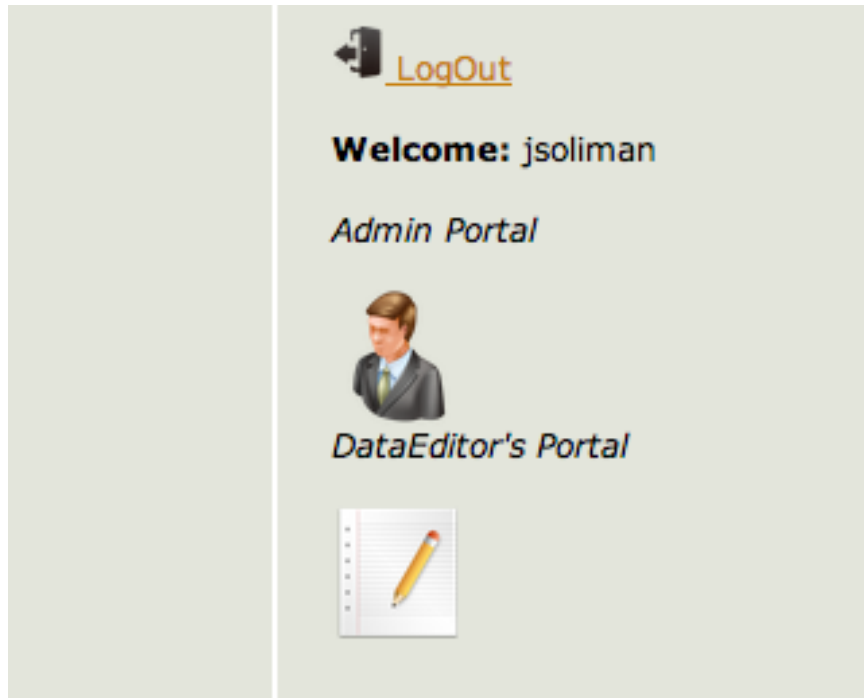
2. Clicking on **Add/View/Delete Users** leads to a page similar to the "Browse Pyro-prints" page described above.
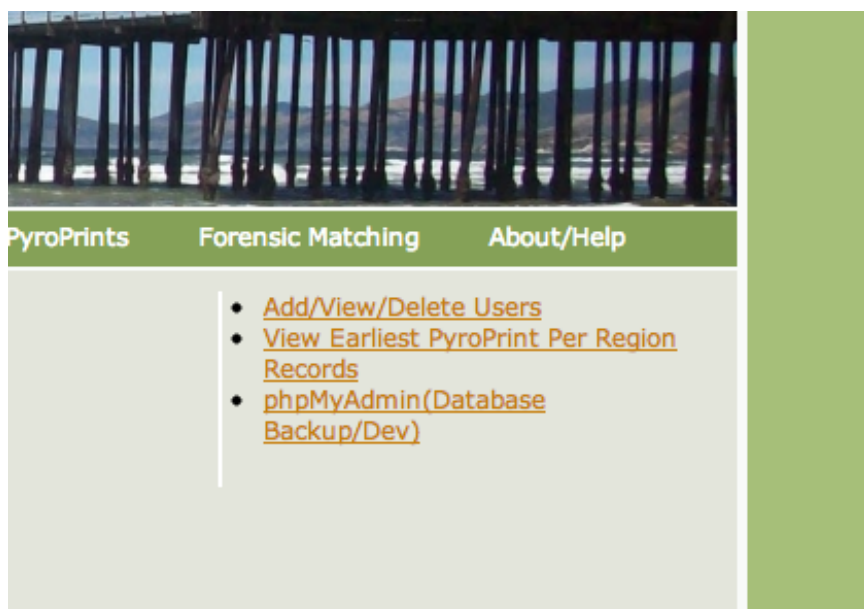
3. Administrators can add, edit, and delete users following similar steps to the ones described in the subsections of 2.7.2.

	User Name	Name	Affiliation	Type	Date
Clear All					
Query					
Edit Delete	jsoliman	Jan Lorenz Soliman	Student	1	0000-00-00 00:00:00
Edit Delete	admin			1	0000-00-00 00:00:00

2.7.3 Limitations

CPLOP 1.0 has functionality for creating, reading, updating, and deleting data. These four operations provide the basis for a persistent storage application. However as stated in Chapter 1, the persistent storage of data only encompasses one part of the MST method. Investigators must be able to run several analyses on pyroprint data.

In order for pyroprinting as an MST method to succeed, the database must fulfill several tasks of interest to MST investigators. MST investigators need to be able to track the changes in strains as the database grows. Support for comparing isolates of unknown origin to isolates with known associated host species to determine the fecal source. The web application used by investigators must meet all the needs of investigators. This includes providing user interfaces for running analyses and ensuring the responsiveness of the application as more data is added.

CPLOP must be updated to provide new features to support MST investigators. CPLOP 1.0 only provided several features relating to the storage of pyroprints while maintaining the provenance structure. However, to perform more complex analyses on the data, which is required by pyroprinting as an MST method, new algorithms must be developed and implemented into CPLOP.

Specifically, functionality for the matching of pyroprints, the clustering of isolates to build strains, and the partial automation and implementation of quality control procedures must be added to support pyroprinting as an MST method.

Chapter 3

Requirements

Based on the previously described MST method, CPLOP must fulfill several require-ments to allow researchers to perform MST investigations. The requirements are split into four sections. The first, Section 3.1.1, describes the usability and user interface aspects of the system. Section 3.1.2 addresses requirements for the database part of CPLOP applica-tion. Section 3.1.3 deals with the building and maintenance of strains in the database and Section 3.1.4 details the matching and comparison methods that must be implemented.

3.1 CPLOP 2.0 Requirements

3.1.1 Application

CPLOP shall have the following:

A1. Functionality for researchers and technicians to create, read, update, and delete data.

A2. A user interface suitable for use by researchers, technicians, and other interested individuals.

A3. User management system that gives the proper privileges to outside users, registered users, data editors, and administrators.

3.1.2 Data

The database of pyroprints shall do the following:

D1. Store information about individual pyroprints and information about their origin, including isolate information.

D2. Provide users with semi-automated way to check for errors and to flag problem pyroprints for future investigation.

3.1.3 Strains

The database shall implement the following regarding clustering and building strains:

S1. A method for organizing isolates into groups based on their pyroprint similarity.

S2. Structures that recognize groups of similar pyroprints as being part of a strain.

S3. Incremental maintenance of strains throughout the life of the database.

S4. The database shall associate individual determined strains with a host species or a group of host species.

3.1.4 Comparisons

The database shall implement the following:

C1. Algorithm for computing the similarity between two isolates or pyroprints of the same ITS region.

C2. The ability to determine, based on the similarity score, if the two pyroprints represent the same ITS region DNA sequences.

C3. Functionality for matching a single or group of pyroprints or isolates against all others in the database and any subset thereof. This algorithm must be efficient and must scale with the size of the database.

C4. Functionality for matching an isolate against a collection of strains.

3.2   Requirements Analysis

The requirements specified in Section 3.1 address the needs of the CPLOP system in order for it to become a useful tool in MST. CPLOP 1.0 provides functionality for fulfilling several requirements related to the managing of data. However it is incomplete in addressing several sections of the requirements including,

3.2.1   Requirements Fulfilled in CPLOP 1.0

CPLOP 1.0 is a persistent storage application that allows for the adding, viewing, updating, and deleting of data. In Section 2.7.2, we discussed several use cases for using the CPLOP 1.0 web interface to add, update, delete, and read data. This fulfills several requirements for CPLOP. First the web interface in CPLOP 1.0 fulfills requirement A2. CPLOP 1.0 was used by MST investigators from December 2012 to April 2013. Next, the create, read, update, and delete operations showcased in Section 2.7.2 fulfill requirement A1, which addresses the issue of adding and manipulating data in the database. Section 2.7.2 also illustrates how an administrator can manage user accounts in CPLOP. This fulfills requirement A3. Requirement D1 in the Data section of the requirements addresses the proper storage of pyroprint data. CPLOP correctly stores the provenance information

36

Requirement	CPLOP 1.0
A1	Fulfilled
A2	Fulfilled
A3	Fulfilled
D1	Fulfilled
D2	Unfulfilled
S1	Unfulfilled
S2	Unfulfilled
S3	Unfulfilled
S4	Unfulfilled
C1	Unfulfilled
C2	Unfulfilled
C3	Unfulfilled
C4	Unfulfilled

Figure 3.1: Not all requirements are fulfilled by CPLOP 1.0

on pyroprints as shown by the data model in Figure 2.12 in Section 2.7.

3.2.2 Unfulfilled Requirements

While CPLOP 1.0 fulfills the requirements for the Application section and one of the Data section requirements, it does not address any of the requirements relating to strains and comparisons. Figure 3.1 describes the requirements that are not fulfilled by CPLOP 1.0.

In order for pyroprinting to succeed as an MST method, the data in CPLOP must be correct. There must be a procedure for performing quality control on the data in the database. In CPLOP 1.0, quality control is performed manually. It is tedious for investigators to manually inspect pyroprints in order to ensure that each pyroprint is valid. Even with manual inspection, erroneous data still manages to enter the database. With no automated procedure for validating incoming data for correctness, CPLOP 1.0 does not fulfill requirement D2.

The data model allows for the storage of all entities in the provenance model for a py-

roprint. However, there is no notion of a strain in CPLOP 1.0. While host species exist in the data model of CPLOP 1.0, it does not have have any link to a strain entity. With no notion of a strain, requirements S1 and S4 are unfulfilled. The notion of a strain must manually be added to CPLOP with an algorithm for building strains. Thus requirement S2 needs to be addressed in CPLOP 2.0. There also exists no functionality in the upload process of CPLOP 1.0 for the incremental maintenance of strains. Requirement S3 addresses the maintenance of the database after strains are built. Functionality addressing S3 is also absent.

CPLOP 1.0 also lacks functionality for comparing pyroprints or isolates together. There is no similarity metric built into the initial version of CPLOP, meaning that requirement C1 is unfulfilled. Thresholds need to be developed to discern similarity. Without thresholds, requirement C2 is not addressed. Requirement C4 describes the need for matching against strains in the database. Since CPLOP 1.0 lacks a notion of strains and matching functionality, C4 is unfulfilled in CPLOP 1.0. Another use case of the database is the comparison of manually-specified subsets of pyroprints or isolates. There is no notion of manually built groupings of pyroprints or isolates meaning requirement C4 is unaddressed in CPLOP 1.0.

Chapter 4

Design and Implementation

CPLOP 1.0 is unable to fulfill the new requirements for CPLOP. We must develop several new algorithms to meet several new use cases for investigators. We must update the database schema to support the implementation of new algorithms for matching and strain maintenance.

Before proceeding, we organize users of CPLOP into four categories: administrators, data uploaders, investigators, and guests. Each user of CPLOP has an associated account. Guests are unregistered users who can view data in CPLOP but cannot make any changes. Investigators can browse data and run matching on datasets and individual isolates, but cannot change any data. Data uploaders can upload data to CPLOP and can manually modify all data in CPLOP except for user accounts. Administrators have the ability to change other user information and can modify all data associated with the database, including user accounts. Requirement A3 describes the need for granting users different roles.

The design of CPLOP 2.0 attempts to extend existing functionality without modifying what is already there. The additions to the data model are new entity sets and relationship sets independent of the ones created for CPLOP 1.0.

CPLOP 2.0 contains several new algorithms, database additions, and new user inter-

faces to meet the new requirements for CPLOP. The additions are outlined below:

1. Algorithm and implementation for matching pyroprints against one another and a UI for running matching and displaying results.

2. Algorithm and implementation for matching isolates against each other and a UI for running matching and displaying results.

3. Functionality for manually grouping together pyroprints and a UI for displaying the groupings.

4. Functionality for manually grouping together isolates and a UI for displaying the groupings.

5. Algorithm for matching pyroprint datasets against each other and all pyroprints in the database. Related UIs for displaying results.

6. Algorithm for matching isolate datasets against each other and all isolates in the database. Related UIs for displaying results.

7. Functionality for performing automated and manual QC procedures. Related UIs for displaying possible erroneous data.

8. Functionality for calculating database statistics and UIs for displaying them.

9. Integration of a clustering algorithm for building strains. UIs for running and configuring clustering as well as browsing clustering results.

10. Algorithm for enhancing isolate matching performance by using the notion of strains.

11. Algorithm for incremental updating which allows new data to be added to strains without having to run the entire clustering algorithm.

```
function PEARSON(firstPyroprint, secondPyroprint, compareLength)
    firstMean ← mean(firstPyroprint)
    secondMean ← mean(secondPyroprint)
    firstStdDev ← standardDeviation(firstPyroprint)
    secondStdDev ← standardDeviation(secondPyroprint)
    sum ← 0
    for i = 0 → compareLength do
        sum+ = ((firstPyroprint[i] − firstMean) * (secondPyroprint[i] −
secondMean)
    end for
    return (sum/(compareLength * firstStdDev * secondStdDev))
end function
```

Figure 4.1: Calculating the Pearson Correlation between two pyroprint vectors.

4.1 Pyroprint Matching

As described earlier in Section 1 and Section 2.6, a similarity metric is required to discriminate between isolates. Isolates are associated with several pyroprints of differing ITS regions. When comparing isolates, the pyroprints from each region must be compared against each other. Pearson correlation coefficient is used to compute the similarity between individual pyroprints.

CPLOP 1.0 lacks the functionality to match pyroprints. Before the development of CPLOP 2.0, investigators calculated the Pearson scores between pyroprints using Microsoft Excel. This is tedious for investigators, because peak heights from a specific pyroprint must be manually copied to an Excel spreadsheet before the calculation can occur.

The calculation of the Pearson correlation is important to studies involving pyroprint data. For example, a previous study looked at the reproducibility of comparisons when truncating the pyroprint data. The study found that discrimination between pyroprints of the 23-5S region is more reproducible with a peak height vector of length 93 instead of the full-length (104). This investigation was done manually using spreadsheets. By allowing investigators to tweak the parameters of matching calculations, such as the size of the input vectors, CPLOP can be used as a tool to facilitate similar studies in the future.

Q1. SELECT pHeight FROM Histograms WHERE Histograms.pyroID = ⟨firstPyroprint.pyroID⟩ ORDER BY position;

Q2. SELECT pHeight FROM Histograms WHERE Histograms.pyroID = ⟨secondPyroprint.pyroID⟩ ORDER BY position;

Q3. SELECT pyroID, pHeight FROM Histograms LEFT JOIN Pyroprints ON Histograms.pyroID = Pyroprints.pyroID WHERE Histograms.pyroID = firstPyroprint.pyroID AND appliedRegion = region;

Figure 4.2: SQL Queries used to obtain peak height histograms for each pyroprint.

To meet the above use cases, CPLOP 1.0 needs to be updated to allow for automated matching between pyroprints. Requirements C1, C2, C3, and C4 describe the need for matching in CPLOP 2.0.

We have developed and implemented an algorithm for matching pyroprints together. The algorithm for matching is described in Figure 4.3. The SQL query templates for obtaining the histograms are shown in Figure 4.2. Each numbered item in the figure describes a query template. The parameter between the ⟨ and ⟩ varies depending on the algorithm. In this case, ⟨firstPyroprint.pyroID⟩ is the identifier for the first pyroprint to be compared and ⟨secondPyroprint.pyroID⟩ is the identifier for the second pyroprint to be compared. Queries Q1 and Q2 are used in calculating the Pearson correlation between pyroprints. Query Q3 is used to obtain all pyroprints for a specific region.

A Pearson correlation is generated between the input pyroprint and each member of the database with the same region. Two thresholds are used to determine similarity and dissimilarity. If the score is above the α threshold, then the two pyroprints are similar is added to the primary results. If the score is between the α and β thresholds, then the two pyroprints are not similar but also not dissimilar. Thus, the result set is added to the secondary results. They are α and β. The meaning of both thresholds are described in Section 2.6.2.

The implemented matching functionality gives investigators the ability to vary the pa-

```
function MATCHALLPYROPRINTS(matchPyroprint,region, primary, secondary)
    primaryMatches ← ∅
    secondaryMatches ← ∅
    for all setPyroprint in getPyroprintsByRegion(region) do
        score ←PearsonCorrelation(matchPyroprint, setPyroprint)
        if score>primary then
            primarySet ← primarySet ∪ (matchPyroprint, setPyroprint,
                                                    score)
        end if
        if score > secondary AND score < primary then
            secondaryMatches ← secondaryMatches
                                      ∪(matchPyroprint, setPyroprint, score)
        end if
    end for
    return (primaryMatches, secondaryMatches)
end function
```

Figure 4.3: Matching a single pyroprint against the database.

rameters to the matching algorithm to meet different needs. This includes the α threshold, β threshold, and length of comparison. For example, when investigating possible QC procedures, the first nine dispensations of a pyroprint were matched against the first nine dispensations of all the pyroprints in the database to see if low correlation scores between a subset of dispensations could find erroneous data.

The implementation of the matching algorithm also provides investigators with the ability to inspect a specific pyroprint. If an investigator suspects that a machine error may have occurred with a specific run, then the pyroprint can be matched with other pyroprints for the same isolate to see if there is a low correlation. A significantly low correlation means that a possible error occurred.

To improve the performance of calculating Pearson correlations, the standard deviation and mean are saved in the database. Each pyroprint is stored with an associated mean and standard deviation value that is calculated at upload time. The extensions to the data model are shown in Figure 4.4.

We implemented this extension as a single table called PyroStats. The PyroStats table
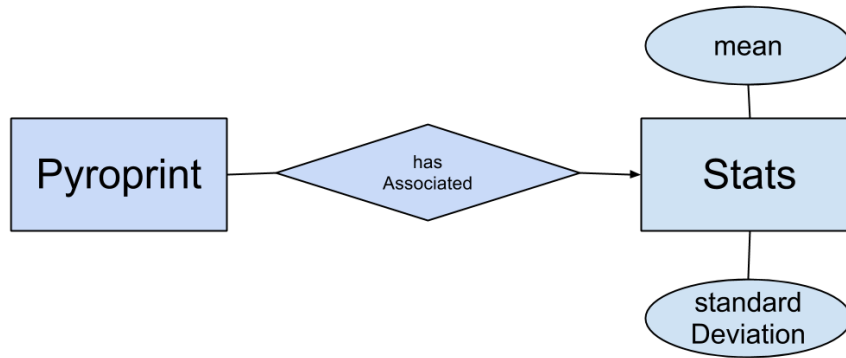
Figure 4.4: Pyroprint statistics extension

structure consists of columns for the pyroprint id, pyroprint mean, and pyroprint standard deviations. By adding PyroStats as a separate table, we were able to add functionality to CPLOP without having to change previous functionality.

Several changes must be made to the user interface to allow for matching functionality. Investigators must be able to select pyroprints to be matched against each other. This requires changes to the UI to accommodate the selecting pyroprints for comparison. Matching also requires a UI change to display the result of matching two pyroprints together.

We have developed a page for displaying the results of the matching functionality. We have also created a link on the "Browse Pyroprints" page that allows investigators to match a single pyroprint against the entire database. Figure 4.5 shows a sample result of pyroprint matching.

4.2   Isolate Matching

While matching single pyroprints together has uses in quality control and understanding the nature of the histograms, this functionality does not provide investigators with the ability to determine the source of an unknown isolate. The unknown isolate must be matched against all isolates in the database. CPLOP 1.0 also lacked this functionality. In order to

Matching Results

Matching **Cats** against **Swine 16-23**

Primary Filter: 0.99
Secondary Filter: 0.98

Note: Matches against the same pyroprint are omitted

Download as csv here

Primary Results (Greater than 0.99)

The first data set matched against the following hosts:

Isolate ID (Pyroprint ID)	Region	Matched Hosts
Ct-001 (4131)	16-23	Pig/Swine (1),
Ct-006 (4135)	16-23	Pig/Swine (1), Pig (2),
Ct-009 (4137)	16-23	Pig/Swine (1), Pig (5),
Ct-010 (4138)	16-23	Pig/Swine (1), Pig (1),
Ct-011 (4139)	16-23	Pig/Swine (1), Pig (6),
Ct-015 (4142)	16-23	Pig/Swine (1), Pig (3),
Ct-018 (4145)	16-23	Pig/Swine (2), Wild Pig (1), Pig (5),
Ct-019 (4146)	16-23	Pig/Swine (3), Wild Pig (2), Pig (5),
Ct-020 (4147)	16-23	Pig/Swine (1), Pig (4),
Ct-022 (4148)	16-23	Pig/Swine (1), Pig (3),
Ct-024 (4150)	16-23	Pig/Swine (1),
Ct-027 (4151)	16-23	Pig (2),

Secondary Results (Between 0.98 and 0.99)

The first data set matched against the following hosts:

Isolate ID (Pyroprint ID)	Region	Matched Hosts
Ct-001 (4131)	16-23	Pig (1),
Ct-006 (4135)	16-23	Pig (6),
Ct-007 (4136)	16-23	Pig/Swine (1), Pig (1),
Ct-009 (4137)	16-23	Pig (4),

Figure 4.5: Pyroprint matching results

45

```
function COMPAREISOLATES(firstIsolate, secondIsolate, region, length)
    scoreSum ← 0
    count ← 0
    for all pyroprint1 in getPyroprints(firstIsolate, region) do
        for all pyroprint2 in getPyroprints(secondIsolate, region) do
            scoreSum+ = Pearson(pyroprint1, pyroprint2, length)
            count+ = 1
        end for
    end for
    return (scoreSum/count)
end function
```

Figure 4.6: Calculating the Average Pearson Correlation between the pyroprints of two isolates for a given ITS region

complete our MST method, algorithms for matching isolates together must be developed and implemented.

To calculate the similarity score between isolates, a Pearson score must be computed between the pyroprints of the two isolates being compared. For two isolates being compared, a Pearson score is computed between pyroprints of the same ITS region. When an isolate has several pyroprints for an ITS region, the average Pearson score is calculated for each pyroprint being compared. This average Pearson score between pyroprints in a given region is called the *region similarity score*. More information on calculating the similarity between isolates is provided in Section 2.6.3.

Figure 4.6 describes the algorithm for comparing two isolates together. We implemented this algorithm using the same Pearson correlation calculation used in matching Pyroprints.

When comparing an isolate to the entire database, two thresholds are used. If the region similarity score between two isolates is greater than a specified α threshold then it is added to the primary results. If the region similarity score is between the α and β thresholds then it is added to the secondary results. We have developed and implemented both an algorithm for comparing pairs of isolates and an algorithm for comparing a single isolate against all

```
Q4.  SELECT pyroID FROM Pyroprints WHERE isoID = ⟨firstIsolate⟩ AND ap-
     pliedRegion = ⟨region⟩;

Q5.  SELECT isoID FROM Isolates;
```

Figure 4.7: SQL Queries used to obtain isolate data.

isolates in the database. Figure 4.9 describes the algorithm for matching a single isolate against all the isolates in the database.

Figure 4.7 describes the queries for obtaining data from the database for us in isolate matching. Query Q4 is used in the *compareIsolates()* function. Query Q5 is used in the *MatchAllIsolates()* function.

The UI must be able to display results for multiple regions. This UI must also allow for investigators to select isolates and have a results page that displays results in a similar way as the results page for matching pyroprints.

We have developed a page for displaying the results of isolate matching. We have also created a link on the "Browse Isolates" page that allows investigators to match a single isolate against the entire database. Figure 4.8 shows a sample results page for isolate matching.

4.3   Datasets

A common need for investigators is to match a subset of pyroprints or isolates against another subset or the entire database. This is done by manually grouping pyroprints together in CPLOP 2.0. Students in Cal Poly's Biology department often look at specific host species or environmental samples. For example, a student looking at only bulls from a specific ranch would have no way to group together pyroprints within the constraints of CPLOP 1.0. Also, the initial version does not provide functionality for grouping pyroprints together in a way specified by a user. CPLOP 2.0 introduces the notion of a *Dataset*.
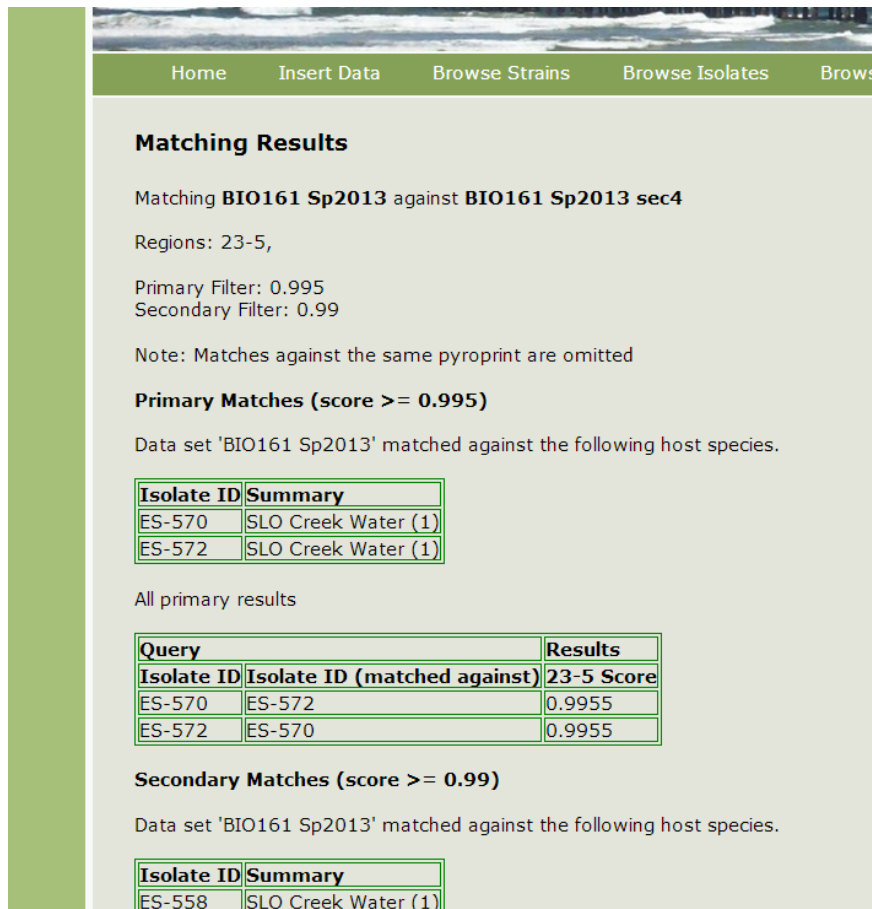
Figure 4.8: Isolate matching results

```
function MATCHALLISOLATES(matchIsolate, primary, secondary)
    primaryMatches ← ∅
    secondaryMatches ← ∅
    for all setIsolate in getAllIsolates() do
        regionScores ← ∅
        for all region in getRegions() do
            regionScores  ←  regionScores∪(region,  compareIso-
lates(matchIsolate, setIsolate, region))
        end for
        if allScoresGreater(regionScores, primary) then
            primarySet ← primarySet ∪ (matchIsolate, setIsolate,
                                        score)
        end if
        if score > secondary AND score < primary then
            secondaryMatches ← secondaryMatches
                                    ∪(matchIsolate, setIsolate, score)
        end if
    end for
    return (primaryMatches, secondaryMatches)
end function
```

Figure 4.9: Matching a single isolate against the database.

A dataset is a group of isolates or pyroprints that are manually grouped together by a data editor or administrator. Since pyroprints and isolates are two separate entities in our data model, we have developed two kinds of datasets.

Data sets should be independent of their encompassing data data. For example if a pyroprint is removed from a dataset, only the link between the dataset and pyroprint should be removed. Also, if a dataset is deleted, then any pyroprint or isolate data associated with the dataset should still exist in the database.

4.3.1 Pyroprint Datasets

A pyroprint dataset is a group of pyroprints that are manually grouped together by a user. The purpose of pyroprint datasets is to allow researchers to group pyroprints of note together for an experiment. Pyroprint data sets can be compared against one another or the
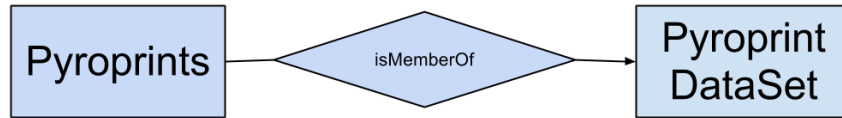
Figure 4.10: Pyroprint Dataset extension
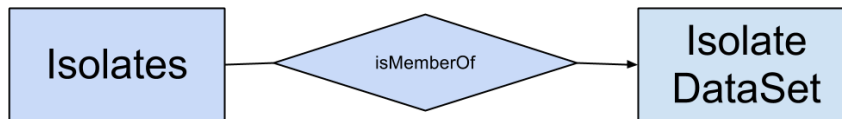
Figure 4.11: Isolate Dataset extension

entire database. Requirement C3 describes the need for comparisons in CPLOP 2.0. To build datasets into CPLOP, several changes must be made to the data model. A pyroprint dataset is essentially a nonexclusive grouping of pyroprints. Several tables must be added to associate pyroprint datasets with pyroprints. First a table representing pyroprint datasets must be created. The pyroprint dataset table holds metadata about the investigator, date of creation, and its name. Another table must be created to link together the pyroprint dataset table and each pyroprint associated with it. Figure 4.10 shows additions to the data model.

To implement this we created two tables. The first table, called PyroprintDataSet, consists of columns for an unique identifier, name, date of creation, investigator, and description. The second table, called PyroprintDataSetLink, consists of columns for a pyroprint ID and a pyroprint data set ID. The PyroprintDataSetLink table links pyroprints into datasets.

4.3.2 Isolate Datasets

Like pyroprint datasets, isolate datasets are groups of isolates that are manually organized by a user. The purpose of isolate datasets is to allow researchers to group isolates of note together for a study. Isolate data sets can be compared against one another or the

entire database.

In order to build isolate datasets into CPLOP, several changes must be made to the data model. These changes are similar to those necessary for building pyroprint datasets into CPLOP. First, a table representing isolate datasets must be added. The dataset table holds metadata about the investigator, date of creation, and its name. Another table must be added to link together the isolate dataset table and the isolates associated with it. Figure 4.11 shows additions to the data model.

To implement this we created two tables. The first table, called IsolateDataSet, consists of columns for an unique identifier, name, date of creation, investigator, and description. The second table, called IsolateDataSetLink, consists of columns for a isolate ID and a isolate data set ID. The IsolateDataSetLink table links isolate into datasets.

4.3.3   Dataset UI

Investigators using CPLOP must be able to view and modify the created datasets. Two pages for creating datasets must be added, one for pyroprint datasets and another for isolate datasets. The creation of datasets must be integrated with the way pyroprints and isolates are browsed in CPLOP 1.0. After the creation of a dataset, users must be able to add new pyroprints or isolates to the dataset, delete pyroprints or isolates from the existing set, and delete the dataset as a whole. Figure 4.12 shows the interface for working with pyroprint datasets. Figure 4.13 shows the interface for working with isolate datasets.

4.4   Dataset Matching

The purpose behind creating datasets is to allow investigators to match pyroprints and isolates. Isolate datasets can be compared against each other and the isolates in the entire

Displaying information for data set 11.

ExperimentID	Name	Date	Investigator	Description	Flag
11	Swine 16-23	2005-07-12 00:00:00	J. VanderKelen		0

Member Pyroprints

pyroID	appliedRegion	isoID
275	16-23	Sw-029
276	16-23	Sw-030
277	16-23	Sw-031
278	16-23	Sw-032
3876	16-23	Sw-032
4167	16-23	Sw-007
4168	16-23	Sw-018
4169	16-23	Sw-021
4170	16-23	Sw-028
4171	16-23	Sw-029
4172	16-23	Sw-030
4173	16-23	Sw-031
4174	16-23	Sw-042
4175	16-23	Sw-047

Figure 4.12: Pyroprint Dataset Interface

Displaying information for data set 3.

id	Name	Date	Investigator	Description
3	ES 251 - 258	2012-10-29 00:00:00	Jan Lorenz	Environmental Samples 251 - 258

Member Isolates

isoID	commonName	hostID
ES-251	SLO Creek Water	Higuera Bridge
ES-252	SLO Creek Water	Higuera Bridge
ES-253	SLO Creek Water	Higuera Bridge
ES-254	SLO Creek Water	Higuera Bridge
ES-255	SLO Creek Water	Higuera Bridge
ES-256	SLO Creek Water	Higuera Bridge
ES-257	SLO Creek Water	Higuera Bridge
ES-258	SLO Creek Water	Higuera Bridge

Figure 4.13: Isolate Dataset Interface

database. Pyroprint datasets can be compared against each other and pyroprints in the entire database. However isolate datasets may not be compared against pyroprint datasets.

4.4.1   Pyroprint Dataset Matching

Investigators often need to look at pyroprints from several isolates at a time. This often involves matching pyroprints against all other pyroprints in the database. It is tedious to go through each pyroprint, run matching against the entire database, and then save the results. Investigators want to match a single pyroprint or group of pyroprints against only a subset of the database. Requirement C3 describes the need for matching subsets of pyroprints together.

For these reasons, the single pyroprint matching algorithm described in Section 4.1 needs to be extended to handle several pyroprints as an input. The pseudocode in Figure 4.14 describes the matching algorithm applied to datasets.

The query used to pull pyroprints for a data set is shown in Figure 4.15. Query Q6 is used in the data set matching algorithm described in Figure 4.14.

A Pearson coefficient is calculated between every pair of pyroprints in the two input sets. Like the individual matching algorithm, if the score is above the α, then the result set is added to the primary results set. If the score is between the α and β thresholds, then the result set is added to the secondary results set.

Since the output data of single-pyroprint matching and dataset-matching have the same structure, the UI for single matching can be extended for matching datasets. The outputted results allow investigators to easily find the host species of *E. coli* pyroprints that matched between the two input sets.

```
function MATCHPYROPRINTDATASETS(firstSet, secondSet, primary, sec-
ondary, length)
    firstPyroprints ← getPyroprintsByDataSet(firstSet)
    secondPyroprints ← getPyroprintsByDataSet(secondSet)
    primaryMatches ← ∅
    secondaryMatches ← ∅
    for all firstPyroprint in firstPyroprints do
        for all secondPyroprint in secondPyroprints do
            score ←    PearsonCorrelation(firstPyroprint,    secondPyroprint,
length)
            if score>primary then
                primarySet ← primarySet∪
                                  (matchPyroprint, setPyroprint, score)
            end if
            if score > secondary AND score < primary then
                secondaryMatches ← secondaryMatches
                                        ∪ (matchPyroprint, setPyroprint, score)
            end if
        end for
    end for
    return (primaryMatches, secondaryMatches)
end function
```

Figure 4.14: Matching datasets against each other

```
Q6. SELECT pyroID FROM PyroprintDataSetLink WHERE dataSetId =
    ⟨firstSet.id⟩;
```

Figure 4.15: SQL Query used to pyroprints from datasets.

```
function MATCHISOLATEDATASETS(firstSet, secondSet, primary, secondary,
length)
    firstIsolates ← getIsolatesByDataSet(firstSet)
    secondIsolates ← getIsolatesByDataSet(secondSet)
    primaryMatches ← ∅
    secondaryMatches ← ∅
    for all firstIsolate in firstIsolates do
        for all secondIsolate in secondIsolates do
            regionScores ← ∅
            for all region in getRegions() do
                regionScores    ←    regionScores∪(region,    compareIso-
lates(firstIsolate,                                    secondIsolate,
region))
            end for
            if allScoresGreater(regionScores, primary) then
                primarySet ← primarySet ∪ (matchIsolate, setIsolate,
                                        score)
            end if
            if score > secondary AND score < primary then
                secondaryMatches ← secondaryMatches
                                    ∪(matchIsolate, setIsolate, score)
            end if
        end for
    end for
    return (primaryMatches, secondaryMatches)
end function
```

Figure 4.16: Matching datasets against each other

4.4.2   Isolate Dataset Matching

Investigators often need to look at several isolates of unknown origin at a time. It is
tedious to go through each isolate, run matching against the entire database, and then save
the results. Investigators want to match a single isolate or group of isolate against only a
subset of the database. Requirement C3 describes the need for matching subsets of isolates
together.

For these reasons, the single isolate matching algorithm described in Section 4.2 needs
to be extended to handle several isolate as an input. The pseudocode in Figure 4.16 de-
scribes the matching algorithm applied to datasets.

Q7. SELECT isoID FROM IsolateDataSetLink WHERE dataSetId = ⟨firstSet.id⟩;

Figure 4.17: SQL Query used to isolates from datasets.

The query used to pull pyroprints for a data set is shown in Figure 4.17. Query Q7 is used in the data set matching algorithm described in Figure 4.16.

A region similarity score for each region is calculated between every pair of isolates in the two input sets. Like the individual matching algorithm, if the score for each region is above the α threshold, then the result set is added to the primary results set. If the score for each region is between the α and β thresholds, then the result set is added to the secondary results set.

Since the results of single-isolate matching and dataset-matching have the same structure, the UI for single matching can be extended for matching datasets. The outputted results allow investigators to easily find the host species of *E. coli* isolates that matched between the two input sets.

4.5   Quality Control

As described in Section 2.6.5, ensuring the correctness of data in the database is a difficult problem for administrators and data uploaders. During the deployment of CPLOP 1.0, administrators must manually open Pyrorun files using Pyromark software and manually look through the raw histograms to find cases of double or shoulder peaks. This is done to prevent bad data from being inserted into the database. This adds time to the data upload process and slows down the building of the pyroprint library. The need for semi-automated quality control is described in requirement D2.

Having erroneous data in the database also causes problems for studies. In most cases, pyroprints from the same isolate have a Pearson coefficient greater than $\beta = 0.99$. However

some pyroprints from the same isolate have Pearson coefficients with smaller numbers due to machine error.

Quality control measures in CPLOP is split into two parts. The first part describes quality control processes that analyze pyroprints to be uploaded and existing database pyroprints, and then flag those that may be erroneous. The second part details the features built into CPLOP that support manual quality control.

4.5.1 Automated QC Support

Regions from the same isolate may be pyroprinted several times. One way to determine if a pyroprint is erroneous is to compare it against pyroprints of the same isolate. On occasion this is done by pyroprinting the region several times and comparing all the replicated pyroprints against each other. If there are two pyroprints for a particular region with a low Pearson correlation score, we withhold judgment of either pyroprint until more data is added to the database. For cases of three or more pyroprints, the set of pyroprints for a region is clustered using hierarchical clustering, which will be described in more detail in Section 4.7. If multiple clusters form, then the clusters that do not have the highest membership are flagged as possible erroneous.

We have implemented this algorithm into CPLOP. However, at this time, most isolates contain at most two pyroprints for a given region. When a low correlation occurs between a pair of pyroprints of the same isolate and region we are unsure which pyroprint is correct and cease to make a judgment until there is more data.

4.5.2 Manual QC Support

CPLOP 1.0 lacks manual QC functionality. If an administrator or data uploader learned that pyroprints associated with a Pyrorun file or sample was erroneous, then all the user
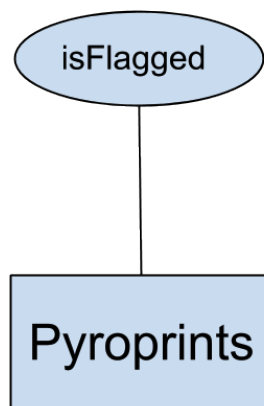
Figure 4.18: Pyroprint data model extension

could do is delete the pyroprints from the database.

Erroneous pyroprint data may be useful in developing future QC tools. CPLOP 2.0 should allow data uploaders and administrators to flag pyroprints as erroneous or possibly erroneous. This allows bad pyroprint data to be excluded from analyses such as matching or clustering, while allowing future developers to use the erroneous pyroprints as test data for future QC techniques.

In order to build QC into the CPLOP model, a new attribute must be added to the pyroprint table. The attribute is a flag which, when set, signifies that the pyroprint may be erroneous. We built this attribute into CPLOP as a new table with columns mapping pyroprint ids to a boolean flag. Figure 4.18 illustrates the pyroprint flag extension.

Several UI changes need to be made to add this functionality. The UI for browsing pyroprints must be extended to include a link for marking a pyroprint as erroneous. A new UI must be created to allow administrators and data uploaders to view and search erroneous pyroprints.

We have implemented features for flagging pyroprints that a data editor or administrator thinks is suspect. We have implemented this UI as a link in the "Administrator Portal" page. Figure 4.19 shows the interface for browsing flagged pyroprints. The "Browse Pyroprints"
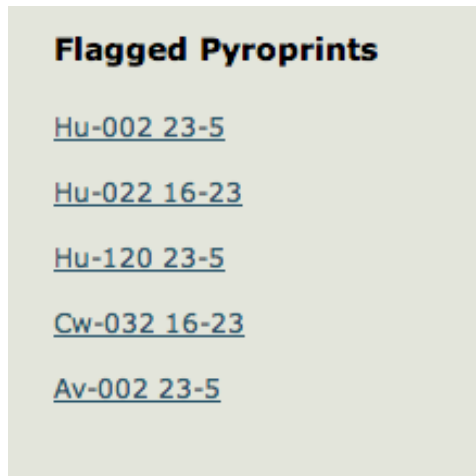
Figure 4.19: Flagged Pyroprints Screen

page has also been extended to include a link for flagging and unflagging a pyroprint as erroneous.

4.6 Application Modifications

CPLOP 1.0 needs to be extended to meet several new use cases. It must also address performance concerns as the database scales.

Investigators often want to know the current status of the database in order to include this information in their studies. Often investigators need to know database statistics such as the number of isolates, the number of pyroprints, and the number of isolates for each host species. However, CPLOP 1.0 has minimal functionality for doing this.

A new page for viewing database statistics must be created. This page includes the following:

1. The total number of isolates in the database.

2. The total number of pyroprints in the database.

```
Q8.  SELECT COUNT(*) FROM Isolates;

Q9.  SELECT COUNT(*) FROM Pyroprints;

Q10.  SELECT DISTINCT(Isolates.commonName), COUNT(Isolates.isoID) FROM
      Isolates LEFT JOIN HostSpecies ON Isolates.commonName = Host-
      Species.commonName GROUP BY HostSpecies.commonName;

Q11.  SELECT    DISTINCT(Isolates.commonName),    COUNT(Pyroprints.isoID)
      FROM Isolates LEFT JOIN Pyroprints ON Isolates.isoID = Pyroprints.isoID
      GROUP BY Isolates.commonName;

Q12.  SELECT DISTINCT(pyroID) FROM Pyroprints LEFT JOIN Isolates ON Iso-
      lates.pyroID = Pyroprints.pyroID WHERE Isolates.isoID IS NULL;

Q13.  SELECT DISTINCT(pyroID) FROM Pyroprints LEFT JOIN Histograms ON Py-
      roprints.pyroID = Histograms.pyroID WHERE Histograms.pyroID IS NULL;
```

Figure 4.20: SQL Queries used to calculate database statistics.

3. The number of isolates associated with each host species.

4. The number of pyroprints associated with each isolate.

5. Isolates with missing pyroprint data.

6. Pyroprints with missing isolate data.

7. Pyroprints with missing histogram data.

The queries for pulling statistics from the database are shown in Figure 4.20. Queries Q8 and Q9 retrieve the total number of isolates and pyroprints respectively. Query Q10 calculates the number of isolates for each host species. Query Q11 counts the number of pyroprints for each isolate. Query Q12 retrieves the pyroprints with missing isolate data. Query Q13 retrieves the pyroprints with missing histogram data.

The UI modifications meet the needs of requirement A2. It suits the needs of administrators, data uploaders, and guest users.

We have implemented this feature as a "Database Statistics" page, accessible from the Data Editor portal. Figure 4.21 shows the statistics user interface.

CPLOP Statistics

Isolate Statistics

# of Isolates: 4385

Isolates missing region 16: 273

Isolates missing region 23: 59

# of Hosts: 34

# of Isolates By Host:

36yr female: 1

4yr female: 1

56yr female: 1

Cat: 65

Chicken: 44

Cliff Sparrow: 28

cougar: 1
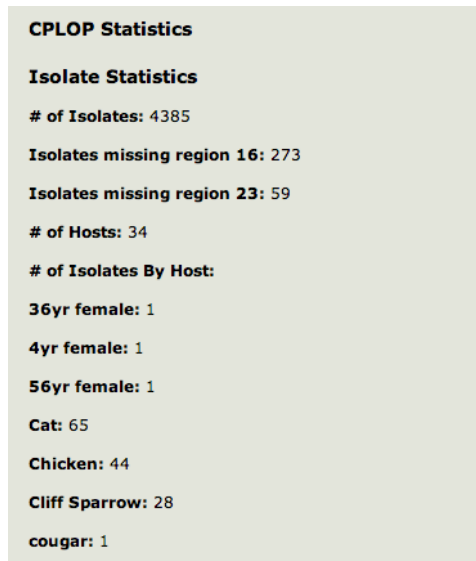
Figure 4.21: CPLOP Statistics Screen

4.6.1 Caching

The calculation of database statistics is computationally expensive. Recalculating the statistics each time a user browses the page is slow and forces the user to wait. In CPLOP 1.0, there existed functionality for viewing isolates with missing pyroprints, pyroprints with missing isolates, and pyroprints with missing histograms. However, as the database grew, the generation of this list became more extensive. For this reason, the queries for calculating the above statistics must be cached inside the database for easy retrieval. This is done using MySQL's query caching functionality. We cache queries for viewing pyroprints with missing isolates and pyroprints with missing histograms.

Query caching will not work with the "Missing Regions" page. The "Missing Regions" page displays isolates that are missing pyroprints in a specific ITS region and the region(s) that are missing. Since the region missing from each isolate must be retrieved from the database, query caching cannot be performed without caching a query for each isolate. Thus, regions missing for each isolate are stored in the database and recalculated each time data is added.

Q14. SELECT isoID, region FROM MissingRegion;

Figure 4.22: SQL Queries used to obtain missing pyroprint data.

Figure 4.23: Extension for saving missing regions

Figure 4.23 shows the necessary extension to the data model to allow the caching of missing regions. We implement this extension as a new table. The table has two columns. One of the columns holds the isolate id. The other holds a region. The table maps isolate ids to missing regions. Figure 4.22 shows the query for retrieving isolates with missing pyroprints.

4.7   Clustering

As stated in Section 2.6.4, an integral part of the Pyroprinting MST method is the building of strains. An important process in an MST investigation is determining the source of an unknown sample. In this case, each isolate of unknown origin in the sample must be compared to a database of known isolates to determine its source. As the database scales up, the number of comparisons that must be made for each unknown isolate increases with the size of the database. Other questions in *E. coli* MST investigations involve the tracking of the bacteria across several environments or hosts.

To facilitate the process of determining an isolate's source and to support investigations across different environments or hosts, CPLOP must be extended to support the notion of a strain.

As described in Section 2.6.4, biological strains describe a group of related isolates. All

members of a biological strain must be similar in some way. In order to build the notion of strains into the database, an algorithm must be developed to group together similar isolates. Clustering is an algorithm for organizing objects together based on similarity. Clustering is a fitting algorithm to use for building strains. The groups formed from clustering algorithms are called *clusters*. The clusters formed after running clustering on isolate data result in groups of isolates with a similar characteristic. The similarity metric used by clustering to build strains is the Pearson correlation score between pyroprints of the same ITS region.

Clustering must be integrated into the database in order to facilitate the building of strains. There are many clustering algorithms that exist in the literature.

Partitional clustering algorithms work by developing cluster centroids and iteratively assign and re-assign database points based on the calculated centroid [5]. A *centroid* is defined as a single object selected to represent a cluster. *K-Means clustering* works by iteratively adding the item with the minimum distance to a centroid and then recalculating the centroid [5]. Partitional algorithms like K-Means work well when there is a general estimate of the number of partitions. Thus, K-Means Clustering, would not be reliable because new and diverse data is constantly added to the database.

Hierarchical clustering, another group of clustering algorithms, may be more suitable due to no *apriori* knowledge of the number of partitions. Hierarchical clustering works in two ways [5]. The algorithm either groups together pairs of items with a maximum similarity value or separates items with the minimum similarity value. In the case of grouping together items with maximum similarity, the following steps describe the algorithm

1. Assign each item to a cluster. Compute the similarity score between all clusters.

2. Find the pair of items with the highest similarity score and merge them into the same cluster. Compute the similarity score between the new set of clusters

3. Repeat until all items are members of a single cluster.
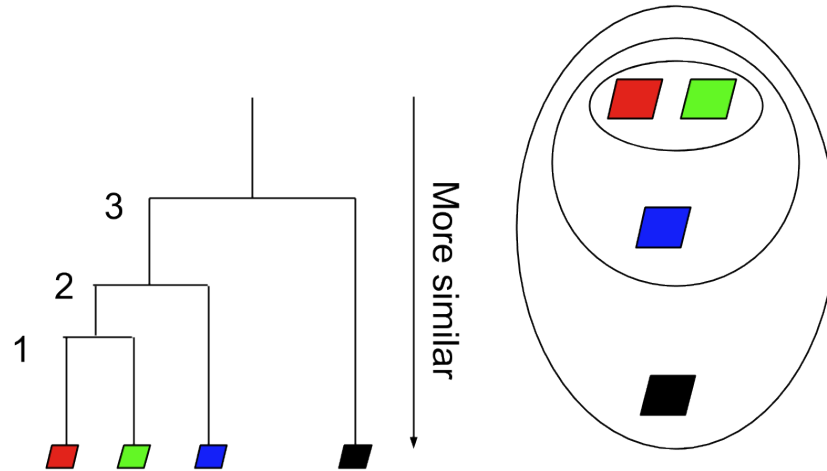
Figure 4.24: Example Agglomerative Clustering. The numbers correspond to the order that items are clustered together.

Figure 4.24 illustrates the hierarchical clustering algorithm. The hierarchy created is called a *dendrogram*. Items that are clustered together earlier in the algorithm have a higher similarity score than those clustered later. Hierarchical clustering very nicely solves the elusive problem of expected number of strains since it relates data points to other data points to build clusters.

The standard hierarchical clustering method used in biology is *agglomerative clustering*. In agglomerative clustering, each item starts in its own cluster and pairs of clusters are iteratively merged together. The two parameters in agglomerative clustering are the metric for comparing items and the threshold for discerning similarity.

Unfortunately agglomerative clustering is inadequate for use in CPLOP. If agglomerative clustering is used, whenever new data is added to CPLOP, the entire clustering computation has to be repeated. A new algorithm must be developed and integrated into CPLOP.

4.7.1 OHClust!

In CPLOP we use *OHClust!* (Ontology-based Hierarchical Clustering), an algorithm developed by Aldrin Montana, a student in the computer science department at Cal Poly for use in CPLOP. *OHClust!* is a modification of agglomerative clustering that is better adapted for analysis of data for studies conducted by students and researchers.

OHClust! uses metadata to cluster isolates. Metadata is defined as data that contains provenance information. Metadata provides a context for which isolates are to be clustered. The metadata order is specified as an *ontology*. An ontology is a hierarchical structure that specifies what order metadata is clustered first. *OHClust!* allows isolates to be clustered based on a specified ontology. The ontology specifies what metadata to use to determine the order of the clustering. The motivation behind using an ontology is that data with metadata that this the same may have a stronger relationship than data with different metadata. For example, pyroprints from the same host species, such as cattle, may be more related than pyroprints from different host species, such as cattle and birds. Data points can be clustered based on associated metadata instead of by using a similarity metric [8]. Figure 4.25 illustrates a sample CPLOP ontology.

OHClust! recursively clusters at each level of the ontology. Although it it hierarchical, it is specified as a list of metadata fields. For example, an ontology of "'Sample', 'Sample Date', 'Host', 'Host Species'" causes isolates to be clustered with isolates in the same sample, then isolates with the same sample data, then isolates with the same host, and finally isolates with the same host species. This contrasts agglomerative clustering, which organizes data points on similarity alone [8].

Since *OHClust!* is an extension of agglomerative hierarchical clustering, it requires a method for determining the similarity between isolates and a threshold for discerning similar from non-similar isolates.
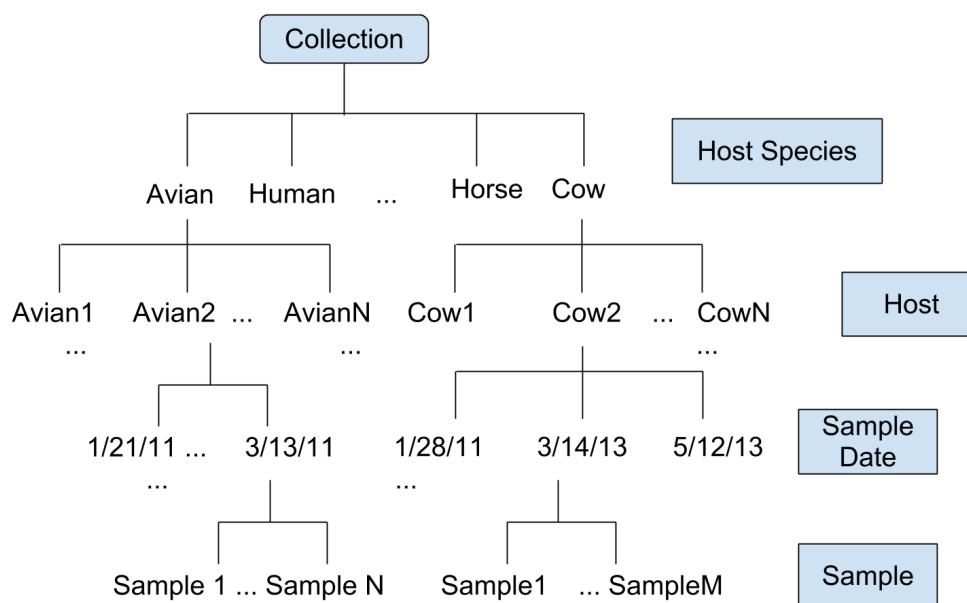
Figure 4.25: CPLOP Ontology

OHClust! uses two thresholds for categorizing isolates into three groups. These two thresholds are the same specified in Section 2.6.2. The two thresholds are $\alpha = 0.995$ and $\beta = 0.99$. Similar isolates are those with $sim(I_a, I_b) \geq \alpha$. Isolates with $sim(I_a, I_b) < \beta$ are dissimilar. Isolates with $\alpha > sim(I_a, I_b) \geq \alpha$ are defined as "squishy."

The method used for determining similarity compares the Pearson score between pyroprints of each ITS region. However since each isolate may have several pyroprints for a given region, the average score of all pairwise Pearson scores for a given region is used.

The *OHClust!* algorithm starts by clustering the isolates in the leaf nodes in the ontology. Once all of the leaf nodes are clustered, the clusters are propagated up to their parent. At this point the inter-cluster distances are recomputed. Hierarchical clustering is then performed again on the recomputed clusters. This is then done recursively for each parent node until it gets to the top of the ontology. The psuedo-code for the *OHClust* algorithm is provided in Figures 4.26, 4.27, and 4.28. Figure 4.26 illustrates how the ontology is traversed. Figure 4.27 describes how the clustering is implemented and Figure 4.28 shows

```
function CLUSTERDATASTRUCTURE(N)
    C ← 0
    if N = null then
        return C
    end if
    if |children(N)|! = 0 then
        for n ∈ children(N) do
            C ← C ∪ clusterDataStructure(n_i)
        end for
        C ← C ∪ performHierarchical(C)
    else
        C ← performHierarchical(C_n)
    end if
    return C
end function
```

Figure 4.26: Traversing the data structure for Clustering

how the inter-cluster distance is recomputed at each level.

OHClust uses the ontological structure to partition a given dataset and to cluster these partitions in a directed order. Using this ontology provides flexibility in user-specified ontologies. Investigators can modify ontologies for specific studies. For example, a researchers looking at how strains change over time can specify an ontology using the date of collection. Other advantages to using an ontology are scalability in incremental updates and access to more information in being able to query sub-relationships in the analyzed data.

For more information on *OHClust*, refer to [8].

4.7.2   Data Model Changes

The data model must be extended to include strains. Since *OHClust* uses two thresholds, α and β, to categorize isolate comparisons as similar, dissimilar, and "squishy," clusters computed at each threshold must be stored in the database.

Similar isolates are built into the database as strains and stored with a threshold value

```
function PERFORMHIERARCHICAL(C)
    C' ← C
    (c_a, c_b) ← 0
    S[j,k] ← recomputeDistances(C')
    for c_j, c_k ∈ C' do
        S[j,k] ← sim(c_j, c_k)
    end for
    while |C'| > 1 and sim(c_a, c_b) ≥ α do
        (c_a, c_b) ← argmin(S[j,k])
        if sim(c_a, c_b) ≥ α then
            C' ← C' ∪ combineClusters(c_a, c_b)
            S[j,k] ← recomputeDistances(C')
        end if
    end while
    for c_i ∈ C do
        if |c_i = 1 then
            W ← W ∪ c_i
        end if
    end for
    return C
end function
```

Figure 4.27: Hierarchical Clustering

```
function RECOMPUTEDISTANCES(C)
    for c_j, c_k ∈ C do
        if sim(c_i, c_k) ≥ α then
            S[j,k] ← 1
        else if sim(c_i, c_k) < β then
            S[j,k] ← 0
        end if
    end for
    return S
end function
```

Figure 4.28: Recompute Distances after clustering leaf nodes
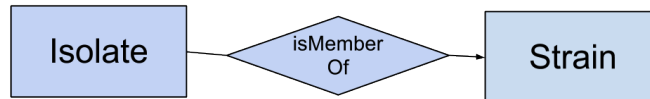
Figure 4.29: Strain data model extension

Figure 4.30: Browse Strains Interface

of 0.995. Clusters that occur in the "squishy" region are stored as strains with a threshold value of 0.99. The threshold value is saved in order to differentiate between strains formed with a threshold value greater than α or strains formed in the "squishy" region between α and β.

CPLOP needs to be extended to include the running of *OHClust* to build strains. Figure 4.29 shows the extension needed to add strains to the data model. First, an entity must be added to associate isolates with strains. Requirements S1, S2, and S4 describe the need for building strains. Each strain may have many isolates.

The UI must allow all users to navigate strains and the isolates within. Figure 4.30 shows the UI for browsing strains.

```
function MatchAllIsolates(matchIsolate, primary, secondary)
    primaryMatches ← ∅
    secondaryMatches ← ∅
    for all cluster in getAllClusters() do
        representativeIsolate ←getRepresentative(cluster)
        score ← matchIsolates(matchIsolate, representativeIsolate)
        if score > primary then
            clusterIsolates ← getIsolatesByCluster(cluster)
            matchSet ← MatchDataSets(matchIsolate,
                              clusterIsolates, primary, secondary)
            primarySet ← primarySet ∪ matchSet
        end if
        if score > secondary AND score < primary then
            clusterIsolates ← getIsolatesByCluster(cluster)
            matchSet ← MatchDataSets(matchIsolate,
                              clusterIsolates, primary, secondary)
            secondaryMatches ← secondarySet ∪ matchSet
        end if
    end for
    return (primaryMatches, secondaryMatches)
end function
```

Figure 4.31: Matching an isolate against the database.

We have built the necessary entity and relationship sets for storing strains in the database. In addition, we have built UIs for browsing strains at both threshold levels.

4.8 Matching Against Strains

One feature of strains is that each member of a strain is similar to other members of the strain. This could be advantageous for matching against the database. When an unknown isolate is encountered, it must be compared with each other isolate in the database. As the database size grows, the number of comparisons increases as well. This will lead to a slowdown in performance.

Limiting the set of comparisons could provide a significant performance boost. Using strains can decrease the number of comparisons made. A new matching algorithm is

70
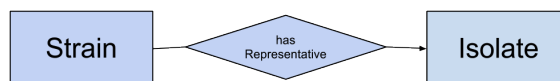
Figure 4.32: Strain representative data model extension

proposed based on limiting the comparison set.

When an isolate must be matched against every isolate in the database, instead of calculating the similarity for all items in the database, it is compared against a representative from each strain first.

If the isolate does not match the strain, then it is not compared to the isolates in that strain. By comparing against strain representatives first, the comparison set can be culled significantly. The algorithm is described in Figure 4.31.

4.8.1   Strain Representatives

In order to implement the above described matching algorithm, structures must be added to the database to allow for the storage of strain representatives. Figure 4.32 describes the changes to the data model. Each strain chooses an isolate as a representative based on the average of the similarity scores to each other item in the strain. The isolate with the maximum average similarity score to each member isolate is chosen as the representative. This isolate is called the representative of the strain. We implemented this algorithm for choosing strain representatives. Once strains are created or new data is inserted, representatives are calculated for newly created or modified strains.

The data model must be extended to associate isolates with strains. Each strain should be associated with an isolate. We achieve this by creating a link table, called Strain-IsolateLink, between strains and their member isolates. The entity relationship is shown in Figure 4.32.

4.9 Incremental Updating

Running *OHClust* is expensive and could take anywhere from several minutes to several hours based on the size of the database. The database would not be able to handle clustering when many users are uploading large amounts of data at once. Requirement S2 describes the need for incrementally adding to the strain structure.

One advantage of *OHClust* is that it can be incrementally updated when new data is introduced. This allows strains in the database to be updated without running the entire *OHClust* algorithm again. When a new isolate is added to the database, it is placed in its correct place in the ontology. It is then compared with each isolate in each cluster at the ontology node. If the comparison yields a similarity then it is added to that cluster. The cluster is propagated up the ontology and added to the appropriate strain.

4.10 Summary of Data Model Changes

Seven of the new features built into CPLOP required changes to the dataset. Figure 4.34 illustrates the changes to the CPLOP 1.0 data model. Figure 4.33 describes the new tables and their attributes. We added tables and relations to add functionality for Pearson correlation calculation, isolate datasets, pyroprint dataset, quality control flagging, database statistics, strain maintenance, and strain representatives.

Strain
id
clusterName
date
diameter
mean
threshold
summary
representativeId

IsolateScores
isoID1
isoID2
region
score
length

PyroprintDataset
Id
Name
Date
Investigator
Description

IsolateDataset
Id
Name
Date
Investigator
Description

Pyrostats
pyroID
mean
stdDev
length

PyroprintDatasetLink
datasetID
pyroID

IsolateDatasetLink
datasetID
isoID

StrainIsolateLink
strainID
isoID

FlaggedPyroprints
id
pyroID
reason
dateFlagged

Figure 4.33: **CPLOP 2.0 New Tables**
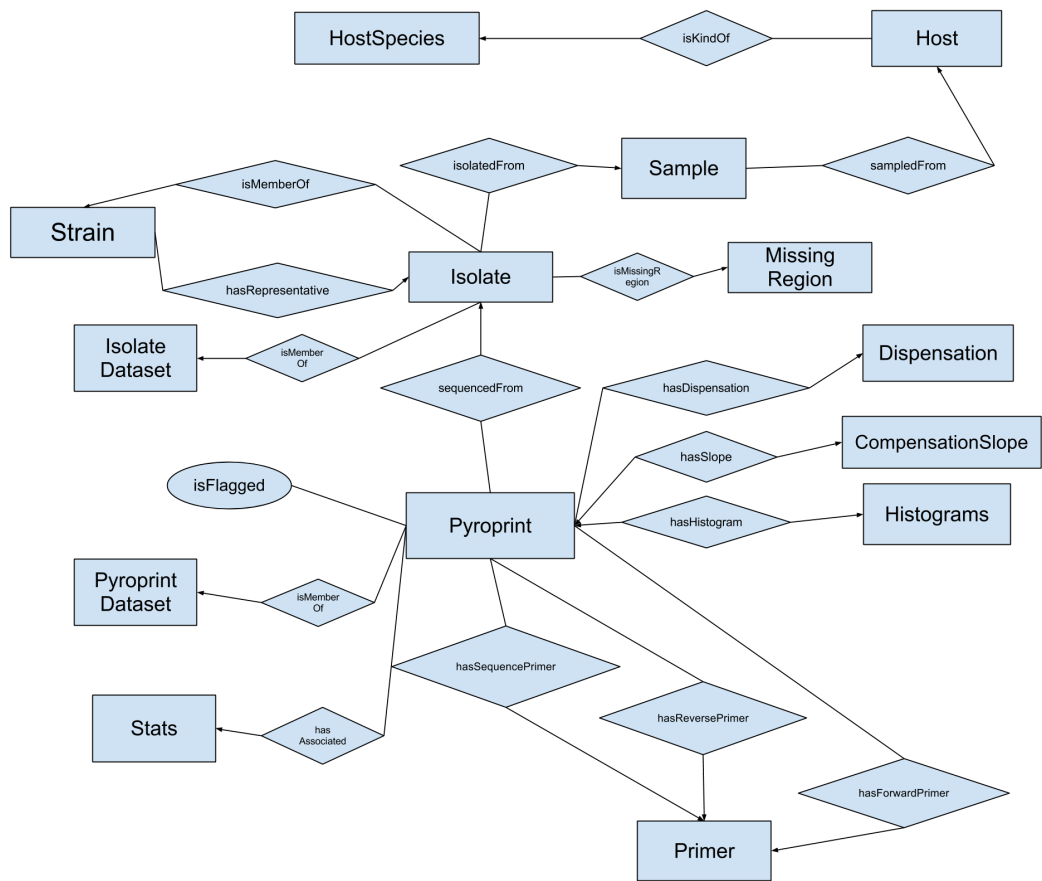
Figure 4.34: **CPLOP 2.0 E-R Diagram**

Chapter 5

Evaluation

CPLOP 2.0 is evaluated on two different levels. First, each feature added in Chapter 4 is traced to the requirements described in Chapter 3.1 to show that each is complete. Second, we describe several case studies showing CPLOP 2.0's use as an application for MST.

5.1 Requirement Tracing

Each feature developed and described in Chapter 4 is laid out in this section and given an identifier. We map each identifier to a feature in Chapter 3.1. As shown in Figure 3.1 in Chapter 3.1, Section 3.2.2, requirements A1 through A3 and D1 were fulfilled by CPLOP 1.0. They are ignored in this section.

5.1.1 Features

The following features describe all of the implemented items in Chapter 4 for the purpose of mapping them to requirements.

F1. Data editors and administrators can match two pyroprints or isolates together by

using the match UI. The user must create a dataset with the pyroprints or isolates of interest. The user then provides the pyroprints or isolates of interest, a α, and a secondary threshold. The user can match two datasets against each other or a dataset against the entire database. The user is given the results of the matching output under two categories, one for those above the α, and one for those between the α and β thresholds. Described in Sections 4.1 and 4.4. Fulfills requirement C1, C2, and C3.

F2. From the browse pyroprints UI, an administrator or data editor can select pyroprints and add them to a dataset. From the browse isolates UI, an administrator or data editor can select isolates and add them to a dataset. Described in Section 4.3.1 and Section 4.3.2.

F3. Inserted data is run through several quality control procedures before being inserted. Possible erroneous data is flagged. The administrator or data editor can view flagged data and choose to delete it or unflag it. Described in Sections 4.5, 4.5.1, and 4.5.2. Fulfills requirement D2.

F4. Administrators and data editors can view the statistics page from the data editors portal that shows various statistics on the database. The page loads promptly with little delay. Described in Sections 4.6 and 4.6.1.

F5. Administrators can run *OHClust* on the database and can specify parameters relating to the ontology and threshold values used. The results are saved as strains and are viewable in the browse strains page. Described in Sections 4.7, 4.7.1, and 4.7.2. Fulfills requirements S1, S2, and S4.

F6. After running data set matching for pyroprints or isolates, the results displays results relating to the strains that were matched. In addition, administrators and data editors can choose to use strain matching to increase the performance of the matching algo-

Features	F1	F2	F3	F4	F5	F6	F7
Requirements							
D2			fulfilled				
S1					fulfilled		
S2					fulfilled		
S3							fulfilled
S4					fulfilled		
C1	fulfilled						
C2	fulfilled						
C3	fulfilled					partially fulfilled	
C4						fulfilled	

Figure 5.1: Requirements fulfilled by CPLOP 2.0

rithm for isolates. Described in Sections 4.8 and 4.8.1. Partially fulfills requirement C3 and fulfills requirement C4.

F7. When data is inserted, each new isolate is compared for similarity with each existing strain. It is added to the strain with the highest similarity score. Described in Section 4.9. Fulfills requirement S3.

The table in Figure 5.1 summarizes the traceability matrix for CPLOP 2.0 requirements.

5.2 Case Studies

CPLOP 1.0 was hardly used as an MST application. CPLOP 2.0 is currently used in several MST investigations at Cal Poly. This is significant because MST investigations were the initial motivation for developing the application. The first is an investigation is by a Biology Master's student looking at the relationship between the *E. coli* in bulls and squirrels, the second investigation is tracking environmental samples in the surrounding areas to host species.

5.2.1 Bull Run Study

A graduate student in the Biology department at Cal Poly, Joshua Dillard, investigated whether *E. coli* are transferred between cohabitating animals [2]. For this study he chose to look at bulls. He also investigated whether a dominant strain exists for bovines.

Josh used the Cal Poly Bull Test to set up his experiment. The Cal Poly Bull Test is an annual event held by the Animal Science department where bulls are brought to Cal Poly from different ranches in California in order to be tested and vaccinated. The bulls are housed in five separate pens. They arrive at Cal Poly in May and are sold in September.

One of the hypotheses in Josh's study is that the environment factors into the strains of *E. coli* found. The first expectation is for bulls from the same farm to have similar *E. coli* strains than those from different farms [2]. As bulls from different ranches spend time together, bulls share *E. coli* with each other. The second expectation is that the *E. coli* strains for all of the cattle will be more similar after the bulls have been cohabitating for four months.

Another hypothesis in the study is that rodents on campus such as squirrels may be exposed to *E. coli* from bulls. The expectation is to see significant similarity between bovine strains of *E. coli* and isolates from squirrels in the region.

In order to test this, fecal samples were obtained during two separate Bull Test events, one from May 2011 to September 2011 and the other from May 2012 to September 2012. 175 bulls were sampled in May 28th, 2011. 144 were sampled on September 3rd, 2011, the end of the Bull Test 2011. 180 bulls were sampled on May 12th 2012. 139 bulls were sampled on September 1st, 2012, the end of the Bull Test 2012 [2].

The fecal samples were isolated and cultured. The cultures were pyroprinted for the 16S-23S and 23S-5S regions. The pyroprints were uploaded to the database using the web UI. Josh then performed an analysis on the data using an implementation of *OHClust!*,

developed by Aldrin Montana.

The study found that the distribution of strains increased after bulls were housed together for 4 months. He also found that the number of strains decreased per home ranch after four months of cohabitation. Both of these results support his initial hypotheses on whether bulls share *E. coli* when housed together [2]. In addition, he found that isolates from squirrels on campus often clustered with isolates from the bulls in the study supporting his hypothesis that squirrels are exposed to *E. coli* from bulls on campus.

This study was facilitated by the CPLOP database and analysis tools. A total of 638 bulls were sampled during two Bull Test events. From these samples, 1276 isolates were pyrosequenced. Given the significant size of his samples, it would have been difficult to manually cluster his data together.

In this study, data was stored on the database. The web application allowed investigators easily browse and organize data. Since the data is easily accessible on the web, it allows developers to create applications for analysis. Aldrin's *OHClust!* application is able to communicate with the database to have up to date data each time it is run. The application connected to the MySQL database stored on the CPLOP server in order to access data. The use of a clustering application shows the usefulness of building a database to store pyroprint data. Researchers can create their own applications to run analyses on pyroprint data depending on the study.

5.2.2 San Luis Obispo Creek

The Environmental Biotechnology Institute (EBI) at Cal Poly is using CPLOP for several MST investigations. Each investigation hopes to find the source of fecal contamination in several areas in order to help determine techniques for halting current contamination and preventing future ones.

The first MST investigation done by the EBI using pyroprinting is San Luis Obispo Creek (SLO Creek). The EBI is looking at fecal contamination in the creek that runs through the city of San Luis Obispo. In 2004, the Central Coast Regional Water Quality Control Board found high traces of *E. coli* in SLO Creek. They traced the source to a leaking sewer pipe nearby. The pipe was repaired, however the contamination continued.

The goal of this study is to figure out what species of animal is the source of the *E. coli*. Investigators walked through SLO creek and found that pigeons, bats, and humans are the most likely culprits. When sampling an isolating *E. coli* from each source, the investigators excluded bats as a possible cause due to low traces *E. coli* in their feces.

Several samples were collected from the creek. The sample cultures were pyroprinted and stored in CPLOP. They were then clustered with the database using *OHClust!*. Preliminary results show that the unknown isolates matched pigeons more than an other host species.

There are more isolates from SLO creek being pyroprinted as part of two courses, a general microbiology course (MCRO 224) and a molecular biology course (BIO 161). Students from MCRO 224 obtain environmental water samples from SLO Creek. They pass these samples to students in BIO 161 lab. The students in BIO 161 isolate cultures from samples, pyroprint them, and store them into the database. The BIO 161 students then use the matching functionality described by Feature F6 in Section 5.1.1 as a part of their lab assignment. They match isolates from their lab section against the entire database to determine possible sources. They will also match isolates from their lab section against isolates from all other lab sections to see if any possible similarities between the isolates given to the classes. The BIO 161 students report the results of their matching back to the students in MCRO 224.

A new initiative of the study is to increase the number of isolates in the database from wild animals and other underrepresented host species in the database. The plans for this

collection and expansion are for Summer 2013.

CPLOP supports the SLO Creek study in several ways. The storage functionality allows investigators to easily add data to the database. CPLOP allowed for the execution of *OHClust* like in the Bull Test study. This is also the first study to leverage matching functionality from CPLOP 2.0 in order to investigate the source of unknown isolates. Without the matching functionality built into CPLOP 2.0, students would not be able to investigate unknown isolates in their BIO 161 lab. The matching functionality will be used in future iterations of MCRO 224.

Chapter 6

Future Work

While CPLOP 2.0 has analytical tools for clustering and comparing data, it lacks functionality in several areas. Several UIs were built while developing new features, however, major aesthetic changes have not been done to the application as a whole.

6.1   Peak Width and Peak Area

As stated in Section 2.5, peak widths and peak areas are stored in the database. CPLOP 2.0 does not have any interface for browsing or taking advantage of this data. The current CPLOP UI assumes that pyroprint histograms consist of peak height values. There is currently no way for users to view peak width and peak area values.

The analyses presented in this thesis use peak height to represent pyroprint histograms. However, the analyses presented may be modified to run using peak area or peak width. The next version of CPLOP must allow users to specify which of the three histogram representations to use in matching, clustering, or quality control. Further studies could compare the results of our implemented analyses using different histogram representations.

6.2 Visualization

Scientific visualization of scientific data is a branch of computer science devoted to visually presenting scientific data to enable researchers to understand their data [7]. CPLOP 2.0 contains a wealth of biological data, but lacks ways to visualize it.

Sample, isolate, and pyroprint data is presented to users as a list of rows in a table. The matching and clustering analyses also display results in a tabular form. This does not allow researchers to easily obtain insight on the data.

There is no way to view relationships between clusters in CPLOP 2.0. Cluster visualization, a subset of scientific visualization, is a technique for visualizing clusters by showing the relationships between clusters and between the data points within. Building cluster visualization into CPLOP 2.0 could help researchers learn about the the properties of strains. Work was started on visualizing clusters based on size in CPE 572, a graduate level computer graphics course at Cal Poly. It must be continued in order to be useful for illustrating the relationships between clusters.

6.3 Quality Control

More work can be done on detecting erroneous data in the database. By storing erroneous pyroprints and tagging them as such, machine learning techniques may be applied to the data in order to predict whether incoming data is erroneous. In addition, data stored in the Pyrorun file, such as the raw light emission values may be useful for determining whether there are double or shoulder peaks in a pyroprint.

Chapter 7

Conclusion

New features were successfully built into CPLOP 1.0. The updated version, CPLOP 2.0, provides analyses to support investigators in MST pursuits. CPLOP 2.0 features functionality for organizing isolates through the use of datasets. Investigators may now use CPLOP to match isolates of unknown origin against each other. They may now build and save strains into the database by using the *OHClust!* algorithm. Lastly, strains are maintained in the database by incrementally adding new data to existing strains. This prevents users from having to recluster the database each time data is added.

CPLOP 2.0 fulfills all of the requirements for a database application supporting pyroprinting MST. It is now is a fully featured database application with tools for addressing MST, such as the creation of strains and the matching of isolate of unknown origin to known isolates. The creation of a database to house isolate data was integral in investigating the movements of *E. coli* strains in the Bull Test study. It is currently used by several groups researching *E. coli* contamination in SLO Creek and Pennington Creek. Finally it has become a tool used by several of courses in the Biology department at Cal Poly.

Bibliography

[1] R. Conrad. Lettuce from California's Salinas Valley recalled over E. coli concerns. http://www.wmctv.com/story/5515482/lettuce-from-californias-salinas-valley-recalled-over-e-coli-concerns?clienttype=printable.

[2] J. Dillard, J. Kent, D. Britton, T. Branck, P. Frey, Amandaand McCeesh, J. VanderKelen, C. Kitts, and M. Black. E. coli strain demographics and transmission in cattle. 2013.

[3] IEH epidemiology and genetic fingerprinting. http://www.iehinc.com/index.php/epidemiology-genetic-fingerprinting/.

[4] M. A. Jensen, J. A. Webster, and N. Straus. Rapid identification of bacteria on the basis of polymerase chain reaction-amplified ribosomal dna spacer polymorphisms. *Applied and Environmental Microbiology*, 59:945–952, apr 1993.

[5] B. Liu. *Data Mining: Exploring Hyperlinks, Contents, and Usage Data, Springer*. Springer, 2008.

[6] M. C. Maiden, J. A. Bygraves, E. Feil, G. Morelli, J. E. Russell, R. Urwin, Q. Zhang, J. Zhou, K. Zurth, D. A. Caugant, I. M. Feavers, M. Achtman, and B. G. Spratt. Multilocus sequence typing: A portable approach to the identification of clones within populations of pathogenic microorganisms. *Proceedings of the National Academy of Sciences*, 95:3140–3145, mar 1998.

[7] B. H. McCormick. Visualization in scientific computing. *SIGBIO Newsl.*, 10(1):15–21, Mar. 1988.

[8] A. Montana. Algorithms for library-based microbial source tracking. Master's thesis, California Polytechnic State University - San Luis Obispo, 2013.

[9] A. Montana, A. Dekhtyar, E. Neal, M. Black, and C. Kitts. Chronology-sensitive hierarchical clustering of pyrosequenced dna samples of e. coli: A case study. pages 155–159, 2011.

[10] Qiagen. Frequently asked questions - pyrosequencing. http://www.pyrosequencing.com/DynPage.aspx?id=7481.

[11] M. Ronaghi. Pyrosequencing Sheds Light on DNA Sequencing. *Genome Research*, 11(1):3–11, Jan. 2001.

[12] M. Ronaghi, S. Karamohamed, B. Pettersson, M. Uhlen, and P. Nyren. Real-time dna sequencing using detection of pyrophosphate release. *Comparative and Functional Genomics*, 3:51–56, feb 2002.

[13] D. Schoch. Five years after deadly e. coli outbreak, salinas valley farmers struggle to rebound. *San Jose Mercury News*, November 2011.

[14] T. M. Scott, J. B. Rose, T. M. Jenkins, S. R. Farrah, and J. Lukasik. Microbial source tracking: Current methodology and future directions. *Appl. Environ. Microbiol.*, 68:5796 – 5803, dec 2002.

[15] D. Shealey. Exploration of pyroprinting for environmental forensics. Master's thesis, California Polytechnic State University - San Luis Obispo, 2013.

[16] J. M. Simpson, J. W. Santo Domingo, and D. J. Reasoner. Microbial source tracking: state of the science. *Environ Sci Technol*, 36(24):5279–88, 2002.

[17] J. L. Soliman, A. Dekhtyar, J. Vanderkellen, A. Montana, M. Black, E. Neal, K. Webb, C. Kitts, and A. Goodman. Microbial source tracking by molecular fingerprinting. pages 617–619, 2012.

[18] D. M. Stoeckel and V. J. Harwood. Performance, design, and analysis in microbial source tracking studies. *Applied and Environmental Microbiology*, 73:2405 – 2415, apr 2007.

[19] J. Wapps. FDA launches salinas valley e coli probe. *CIDRAP*, September 2006.

Appendix A

CPLOP 1.0 Database MySQL Statements

The following describes the MySQL CREATE TABLE statements for CPLOP 1.0.

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";


--
-- Database: `CPLOP`
--



-- --------------------------------------------------------



--
-- Table structure for table `compensation_slope`
--



CREATE TABLE IF NOT EXISTS `compensation_slope` (
```

```
  `pyrogram_num` decimal(50,0) NOT NULL,

  `position` decimal(50,0) NOT NULL,

  `level` decimal(10,0) NOT NULL,

  `drop_off_value` int(11) NOT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `Dispensation`

--



CREATE TABLE IF NOT EXISTS `Dispensation` (

  `dsName` varchar(15) NOT NULL default '',

  `dispSeq` varchar(200) default NULL,

  PRIMARY KEY  (`dsName`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `FirstPyroDate`

--



CREATE TABLE IF NOT EXISTS `FirstPyroDate` (

  `isoID` varchar(50) NOT NULL,

  `region` int(50) NOT NULL,
```

```
  `PDate` varchar(50) NOT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

-- --

```
--
-- Table structure for table `FirstPyroprintDate`
--


CREATE TABLE IF NOT EXISTS `FirstPyroprintDate` (

  `IsoID` varchar(20) NOT NULL,

  `Region` varchar(50) NOT NULL,

  `Date` int(50) NOT NULL,

  PRIMARY KEY  (`IsoID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

-- --

```
--
-- Table structure for table `Histograms`
--


CREATE TABLE IF NOT EXISTS `Histograms` (

  `pyroID` int(11) NOT NULL default '0',

  `position` int(11) NOT NULL default '0',

  `pHeight` decimal(8,4) default NULL,

  `cPeakHeight` decimal(8,4) default NULL,
```

```
    `nucleotide` char(1) default NULL,

    `PeakArea` float NOT NULL,

    `PeakWidth` float NOT NULL,

    `BaselineOffset` float NOT NULL,

    `SignalValue` float NOT NULL,

    PRIMARY KEY  (`pyroID`,`position`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `Host`

--



CREATE TABLE IF NOT EXISTS `Host` (

    `commonName` varchar(50) NOT NULL,

    `hostID` varchar(50) NOT NULL default '',

    PRIMARY KEY  (`commonName`,`hostID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `HostSpecies`

--



CREATE TABLE IF NOT EXISTS `HostSpecies` (
```

```
  `latinName` varchar(50) default NULL,

  `commonName` varchar(50) NOT NULL,

  `letterCode` varchar(10) default NULL,

  PRIMARY KEY  (`commonName`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `Isolates`

--



CREATE TABLE IF NOT EXISTS `Isolates` (

  `isoID` varchar(15) NOT NULL default '',

  `FreezerLocation` varchar(50) default NULL,

  `TANOTES` varchar(300) default NULL,

  `dateStored` varchar(15) default NULL,

  `PyroPrintDate` varchar(50) default NULL,

  `userName` varchar(50) default NULL,

  `sampleID` int(11) default NULL,

  `commonName` varchar(50) NOT NULL,

  `hostID` varchar(50) NOT NULL,

  `tag` varchar(10) default NULL,

  PRIMARY KEY  (`isoID`),

  KEY `commonName` (`commonName`),

  KEY `hostID` (`hostID`,`commonName`),

  KEY `sampleID` (`sampleID`,`commonName`,`hostID`),
```

```sql
  KEY `userName` (`userName`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--
-- Table structure for table `Primer`
--


CREATE TABLE IF NOT EXISTS `Primer` (
  `primerName` varchar(40) NOT NULL default '',
  `sequence` varchar(40) default NULL,
  PRIMARY KEY  (`primerName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--
-- Table structure for table `Pyroprints`
--


CREATE TABLE IF NOT EXISTS `Pyroprints` (
  `pyroID` int(11) NOT NULL default '0',
  `fileName` varchar(150) default NULL,
  `appliedRegion` varchar(20) default NULL,
  `pcrDate` varchar(15) default NULL,
  `wellID` varchar(5) default NULL,
```

```sql
  `pcrMachine` varchar(20) default NULL,

  `sequencerMachine` varchar(20) default NULL,

  `dsName` varchar(205) default NULL,

  `forPrimer` varchar(40) default NULL,

  `revPrimer` varchar(40) default NULL,

  `seqPrimer` varchar(40) default NULL,

  `pyroPrintedTech` varchar(100) default NULL,

  `pyroPrintedDate` varchar(15) default NULL,

  `isoID` varchar(15) default NULL,

  `tag` varchar(10) default NULL,

  PRIMARY KEY  (`pyroID`),

  KEY `dsName` (`dsName`),

  KEY `forPrimer` (`forPrimer`),

  KEY `revPrimer` (`revPrimer`),

  KEY `seqPrimer` (`seqPrimer`),

  KEY `pyroPrintedTech` (`pyroPrintedTech`),

  KEY `isoID` (`isoID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--

-- Triggers `Pyroprints`

--

DROP TRIGGER IF EXISTS `CPLOP`.`TEST1`;

DELIMITER //

CREATE TRIGGER `CPLOP`.`TEST1`

AFTER INSERT ON `CPLOP`.`Pyroprints`

 FOR EACH ROW BEGIN
```

```
END

//

DELIMITER ;


-- ---------------------------------------------------------



--

-- Table structure for table `Samples`

--


CREATE TABLE IF NOT EXISTS `Samples` (

  `sampleID` int(11) NOT NULL default '0',

  `commonName` varchar(50) NOT NULL,

  `hostID` varchar(50) NOT NULL,

  `location` varchar(50) default NULL,

  `dateCollected` varchar(15) default NULL,

  PRIMARY KEY  (`sampleID`,`commonName`,`hostID`),

  KEY `hostID` (`hostID`,`commonName`),

  KEY `commonName` (`commonName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- ---------------------------------------------------------



--

-- Table structure for table `Users`
```

--

```sql
CREATE TABLE IF NOT EXISTS `Users` (

  `userName` varchar(50) NOT NULL default '',

  `passwordCheck` varchar(50) default NULL,

  `type` varchar(50) default NULL,

  PRIMARY KEY  (`userName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------
```

Appendix B

CPLOP 2.0 Database MySQL Statements

The following describes the MySQL CREATE TABLE statements for CPLOP 2.0.

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";


--

-- Database: `CPLOP`

--


-- --------------------------------------------------------


--

-- Table structure for table `compensation_slope`

--
```

```
CREATE TABLE IF NOT EXISTS `compensation_slope` (

  `pyrogram_num` decimal(50,0) NOT NULL,

  `position` decimal(50,0) NOT NULL,

  `level` decimal(10,0) NOT NULL,

  `drop_off_value` int(11) NOT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `Dispensation`

--



CREATE TABLE IF NOT EXISTS `Dispensation` (

  `dsName` varchar(15) NOT NULL default '',

  `dispSeq` varchar(200) default NULL,

  PRIMARY KEY  (`dsName`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------



--

-- Table structure for table `PyroprintDatasetLink`

--



CREATE TABLE IF NOT EXISTS `PyroprintDatasetLink` (

  `PyroprintDatasetLinkID` int(75) NOT NULL auto_increment,
```

```sql
  'ExperimentID' int(75) NOT NULL,

  'PyroprintID' int(75) NOT NULL,

  PRIMARY KEY  ('PyroprintDatasetLinkID'),

  KEY 'ExperimentIDIndex' ('ExperimentID'),

  KEY 'PyroprintIDIndex' ('PyroprintID')

) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1452 ;


-- --------------------------------------------------------


--

-- Table structure for table 'PyroprintDataset'

--


CREATE TABLE IF NOT EXISTS 'PyroprintDataset' (

  'ExperimentID' int(11) NOT NULL auto_increment,

  'Name' varchar(100) character set ascii NOT NULL,

  'Date' datetime NOT NULL,

  'Investigator' varchar(200) NOT NULL,

  'Description' varchar(300) NOT NULL,

  'Flag' tinyint(1) NOT NULL,

  PRIMARY KEY  ('ExperimentID'),

  UNIQUE KEY 'UniqueName' ('Name')

) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=184 ;


-- --------------------------------------------------------


--
```

```sql
-- Table structure for table 'FirstPyroDate'
--


CREATE TABLE IF NOT EXISTS 'FirstPyroDate' (
  'isoID' varchar(50) NOT NULL,
  'region' int(50) NOT NULL,
  'PDate' varchar(50) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------


--
-- Table structure for table 'FirstPyroprintDate'
--


CREATE TABLE IF NOT EXISTS 'FirstPyroprintDate' (
  'IsoID' varchar(20) NOT NULL,
  'Region' varchar(50) NOT NULL,
  'Date' int(50) NOT NULL,
  PRIMARY KEY  ('IsoID')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;



-- --------------------------------------------------------


--
-- Table structure for table 'FlaggedPyroprints'
--
```

```
CREATE TABLE IF NOT EXISTS `FlaggedPyroprints` (

  `flagId` int(11) NOT NULL auto_increment,

  `pyroID` int(11) NOT NULL,

  `reason` varchar(100) NOT NULL,

  `dateFlagged` datetime NOT NULL,

  PRIMARY KEY  (`flagId`),

  KEY `ForeignKey` (`pyroID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;


-- --------------------------------------------------------


--

-- Table structure for table `Histograms`

--


CREATE TABLE IF NOT EXISTS `Histograms` (

  `pyroID` int(11) NOT NULL default '0',

  `position` int(11) NOT NULL default '0',

  `pHeight` decimal(8,4) default NULL,

  `cPeakHeight` decimal(8,4) default NULL,

  `nucleotide` char(1) default NULL,

  `PeakArea` float NOT NULL,

  `PeakWidth` float NOT NULL,

  `BaselineOffset` float NOT NULL,

  `SignalValue` float NOT NULL,

  PRIMARY KEY  (`pyroID`,`position`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- ----------------------------------------------------------


--
-- Table structure for table `Host`
--


CREATE TABLE IF NOT EXISTS `Host` (
  `commonName` varchar(50) NOT NULL,
  `hostID` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`commonName`,`hostID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- ----------------------------------------------------------


--
-- Table structure for table `HostSpecies`
--


CREATE TABLE IF NOT EXISTS `HostSpecies` (
  `latinName` varchar(50) default NULL,
  `commonName` varchar(50) NOT NULL,
  `letterCode` varchar(10) default NULL,
  PRIMARY KEY  (`commonName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

-- --

--
-- Table structure for table `IsolateDataset`
--

```
CREATE TABLE IF NOT EXISTS `IsolateDataset` (
  `id` int(11) NOT NULL auto_increment,
  `Name` varchar(100) character set ascii NOT NULL,
  `Date` datetime NOT NULL,
  `Investigator` varchar(200) NOT NULL,
  `Description` varchar(300) NOT NULL,
  `Flag` tinyint(1) NOT NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `UniqueName` (`Name`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=7 ;
```

-- --

--
-- Table structure for table `IsolateDatasetLink`
--

```
CREATE TABLE IF NOT EXISTS `IsolateDatasetLink` (
  `id` int(75) NOT NULL auto_increment,
  `datasetID` int(75) NOT NULL,
  `isoID` varchar(10) NOT NULL,
```

```
  PRIMARY KEY  (`id`),

  KEY `datasetID` (`datasetID`),

  KEY `IsolateIDIndex` (`isoID`)

) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=55 ;



-- --------------------------------------------------------



--

-- Table structure for table `Isolates`

--



CREATE TABLE IF NOT EXISTS `Isolates` (

  `isoID` varchar(15) NOT NULL default '',

  `FreezerLocation` varchar(50) default NULL,

  `TANOTES` varchar(300) default NULL,

  `dateStored` varchar(15) default NULL,

  `PyroPrintDate` varchar(50) default NULL,

  `userName` varchar(50) default NULL,

  `sampleID` int(11) default NULL,

  `commonName` varchar(50) NOT NULL,

  `hostID` varchar(50) NOT NULL,

  `tag` varchar(10) default NULL,

  PRIMARY KEY  (`isoID`),

  KEY `commonName` (`commonName`),

  KEY `hostID` (`hostID`,`commonName`),

  KEY `sampleID` (`sampleID`,`commonName`,`hostID`),

  KEY `userName` (`userName`)
```

```sql
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- ----------------------------------------------------------


--
-- Table structure for table `IsolateScores`
--


CREATE TABLE IF NOT EXISTS `IsolateScores` (

  `isoID1` varchar(10) NOT NULL,

  `isoID2` varchar(10) NOT NULL,

  `region` varchar(10) NOT NULL,

  `score` float NOT NULL,

  `length` int(10) NOT NULL,

  PRIMARY KEY  (`isoID1`,`isoID2`,`region`,`length`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- ----------------------------------------------------------


--
-- Table structure for table `Primer`
--


CREATE TABLE IF NOT EXISTS `Primer` (

  `primerName` varchar(40) NOT NULL default '',

  `sequence` varchar(40) default NULL,

  PRIMARY KEY  (`primerName`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------



--

-- Table structure for table `Pyroprints`

--


CREATE TABLE IF NOT EXISTS `Pyroprints` (

  `pyroID` int(11) NOT NULL default '0',

  `fileName` varchar(150) default NULL,

  `appliedRegion` varchar(20) default NULL,

  `pcrDate` varchar(15) default NULL,

  `wellID` varchar(5) default NULL,

  `pcrMachine` varchar(20) default NULL,

  `sequencerMachine` varchar(20) default NULL,

  `dsName` varchar(205) default NULL,

  `forPrimer` varchar(40) default NULL,

  `revPrimer` varchar(40) default NULL,

  `seqPrimer` varchar(40) default NULL,

  `pyroPrintedTech` varchar(100) default NULL,

  `pyroPrintedDate` varchar(15) default NULL,

  `isoID` varchar(15) default NULL,

  `tag` varchar(10) default NULL,

  PRIMARY KEY  (`pyroID`),

  KEY `dsName` (`dsName`),

  KEY `forPrimer` (`forPrimer`),
```

```sql
  KEY `revPrimer` (`revPrimer`),

  KEY `seqPrimer` (`seqPrimer`),

  KEY `pyroPrintedTech` (`pyroPrintedTech`),

  KEY `isoID` (`isoID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--

-- Triggers `Pyroprints`

--

DROP TRIGGER IF EXISTS `CPLOP`.`TEST1`;

DELIMITER //

CREATE TRIGGER `CPLOP`.`TEST1` AFTER INSERT ON `CPLOP`.`Pyroprints`
 FOR EACH ROW BEGIN


END
//
DELIMITER ;


-- --------------------------------------------------------


--

-- Table structure for table `PyroPrintStatus`

--


CREATE TABLE IF NOT EXISTS `PyroPrintStatus` (

  `pyroprintstatusid` int(10) NOT NULL auto_increment,

  `PyroPrintId` int(10) NOT NULL,
```

107

```sql
  'Status' int(2) NOT NULL default '0',

  PRIMARY KEY  ('pyroprintstatusid')

) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;


-- --------------------------------------------------------


--

-- Table structure for table 'PyroStats'

--


CREATE TABLE IF NOT EXISTS 'PyroStats' (

  'pyroID' int(11) NOT NULL,

  'mean' float NOT NULL,

  'std_dev' float NOT NULL,

  'length' int(11) NOT NULL,

  PRIMARY KEY  ('pyroID','length')

) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--

-- Table structure for table 'Samples'

--


CREATE TABLE IF NOT EXISTS 'Samples' (

  'sampleID' int(11) NOT NULL default '0',

  'commonName' varchar(50) NOT NULL,
```

```
`hostID` varchar(50) NOT NULL,

`location` varchar(50) default NULL,

`dateCollected` varchar(15) default NULL,

PRIMARY KEY  (`sampleID`,`commonName`,`hostID`),

KEY `hostID` (`hostID`,`commonName`),

KEY `commonName` (`commonName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--
-- Table structure for table `Strain`
--


CREATE TABLE IF NOT EXISTS `Strain` (
  `id` int(11) NOT NULL auto_increment,
  `cluster_name` varchar(100) NOT NULL,
  `date` datetime NOT NULL,
  `diameter` float NOT NULL,
  `mean` float NOT NULL,
  `threshold` float NOT NULL,
  `summary` varchar(400) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `Unique` (`threshold`,`cluster_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;


-- --------------------------------------------------------
```

```
--
-- Table structure for table `StrainIsolateLink`
--


CREATE TABLE IF NOT EXISTS `StrainIsolateLink` (

  `id` int(11) NOT NULL auto_increment,

  `cluster_id` int(11) NOT NULL,

  `isolate_id` varchar(15) NOT NULL,

  `date_time` datetime NOT NULL,

  PRIMARY KEY  (`id`),

  KEY `Index` (`cluster_id`),

  KEY `Isolate Id Index` (`isolate_id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;


-- --------------------------------------------------------


--
-- Table structure for table `Users`
--


CREATE TABLE IF NOT EXISTS `Users` (

  `userName` varchar(50) NOT NULL default '',

  `passwordCheck` varchar(50) default NULL,

  `type` varchar(50) default NULL,

  PRIMARY KEY  (`userName`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------



--
-- Constraints for dumped tables
--


--
-- Constraints for table `PyroprintDatasetLink`
--
ALTER TABLE `PyroprintDatasetLink`
  ADD CONSTRAINT `PyroprintDatasetLink_ibfk_1`
  FOREIGN KEY (`ExperimentID`)
  REFERENCES `PyroprintDataset` (`ExperimentID`)
  ON DELETE CASCADE;


--
-- Constraints for table `FlaggedPyroprints`
--
ALTER TABLE `FlaggedPyroprints`
  ADD CONSTRAINT `FlaggedPyroprints_ibfk_1`
  FOREIGN KEY (`pyroID`)
  REFERENCES `Pyroprints` (`pyroID`)
  ON DELETE NO ACTION ON UPDATE NO ACTION;
```

```
--

-- Constraints for table `IsolateDatasetLink`

--

ALTER TABLE `IsolateDatasetLink`

  ADD CONSTRAINT `IsolateDatasetLink_ibfk_1`

  FOREIGN KEY (`isoID`)

  REFERENCES `Isolates` (`isoID`),

  ADD CONSTRAINT `IsolateDataSetLink_fk_1`

  FOREIGN KEY (`datasetID`)

  REFERENCES `IsolateDataset` (`id`)

  ON DELETE CASCADE;


--

-- Constraints for table `StrainFamilyToStrainLink`

--

ALTER TABLE `StrainFamilyToStrainLink`

  ADD CONSTRAINT `StrainFamilyToStrainLink_ibfk_2`

  FOREIGN KEY (`strainFamilyID`)

  REFERENCES `StrainFamilyToStrainLink` (`id`)

  ON DELETE CASCADE,

  ADD CONSTRAINT `StrainFamilyToStrainLink_ibfk_1`

  FOREIGN KEY (`strainID`)

  REFERENCES `Strain` (`id`)

  ON DELETE CASCADE;


--

-- Constraints for table `StrainIsolateLink`
```

```
--

ALTER TABLE `StrainIsolateLink`

  ADD CONSTRAINT `StrainIsolateLink_ibfk_2`

  FOREIGN KEY (`isolate_id`)

  REFERENCES `Isolates` (`isoID`),

  ADD CONSTRAINT `StrainIsolateLink_ibfk_1`

  FOREIGN KEY (`cluster_id`)

  REFERENCES `Strain` (`id`)

  ON DELETE CASCADE;


--

-- Constraints for table `StrainRepPyroprint`

--

ALTER TABLE `StrainRepPyroprint`

  ADD CONSTRAINT `StrainRepPyroprint_ibfk_1`

  FOREIGN KEY (`cluster_id`)

  REFERENCES `Strain` (`id`);
```