2013 IEEE 13th International Conference on Data Mining Workshops

Ontological Hierarchical Clustering for Library-based Microbial Source Tracking

Aldrin Montana, Alex Dekhtyar
Department of Computer Science
Cal Poly
San Luis Obispo, California
drinmontana@acm.org
dekhtyar@calpoly.edu

Michael Black, Chris Kitts
Department of Biology
Cal Poly
San Luis Obispo, California
{mblack, ckitts}@calpoly.edu

Anya Goodman
Department of Chemistry
Cal Poly
San Luis Obispo, California
agoodman@calpoly.edu

***Abstract*—Pyroprinting is a novel library-based microbial source tracking method developed by the Biology department at Cal Poly, San Luis Obispo. Biologists conducting research using pyroprinting rely on methods for partitioning collected bacterial isolates into bacterial strains. Clustering algorithms are often used for bacterial strain analysis of organisms in computational biology. Agglomerative hierarchical clustering, a commonly used clustering algorithm, is inadequate given the nature of data collection for pyroprinting. While the clusters produced are acceptable, pyroprinting requires a method of analysis that is scalable and incorporates useful metadata into the clustering process. We propose an ontology-based hierarchical clustering algorithm OHClust!, a modification of the agglomerative hierarchical clustering algorithm. OHClust! uses metadata associated with the data being clustered to direct the order in which the individual data points and subclusters are compared to each other. This paper describes OHClust! and compares it to agglomerative hierarchical clustering.**

I. Introduction

To maintain the cleanliness of “fishable and swimmable” bodies of water, and improve the cleanliness of other bodies of water, health and environmental protection agencies have been interested in the ability to track sources of fecal contamination at the species level [1], [2], [3]. Identifying the animal species from which a fecal contamination originates is an important first step in controlling fecal contamination and managing bacterial risks. It enables natural resource managers to address the problem of fecal contamination at its source. Investigation of fecal contamination is especially necessary in waters used for human recreation where dangerous pathogens are transferable. This paper addresses the application of a clustering algorithm, OHClust!, for comparing DNA fingerprints of fecal bacterial isolates (individuals) for a new library-dependent microbial source tracing (MST) technique, *pyroprinting*, developed at Cal Poly.

Library-dependent MST techniques attempt to identify sources of fecal contamination at the level of host species using a library, or database, containing DNA fingerprints of bacteria. In the proposed technique, the role of such a “DNA fingerprint” is played by a *pyroprint*, the digital output of a DNA sequencing technique known as *pyrosequencing*.

The workflow for pyroprinting starts with the collection of a large number of isolates from a variety of known species which are hosts of a selected bacteria. The collection of bacterial isolates from known host species continues until the library is sufficiently large. A particular portion of each isolate’s genome, which has a number of regions with high DNA variability, is pyrosequenced [4], [5]. The DNA fingerprint produced by pyrosequencing, called a *pyroprint*, is then archived in a database.

The pyroprint database is used in MST as follows. A bacterial isoloate is collected from the environment, referred to as an *environmental isolate*, and processed using the aforementioned pyrosequencing procedure. A single isolate may be pyrosequenced multiple times in the same, or a different, genomic region. The resulting pyroprint(s) are compared to pyroprints stored in the pyroprint database. The host species of matched pyroprints [1] from the database provide insight into the animal species responsible for leaving the environmental isolate behind.

This process can be sped up significantly by comparing environmental isolates against bacterial strains rather than individual isolates. For this, a process of organizing a digital collection of bacterial isolates into strains is needed.

This paper describes a hierarchical clustering algorithm Ontological Hierarchical Clustering (OHClust!), which is based on agglomerative hierarchical clustering. When bacterial isolates are collected, a lot of different metadata about each isolate is made available to the isolate database: date/time of collection, location, host identity, host species, and more. OHClust! is designed to use bacterial isolate and pyroprint metadata to drive its behavior.

The rest of the paper is organized as follows. Section II describes the data and similarity functions used for the analysis in this paper, as well as describes why clustering is relevant to the problem. Section III discusses the details of the clustering algorithm while Section IV describes relevant related work. Section V evaluates OHClust! in comparison to agglomerative hierarchical clustering and finally, Section VI mentions future work and conclusions.

[1]pyroprints that represent genomic regions of isolates from the same bacterial strain

II. Background

A. Data

There are two primary biological entities being analyzed in this paper–bacterial isolates and pyroprints. Bacterial isolates, or isolates for brevity, are bacterial cultures grown from a pure culture obtained from a clinical specimen. A *pyroprint* is the result of pyrosequencing special regions inside bacterial DNA called intergenic transcribed spacer (ITS) regions. Isolates have two ITS regions of interest that exist in several copies in the genome. At the heart of the pyroprinting MST method lies amplification of all copies of a specific ITS region in a bacterial genome and pyrosequencing the obtained mix of DNA fragments. Pyrosequencing is a DNA sequencing technique that given one DNA strand synthesizes the opposite strand. Each time a successful reaction yields addition of one or more nucleotides to the synthesized strand, light is emitted, and the pyrosequencing equipment captures and measures it. The light emission graph for the mix of DNA fragments for the same ITS region, discretized as described below, is called a *pyroprint* and is used as a digital fingerprint In our work each pyroprint consists of up to 104 light emittance values, each corresponding to a *dispensation event* during the pyrosequencing process: a release of one of the nucleotide (A,T,C,G) reagents into the sequencing process, and a dispensation sequence defining the order of dispensation events.

Clustering is done based on isolate similarity, which is calculated based on pyroprint similarity. When comparing pyroprints, there is a 5-tuple of parameters, known as a *protocol*, that must be identical for the comparison to be meaningful. The protocol is formally defined as follows:

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

where R_k is an ITS region, d is the dispensation sequence, and Pr_f, Pr_r, and Pr_s are forward, reverse, and sequencing primers, respectively.

We use the following notation to represent each entity and their relationships:

$$\mathcal{I} = \{I_1, \ldots, I_n\} \qquad \mathcal{R} = \{R_1, \ldots, R_k\}$$
$$\mathcal{P} = \langle \mathcal{T}, \{\langle I_1, p_1 \rangle, \ldots, \langle I_m, p_m \rangle\} \rangle,$$

where $\mathcal{I}$ represents the set of all bacterial isolates in the input dataset and $\mathcal{R}$ represents the set of all ITS regions of interest [2]. The set of all pyroprints, $\mathcal{P}$, is represented as a tuple consisting of: (1) the protocol used ($\mathcal{T}$), and (2) the set of pyroprints corresponding to the protocol tupled with the bacterial isolate I_m from which pyroprint p_m was constructed. Generally, an individual pyroprint p_m is easily represented as a tuple of the form:

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

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

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

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

Here d is a dispensation sequence and $\bar{p}$ is a vector of values that represent *peak light heights*, i.e., specify the highest light intensity achieved during the time of a single dispensation event from the dispensation sequence d. Each light emittance value p_i of a pyroprint $\bar{p}$ may be 0 or any positive real number.

B. Similarity Functions

To compare two pyroprints with matching protocols, $\bar{p}$ and $\bar{q}$, we use Pearson's correlation:

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

We identify three categories of similarity between a pair of pyroprints: definitely similar, *"squishy"*, and definitely dissimilar. The categorization of pyroprint similarities into these three categories is controlled by two thresholds, $\alpha = 0.995$, the upper threshold, and $\beta = 0.99$, the lower threshold[3]. A thresholded version of Pearson's correlation is sometimes used to transform pyroprint similarities corresponding to these three categories:

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

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

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

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

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

The similarity between two bacterial isolates, I_α, I_β, is then computed as the average of the similarities computed between corresponding ITS regions or 0 if the similarity between any two ITS regions is too low:

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

[2]this paper considers $k = 2$ ITS regions: R_1 = "$16 - 23$" and R_2 = "$23 - 5$"

[3]Generally speaking α and β values are parameters of the pyroprint evaluation process. We use the values specified as they tend to minimize the incidence of true negatives and false positives. The work on establishing the thresholds was performed by Shealy as her (unpublished) senior project, and has been described, with proper attribution, in Montana's M.S. thesis [6].

C. Bacterial Strains and Clusters

A bacterial strain is a biological concept that describes a group of bacterial isolates, from a given species, that are identical based on the method used to identify them. Informally, bacterial strains are groups of bacterial isolates that have the same genotype and are distinguished from those with a different genotype. Formally, a bacterial strain is a set S of bacterial isolates, where each bacterial isolate $I_j \in S$ has a high level of similarity to each other bacterial isolate $I_k \in S$, and is substantially different from other isolates $I' \notin S$. The similarity between bacterial isolates is determined using pyroprints which function as DNA fingerprints of each ITS region. The definition of a bacterial strain is given by the following notation:

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

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

Here, $\mathcal{S}$ represents the set of all determined strains S_1 to S_m. Using the similarity function $sim(I_a, I_b)$ [4], determining the subset of bacterial isolates $\{I_1, \ldots, I_c\}$ from all bacterial isolates $\mathcal{I} = \{I_1, \ldots, I_n\}$, that belong to each strain, $\mathcal{S} = \{S_1, \ldots, S_m\}$, is done by incorporating bacterial isolates with a high similarity into the same strain S_i. This definition shows the problem of identifying bacterial strains as a clustering problem. In this way, the interpretation of constructed clusters is straightforward–each cluster corresponds to a potential bacterial strain.

III. OHClust!

In working with bacterial isolate data, we encountered the need to cluster collections of bacterial isolates while keeping some information about the isolate origins in mind. For example, in one case, a collection of bacterial isolates was obtained from a single individual over a period of six month, with different collection methods used each month to collect a subset of the isolates. When analyzing the data, biologists wanted to break the strain discovery into first determining the strains for each collection method within each month, then, by putting together all strains for a single month, and only then to combine the strains from different months into a single collection.

OHClust! was developed to incorporate isolate and pyroprint metadata into the clustering process in a way that allows for analysis of relationships in sub-groups of the data. Because a quantitative measure for metadata similarity is difficult, OHClust! guides the order of comparisons made during clustering by expressing pyroprint and isolate metadata as an ontology, or hierarchical structure.

A. Metadata and the Ontological Structure

An ontology is represented as an n-ary tree and constructed based on user-specified metadata attributes. Use of ontologies to drive the clustering process is the key distinction between OHClust! and agglomerative hierarchical clustering.

[4]described in Section II-B

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

Fig. 1: Example of a three level ontology with specific and general partitions.

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

Fig. 2: The format for defining an ontology.

During clustering, agglomerative clustering iteratively clusters together data points (or subclusters) that are closest to each other. In contrast, OHClust! uses the ontology to construct a hierarchical view of a data set, clustering subsets of the data throughout the hierarchy. An ontology is a hierarchical structure that allows OHClust! to form clusters as agglomerative clustering does but for a sub-group of the data set.

B. Ontology

1) Ontology Construction: An ontology is constructed using the format in Figure 2. There are two different kinds of *Options*: *LocalOpt*, options particular to a level of the ontology, and *GlobalOpt*, options set for every level of the ontology. Briefly, each *Option* has the following effect [5]:

TimeSensitive:	Force clustering of children to be in chronological order.
SimilarCorr:	Only cluster data points above the α threshold.
SquishyCorr:	Only cluster data points above the β threshold.
Static:	If data point labels do not match any partition for the corresponding metadata type, the data point is stored at the lowest (most specific) partition it was able to reach.
Dynamic:	If data point labels do not match any partition for the corresponding metadata type, a new partition is created.
Transform:	Apply transformation of definitely similar and definitely dissimilar isolates, mentioned in section II-B.

2) Ontology Usage: After construction, the ontology is used by OHClust! in the following manner: (1) Isolates are added. (2) Clustering takes place within each node of the ontology in the bottom-up fashion. (3) Agglomerative clustering is used for clustering within each node. (4) Clusters are percolated up the ontology until the root is reached.

The metadata attributes for each isolate added to the ontology are compared to each *partition* in the ontology. The partition that matches a metadata attribute determines the correct edge, or path, down which the isolate should be propagated. This repeats at every level until the isolate is placed in the appropriate node. Metadata attributes represent columns of the database whereas partitions are distinct values of a metadata attribute used to separate sub-groups of a

[5]Feature options in bold are default options for each feature

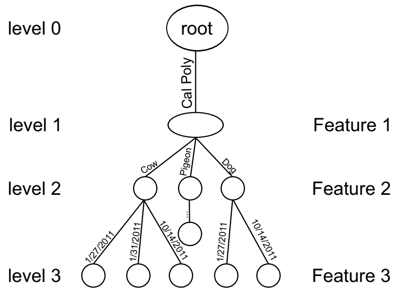

Fig. 3: A three level ontology corresponding to the ontology format in Figure 1.

data set. As an example, the metadata attributes that are important for the general ontology used for CPLOP (Cal Poly Library of Pyroprints), the database used to store all collected isolates of *E. Coli*, are *commonName*, *hostID*, *sampleID*, and *dateCollected* [7]. The level of the ontology corresponding to the *commonName* attribute have several partitions including *dog*, *cat*, *cow*, etc.

C. OHClust! Algorithm

Once the ontology is constructed and populated with the isolates, it is ready to be clustered. The clustering step applies agglomerative clustering recursively on the data contained in the ontology, starting at the leaves. When clustering, all similarities between clusters are calculated and stored in a similarity matrix. The function ontologicalCluster (Figure 4) iterates over an ontology, applying agglomerative clustering within each node of the ontology and percolating clusters up the ontology. It is a recursive function, where the first call is passed the root of the ontology as N. Each set of clusters produced from a node is passed to the parent of N. Once all siblings of N has been clustered, then the parent of N, itself, may be clustered.

IV. Related Work

A. Incremental Clustering

This section discusses incremental clustering algorithms developed for generic datasets. These algorithms were developed to incrementally cluster data streams or datasets too large to fit in main memory.

BIRCH, Balanced Iterative Reducing and Clustering using Hierarchies, is a clustering method developed by Zhang, et al. to address the problem of clustering large datasets and minimizing I/O costs [8]. BIRCH incrementally clusters numerical data by doing a scan of the target dataset until memory constraints are reached. Each chunk of data is added to an N-ary tree, called a cluster feature (CF) tree, which represents clusters as internal nodes and maintains only aggregate information for each cluster. The aggregate information, and clustering requirements, are computed based on basic algebraic functions that are additive and easily computed in a single scan of the data. This approach uses the CF Tree as a fast, guiding method for directing a data point to the appropriate cluster. However, determining an appropriate, meaningful CF representation is difficult.

```
function ONTOLOGICALCLUSTER($D$, $N$, $T$)
    $C \leftarrow \emptyset$
    if $N = null$ then        ▷ If node is null; return emptyset
      return $C$
    end if
    if $|\text{children}(N)| > 0$ then
        for $n_i \in \text{children}(N)$ do        ▷ cluster data in each child
            $C \leftarrow C \cup \text{ontologicalCluster}(D, n_i)$
            if isTimeSensitive($N$) then
                agglomerativeCluster($D$, $C$, $T$)
            end if
        end for
        if !isTimeSensitive($N$) then
            agglomerativeCluster($D$, $C$, $T$)
        end if
    else                              ▷ If leaf node; cluster data
        agglomerativeCluster($D$, $C$, $T$)
    end if
      return $C$
end function
```

Fig. 4: Pseudo-code for clustering in OHClust!.

Young, et al.'s work on incremental clustering takes a partitional approach to clustering [9]. The described work utilizes a competitive learning algorithm that adjusts itself based on a calculated pseudo-entropy of the clusters such that a tradeoff is made between updating centroids aggressively earlier in the clustering process (earlier iterations and/or updates) and updating centroids conservatively later to ensure cluster stability. Additionally, there is a credit-based algorithm for preventing centroid starvation which is computed based on a fixed value or the previously calculated pseudo-entropy such that centroids selected for updates lose credit and centroids that are starving accumulate credit over time. Since the number of clusters that may be formed from clustering pyroprints is completely unknown, and will grow unpredictably as more data is gathered from different regions, a partitional scheme with a fixed number of centroids is not ideal.

B. Incremental Clustering for 16S rRNA Sequences

This section discusses incremental clustering algorithms that were developed specifically for clustering DNA sequence reads. Incremental clustering algorithms described in this section are all derived from a greedy incremental sequence clustering algorithm developed in 1998 by Holm and Sander [10], hereafter referred to as SeqClustering for brevity. The motivation for this incremental clustering algorithm was to effectively handle continuously growing biological datasets, specifically protein sequences, and to efficiently determine and convey clusters of similar protein sequences. Due to the high volume of redundant protein sequence reads, Holm and Sander decided that each cluster should be represented by a single data point, or sequence read. By using a representative data point for each cluster, it was much more efficient to compute cluster membership for each unclustered data point and to convey meaningful, non-redundant information for understanding each cluster. Variants of this clustering algorithm refer to the representative sequence read of a cluster as a *seed*. This algorithm, as well as its variants such as CD-HIT [11], UCLUST [12], and DySC [13], cluster DNA sequences by using a single sequence as a cluster representative to which unclustered DNA sequences are aligned against. The comparison metric used is typically a sequence alignment.

CD-HIT is a SeqClustering variant with several variants: CD-HIT-2D, CD-HIT-EST, CD-HIT-EST-2D [11], [14]. CD-HIT uses short word filtering instead of sequence alignments as its comparison metric to improve performance. Short word filtering verifies that the compared sequences share a minimum number of identical short substrings, known as *words*, such as dipeptides, tripeptides, etc.

Dynamic seed clustering (DySC), was developed by Zheng, et al. [13]. DySC is a clustering algorithm developed for reads of the 16S rRNA marker gene commonly used in microbial studies. DySC uses both fixed and dynamic seeds. A fixed seed in DySC is equivalent to seeds in SeqClustering and dynamic seeds have no equivalent. Using dynamic seeds, DySC constructs *pending clusters*, clusters containing reads that are not within a similarity threshold to the fixed seeds. Pending clusters are transient and will either join a fixed seed cluster or become a fixed seed cluster after reaching a specified size. Zheng, et al. claim that DySC is able to improve cluster quality while maintaining comparable runtime compared to UCLUST and CD-HIT [13].

Yooseph, et al. worked on incremental clustering of microbial metagenomic sequence data [15]. It is a three stage clustering process based on CD-HIT. **First Stage**. patterns are combined with clusters in three steps, each with 90%, 75%, and 60% identity thresholds, respectively. Incremental clustering takes this approach of decreasing identity thresholds for efficiency and quality. For efficiency, CD-HIT-2D runs faster at high thresholds (90%) than at low thresholds (60%). For quality, the parallel implementation of CD-HIT-2D being used by Yooseph, et al. at the time (2008) could only assign patterns to the first cluster whose similarity met the threshold. **Second Stage**. Patterns from the data set not incorporated into clusters in stage one are clustered together using CD-HIT at 90%, 75%, and 60% identity. The clusters formed here are referred to as *core clusters* by Yooseph, et al. **Third Stage**. Two similarity measures are used to join *big core clusters* (cluster with cardinality $\geq$ 20) with each other and small core clusters or singleton clusters (cluster with cardinality = 1) with final clusters, respectively FFAS and PSI-BLAST.

Unfortunately, determining a good seed for a cluster when clustering isolates is difficult. For this reason, many of the SeqClustering variants are not ideal for the work described in this paper. OHClust! takes the approach of first clustering data points that are definitely similar. The clusters formed are small and very tight, and can be used instead of a representative data point. This is useful due to fluctuation and error involved in pyroprinting. While many of these algorithms may be applicable to incrementally clustering isolates, it would require extra effort to also investigate relationships in subsets of the data. Additionally, many of these algorithms use more of a partitional approach as opposed to a hierarchical approach, which is not ideal for clustering isolates.

Additional related work include feature guided clustering, as well as the use of specific data to guide clustering of particular data sets. Clustering algorithms in these fields are very interesting as they take a mathematical approach to guiding the order of clustering through the use of subspaces [16]. Unfortunately, determining just how closely related to OHClust! subspace, or mathematically, guided clustering is is difficult to determine. It sounds incredibly similar in the fact that the clustering is being guided to group or compare certain data points prior to other data points given some useful information. However, it also seems that the work described by Chopra, et al. takes the approach of quantifying relationships between data points. Clustering that utilizes subspaces of a data set takes a more multi-dimensional quantitative approach to data point comparison whereas OHClust! is primarily applicable to single-dimensional data point comparison with guidance being given through categorical means.

V. Evaluation

For our evaluation, we use agglomerative clustering as a baseline for evaluating OHClust!. This is because biologists consider agglomerative clustering results to be acceptable. Preliminary results on the comparison between OHClust! and agglomerative clustering were reported in a previous case study [**?**]. However, here the comparison is investigated more thoroughly using larger datasets.

A. Notation and Evaluation Metrics

For our evaluation, we use the same notation described by Wagner and Wagner [17]. Let $\mathcal{C}$ and $\mathcal{C}'$ be the set of clusters produced by OHClust! and agglomerative clustering, respectively. When comparing the two sets of clusters, we must consider four disjoint sets of isolate pairs:

$$\begin{aligned}
S_{00} &= \{I_j, I_k \notin C_a \in \mathcal{C} \wedge I_j, I_k \notin C'_b \in \mathcal{C}'\}\\
S_{10} &= \{I_j, I_k \in C_a \in \mathcal{C} \wedge I_j, I_k \notin C'_b \in \mathcal{C}'\}\\
S_{01} &= \{I_j, I_k \notin C_a \in \mathcal{C} \wedge I_j, I_k \in C'_b \in \mathcal{C}'\}\\
S_{11} &= \{I_j, I_k \in C_a \in \mathcal{C} \wedge I_j, I_k \in C'_b \in \mathcal{C}'\}
\end{aligned}$$

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

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

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

For the comparison of clusters produced by OHClust! and agglomerative clustering, we consider three comparison metrics: Jaccard Index, Dice's coefficient, and Rand Index. The Jaccard Index and Rand Index are explained very well by Wagner and Wagner [17].

The Jaccard Index is applied to two sets of clusters $\mathcal{C}$ and $\mathcal{C}'$ as follows:

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

where n_{11} represent the intersection, or agreement, between $\mathcal{C}$ and $\mathcal{C}'$, and $n_{10}+n_{01}+n_{11}$ represents the union, or isolate pairs that are in the same cluster in either $\mathcal{C}$, $\mathcal{C}'$, or both. The only isolate pairs not considered in the Jaccard Index function are pairs that are in different clusters in both $\mathcal{C}$ and $\mathcal{C}'$. However, these pairs are accounted for in Dice's coefficient.

Dice's coefficient for comparing clusters is:

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

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

In contrast, the Rand Index considers isolate pairs in the set S_{00} as part of the similarity between two sets of clusters. The formulation of the Rand Index we use is:

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

Notice that the denominator $\frac{n\,(n-1)}{2}$ is the number of isolate pairs being considered ($\binom{n}{2} = \frac{n\,(n-1)}{2}$). The numerator $(n_{11} + n_{00})$ represents the isolate pairs with the same cluster assignment, or classification, in both sets of clusters $\mathcal{C}$ and $\mathcal{C}'$. The Rand Index is a simple comparison metric that counts the number of agreements between each set of clusters and then divides by the total number of isolate pairs considered.

B. Evaluation Data

Unfortunately, the evaluation included in this paper was unable to run efficiently on large datasets. While code has been improved since, the evaluation has not yet been updated and so this paper describes evaluations on smaller, randomly generated datasets.

To create randomized datasets, new data was generated by applying genetic algorithm concepts–crossover and mutation–to existing actual data. Two isolates are selected at random, multiple times, from a random pool of a given size. If the two isolates are identical, then they are simply duplicated with a new ID. If they are not identical, then there is an 80% chance that a crossover will occur between the two isolates. In the event that a crossover does not occur, then mutation will occur in one of the two isolates selected at random with equal probability.

a) Crossover.: Two pyroprints, p_a^i and p_b^i, are selected at random from the same ITS region R_i of isolates I_a and I_b. The crossover point is randomly chosen from the last 25% of the pyroprint vector. Each peak height from the crossover point to the end of each pyroprint vector is swapped. There is then a 25% chance that a second, or double, crossover will occur, in which case the second crossover point is randomly chosen from the $25\% - 50\%$ of the pyroprint vector. The length of the second crossover is 25% of the pyroprint vector length.

b) Mutation.: An isolate is chosen at random. From this isolate, a pyroprint is chosen at random. Each peak height in the selected pyroprint vector is scaled $X \in [-15, +15]\%$, with each peak height being mutated independently of others.

While many of the parameters were arbitrarily set, the main goal for this data is to increase our dataset size without simply duplicating randomly selected data points. Additionally, because the accuracy of resulting clusters are compared between OHClust! and agglomerative clustering, the results should be similar for the same pool of isolates regardless of the actual relationships present in the data. To provide some form of control, however, each algorithm is also run on the full CPLOP dataset.

C. Hypothesis

For clustering performance, it is expected that OHClust! will perform similarly to agglomerative clustering for a single or initial run. For smaller datasets, the overhead associated with building and filling the an ontology used by OHClust! is likely to worsen performance when compared to agglomerative clustering. However, for large datasets, the partitioning of data points into $O(p^k)$ paths of an ontology should make OHClust! perform better.

For cluster results, it is expected that OHClust! will produce clusters with a high Jaccard, Rand, and Dice similarity to clusters produced by agglomerative clustering. Preferably at least two measures will produce a similarity of at least 70%.

D. Results

The results of our evaluation was done using a machine with two 2GHz 8-core Intel Xeon E5-2650 processors and 64 GB of RAM. For our evaluation we consider the following information: total runtime, runtime for each update, initial runtime, cluster similarity.

1) Test Ontologies: The ontologies pictured in Figures 5 and 6 are used for all OHClust! tests. The ontology pictured in Figure 5 is called the *fixed ontology*, whereas the ontology in Figure 6 is called the *dynamic ontology*.

The fixed ontology has a depth of only two, but uses all distinct metadata values in the database as an edge to a child node. In this case, the similarity between OHClust! and agglomerative clustering should be similar. The dynamic ontology is used to better understand the influence of the dimensions of the ontology on performance. The height is

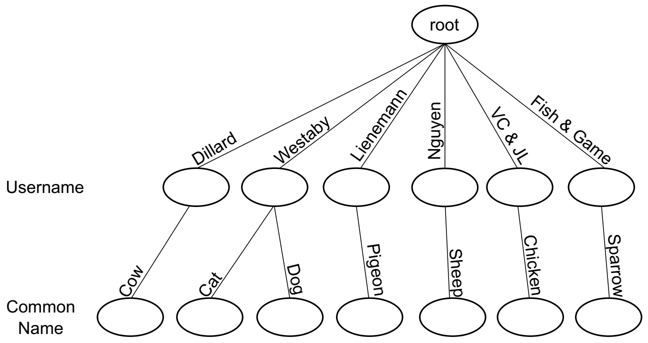

Fig. 5: A portion of the fixed ontology used for most tests consisting of the seven largest partitions of the data set.

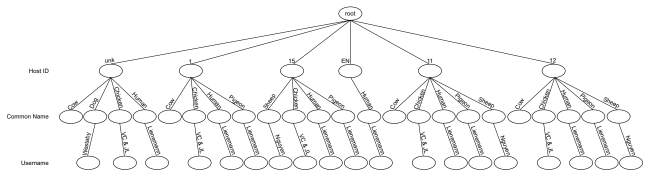

Fig. 6: A portion of the ontologies with varying depths used for determining influence of ontology on performance. Only shows the partitions with the most data present.

varied by cumulatively adding levels, or metadata attributes of interest, representing experimental complexity. The width is varied by including more partitions per level, or distinct values of each metadata attribute. This represents the complexity or generality of specific metadata attributes.

2) OHClust! Performance: The graph in Figure 7 shows performance times for OHClust! vs agglomerative clustering on datasets of size 500, 1000, 1500, 2000, and 2500. This graph is particularly meaningful because it is for fixed data set sizes. This provides the best "apples to apples" comparison of the performance of both OHClust! and agglomerative clustering because OHClust! is able to accommodate incremental updates whereas agglomerative clustering is not. However, even in this case where there are no incremental updates, OHClust! clearly performs significantly better.

Figure 8 shows the cumulative run time for each given incremental update for an initial size of 100 isolates and 20 iterations of adding 15 isolates. As expected, OHClust! significantly outperforms agglomerative clustering when incremental updates are considered. In the future, after both clustering implementations have been improved, this evaluation will be run on larger data sets in order to ensure that this trend is consistent for cases where the overhead of the ontology or memory usage becomes a limitation.

In Figure 9, we see individual run times for each incremental update with an initial size of 100 isolates and 20 iterations of adding 15 isolates. Given the data in Figure 8, this behavior is to be expected. This seems to verify that the improved performance of OHClust! over agglomerative clustering is not merely due to the behavior seen in Figure 7, but also in an ability to handle incremental updates more efficiently.

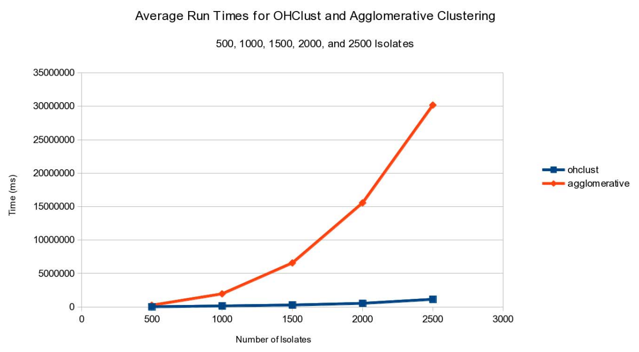

Fig. 7: Runtime for OHClust! vs Agglomerative for fixed data set size of 500, 1000, 1500, 2000, and 2500 isolates.

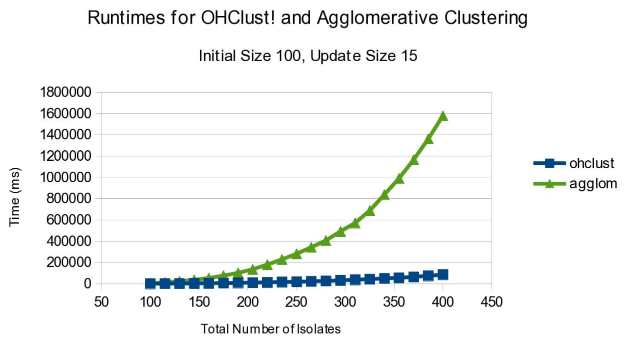

Fig. 8: Total Runtime for OHClust! vs Agglomerative with initial size 100 and 20 updates of 15%.

Table I suggests the ontology depth has minimal impact on runtime. While increasing the number of isolates in the ontology affects runtime, varying the size of the ontology for the same size dataset has almost no impact on total runtime (about $1 - 2\%$ in most cases). I suspect that this is largely due to some branches of the ontology being more heavily populated than others, especially branches that are "useless." Useless branches represent combinations of metadata values that are not represented in the database. For example, Figure 6 shows only the useful paths of the ontology. Isolates labelled with "CA Dept Fish and Game" do not have labels matching any of the six common names and so the "CA Dept Fish and Game" path is omitted. In order to better evaluate the influence of ontology depth in the future, a more balanced data set, with regards to isolate metadata, is necessary.

The results in Table II show minimal impact on the performance of OHClust! using ontologies of varying width. A three level ontology was used with three nodes on each level for the first test and adding another node on each level for each subsequent test. For a random data set and a poorly balanced ontology, the average total runtime increases 0.81% from three nodes on each level to six nodes on each level. The levels of the ontology were the the same as the "large.ont" ontology used in Table I.

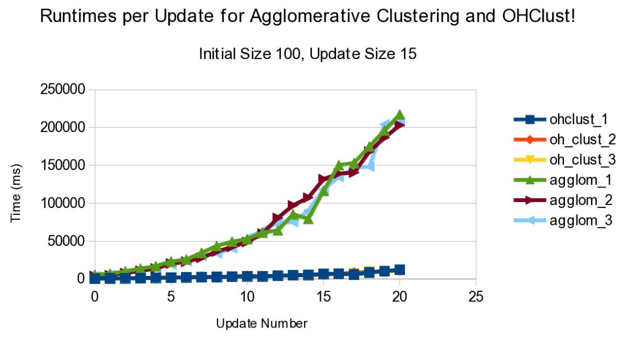

Fig. 9: Runtime for incremental updates for OHClust! vs Agglomerative with initial size 100 and 20 updates of 15%.

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

TABLE I: Cluster performance for varying depths of an ontology.

3) Clustering Similarity: Interestingly, the cluster similarity evaluation metrics presented in Table III do not agree as much as was initially predicted. While the Rand Index agrees with the hypothesis that cluster results between OHClust! and agglomerative clustering should be very similar, Jaccard and Dice both seem to disagree in some cases. It is possible that the amount of single isolate clusters affect these metrics negatively since neither Dice or Jaccard consider n_{00} to improve similarity.

Another evaluation was run for all data points that are present in clusters of size ≥ 5. These evaluations are presented in Table IV, however, it seems that the evaluation metrics report lower similarity for clusters of size ≥ 5. This definitely does not agree with our initial hypothesis, however the high Rand Index values in Table III in comparison to the Jaccard and Dice metrics seems to indicate a high similarity in some respect. While it's clear why Rand would fall if single isolate clusters are excluded, the removal of single isolate clusters seems like it should greatly improve the Jaccard and Dice coefficients, but the cause is currently unknown.

Additionally, while the data set of size 400 has high agreement and coincides with our predictions, the tests with data set sizes 300 and 500 have very low Jaccard and Dice similarity. Although, the rand index is still very good. The tests with 300 and 500 sized data sets used the same pool of random isolates. However, the test with 400 data points was ran with a different pool of isolates. It's possible that one data set simply

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

TABLE II: Cluster performance for varying widths of an ontology.

Num Isolates	Transform?	Jaccard	Dice	Rand
300	Yes	0.3486	0.0967	0.9097
400	No	0.7934	0.7649	0.9004
500	Yes	0.3907	0.0896	0.9302
750	Yes	0.2503	0.0451	0.9324
1000	No	0.7295	0.4201	0.9221

TABLE III: Cluster similarity results for various sized data sets.

had more isolates that were similar. This would have to be addressed in more extensive evaluations in the future.

VI. Future Work and Conclusions

A. Future Work

There are many aspects of pyroprint analysis and SPAM that can be extended or improved in the future. While the application of OHClust! provides greater insight into sub-structures of clusters given metadata of interest, work on using this information to influence scientific workflows would be an interesting long-term goal.

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

VII. Conclusion

This paper has addressed the analysis of some aspects of OHClust! in comparison to agglomerative clustering. While initial evaluation results seem to heavily favor OHClust!, there are some issues with the evaluation conducted, as well as many other aspects to evaluate. However, the performance comparisons seem very promising, and continued use of OHClust! by students and faculty in the Biology dept at Cal Poly may obviate the practical uses of using various ontologies for insight into isolate relationships. Finally, although some similarity metrics do not agree as much as initially believed, the Rand Index still shows very high similarity between OHClust! and agglomerative clustering. For the application of pyroprinting, OHClust! seems to be a better choice compared to agglomerative clustering.

Num Isolates	Transform?	Jaccard	Dice	Rand
300	Yes	0.0735	0.0308	0.8058
400	No	0.4118	0.5726	0.5911
500	Yes	0.0608	0.0233	0.8200
750	Yes	0.0424	0.0121	0.8635
1000	No	0.2025	0.1894	0.6271

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

ACKNOWLEDGMENT

This work was supported in part by grants from the W.M. Keck Foundation and the National Science Foundation grant number 1140828. Special thanks to Emily Neal, whose M.S. thesis was the first to apply OHClust! and was very important for figuring out initial implementation details for the algorithm.

REFERENCES

[1] 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.*, vol. 68, pp. 5796–5803, Dec 2002.

[2] J. M. Simpson, J. W. S. Domingo, and D. J. Reasoner, "Microbial source tracking: State of the science," *Environmental Science and Technology*, vol. 36, pp. 5279–5288, Dec 2002.

[3] T. R. Desmarais, H. M. Solo-Gabriele, and C. J. Palmer, "Influence of soil on fecal indicator organisms in a tidally influenced subtropical environment," *Applied and Environmental Microbiology*, vol. 68, pp. 1165–1172, Mar 2002.

[4] E. Neal, C. Sabatini, W. Tang, M. Black, and C. Kitts, "Demographics of e. coli strains in the human gut using pyroprints: A novel mst method," Jan 2012.

[5] W. Tang, C. J. Sabatini, E. R. Neal, A. Goodman, A. Dekhtyar, K. McGaughey, M. W. Black, and C. L. Kitts, "Investigating changes in prevalent human escherichia coli strains with different collection methods using pyroprinting: A novel mst method," May 2012.

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

[7] J. L. Soliman, "Cplop: Cal poly library of pyroprints," Master's thesis, California Polytechnic State University, San Luis Obispo, California, 2013.

[8] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96. New York, NY, USA: ACM, 1996, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/233269.233324

[9] S. Young, I. Arel, T. P. Karnowski, and D. Rose, "A fast and stable incremental clustering algorithm," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, 2010, pp. 204–209.

[10] L. Holm and C. Sander, "Removing near-neighbour redundancy from large protein sequence collections," *Bioinformatics*, vol. 14, no. 5, pp. 423–429, 1998.

[11] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006.

[12] R. C. Edgar, "Search and clustering orders of magnitude faster than blast," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[13] Z. Zheng, S. Kramer, and B. Schmidt, "Dysc: software for greedy clustering of 16s rrna reads," *Bioinformatics*, vol. 28, no. 16, pp. 2182–2183, 2012.

[14] W. Li, L. Jaroszewski, and A. Godzik, "Tolerating some redundancy significantly speeds up clustering of large protein databases," *Bioinformatics*, vol. 18, no. 1, pp. 77–82, 2002.

[15] S. Yooseph, W. Li, and G. Sutton, "Gene identification and protein classification in microbial metagenomic sequence data via incremental clustering," *BMC bioinformatics*, vol. 9, no. 1, p. 182, 2008.

[16] P. Chopra, J. Kang, J. Yang, H. Cho, H. S. Kim, and M.-G. Lee, "Microarray data mining using landmark gene-guided clustering," *BMC bioinformatics*, vol. 9, no. 1, p. 92, 2008.

[17] S. Wagner and D. Wagner, "Comparing clusterings–an overview," Universität Karlsruhe, Fakultät für Informatik, Tech. Rep. 2006-04, 2007. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000011477