COMPUTATIONAL PAN-GENOMICS: ALGORITHMS AND APPLICATIONS

by

Alan Cleary

A dissertation proposal submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

August, 2017

ACKNOWLEDGEMENTS

I would like to thank my Advisor, Dr. Brendan Mumey, for his support and guidance. I would also like to thank Andrew Farmer at the National Center for Genome Resources for his support and patience with my feeble understanding of biology.

Funding Acknowledgment

This work was kindly supported by the National Center for Genome Resources, United States Department of Agriculture Agricultural Research Service project funding for the Legume Information System, National Science Foundation Integrative Organismal Systems award number 1444806, National Science Foundation Advances in Biological Informatics award number 1542262, and Google Summer of Code.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - CONTINUED

LIST OF FIGURES - CONTINUED

LIST OF FIGURES - CONTINUED

ABSTRACT

As the cost of sequencing DNA continues to drop the number of sequenced genomes rapidly grows. In the recent past, the cost dropped so low that it is no longer prohibitively expensive to sequence multiple genomes for the same species. This has led to a shift from the single reference genome per species paradigm to the more comprehensive pan-genomics approach, where populations of genomes from one or more species are analyzed together.

The total genetic/genomic content of a population is vast, requiring algorithms for analysis that are more sophisticated and scalable than existing methods. Furthmore, existing algorithms are generally not intended for analyzing populations, let alone without a reference, and so they are innapropriate for pan-genome analysis. The focus of this dissertation proposal is the exploration of new algorithms and their applications to pan-genome analysis, both at the nucleotide and genic resolutions.

CHAPTER ONE

INTRODUCTION

1.1 Motivation

Since the publication of the first working draft of the human genome in 2001, the National Human Genome Research Institute (NHGRI), a division of the National Institutes of Health (NIH), has been tracking the cost of sequencing a human-sized genome each year; see Figure 1.1. Not only has the cost of sequencing a human genome generally decreased, starting in January of 2008, the rate at which the cost was decreasing surpassed Moore's Law [75]. Subsequently, the number of genomes sequenced per year has been growing at an exponential rate [81], with research institutions and genome sequencing consortia sequencing multiple genomes per species at a massive scale[1].

Although the cost of sequencing genomes has drastically declined in the past few years, the common approach to genomics is still "reference-centric"; that is, a single *reference* genome is used as a representative for a particular species. Since a reference genome is the sequence of a single individual or a mosaic of individuals as a single linear sequence, this approach is extremely biased. It is a relic of the foundational period of genomics when sequencing multiple genomes of a particular species was limited by technological and budgetary concerns. Only quite recently has

[1]See `http://www.1000genomes.org/`, `http://www.1001genomes.org/`, `http://www.100kgenome.vetmed.ucdavis.edu/`, and [43, 58].

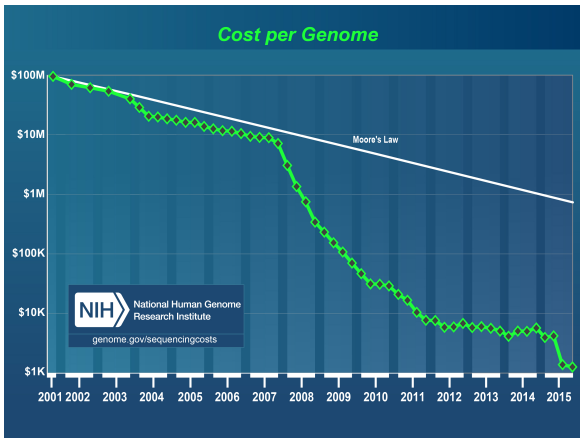

Figure 1.1: The cost of sequencing a human-sized genome per year according to the NHGRI [52].

genomics begun to expand from a single reference per species paradigm into a more comprehensive pan-genome approach that analyzes multiple individuals together.

As a young field, there is still much foundational work to be done [75]. Specifically, there is a need for efficient data structures, algorithms, and statistical methods to perform bioinformatic analyses of pan-genomic data. Addressing these computational needs is the focus of this dissertation proposal.

1.2 Preliminaries

Deoxyribonucleic acid (DNA) is the molecule that encodes the "blueprint" of all known living organisms and many viruses. A sequence of DNA is composed of monomer units called nucleotides. Each nucleotide is composed of one of four nucleobases: cytosine (C), guamine (G), adenine (A), and thymine (T). Most DNA molecules consist of two strands coiled around each other to form a double helix. For each nucleotide in one strand, there is a complementary nucleotide in the other strand: A $\rightarrow$ T, C $\rightarrow$ G, G $\rightarrow$ C, and T $\rightarrow$ A. A nucleotide and its complement are

referred to as a *base pair*. The units of length that are typically used to describe the length of a sequence of DNA are listed in Table 1.1.

In this work, we discuss De Bruijn graphs of the variety commonly used for genome assembly, rather than the classical data structure from which they are derived [29, 39]. A De Bruijn graph is an abstract graph that represents overlap information in a set of DNA sequences [50, 83]. Specifically, each unique length k subsequence (k-mer) in the set of DNA sequences is represented by a node in the De Bruijn graph. If the last $k-1$ characters of a k-mer overlap with the first $k-1$ characters of another k-mer in one or more of the DNA sequences in the set, then the corresponding De Bruijn nodes are connected by a directed edge, the orientation of which reflects the k-mers' ordering in the DNA sequences. A De Bruijn graph can be compressed by collapsing non-divergent chains of nodes into a single super-node [59]. Additionally, a De Bruijn graph can be colored by giving each genome's path through the graph a unique color [53]. A bubble in a De Bruijn graph is a pair of subpaths that have the same start and end nodes but are otherwise vertex disjoint.

A *repeat* is an exact sequence of nucleotides or genes that occurs multiple times in a genome. Similarly, a synteny block is a sequence of nucleotides or genes that is present, or *conserved*, across related genomes. Unlike repeats, instances of a synteny block may vary within or between genomes. Identifying synteny blocks and their variations is a fundamental bioinformatics analysis, for example, they are integral to inferring phylogenies and characterizing functional variation within a group of related species [32].

Variation in a genome refers to deviation from another sequence, typically a reference. The basic types of variation at the nucleotide resolution are single nucleotide polymorphisms (SNPs, pronounced "snips"), the insertion or deletion of contiguous sequence (indels), novel repeats, and inversions - the reversal of a

Table 1.1: DNA Length Measurements and Abbreviations

Unit	Abbreviation	Length
base pair(s)	bp	A single DNA nucleotide and its reverse compliment
kilo base pairs	kb	$1,000$bp
mega base pairs	Mb	$1,000,000$bp
giga base pairs	Gb	$1,000,000,000$bp

repeat. When such polymorphisms occur in the sequence of a gene, especially a gene duplicated across species, the instances of the gene are called *paralogous*. Variations at the genic resolution are called structural variations and include the presence, absence, and variation of copy number[2], polyploidy[3], segmental and/or tandem duplication[4], and inversions.

1.3 Pan-Genomics

The term "pan-genome" was first coined by Sigaux in [88] and was used to describe a public database containing an assessment of genome and transcriptome alterations in major types of tumors, tissues, and experimental models. The term was later revitalized by Tettelin et al. in [94] to describe a microbial genome by which genes were in the core (present in all strains) and which genes were dispensable (missing from one or more of the strains). The term is now used as an umbrella to describe any collection of genetic or genomic sequences from the same or similar species to be analyzed jointly or to be used as a reference.

Today, there exist a variety of tools for pan-genomic analysis [95]. Unfortunately, they are limited in the size and number of genomes they can analyze. This has

[2]Copy number refers to the number of times a gene is repeated.

[3]Polyploidy refers to the presence of more than two paired sets of chromosomes in an organism.

[4]Segmental duplication refers to the near perfect replication of a sequence of genes. Tandem duplication refers to the (segmental) duplication being collocated with the duplicated sequence.

generally been due to insufficient computing hardware and improper representation of the data (non-graphical). Fortunately, due to recent advances in computing hardware and algorithms, these are no longer blocking issues.

1.4 Research Questions

The broad question this proposal will address is:

Question 1

What data structures, algorithms, and statistical methods can be used to perform bioinformatic analyses of pan-genomic data?

This question is indeed broad and so this proposal will primarily focus on the following sub-questions:

Question 2

Pan-genomic data are commonly represented in a graph data structure; how can this data structure be utilized to enable novel and efficient methods of bioinformatic analysis?

Question 3

Given the potentially massive size of pan-genomic data, how can these data structures and algorithms be distributed/parallelized?

Question 4

The interpretation of biological data and the results of their analysis are tasks performed by domain experts, often with the aid of visualization tools. How can pan-genomic data and the results of their analysis be visualized effectively for the purpose of interpretation?

1.5 Organization

The research questions are addressed in the following chapters: Chapter 2 addresses Questions 2 through 4, with an emphasis on Question 2, by introducing a novel method for mining synteny from pan-genome De Bruijn graphs with explorations in visualization and phylogeny construction. Chapter 3 addresses Questions 2 through 4, with an emphasis on Question 2, by introducing another novel method for mining synteny from pan-genome De Bruijn graphs with explorations in visualization and machine learning. Chapter 4 addresses Questions 2 through 4, with an emphasis on Question 4, by introducing an interactive visual data-mining tool for mining micro-synteny at the genic resolution with explorations in distributed algorithms. Chapter 5 proposes a dissertation based on the discussion in Chapters 2 through 4. And Chapter 6 concludes this dissertation proposal.

CHAPTER TWO

APPROXIMATE FREQUENT SUBPATHS

Mining synteny from populations of genomes is an important problem. In this Chapter we will address this problem, and subsequently research Questions 2 through 4 from Chapter 1, with an emphasis on Question 2, by considering the problem of mining Approximate Frequent Subpaths (AFSs) from a pan-genomic De Bruijn graph, where each genomic sequence corresponds to a path in the graph. We formalize the AFS problem, discuss its computational complexity, and describe an effective algorithm for mining AFSs. We also discuss results of the algorithm applied to a yeast pan-genomic data set, which corroborate the existing yeast knowledge base.

2.1 Introduction

Recently, pan-genomes have been represented using colored De Bruijn graphs [4, 71, 74], where each genome's path through the graph is given a unique color [53]. We propose a novel method of path-graph analysis based on *Approximate Frequent Subpaths*, that is, subpaths through a graph that are approximately followed by some subset of the paths through the graph, and use it to analyze the pan-genome of 10 yeast genomes. We chose yeast as our target organism because it is a well studied model system with some published comparative genomic and pan-genomic work. This allowed us to use the existing knowledge base to assess the quality of the subpaths found by our algorithm.

2.2 Related Work

The AFS problem is similar to the Frequent Itemset Mining (FI) problem, which identifies sets of items that frequently occur together in a database of transactions [1]. Unlike the AFS problem, FI does not consider the structure of the graph or the ordering of nodes when mining itemsets. There are variations of FI that are tolerant to noise in the data [14, 67]. These require that the supporting transactions (rows) in the transaction matrix meet a *row error threshold* constraint and the items (columns) meet a *column error threshold* (support) constraint.

A graph-specific problem similar to the AFS problem is Frequent Subgraph Mining (FS), which is concerned with finding frequent subgraphs in a database of graphs. It is a well studied problem [55] commonly applied to biological datasets [45, 47, 61]. Unfortunately, the existing methods are not tolerant to error, nor do they scale to pan-genomic data. For example, in [45] the data sets on which the authors evaluate their "scalable" methods are equivalent in size to what would be considered a small pan-genome, such as a microbial pan-genome [71, 73]. A graph-specific problem more closely related to the AFS problem is Exact Frequent Subpath Mining (EFS) [41]. While EFS is different from FI in that it considers the structure of the graph to achieve more efficient running time and more accurate results, it is incapable of mining frequent subpaths whose supporting paths contain some error.

Recently, bioinformatics tools have started to explore multiple genomes in analyses. Several tools have evolved to take advantage of information from multiple, closely related genomes (species, strains/lines) to perform bioinformatic analyses, such as variant detection without the bias introduced from using a single reference. For example, Cortex [53] uses colored De Bruijn graphs for *de novo* assembly[1] and

[1]De novo assembly is the assembly of a genome without the aid of a reference.

genotyping of variants across samples, intelligently using information across species to improve the analysis for each individual. It characterizes different types of variation (bubbles) within a population, but only on a per bubble basis, whereas an AFS may contain several bubbles that compose a larger, more interesting biological structure. Another tool, Sibelia [73] uses a heuristic approach to identify nested syntenic blocks within strains of the same species. Though Sibelia does not require a reference, it cannot scale beyond small populations of microbial genomes, and the results are likely to include spurious alignments due to the repeated modification of the input sequence to facilitate the progression of the algorithm [84].

2.3 Problem Definition

We define a *path* as a sequence of nodes in a graph. An approximate frequent subpath (AFS) is characterized by a subpath and a set of supporting paths that generally traverse the subpath. An approximate subpath is *frequent* if there are a sufficient number of paths supporting it. We assume the following input and parameters are supplied:

- A graph G and set of paths P within G. In this application, G corresponds to a compressed De Bruijn graph (CDBG) for a given pan-genome (consisting of a large set of genomic sequences). Each genomic sequence corresponds to a path p in G; P is the collection of these paths.

- Parameters: $\epsilon_r, \epsilon_c, \in [0, 1]$, $\mu_i \geq 0, \text{minsup} \in \mathbb{N}$

  The *row error threshold*, ϵ_r, controls how many nodes on a subpath are allowed to be missing from a supporting path. Similarly, the *column error threshold*, ϵ_c, controls how many supporting paths must support any one node in the AFS. The insertion error threshold, μ_i, controls how many nodes can be "inserted"

into a supporting path. A supporting path node is considered inserted if it does not belong to the AFS but lies between two nodes that do.

Suppose $p = \langle n_1, n_2, \ldots, n_L \rangle \in P$, where n_i is the ith node visited by p and L is the length of the path. We define the subpath $p[i, j] = \langle n_i, n_{i+1}, \ldots, n_j \rangle$. Two subpaths $p[i, j]$ and $p[i', j']$ are *index-disjoint* if and only if their index intervals are disjoint, that is, $[i, j] \cap [i', j'] = \emptyset$.

Definition 1. *An* approximate frequent subpath *(AFS) is a tuple* (p_a, S)*, where* p_a *is an* anchor path *in* G *and* S *is a set of supporting subpaths from paths in* P*.*

Note, p_a is just a path in G and need not be a subpath of any path in P. We define the requirements on S as as follows:

A subpath $p[i, j]$ *supports* an anchor path p_a if the following conditions are met:

$$|\text{match}(p_a, p[i, j])| \geq (1 - \epsilon_r) \cdot |p_a| \tag{2.1}$$

$$|p[i, j]| - |\text{match}(p_a, p[i, j])| \leq \mu_i \cdot |p_a|, \tag{2.2}$$

where

$$\text{match}(p_1, p_2) = p_1 \cap p_2,$$

treating p_1 and p_2 as multisets of nodes, so that repeated nodes are counted. Lastly, we require the subpaths in S to be index-disjoint and

$$|\{p[i, j] \in S : n \in \text{match}(p_a, p[i, j])\}| \geq (1 - \epsilon_c) \cdot |S|, \forall n \in p_a \tag{2.3}$$

$$|S| \geq \text{minsup}. \tag{2.4}$$

Here constraint (2.1) says that each subpath $p[i, j]$ must contain all but an ϵ_r-fraction of the anchor path's nodes and constraint (2.2) says that the number of

inserted nodes in the subpath (the LHS of (2.2)) is at most some multiple $\mu_i \geq 0$ of the length of the anchor path. Constraint (2.3) requires that each node in the anchor path is contained in all but an ϵ_c-fraction of the supporting subpaths in S and constraint (2.4) enforces that the number of supporting subpaths in S meets the minimum support requirement.

We define the *support* of an AFS as

$$\text{support}(p_a, S) = |S|. \tag{2.5}$$

Our goal is to find AFSs that have high support. As such, an important point to consider is that, unlike FIs, AFSs do not meet the Apriori Property [1], that is, subpaths of a valid AFS anchor path are not necessarily valid AFS anchor paths themselves.

Lemma 2.3.1. *Subpaths of a valid AFS anchor path are not necessarily valid AFS anchor paths themselves.*

Proof. This can be proven directly from the AFS problem definition. Observe that every supporting subpath of a valid anchor path need not support every node in the anchor path, but rather an ϵ_r fraction of the nodes. This means a supporting subpath may not necessarily support an ϵ_r fraction of the nodes in some subpath of the anchor path. Therefore, each supporting subpath of a valid anchor path may not necessarily support each of the anchor path's subpaths, implying the subpaths of the anchor path are not necessarily valid AFS anchor paths themselves. □

Figure 2.1 shows an example AFS.

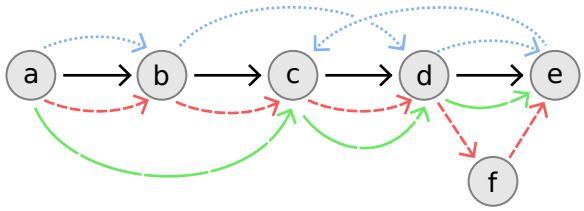

Figure 2.1: Example of an AFS: The anchor path is $p_a = \langle a, b, c, d, e \rangle$ and there are three potential supporting subpaths. If minsup $= 3$, $\epsilon_r = \frac{1}{5}$, $\epsilon_c = \frac{1}{3}$, and $\mu_i = \frac{1}{5}$, then p_a is a valid AFS. Conversely, if ϵ_r, ϵ_c, or μ_i are set to 0, then p_a is not a valid AFS.

2.3.1 Problem Complexity

Counting exact frequent itemsets is known to be #P-complete [101]. It can be seen that the exact frequent itemset problem can be reduced to the AFS problem by assuming G is complete and setting $\epsilon_r, \epsilon_c = 0, \mu_i = |G|$. Thus, just counting the number of AFSs is also #P-complete.

Furthermore, the problem of deciding whether there exists a set S of supporting paths for a given candidate anchor path p_a is NP-complete:

Lemma 2.3.2. *Deciding if there exists a set S of supporting paths for a given candidate anchor path p_a is NP-complete.*

Proof. The proof is reduction from SET-COVER. Given a SET-COVER instance, (X, C, k), where X is a set of elements and C is a collection of sets of elements, we want to know whether there is a cover $C' \subset C$ such that C' covers X and $|C'| \leq k$. Construct a graph G whose nodes are X plus an additional new node y. Let $p_a = \langle x_1, \ldots, x_n, y \rangle$. Let P be defined as follows: For each $A \in C$ construct a path p_A that visits the nodes in A (and only those nodes). Add an additional trivial path p_y that just visits node y. Let $\epsilon_r = 1, \epsilon_c = 1 - \frac{1}{k+1}, \mu_i = 0, \text{minsup} = 1$ and $\text{match}_{\text{set}}$ is the path matching function. Because of constraint (2.3), p_y must belong to any solution S. For $n = y$, the LHS of (2.3) is 1, no matter what other paths are added to S.

Thus, $1 \geq \frac{1}{k+1}|S|$, or equivalently, $|S| \leq k+1$. Finally, in order for (2.3) to hold for the other nodes of p_a, each node $x \in p_a$ must be visited by at least one path in S. Thus, there does not exist a set S of supporting paths for p_a in G unless there exists a set cover $C' \subset C$ such that $|C'| \leq k$.

Conversely, we observe that if anchor path p_a in G has a set S of supporting paths, then a set $C' \subset C$ can be constructed such that C' covers X and $|C'| \leq k$. This is because the previously defined mapping between the collection of sets C and the collection of paths $P \setminus \{p_y\}$ is a bijection, meaning for each path in $S \setminus \{p_y\}$ there is a set in C that corresponds to the same nodes in G. The collection of all such sets is C', so $|C'| = |S| - 1$. Since $\epsilon_c > 0$, each node in G exists in at least one path in S and, therefore, in at least one set in C', other than y, thus covering X. By construction, node y is only ever supported by path p_y, and path p_y always supports node y and no others. We have shown that this constraint yields the inequality $1 \geq \frac{1}{k+1}|S|$, or equivalently, $|S| \leq k+1$. Therefore, since $|C| = |S| - 1$, it follows that $|C'| \leq k$. $\square$

2.4 Algorithm

Our AFS-finding algorithm uses a bottom-up approach to determine candidate anchor paths of increasing length. This search is organized using a search tree T, with the root representing an empty path. The first level in T represents all paths of length 1, in other words, each node in the CDBG G is the starting point of some path. Each tree node v has branches to all neighboring nodes in G. Thus, a path from the root to v represents a path in G. We use a greedy strategy to explore T, using a priority queue Q to maintain a *frontier* of nodes whose children have not yet been explored. Nodes in Q are ordered by their support (2.5); so the most promising frontier nodes are explored first. Since the size of T is exponential, we limit the number of nodes

expanded to a user-specified maximum value and do not explore a node further if its support falls below minsup.

By Lemma 2.3.1, it is NP-hard to determine the set of supporting subpaths for a given anchor path p_a. In order to cope with this, we relax the problem to allow *fractional* supporting subpaths. The idea is to introduce a real-valued variable $x_{p[i,j]} \in [0,1]$ for each supporting subpath $p[i,j] \in S$. This formulation allows for an efficient approach to checking if a candidate anchor path forms an AFS using linear programming as outlined below:

1. Given a candidate p_a, for each path $p \in P$: compute a set of potential supporting subpaths

   $$C_p = \{p[i,j] : p[i,j] \text{ satisfies (2.1) and (2.2)}\}.$$

   This can be done efficiently using dynamic programming. Let $C = \cup_p C_p$ be all possible candidate supporting subpaths.

2. Create a $0,1$ matrix M that has $|p_a|$ columns. For each $p[i,j] \in C$, add a new row r to M using the following rule: if node $p_a[k] \in \text{match}(p_a, p[i,j])$ then put a 1 in column k, otherwise put a 0.

3. The problem is now to select a subset of the rows of M that corresponds to an index-disjoint collection of subpaths meeting constraints (2.3) and (2.4). The subsets $p[i,j]$ corresponding to these rows will form S. Observe that this problem can be formulated as an integer linear program involving variables $x_{p[i,j]}$ for all paths $p[i,j] \in C$. By allowing fractional support, this ILP becomes an LP that can be solved efficiently.

We note that the problem of finding candidate anchor paths remains hard. This because by Lemma 2.3.2 it is not always possible to find longer AFSs by extending the anchor paths of shorter AFSs by a fixed length.

2.4.1 Time complexity

As mentioned before, the size of search tree T is exponential relative to the size of the graph, so our algorithm has exponential run-time. By bounding the number of expansions that can be performed in the tree and using a relaxed ILP to compute sets of supporting paths, the algorithm can still be used to find interesting results on real data sets, as will be shown in Section 2.5.

2.5 Experimental Results

We implemented our algorithm as a Java program, using CPLEX[2] to solve the linear programs for computing fractional support and taking advantage of obvious opportunities for parallelization. Specifically, the exploration of different subpaths in the search tree T and the computation of supporting subpaths are independent of other such computations, enabling these tasks to be performed in parallel. All experiments we performed on an HP DL580 Symmetric multiprocessing (SMP) 64-core server with 1 TB RAM. The algorithm was able to identify interesting AFSs for the yeast data set in a few hours.

2.5.1 *Saccharomyces cerevisiae*

Saccharomyces cerevisiae (Yeast) is a well studied model system with some published pan-genomic work [8, 31]. With a genome size of approximately 12 Mb, it is tractable for algorithm testing. It is highly diverse, economically relevant, and

[2] `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

Table 2.1: Yeast Strains

Strain	Source	Region
AWRI796	Wine	South Africa
BC187	Wine	United States
CLIB215	Bakery	New Zealand
CLIB324	Bakery	Vietnam
DBVPG6044	Wine	West Africa
L1528	Wine	Chile
LalvinQA23	Wine	Portugal
Red Star	Bakery	United States
VL3	Wine	France
YS9	Bakery	Singapore

its multiple industrial applications enable interesting functional genomics. Our yeast pan-genome was built with assemblies from the *Saccharomyces* Genome Database[3]. The original dataset consisted of 55 assemblies from 48 yeast strains. For testing purposes, we created a subset from a wide geographical range. This subset consists of 10 strains, including 6 strains used in wine-making and 4 strains used in bread-making; see Table 2.1.

As a first pass, we searched for highly conserved AFSs. To do so, we used [4] to construct CDBGs at the nucleotide resolution with k-mer sizes in $\{25, 100, 500, 1000\}$. We chose these values because it is still not well understood what k values should be used for pan-genomic analysis. The parameters were arbitrarily selected to be minsup $= 2$ and $\epsilon_r = \epsilon_c = \mu_i = 0.10$. We mined copy number variants from the graph, specifically looking for regions that had expanded compared to other strains. Copy number expansions and contractions can have functional consequences, pointing to regions that are important in adaptation to the environment [30, 57, 99]. We looked in depth at the longest region - measured by the number of graph nodes - that had

[3] `http://www.yeastgenome.org/`

more than two copies in a single genome ($k = 1000$). In this case, there were three copies found in the Red Star assembly, which was derived from a commercial bakery yeast, and one copy found in the LalvinQA23 assembly, which is a wine yeast. We hypothesized that these regions would contain genes with important functions in bread making, given that its copy number is increased in Red Star. Each of the regions contained one full-length gene: *HFI1*. The HFI1 gene functions in chromatin modification, DNA damage repair and transcriptional regulation. This gene has, indeed, been found to be important in bread making. Specifically, it has a critical role in air-drying stress, which occurs during bread making [87] and is also required for anaerobic, but not aerobic, respiration. Anaerobic respiration is important in both bread making and alcohol generation [54].

The HFI1 AFS that we analyzed had representatives from only two species, yet the full-length HFI1 gene was found in 8 of the 10 yeast assemblies and a partial gene in an additional assembly. A Muscle multiple sequence analysis [33] and visualization in Jalview [16] (version 2.9.0b2) of the HFI1 gene shows only six variant sites of 1467 nucleotides; see Figure 2.2.

The reason that only four HFI1 genes from two strains were put in the AFS that we considered is likely due to the stringency of the k=1000 constraint, which grouped the four identical HFI1 genes but excluded the others, which differed at 1 or 2 bp for a total of 6 polymorphic sites among the strains over the 1467 bp of the HFI1 gene. The three Red Star and the LalvinQA23 HFI1 genes either represent very recent duplications, gene conversion, or transfer events, given their high sequence identity. Indeed, this is supported by a neighbor joining tree (generated within Jalview) for the HFI1 gene that shows no differentiation between these genes; see Figure 2.3. Though these two strains do not share an obvious recent ancestor, geography, or industry that would explain the presence of identical copies of the HFI1 gene, active

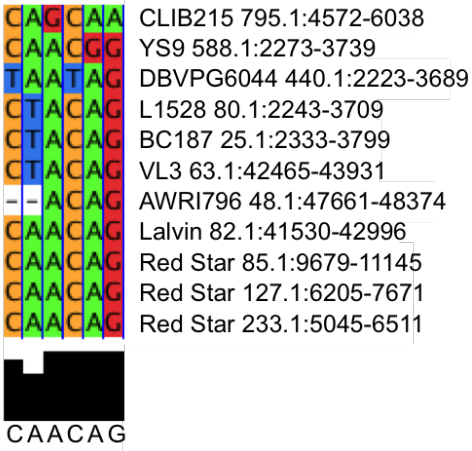

Figure 2.2: Muscle multiple sequence alignment of the HFI1 genes showing only the positions where differences occur. Yeast names are followed by scaffolds and nucleotide positions.

gene interchange has been suggested in yeast, especially in regard to genes with copy number differences between strains [31], though more work would be needed to explore this possibility.

We recognize that genome assemblies are not exact replicates of the genome from which they are derived because of sequencing bias and error, as well as artifacts in the assembly, such as chimeric regions, regions that are missing from the assembly, and repetitive and/or duplicated regions that are overcollapsed, overexpanded, or simply overcorrected to make the different copies appear more similar than they are. Particularly relevant here is that without experimental validation we cannot confirm that the three Red Star HFI1 genes in the assembly match those in the genome, nor that we are not missing additional copies (or assembling extra copies) from Red Star or other strains. Nevertheless, our goal here is to develop a pan-genomics algorithm using the yeast assemblies as a model system rather than focusing on yeast pan-genomics, per se. As such, even with the shortcomings of the assemblies, we have demonstrated that biologically relevant information can be gleaned by applying our

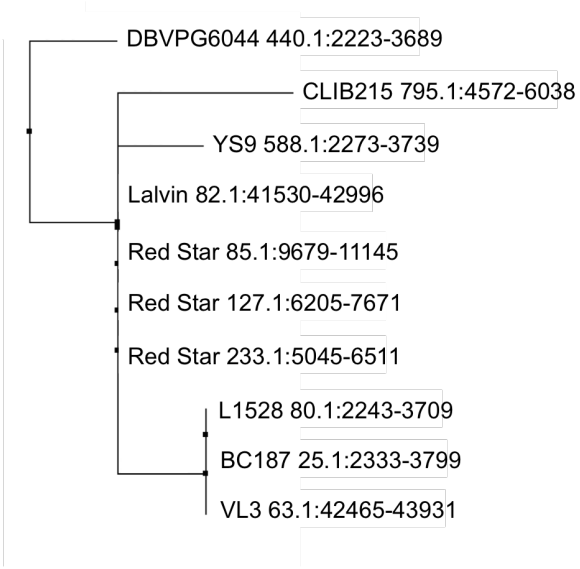

Figure 2.3: Phylogenetic tree of the full-length HFI1 genes.

algorithm to multiple genomes from a single species, generating interesting biological hypotheses that could be followed up with experimental work.

2.6 Conclusions

A strength of pan-genomics algorithms and their ability to analyze multiple related assemblies simultaneously without reference bias is the ability to find patterns both of divergence (novelty) and conservation. Identification of divergent regions (regions that do not fall into AFSs even with lenient parameters) allows the detection of novel genes that are not present in the reference sequence and/or other assemblies from a species. These genes could represent genes obtained through horizontal transfer, hybridization, or strong positive selection, and may have important adaptive functions. Identification of regions that are conserved, on the other hand, allows determination of core gene sets that are required for the species. Identification of unannotated regions that are conserved across the species is also

important. Conservation implies that purifying selection has been active to keep important regions conserved. Finding unannotated, conserved regions can lead to the identification of new genes or important regulatory elements.

Gene copy expansion and contraction, which we demonstrated above, can be an important mechanism of adaptation. Examining the AFS paths can differentiate between tandem duplications, which can be a response to environmental pressures that involved one or more genes, from duplications that resulted from segmental or whole genome duplications (polyploidy) that involve multiple genes. Identification of polyploidy-derived AFSs, whether auto-[4] or allopolyploidy[5], allows the detection of paralogous regions that can be flagged or merged, as desired. In addition, paralogous genes can be identified and their fate determined (gene loss, subfunctionalization, neofunctionalization). For allopolyploidy events, parental contribution and allelic dosage can be determined.

In this work, we are concerned with pan-genomes at the nucleotide level, but pan-genomics is a powerful approach that can be applied at other levels as well. For instance, our algorithm can be applied on the nucleotide level or the amino acid level. It can further be applied at domain, gene, gene family, operon, or molecule (chromosome or plasmid) level. This flexibility allows researchers to tailor the algorithm to their questions and organisms, and to apply the algorithm across different evolutionary scales.

In the following chapters, we will improve upon the AFS problem to handle the full yeast dataset, as well as a variety of legume plant species at the gene and gene family level.

[4]Having two or more sets of chromosomes.
[5]Having three or more sets of chromosomes.

2.7 Acknowledgements

The contents of this chapter was originally presented in [18, 19] and are reprinted here for the purpose of the author's educational theses.

CHAPTER THREE

FREQUENTED REGIONS

As in Chapter 2, here we will address the problem of mining synteny, and subsequently research Questions 2 through 4 from Chapter 1, with an emphasis on Question 2, by considering the problem of identifying regions within a pan-genome De Bruijn graph that are traversed by many sequence paths. We define such regions and the subpaths that traverse them as Frequented Regions (FRs). We formalize the FR problem and describe an efficient algorithm for finding FRs. Subsequently, we propose applications of FRs based on machine-learning and pan-genome graph simplification. We demonstrate the effectiveness of these applications using data sets for the organisms *Staphylococcus aureus* (bacteria) and *Saccharomyces cerevisiae* (Yeast). We corroborate the biological relevance of FRs by identifying introgressions in yeast that aid in alcohol tolerance, and show that FRs are useful for classification of yeast strains by industrial use and visualizing pan-genomic space.

3.1 Introduction

As discussed in Chapter 2, an important problem in population genomics is the identification of synteny, that is, sequences that are (in)exactly preserved within the population. Though effective, the AFS problem presented in Chapter 2 posed a few challenges to devising algorithms, namely the requirement that the graph structures to be identified are subpaths (linear sequences of graph nodes). This could, for example, cause an algorithm to identify each supporting path of a frequent subpath as the anchor path for a different but effectively equivalent frequent subpath. This is

a special case of the more general problem of two anchor paths that are practically identical, perhaps only differing by a single node, being identified as separate frequent subpaths. This causes redundancies in both computation and the result set. For these reasons, in this chapter we present the Frequented Regions (FR) problem, which aims to alleviate these issues by mining regions, rather than subpaths, of a graph that are frequently traversed by a set of paths. By exploiting the structure of the graph we are able to develop an efficient algorithm that effectively mines inexact syntenic regions from pan-genomic graphs. Note, as we will see in Chapter 5, the AFS problem is still relevant.

3.2 Related Work

Given the FR problem's relatedness to the AFS problem, the related work is quite similar. Regardless, the contents of this section are thorough for the sake of completeness.

The FR problem is somewhat similar to other data mining problems, especially Frequent Itemset Mining (FI), which identifies sets of items that frequently occur together in a database of transactions. Specifically, the database is a binary matrix where the columns correspond to items and the rows to transactions. If an item occurred in a transaction, then its cell has a 1, otherwise 0. An itemset is considered *frequent* if each of its items occurred together in *minsup* transactions, where minsup is either a fraction of the transactions in the database or a minimum number of transactions. Transactions in which a frequent itemset's items occur together are called *supporting transactions*, since they support the itemset as being frequent. A seminal FI algorithm, Apriori, was introduced in [1]. It works by constructing itemsets in a bottom-up, combinatorial manner and has an exponential run-time complexity. There are variations of FI that are tolerant to noise in the data [14, 67, 100]. These

require that the supporting transactions in the transaction matrix meet a *row error threshold* constraint and/or the items meet a *column error threshold* constraint. Like exact FI algorithms, these do not consider the structure of the underlying graph and so are likely to generate several false positives.

Also related is Frequent Subgraph Mining, which is concerned with finding subgraphs that frequently occur in a database of graphs [51]. It is a well studied problem [55] commonly applied to biological datasets [45, 47, 61]. It could be applied to paths through a graph by treating each path as a different graph, but little work has been done with regards to discovering approximate solutions or scalability [65]. Another related graph problem is Frequent Subpath Mining, which is described in [41]. The author shows that the problem of mining exact frequent subpaths is similar to FI, but differs in that the structure of the graph can be exploited to achieve more efficient running time. Unfortunately, the algorithm is incapable of mining frequent subpaths whose supporting paths contain some error. Furthermore, like Apriori, the algorithm works by constructing subpaths in a bottom-up, combinatorial manner and has exponential run-time complexity.

There exist a variety of tools for pan-genome analysis [95]. Unfortunately, few of these are concerned with mining synteny [64] or scale beyond populations of microbial genomes, both of which are needs of the pan-genomics community [72]. One exception to the lack of synteny-based approaches is Sibelia [73], which determines syntenic regions from pan-genomes by iteratively eliminating bubbles in a De Bruijn graph with the sequence modification algorithm [84]. Since all bubbles are eventually merged into a single syntenic region, regions that are truly divergent will be falsely identified as syntenic, requiring the user to manually differentiate between false and true positives - a tedious task. Furthermore, this method also does not scale beyond populations of microbial genomes, as noted in Section 3.6.

3.3 Problem Definition

We assume the following input and parameters are supplied: A graph G and set of paths P within G. In our application, G corresponds to a CDBG of a pan-genome composed of multiple genomic sequences, where each sequence corresponds to a path p in G; P is the collection of these paths.

A *frequented region* (FR) is characterized by a set of nodes C and a set of supporting subpaths from P that pass through the nodes in C. There are two error parameters that we consider: 1) what is the minimum fraction of the nodes in C that each subpath must contain; we call this the *penetrance* parameter α, and 2) if a subpath from P leaves C, how soon must it return to C (measured by the length of the corresponding sequence insertion); we call this the *maximum insertion* parameter κ. With this in mind, we formalize the definition of an FR as follows:

Given a path $p \in P$, let $p = \langle n_1, n_2, \ldots, n_L \rangle$, where n_i is the ith node visited by p and L is the length of the path. We define a subpath as $p[i,j] = \langle n_i, n_{i+1}, \ldots, n_j \rangle$ and seq$(p[i,j])$ as the genomic sequence corresponding to $p[i,j]$ G.

Definition 2. *We say $p[i,j]$ is an (α,κ)-supporting subpath for a set of nodes C if and only if*

1. *$n_i, n_j \in C$ and between any two consecutive C nodes in $p[i,j]$, any gap of inserted sequence is at most κ in length; see (3.3) for a computational definition.*
2. *$p[i,j]$ is* maximal *in the sense that it cannot be extended to either the left or right, or rather, $\nexists$ (α,κ)-supporting subpath $p[i',j']$ s.t. $[i,j] \subsetneq [i',j']$.*
3. *$|p[i,j] \cap C| \geq \alpha|C|$.*

Note that (α,κ)-supporting subpaths do not overlap due to the maximality requirement; if they did overlap they could be merged. It is also fairly easy to

identify all of the (α, κ)-supporting subpaths for a given path p and node set C; we just need to identify all maximal runs of C nodes in p whose corresponding sequences have insertions of length at most κ and then check if the run contains at least $\alpha|C|$ distinct nodes from C. Algorithm 7 in Section 3.4 implements this idea.

Definition 3. *A* frequented region *(FR) is a tuple* (C, S)*, where* C *is a set of CDBG nodes and* S *is a set of* (α, κ)*-supporting subpaths of paths from* P*.*

We say C is the *node-set* and S is the *supporting-subpath-set* for the FR. We define the *support* of the FR as

$$\text{support}(C, S) = |S| \tag{3.1}$$

and also define the *average length* of the FR as

$$\text{average-length(FR)} = \frac{\sum_{p[i,j] \in S} |\text{seq}(p[i, j])|}{|S|} \tag{3.2}$$

Figure 3.1 provides an example.

The computational problem considered is to find FRs that have high support and high average length.

3.3.1 Problem complexity

Counting exact frequent itemsets is known to be #P-complete [101]. It can be seen that the exact frequent itemset problem can be reduced to the FR problem by setting $\alpha = 1, \kappa = \infty$. Thus, just counting the number of FRs is also #P-complete.

Unlike the AFS problem in Chapter 2, deciding if there exists a set S of supporting paths for a given candidate node-set C is tractable. This is because there is no parameter that constrains the number of paths that must support each

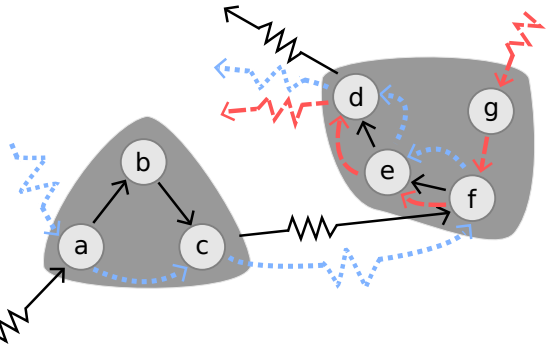

Figure 3.1: Example FRs: Assuming $\alpha \leq \frac{2}{3}$, the left group of nodes $C_L = \{a, b, c\}$ forms an FR with support 2 (from the black (solid) and blue (dotted) paths). The right group of nodes $C_R = \{d, e, f, g\}$ forms an FR with support 3 (from the black, blue, and red (dashed) paths). If C_L and C_R were merged, the merged FR would have support 2, provided the connecting black and blue path segments between nodes c and f each have at most κ insertions.

node in C - ϵ_c in the case of the AFS problem. In Section 3.4, we present an optimal algorithm for computing such sets of supporting paths.

3.4 Algorithm

Due to the vast scale of pan-genomic data, we opted for a simple heuristic approach to find interesting FRs. The basic idea of the algorithm is to find FRs in a agglomerative (bottom-up) fashion, where each CDBG node starts in its own cluster and pairs of clusters are repeatedly merged. When an FR is created for a set of nodes C, the corresponding set of supporting subpaths S must be found. This is accomplished by the ComputeSupport subroutine shown as Algorithm 3.1. Note, in line 7, the `gap` subroutine returns the length of the inserted sequence between the consecutive matching node $m[i]$ and $m[i+1]$ along the path p; each end of

Algorithm 3.1: Compute supporting path segments

```
1: function ComputeSupport(C,p)
2:     Let S = ∅
3:     Let m = [i : p[i] ∈ C]
4:     start = 1
5:     while start ≤ |m| do
6:         i = start
7:         while i < |m|, gap(p, m, i) ≤ κ do
8:             i = i + 1
9:         end while
10:        if (i − start + 1) ≥ α|C| then
11:            S = S ∪ p[m[start], m[i]]
12:        end if
13:        start = i
14:    end while
15:    return S
16: end function
```

$p[m[i], m[i+1]]$ matches a node in C, so the insert gap length is

$$\text{gap}(p, m, i) \;=\; |\text{seq}(p[m[i], m[i+1]])| - |\text{seq}(p[m[i]])| - |\text{seq}(p[m[i+1]])|. \quad (3.3)$$

The idea of the algorithm is to find a pair of existing FRs and merge them such that the newly created FR has the greatest possible support. Specifically, the result of every possible merge (one for each inter-cluster edge) is computed in turn with Algorithm 3.2, which unions the node sets of the two FRs being merged, C_L and C_R, and then computes the corresponding set of supporting subpaths S with Algorithm 3.1. The merge that would yield the FR with the highest support is then performed. The procedure terminates when no merge will yield an FR with positive support. Pseudocode of the procedure is provided in Algorithm 3.3. Line 10 can be computed efficiently using a priority queue of possible merges. When a new FR is formed by merging (C_L, S_L) and (C_R, S_R), other potential merges involving either of

Algorithm 3.2: Evaluate FR merge

```
1: function EvalMerge((C_L, S_L), (C_R, S_R))
2:     Let C = C_L ∪ C_R.
3:     Let S = ∅.
4:     for p ∈ P do
5:         S = S∪ ComputeSupport(C, p)
6:     end for
7:     return S
8: end function
```

these FRs remaining in the queue must be updated to be potential merges with the newly formed FR. Note that S_L and S_R are not actually used by Algorithm 3.2 since a new supporting set must be computed in order to capture supporting subpaths that may not have been present in S_L or S_R.

After Algorithm 3.3 completes, the FRs will form a *hierarchical clustering* with one or more root FRs, that is, FRs that are not contained within another FR. The final step of the algorithm is to filter the FRs with high support. A simple recursive procedure is used to do this, as outlined in Algorithm 3.4. For each root FR (C, S) found, we call Explore$((C, S, 1))$. This will recursively explore the tree in a depth-first manner and report the FR associated with a tree node t if and only if t's support is greater than any of its ancestors (superset FRs). All such reported FRs are considered *interesting* FRs (iFRs). We limit the analyses in Section 3.6 to such FRs.

3.4.1 Time complexity

Suppose the CDBG contains V vertices and E edges and let L be the total length of all the pan-genomic paths in P. Algorithm 3.3 is the main program; it begins with creating a new FR for each node $n \in G$ and finds its support. This can be done in $O(V + L)$ time. Next, we note that there are at most $V - 1$ iterations of the repeat-until loop since each iteration creates an internal node in a binary tree with V

Algorithm 3.3: Agglomerate FRs

1: **procedure** MERGEFRS(G, P)
2: **for** $n \in \text{nodes}(G)$ **do**
3: Let $S = \emptyset$.
4: **for** $p \in P$ **do**
5: $S = S \cup$ COMPUTESUPPORT($\{n\}, p$)
6: **end for**
7: Create FR $(\{n\}, S)$.
8: **end for**
9: **repeat**
10: Compute

$$(C_L, S_L), (C_R, S_R) = \underset{(C_L,S_L),(C_R,S_R)}{\text{argmax}} |\text{EVALMERGE}((C_L, S_L), (C_R, S_R))|$$

11: Let $C = C_L \cup C_R$.
12: Let $S = \text{EVALMERGE}((C_L, S_L), (C_R, S_R))$
13: Create FR (C, S)
14: Mark (C_L, S_L), (C_R, S_R) unavailable for subsq. mergers
15: **until** EVALMERGE$((C_L, S_L), (C_R, S_R)) = \emptyset$
16: **end procedure**

leaves. Determining the next FR merger (internal node) can be done efficiently using a priority queue data structure. After each FR merger happens, the queue needs to be updated. This can done in $O(V + L + V \lg V)$ time; the $O(V + L)$ accounts for computing the support of all new potential FR mergers with the newly-created FR (there are at most V of them) and the $O(V \lg V)$ accounts for the resulting priority queue updates. The overall running time is thus, $O(LV + V^2 \lg V)$.

3.5 Applications

In this section we discuss some applications for FRs including machine learning methods and an approach for pan-genome graph simplification and visualization.

Algorithm 3.4: Report interesting FRs

1: **procedure** EXPLORE($(C, S), m$)
2: **if** support$(C, S) \geq m$ **then**
3: **report** (C, S).
4: **end if**
5: Suppose $(C, S) = \text{merge}(C_L, S_L, C_R, S_R)$
6: Let $m' = \max(m, \text{support}(C, S) + 1)$
7: EXPLORE($(C_L, S_L), m'$)
8: EXPLORE($(C_R, S_R), m'$)
9: **end procedure**

3.5.1 Machine learning with FRs

Like other sequence features, such as SNPs, FRs may provide sequence characteristics that can be used to distinguish or categorize groups of genomes. For example, in Section 3.6 we differentiate between yeast strain genomes based on their industrial origin. In order to evaluate the effectiveness of FRs for this task we model, this as a *multi-class classification* problem in which each strain is annotated with one of the industrial-origin class labels and each iFR is a feature. The problem is then to apply a *supervised learning* algorithm to the feature set, or a subset of the feature set, such that unlabeled strains are labeled with the correct class.

Support Vector Machines (SVMs) [24] have been shown to be an effective approach to classifying genomes based on shared genetic features. Traditionally such classifications are done with Genome Wide Association Studies (GWAS) and/or Principal Component Analysis (PCA) [86], however, SVMs have been shown to have more classification power than GWAS/PCA [10]. Therefore, in this work we use Support Vector Machines for classifying genomes within a pan-genome based on their FR content as described below.

Each example (genome) is represented as an n-dimensional vector (n is the total number of FRs used) in which each individual component of the vector corresponds

to the number of times a certain FR occurs within that genome. We use these vectors as input to our SVM model [78] and evaluate its accuracy in predicting the correct industrial origin based on the FRs they are associated with (see Section 3.6.2.2 for SVM configuration details and for results of this study). However, depending on the parameters used and the size and complexity of the pan-genome, a large number of iFRs may be identified by Algorithm 3.4. These may span a variety of classes, ranging from the pan to strain-specific, among others. As such, *feature selection* is an important task when trying to maximize classification power.

An effective feature selection strategy is one that can identify a small number of features that can discriminate classes based on their attributes. A simple approach based on multinomial distributions which we refer to as *multinomial-filter* is as follows: Suppose that the sequence paths are divided into a set of groups $\{G_1, \ldots, G_k\}$. Let

$$c_{ij} = \sum_{p \in G_i} \text{support}_p(\text{FR}_j), \tag{3.4}$$

be the total support of FR_j for all sequence paths belonging to group G_i. In other words, c_{ij} is a count of the number of times FR_j occurs in G_i. Let $T_i = \sum_j c_{ij}$ be the total count of iFRs found in group G_i and let $T = \sum_i T_i$ be the total across all groups. The frequency with which a random iFR occurs in G_i is then $f_i = {}^{T_i + 1}/_{T}$ (a pseudo-count of 1 is added to the count for each group). The probability of observing the group counts for FR_j in a random FR is multinomially distributed with bin probabilities $\langle f_i \rangle$

$$\Pr(\text{FR}_j) = \frac{n!}{c_{1j}! \cdots c_{kj}!} f_1^{c_{1j}} \cdots f_k^{c_{kj}} \tag{3.5}$$

where $n = \sum_i c_{ij}$. The lower the $\Pr(\text{FR}_j)$ value, the less likely that the observed group counts for that FR occurred by random chance. We note that $\Pr(\text{FR}_j)$ is

similar but not identical to a p-value, as the latter includes the probability of at least as extreme data under the null hypothesis.

3.5.2 Graph simplification

Visualizing a pan-genome graph is a difficult task, especially when dealing with larger, complex genomes. It is relatively simple to visualize pan-genomes at the microbial level and with a limited number of genomes, but when the number of genomes and size increases, the result is an indecipherable "hairball", as depicted in the SplitMEM [71] work. To that end, we would like to use iFRs to create visualizations of pan-genomes that are human parsable, meaningful, and facilitate knowledge discovery.

One approach is to filter what contents of a pan-genome are visualized by selecting a group of FRs and restrict attention to how the pan-genome traverses the corresponding FR node clusters. If we are given a list $L = \{C_1, \ldots C_n\}$ of disjoint FR clusters, we can create a corresponding graph G_L with n vertices, where each vertex represents one of the FR clusters in L. For each sequence path p in the original CDBG, we can trace a path in G_L according to the order in which p supports the clusters in L (see Section 3.6.2.1 for results of this study).

3.6 Experimental Results

The FR finding algorithm was implemented as a Java program called *FindFRs*, which first runs Algorithm 3.3 to construct the FR hierarchy and then runs Algorithm 3.4 to report iFRs. Our implementation exploits obvious opportunities for parallelization, specifically, the for loop in Algorithm 3.2. Besides the main parameters α and κ, we also include two other parameters *minSup* and *minSize* which

require the iFRs output have enough supporting paths and are large enough (number of CDBG nodes), respectively.

We evaluated FindFRs on two datasets: *Staphylococcus aureus* and *Saccharomyces cerevisiae*. The *Staphylococcus aureus* dataset was used to compare against Sibelia [73], the only other program currently available for mining synteny from a pan-genome De Bruijn graph. The *Saccharomyces cerevisiae* dataset was used to illustrate the scalability of the FR-finding algorithm and exhibit a variety of FR-based analyses. We used [4] to construct the CDBGs for each dataset. Experiments were run on a server with 4 Intel Xeon 2.2GHz 32 core processors and 1 TB of RAM, however most data sets could also run on standard desktop PCs, with longer running times.

3.6.1 *Staphylococcus aureus*

In this section we compare FindFRs to Sibelia [73]. In our first experiment we compared the similarity of the results produced by both programs, using the four *Staphylococcus aureus* strains that were used as an example in the original Sibelia paper: JH1, N315, TW20, and MSSA476. Both programs were run with k-mer values in $\{25, 100, 500, 1000\}$. As in Chapter 2, we chose these values because it is not well understood what k values are appropriate for pan-genome analysis. Table 3.1 shows that FindFRs reports a higher number of frequented regions compared to Sibelia's synteny blocks, indicating the finer scale of partitioning of regions based on biological significance. Additionally, we estimated percent overlap between regions reported by each algorithm: For all k-mer values, approximately 98% of the FRs reported by FindFRs overlapped with Sibelia's synteny blocks. We also computed overlap in terms of sequence percentage, or rather, what percentage of the Sibelia block sequences are contained within FindFRs FRs, and vice versa. For the $k = 25$

Table 3.1: Sibelia versus FindFRs comparison. FindFRs: $\alpha = 0.5$, $\kappa = 0$, minSup = 2, minSize = 2. Sibelia: All set to default, except for k values.

S. aureus (4 Strains)	$k = 25$	$k = 100$	$k = 500$	$k = 1000$
Synteny blocks	142	143	132	132
iFRs	10,740	1,334	2,011	1,351
iFR roots	392	122	97	134

CDBG, alignments showed 91% of Sibelia synteny block sequence base pairs were contained within FindFRs FRs and only 82% of FindFRs FR sequence base pairs were contained within Sibelia synteny blocks, indicating FindFRs FRs captured most of Sibelia synteny blocks at the nucleotide base pair level.

In our second experiment, we compared the run times of both programs. This was done with a slightly larger dataset consisting of 31 *Staphylococcus aureus* (≈ 90 Mb) strains. For $k = 25$, Sibelia took 186 minutes, whereas FindFRs took 43 minutes, approximately four times faster.

3.6.2 *Saccharomyces cerevisiae*

Saccharomyces cerevisiae (Yeast) is a well studied model system with some published comparative genomic and pan-genomic work [8, 31], allowing us to use the existing knowledge base to assess the biological quality of the FRs found by our algorithm. Furthermore, yeast is highly diverse, economically important, and its multiple industrial applications enable interesting functional genomics. These attributes, compounded with a genome size of approximately 12 Mb, make yeast an interesting and tractable dataset for algorithm testing.

Our yeast pan-genome was built with assemblies from the *Saccharomyces* Genome Database[1]. The dataset consisted of 55 assemblies from 48 yeast strains

[1] `http://www.yeastgenome.org/`

Table 3.2: The 48 yeast strains with usage (source) and region listed where known.

Strain	Source	Region	Strain	Source	Region
AWRI1631	Wine	South Africa	M22	Fruit/Nectar	Italy
AWRI796	Wine	South Africa	PW5	Wine	Nigeria
BC187	Wine	California	Red Star	Bakery	
BY4741	Laboratory		RM11-1a	Fruit/Nectar	California
BY4742	Laboratory		SEY6210	Laboratory	
CBS7960	Bioethanol	Brazil	SK1	Laboratory	
CEN.PK	Laboratory		T7	Nature	Missouri
CLIB215	Bakery	New Zealand	T73	Wine	Spain
CLIB324	Bakery	Vietnam	UC5	Sake	Japan
CLIB382	Ale	Ireland	UWOPS05_217_3	Fruit/Nectar	Malaysia
D273-10B	Laboratory		VIN13	Wine	South Africa
DBVPG6044	Wine	West African	VL3	Wine	France
EC1118	Wine		W303	Laboratory	
EC9-8	Nature	Israel	X2180-1A	Laboratory	
FL100	Laboratory		Y10	Fruit/Nectar	Coconut
Foster's B	Ale		Y55	Laboratory	
Foster's O	Ale		YJM269	Wine	Austria
FY1679	Laboratory		YJM339	Pathogen	
JAY291	Bioethanol	Brazil	YJM789	Pathogen	
JK9-3d	Laboratory		YPH499	Laboratory	
K11	Sake	Japan	YPS128	Nature	Pennsylvania
Kyokai7	Sake	Japan	YPS163	Nature	Pennsylvania
L1528	Wine	Chile	YS9	Bakery	Singapore
LalvinQA23	Wine	Portugal	ZTW1	Bioethanol	China

over a wide range of industrial applications and geographic origins; see Table 3.2. Again, since k-mer size can affect which genomic features are explicitly represented in the topology of the graph and it is not yet well understood how to choose an appropriate k-mer size for pan-genome analysis, we tested $k \in \{25, 100, 500, 1000\}$. We also varied α and κ, as indicated. Running times ranged from a few minutes for $k = 1000$ to approximately an hour for $k = 25$. Table 3.3 indicates the size of the CDBGs created and the number of iFRs found. Figure 3.2 shows support versus average length for a sample run. We note that Sibelia was not able to process this data set (≈ 600 Mb), so we do not report a comparison.

3.6.2.1 Consistency with Yeast biology

Because yeast is one of the simplest eukaryotes, it has been an important model system, including for genomic research [8, 31]. In addition to being an important laboratory model, yeast is economically important, as it is critical in baking bread,

Table 3.3: Number of CDBG nodes and iFRs found by FindFRs with parameters $\alpha = .0.7$, $\kappa = 0$, minSup $= 5$, and minSize $= 5$.

k	CDBG nodes	iFRs
25	$2,260,767$	$115,585$
100	$1,758,760$	$97,550$
500	$890,055$	$29,766$
1000	$443,764$	$8,994$

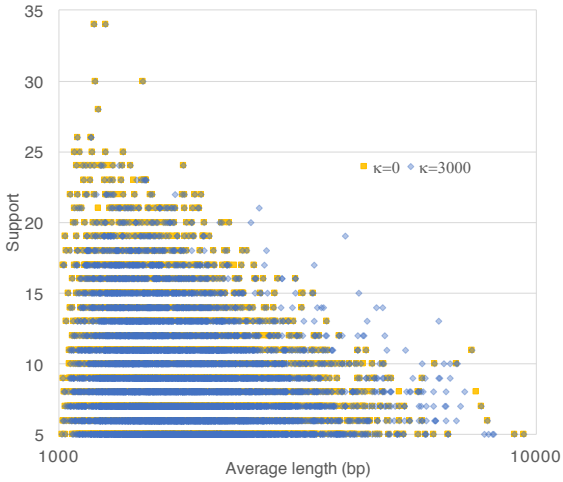

Figure 3.2: Distribution of iFR support versus average length for FRs mined with parameter values $\alpha = 0.7$, $\kappa \in \{0, 3000\}$, minSup $= 5$, and minSize $= 5$. As can be seen, allowing insertions ($\kappa = 3000$) creates some longer FRs.

generating alcoholic beverages, such as wine, sake, and ale, and is also used in generating biofuels. Yeast is also important in other contexts, being found in natural environmental systems, including as pathogens of humans and other organisms.

The ability to thrive in harsh environments, which is required for industrial specialization in yeast, is often obtained through horizontal transfer of sets of genes, or *introgressions* [8, 31]. To determine whether we could uncover some of these introgressed regions, we looked at introgressions that have been validated in EC1118. These introgressions introduce genes that allow EC1118, which is used in wine-making, to be able to grow in the presence of alcohol. EC1118 has three introgressions compared to S288C [80]. Because S288C is not in our dataset, we used FR nodes contained in BY4741 (derived from S288C) to compare to those found in EC1118. All three introgressions were uncovered using our FR approach; see Figure 3.3.

The shortest introgression is a 17 kb introgression on chromosome 14 [80]. Five novel genes are inserted near the telomere between S288C genes YNL037C and YNL038W. The introgression likely comes from the species *Zygosaccharomyces bailii*, with the possible exception of EC1118_1N26_0034g, which could not be found in *Z. bailii* and has counterparts in the *Saccharomyces* genus. The order of the genes is slightly different in *Z. bailii*, however, with EC1118_1N26_0056g occuring before EC1118_1N26_0012g in *Z. bailii*.

The 17 kb introgressed region was found on EC1118 sequence accession FN393084.1 ($k = 500$, $\alpha = 0.7$). Although this sequence was only approximately 25 kb, it contained the introgression and anchoring sequence on both sides; see Figure 3.3a. The order of the green nodes reflects their order in EC1118. Multiple alcohol-related strains have versions of this introgression, as indicated by the thicker green edges as well as green edges that follow alternate paths through the region. Interestingly, some of the alcohol-related strains appear to have a

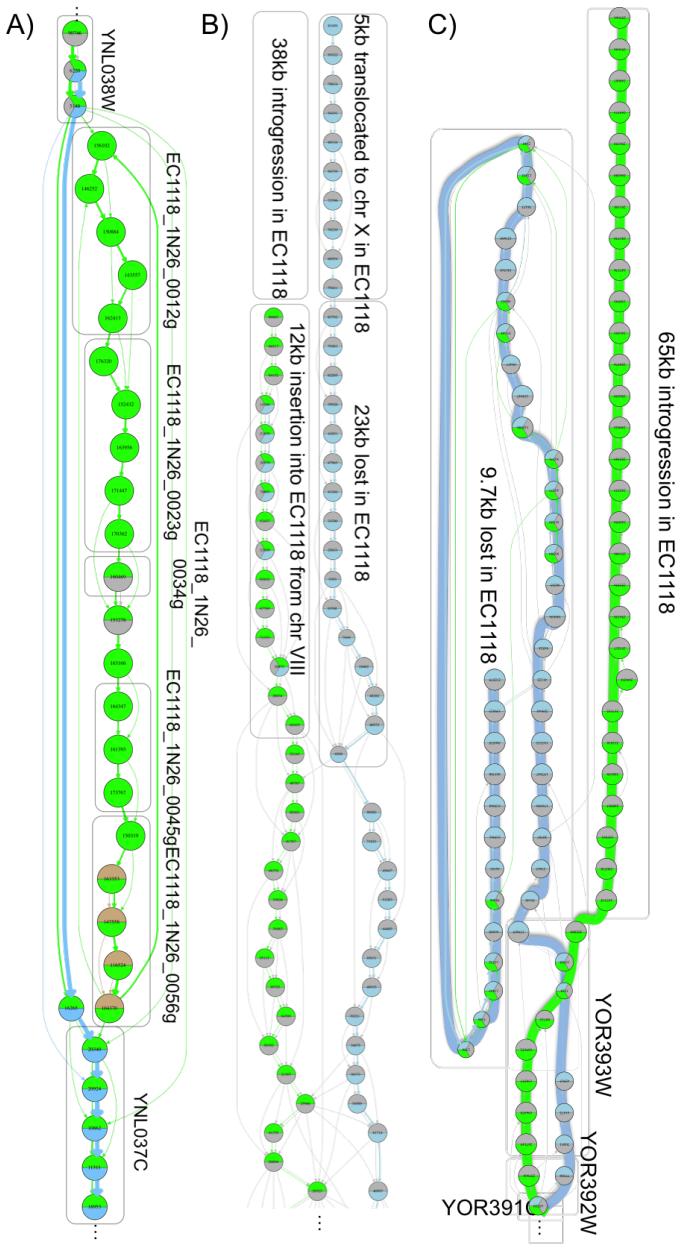

Figure 3.3: The three introgression regions [80] showing FRs from EC1118 (green) compared to S288C (represented by the S288C-derived strain BY4741, light blue). (A) 17 kb introgression on chromosome 14 showing alcohol (green, including wine, sake, ale and bioethanol strains), laboratory (light blue), bakery (brown), and other (gray) strains. (B) A complex introgression event on chromosome 6 shows a 12 kb translocation in EC1118 from chromosome 8, a 23 kb deletion from in EC1118 and a 5 kb region from S288C that is translocated to chromosome 10 in EC1118. Strains included were EC1118 (green), S288C-derived (light blue, strains BY4741, BY4742, FY1679, X2180-1A, and YPH499), and other (gray). (C) A 65 kb novel introgression replaced 9.7 kb of the chromosome 15 telomere in EC1118. Shown are EC1118 (green) and BY4741 (light-blue).

rearrangement matching that of *Z. bailii* with EC1118_1N26_00056g leading into EC1118_1N26_00012g. None of the laboratory strains, including the five S288C-derived strains, show support for any of the introgressed FRs.

Most of the nodes in the introgression occur only in alcohol-related strains (green in Figure 3.3a). This confirms that the region could be important in determining alcohol tolerance. The two nodes which show "other" (gray) strain support include the EC1118_1N26_00034g gene which is not thought to be from *Z. bailii*, but rather, to be from a congeneric strain. [80] found a related gene in RM11-1A while we found it in our dataset in M22 and alcohol-related strains. Both RM11-1A and M22 were isolated from vineyards and might need some alcohol tolerance to survive fermenting fruit, though it is also possible that the gene is unrelated to alcohol tolerance.

The four introgression FRs adjacent to YNL037C, which are part of the EC1118_1N26_00056g gene, appear to be contained in bakery yeasts (brown) as well as alcohol yeasts (green in Figure 3.3a). The EC1118_1N26_00056g gene is involved in activating the proline utilization pathway, though whether this foreign version of the gene increases proline utilization or competes with or replaces the native gene resulting in reduced proline utilization is not clear. Nevertheless, proline is important in both the alcohol and breadmaking industries. Proline can provide a nitrogen source and is the dominant free amino acid in grapes and grape wine [48]. In addition, proline accumulation can confer tolerance to freezing, dessication, and ethanol stress [91, 92, 93]. It is interesting to note that the two bakery yeasts sharing these FRs, CLIB324 (Vietnam) and YS9 (Singapore), are geographically close and likely come from a geographically isolated lineage, which are common in Yeast [66].

The 38 kb EC1118 insertion on one of the telomeres of chromosome 6 [80] was also uncovered in EC1118 ($k = 500$, $\alpha = 0.7$, minSup = 4, minSize = 4). This is a more complicated introgression with some rearrangements also occurring. A 23 kb

segment of chromosome 6 (genes YFL052W to YFL058W) was deleted in EC1118 while an adjacent segment (genes YFL062W-YFL059W) was translocated to one of EC1118's chromosome 10 telomeres. The 38 kb novel insertion merged onto the end of the chromosome, connecting to a 12 kb segment translocated from EC1118's chromosome 8 (genes YHR217C through YHR211W). This segment, in turn, merged with the remaining sequence from chromosome 6, with YHR211W (chromosome 8) and YFL051C (chromosome 6), which are similar on a sequence level, fusing together.

Many of the EC1118 FRs that were translocated to chromosome 6 from chromosome 8 (12 kb) are shared by the S288C-derived accessions (light blue). This is not surprising because the S288C-derived accessions have this region on chromosome 8. As expected, none of the blue edges (S288C-derived) connect the 12 kb chromosome 8 translocation with the rest of chromosome 8. None of the FRs in the 23 kb segment lost in EC1118 show support in EC1118, as expected given that it has been deleted in EC1118. However, none of the FRs in the 5 kb region that was translocated to chromosome 10 in EC1118 are contained in EC1118, either, even though this region occurs on chromosome 10. Presumably, it has evolved away from the S288C version to make it different enough that our parameters did not link the two regions into the same set of nodes. The EC1118 FRs stop approximately 38 kb from the end of chromosome 6. No FRs were found in the 38 kb novel region, presumably because there wasn't enough support to generate FRs. This raises the issue that sometimes it is interesting to look for regions where FRs are absent as they are potentially novel.

The final novel introgression is a 65 kb region that replaced the last 9.7 kb (including genes YOR394-C-A (2 copies), YOR394C, and YOR396W, not shown) of chromosome 15 [80]. For simplicity, only EC1118 (green) and BY4741 (light blue) compared to all others (gray) are shown in Figure 3.3c and their chromosome 15 paths have been highlighted ($k = 500$, $\alpha = 0.7$). The last gene shared by EC1118

Table 3.4: Best AUROC curve for each k-value.

k	support/size-filter	multinomial-filter
25	0.68	0.75
100	0.75	0.79
500	0.72	0.91
1000	0.72	0.9

and BY4741, YOR393W, has some shared FRs and some FRs having diverged. Thereafter, the paths diverge. The EC1118 65 kb novel insertion is shared with other yeast strains. This region is highly conserved as there are not alternate paths through the region. The 9.7 kb region in BY4741, which was deleted in EC1118, actually has FRs that are shared with EC1118, though EC1118 does not have them on chromosome 15. These are telomeric genes that tend to occur on several yeast telomers. This region is shared with other strains as well. It contains many alternate paths through the region and, though smaller, actually contains more (and smaller) FRs than EC1118's 65 kb introgression. The alternate paths and small FRs indicate divergence and rearrangement between strains.

3.6.2.2 Using FRs for classification

As mentioned in Section 3.5.1, we use SVMs as our supervised learning model. Specifically, we applied a one-against-rest multi-class SVM classifier with degree 2 polynomial kernels. We normalized the data using L1 normalization and then used stratified cross-validation to evaluate the effectiveness of the model. In this work, we used this SVM configuration to predict the industrial origin for a given yeast strain.

We implemented the SVM model with the above configuration using the PyML Python library[2]. We then used the model to classify the strains of the yeast dataset by their industrial application, or rather *source* from Table 3.2. The dataset is composed of 55 examples annotated with nine distinct class labels. Given the distribution of the source labels in the data set, we used two-fold stratified cross-validation. For $\alpha \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$ and $\kappa = 0$, we computed the average Area Under Receiver Operating Characteristic (AUROC) curve [7] for 10 iterations of classifier training.

Due to the large number of iFRs (see Table 3.3), we found it necessary to employ feature-selection to reduce the number of iFRs considered by the classifier in order to improve classification power. This was done using two separate methods: 1) iFRs were filtered with minSup $\in \{1, 5, 10, 15, 20\}$ and minSize $\in \{1, 5, 10, 15, 25, 100\}$, which we refer to as the *support/size-filter*, and 2) iFRs were filtered with the *multinomial-filter* feature selection method described in Section 3.5.1, keeping the top 1000, 500, and 250 iFRs based on $\Pr(\mathrm{FR}_j)$-value rank. We chose these feature set sizes to explore if there is an appropriate size for such a classification task. Table 3.4 reports the highest AUROC curve for each k-value and filtering method.

As can be seen, both methods of FR filtration provide feature sets that are effective for classifying yeast by their industrial application, though the multinomial-filter is more effective. It is worth noting that there does not appear to be a specific iFR sample size that yields higher AUROC curve values, so Recursive Feature Selection should be employed to maximize classification power. Note, this can be much more computationally demanding than trying a variety of sizes as we have done. We have omitted classification results for the unfiltered iFR sets because their AUROC curve values were seldom better than guessing.

[2] `http://pyml.sourceforge.net/`

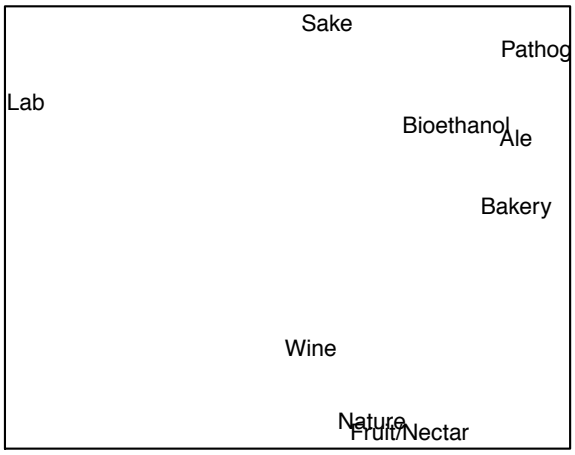

Figure 3.4: Yeast industrial usage multidimensional-scaling plot based on the top 500 discriminative iFRs found with parameter values $k = 500$, $\alpha = 0.7$, and $\kappa = 0$

3.6.2.3 Visualizing pan-genomic space

We also applied *multidimensional-scaling* [25], using the `isoMDS` method in R, to provide a visual representation of the similarity of the industrial uses for yeast based on their iFR content. The ranking method described in Section 3.5.1 was first used to determine the top 500 iFRs by p-value (eqn. 3.5) for discriminating the nine industrial uses from Table 3.2. Distances between usages were found using the Canberra method, an appropriate metric for count-based data, provided by R's `dist` function. The resulting plot, shown in Figure 3.4, provides a visual interpretation of yeast usages based entirely on the aggregate genomic content of each group. As can be seen, the plot shows that certain uses are quite similar, such as Nature and Fruit/Nectar, and Bioethanol and Ale. Conversely, it shows the dissimilarity between uses, most notably Laboratory versus all others.

3.7 Conclusions

Frequented regions provide new approaches to analyze pan-genomic space. While we have described a few examples showing biological relevance above, there are many other biological problems to which our pan-genomic method can be applied [72].

FRs permit analyzing multiple related assemblies simultaneously without reference bias to find patterns both of divergence (novelty) and conservation. Identification of divergent regions (regions that do not fall into FRs even with lenient parameters) allows the detection of novel genes that are not present in the reference sequence and/or in other assemblies from a species. These genes could represent genes obtained through horizontal transfer, hybridization, or strong positive selection, and may have important adaptive functions. On the other hand, identification of regions that are conserved enables the determination of core gene sets that are required for the species. Determining unannotated regions that are conserved across the species is also important. Such conservation could imply that purifying selection has been active to keep important regions conserved. Such regions could also lead to the identification of new genes or important regulatory elements.

Path-based approaches could also be applied at the amino acid level and potentially at the domain, gene, gene family, operon, or molecule (chromosome or plasmid) level. Our FR approach would be best integrated into a visual tool to help researchers understand and explore pan-genomic data, for example, graph visualizations allowing users to expand iFR nodes into the underlying structure or perform analyses on their genetic content, such as multiple sequence alignment. Furthermore, existing annotation data could be superimposed on the graph to guide the user's inquiry.

In the following chapters, we will investigate additional applications of the FR algorithm and propose improvements that will allow it to scale to larger, more complex data sets.

3.8 Acknowledgements

The contents of this chapter was originally presented in [20, 21] and are reprinted here for the purpose of the author's educational theses.

CHAPTER FOUR

GENOME CONTEXT VIEWER

In this Chapter, we again consider the problem of mining synteny, but at the genomic resolution. We will address the problem of mining synteny, and subsequently research Questions 2 through 4 from Chapter 1, with an emphasis on Questions 3 and 4, by presenting the Genome Context Viewer (GCV), a visual data-mining tool that allows users to search across multiple providers of genome data for regions with similarly annotated content that may be aligned and visualized at the level of their shared functional elements. By handling ordered sequences of gene family memberships as a unit of search and comparison, the user interface enables quick and intuitive assessment of the degree of gene content divergence and the presence of various types of structural events within syntenic contexts. Insights into functionally significant differences seen at this level of abstraction can then serve to direct the user to more detailed explorations of the underlying data in other interconnected, provider-specific tools.

4.1 Introduction

In Chapters 2 and 3 we discussed methods for analysing pan-genomes at the nucleotide level. In this chapter we will discuss methods for analysing whole clades, rather than pan-genomes, at the genic level, though these methods may be appropriate for pan-genomes of species that tend to be both large and complex, as will be discussed in Chapter 5.

The advances in sequencing technology and algorithms that has led to unprecedented amounts of sequenced whole genomes has also led to the widespread availability of annotated whole genome assemblies. These annotated assemblies vary widely in terms of the technologies and algorithms used, the complexity of the genomes targeted, the quality of their representation and the magnitude of the phylogenetic distances among them [9]. For many comparative applications, users of these resources are less concerned with low-level details of sequence alignment than with basic questions surrounding functional content and the genomic contexts in which it occurs. Consequently, there is a need for tools that can facilitate the use of contextualized functional annotation as a unit of search and comparison among sets of genomes spanning diverse taxonomic groups. Furthermore, the genomes within such a set of data may be curated by different organizations and so may reside in a distributed set of databases. Since the full compliment of these data may exceed the storage or computational limits of many users, there is a need for tools that can perform comparative analyses on these data in their distributed state.

Here we present the Genome Context Viewer (GCV), a web-based visual data-mining tool for dynamically identifying syntenic genomic segments and enabling interactive exploration of gene content similarities and differences among distributed collections of annotated genomes.

Using GCV, a simple request using a gene known to reside in a region of interest is sufficient to retrieve all genomic segments with similar gene content (regardless of whether a match is present to the query gene itself), align the genes in the returned segments to account for presence/absence, copy number, and structural variation, and present the result in an intuitive interactive view for in depth exploration.

4.2 GCV Application

A *genome context* is a region considered primarily with respect to the ordering and orientation of its functionally significant elements. The primary visualization of a genome context in GCV is a horizontal track in which triangular glyphs represent genes ordered according to their occurrence in the segment, with directionality indicating orientation and intergenic distances represented by the thickness of connecting lines; see Figure 4.1. Colors are assigned to reflect membership in families, providing a visual overview of homologies within and between tracks. The association of each color with a gene family is presented in an interactive legend which can be used to highlight all members of a family present in the view, or to access more information about a family.

GCV uses a service-oriented design to achieve a separation of server-side functions of content match and retrieval from client-side functions of segment alignment and display. This enables federation of data from multiple providers into a single comparative context, depending only on their adoption of a consistent classification into families, or rather, homologous relationships.

The main view of GCV, shown in Figure 4.1, is built to represent a set of genome contexts in terms of their functional content. The most simple form (not pictured) is the *basic* view built from a user-specified set of genes, each of which serves as the *focus* gene of a genome context, flanked by a user-specified number of genes. A more powerful variant on this theme is presented in the *search* view, in which a single gene is specified as the focus of a query track and a user-specified number of flanking genes determines the track extent. Provider services are invoked to locate segments similar in content to the query, and matched segments are aligned to it based on gene family membership and ordering using modified pairwise sequence

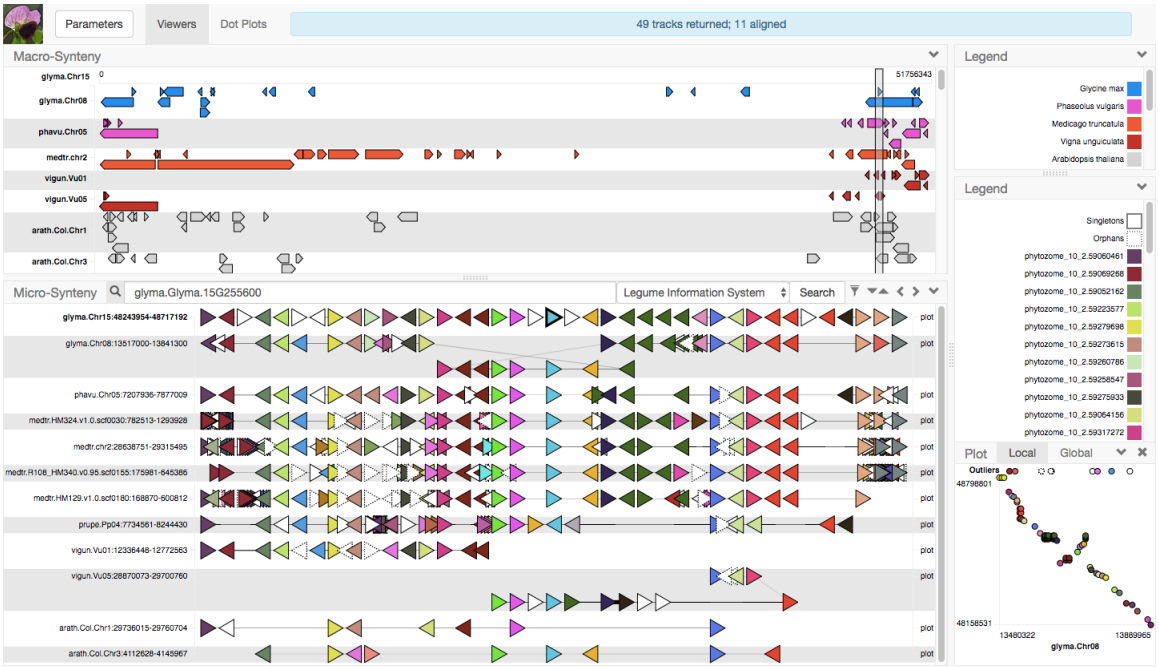

Figure 4.1: A GCV search view exhibiting copy number presence/absence/variation and structural rearrangements. (lower left) Micro-synteny relationships generated from search results aligned with the Repeat algorithm to capture inversion. (upper left) Precomputed macro-synteny blocks indicating the chromosome-scale context of the micro-synteny tracks below. (lower right) A local dot plot of the query track and a selected result track, giving a complementary view of microsynteny features. (upper right) Gene family legend with focus family highlighted.

alignment algorithms. The algorithms operate on the gene family alphabet, rather than the underlying sequence, greatly reducing the computational requirements and allowing them to compute optimal alignments within the context of the responsive user interface. All parameters for defining acceptable gene content similarity and alignment scoring may be altered by the user to suit their specific application. Additionally, further algorithmic modifications make it possible to correctly align inversions and segmental tandem duplications, events occurring frequently at the scale of multi-gene segments in genomes but which are outside the scope of traditional sequence alignment algorithms [32].

Pairwise dot plots are used to represent spatial distributions of corresponding annotations and aid in identifying elements lost, gained or subjected to copy-number alterations. Local plots are composed of the gene family content of the query and result track segments, whereas global plots display all instances of genes from the families of the query track and result track across the chromosome from which the matched syntenic segment was taken. This gives a better sense for the frequency with which members of these families occur outside the matched context and can reveal wider syntenic properties, such as the existence of multiple collinear matches to a region in disparate locations on a single chromosome.

GCV can also display pre-computed macro-synteny blocks in a viewer similar to genome browser feature tracks, with the region corresponding to the current genome context highlighted. Dragging this to a different region triggers a new search, allowing the user to quickly move to regions that may be of interest for higher-level structural properties, such as breakpoints. Ordering and filtering of macro-synteny tracks is coordinated with the corresponding micro-synteny tracks below.

All GCV visualizations provide a context-menu allowing users to download high-quality images of the visualizations as well as the underlying data for further

analysis. Individual elements such as genes, genomic regions, and gene families also provide context-menus whose specific content can be customized to interlink with other resources providing complementary functionality.

4.3 Related Work

The motivation for using gene-families as a unit of search and alignment was two-fold: 1) to emphasize functional content and the genomic contexts in which it occurs and 2) to make these analyses sufficiently performant to be done on-demand across large taxonomic groups in a federated manner. The merit of the first has been sufficiently discussed elsewhere [68, 69], so we will focus on the second.

The "basic" display of GCV is not concerned with similarity of gene content between tracks or with producing track alignments. In this sense, it is similar to the visualizations provided in the Eukaryotic Gene Order Browser [68] and the gene family pages of Phytozome [40].

It is the GCV's "search" view that makes use of algorithms to determine segmental similarity and collinearity by use of the pre-estabished gene family assignments, and which makes it in some sense comparable to tools that are concerned with problems of whole genome synteny comparison. We note however that the alignments that GCV computes dynamically are not whole genome comparisons, but are limited to the determination of segments within whole genomes that are syntenic to the given query track, whose extent determines the computational cost of the subsequent search and alignment problem.

There exist a variety of web-based tools for pairwise and multiple genome synteny search and comparison. The three that are perhaps most related to the GCV are the Plant Genome Duplication Database (PGDD) [63], CoGe's GEvo [70], and Genomicus' PhyloView [69].

PGDD is a database characterizing whole genome duplication events in plant genomes, providing both intra- and inter-genome syntenic relationships. Similar to the GCV, users can perform synteny searches by providing a query gene or can generate a dot plot by selecting two genomes to compare. A locus search[1] will yield a set of precomputed matches between anchor genes for a genomic window of specified size centered on the query gene, rather than an explicitly aligned representation of the query to the resulting tracks; see Figure 4.2. PGDD uses the MCScanX algorithm [98] to precompute collinear blocks, where candidate anchors are based on either pairwise BLAST of the gene models or clustering of genes into homologous groups. Although our algorithmic approach is not geared to producing whole genome alignments, in Section 4.3 we validated the use of our gene family assignments as a surrogate for direct pairwise comparison by computing whole chromosome synteny blocks at the genic resolution. We found that these synteny blocks are similar to those produced by the technique used by PGDD, suggesting that the micro-syntenic blocks reported in the GCV are valid and that precomputed sequence-level / whole genome blocks are not necessary for deriving blocks within a certain locale.

CoGe's GEvo (Genome Evolution Analysis) is one of a large suite of tools for genome synteny analysis[2]. GEvo is the tool within CoGe that is probably most comparable to GCV, as users can perform a search by specifying some genomic feature and flanking region as a search query and selecting what alignment algorithm(s) and corresponding parameters to use. Results are then displayed as pairwise mappings of matched blocks between sequences. The query and search sequences can be loaded from CoGe's database, NCBI, or provided by the user; similarly, CoGe allows any user to upload any number of genomes to use as the target to their analyses, which allows it

[1] `http://chibba.agtec.uga.edu/duplication/index/locus`
[2] `http://genomeevolution.org`

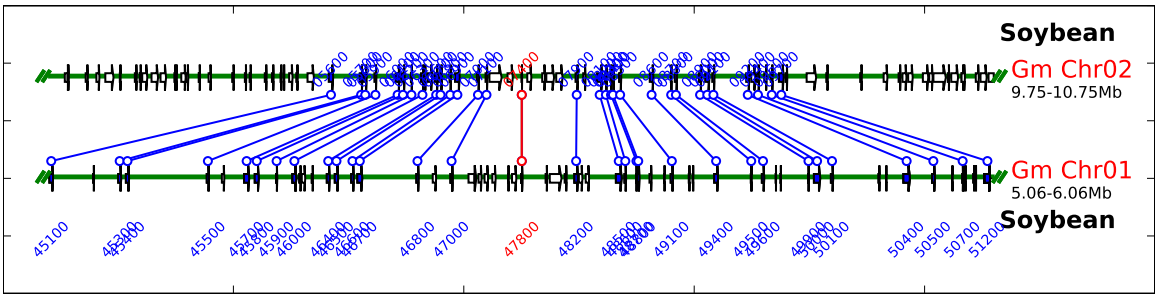

Figure 4.2: The soybean chromosomes 1 and 2 comparison from region showing the same inverted segment in Figure 4.4, produced using the PGDD locus search. Noteworthy is that the inverted block displayed in GCV using its repeat and track merging algorithm shows up as an unmatched set of anchors (with the exception of the central gene in the inverted segment). This has the result that the choice of any of the genes within the interval as the locus to be searched fails to find the two blocks in which they are embedded as being syntenic. GCV's approach finds the blocks as candidates due simply to their similar gene content and initially ignores the ordering which has been disrupted by the structural variation, then applies the client-side modified alignment to detect the presence of the inversion and display accordingly.

to be used regardless of the specific taxonomic focus of any given user. A GEvo search can also be initiated from a pair of genes previously determined to reside in a syntenic region by one of the other tools in the CoGe suite, for example, from a dot in the whole genome comparison dotplots produced by CoGe's SynMap tool, which makes it convenient as a mechanism for generating sequence-level similarity in restricted regions that are predetermined to contain syntenic content. The Gobe viewer for GEvo output is implemented in Flash, and unlike the gene family strategy taken by GCV, emphasizes comparisons of high scoring pairs between pairs of sequences and their relationship to gene structural elements - such as exon structure - making the view more detailed than GCV and hence a little more complex to interpret when large numbers of comparisons are in play. This could obscure the identification of structural events, such as copy number variation and presence/absence variation among gene clusters, although it makes it possible to see sequence-level events below the level

of those that impact annotation, which would be invisible to GCV. The ability to perform on-demand sequence comparisons for genomic segments of interest provides a naturally complementary functionality to the approach taken by GCV and we are developing approaches for utilizing CoGe web services as optional plugins and linkouts through the service framework used by GCV.

Genomicus' PhyloView is similar to GCV in that it compares sequences of genes based on their gene family content. In fact, the alignment view is built from a tree representing the phylogenetic relationships among the returned tracks, where each track is displayed alongside the node it represents in the tree. This is an excellent feature and well suited to the ancestral reconstruction of genomic segments - a key strength of the system. However, for the purposes of federating data across providers, a tight coupling of tracks even with a predetermined tree makes integration with multiple data sources non-trivial. Additionally, rather than compressing, misaligning, or inverting inexact matches, Genomicus simply omits content from other tracks not matching a gene family in the query. This could prevent the user from identifying interesting structure present in many tracks other than the query.

The feature that most fundamentally distinguishes the GCV from these tools is that it only requires each genome's annotations have their gene family assignments pre-computed, which can be done independently on each genome assuming that the family definitions are sufficiently broad to capture most gene content of interest within the taxonomic group. This allows GCV to easily support data federation, only requiring that all service providers have come to agreement on a common set of gene family definitions. This model enables the user to recognize events like copy-number variation and presence/absence variation across large sets of genomic segments from taxonomic groups that span multiple genome data providers.

4.4 Architecture

GCV is a client-side single page Web application and so can be run locally or served as part of a website. It consumes data from one or more service providers that implement the interface defined in a RESTful API[3]. Once it has aggregated data from the providers, GCV creates visualization-specific data representations (micro-synteny alignments, dot plots, and macro-synteny tracks), filters these data according to user-defined criteria, and visualizes the results. The software architecture and data flow is depicted in Figure 4.3.

4.4.1 Technology stack

GCV was implemented using modern Web standards technologies, specifically, Angular[4], ngrx/store[5], and D3[6]. Angular is used to retrieve data from service providers, manage UI components, and mediate communication between the visualizations and the rest of the application. ngrx/store is used to manage application state and give GCV explicit data flow, and D3 is used to draw the various visualizations. Furthermore, the visualizations and their inter-visualization interaction mechanisms have been encapsulated in their own JavaScript library so they can be used independently of GCV.

GCV is service implementation agnostic, only requiring that the services it uses adhere to the RESTful API. Even so, the service implementation used by LIS and LegFed is freely available so that users with similar infrastructures need not re-

[3] See `https://github.com/legumeinfo/lis_context_viewer` for the full API.
[4] `https://angular.io/`
[5] `https://github.com/ngrx/store`
[6] `https://d3js.org/`

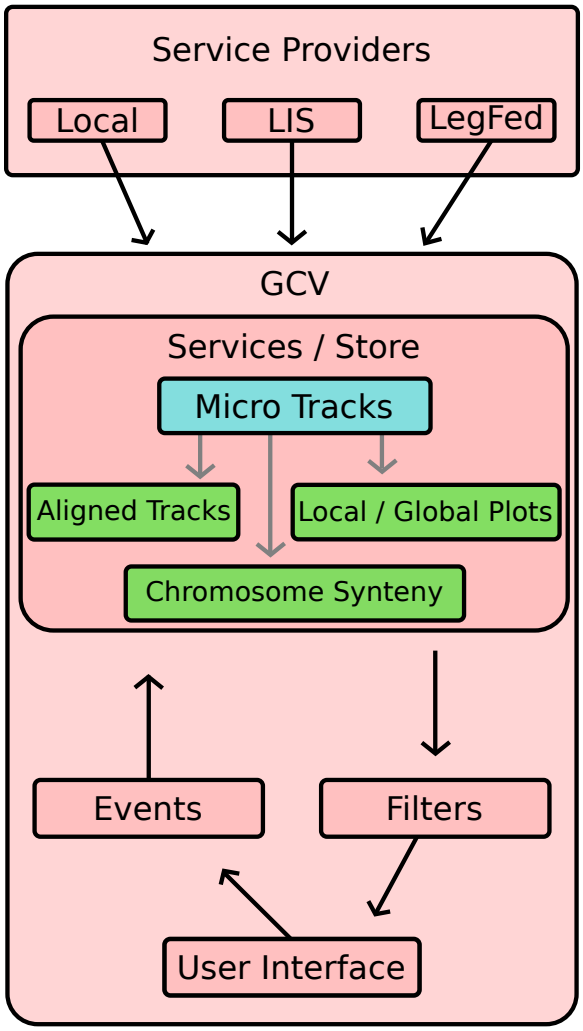

Figure 4.3: The software architecture and data flow of GCV. GCV (an Angular application) consumes data from one or more service providers, one of which may be the user's own computer (local). Angular services are responsible for requesting data from service providers, aggregating the results, and storing the results in ngrx/store. In the search view, whenever new micro track data (blue) is acquired, visualization specific representations are made (green). These representations are then filtered by user controlled criteria and consumed by Angular components (the UI) which draw the data with D3. User interaction with the UI can trigger certain events that will notify the services to update the representations or acquire new data, such as changing alignment parameters or search parameters, respectively.

implement GCV services. It is implemented as a Django application[7] on top of a Chado database [77].

4.4.2 Services

The providers that GCV uses must implement one or more of the RESTful API services. Here we will briefly describe each of the services.

basic micro-synteny tracks: Takes a set of focus genes and the number of neighbors that should flank each gene. It returns the set of tracks centered about the given focus genes.

search micro-synteny tracks: Takes a query track, that is, an ordered list of gene families and search parameters: minimum number of matched families and maximum number of non-matched families between any two matched families. It returns all micro-synteny contexts in the database that meet the given parameter constraints.

gene to query: Takes a focus gene and the number of neighbors as input and passes them to the *basic micro-synteny tracks* service. It then returns the resulting basic track as a query track.

macro-synteny tracks: Takes a reference chromosome and a set of aligned chromosomes as input and returns the synteny blocks of the aligned chromosomes with positions relative to the reference chromosome.

[7] `https://www.djangoproject.com/`

nearest gene: Takes a chromosome and a position on the chromosome and returns the gene nearest to that position on the chromosome.

global plot: Takes a set of gene families and a chromosome as input and returns all the genes on the given chromosome that are members of the given gene families.

An important caveat to consider is that the data provided by a service provider may actually be aggregated from multiple curators or data-stores, as is the case with LIS and LegFed. To enable better interoperability with other sites, such as the primary repository of a particular genome, GCV can use services to load links to relevant external sites. For example:

gene links: Takes a gene as input and returns a list of external links that the gene can be passed to for further inquiry.

4.5 Algorithms

A variety of existing and novel algorithms are used both in GCV and the LIS/LegFed services implementation. Here we present four such algorithms that are integral to the Search view and illustrate the data flow depicted in Figure 4.3.

4.5.1 Track search via AFS supprting paths

Although the implementation of GCV services is left to the service providers, due to the search view's central role in GCV and the non-trivial nature of the corresponding service, we will discuss how the open-source example server used by LIS and LegFed implements the micro-synteny search service.

The problem of finding micro-synteny tracks with similar gene family content and ordering to the query track is an instance of the Fixed-Radius Near Neighbors problem [5]. In the example implementation, the radius is defined by the query parameters - minimum number of *matched* gene families and maximum number of non-matched, or *intermediate*, families between any two matched families - so the search space is 2-dimensional. Gene family ordering (edit distance) could be a third dimension, but we leave the consideration of ordering beyond computing the number of genes between matches as a task for the front-end. This is because track similarity in this sense is dictated by the choice of alignment algorithm and corresponding parameter values. Thus, the tracks returned by the micro-synteny search service as we have defined it are invariant with respect to choices of alignment algorithm and parameters, so changing these in the client does not require a new set of server requests.

It can be seen that the problem of finding micro-synteny tracks as we have defined it is a variation of the AFS supporting paths problem from Chapter 2. Here, though, the ϵ_r constraint is a minimum number of nodes that each supporting path must traverse (*matched*), rather than a fraction, and there is no ϵ_c constraint on the number of subpaths that must support each node in the anchor path. The *intermediate* constraint is equivalent to the AFS μ constraint and minsup = 1. This variation of the AFS supporting path problem is tractable.

The example implementation uses an exact algorithm that works as follows: Given a micro-synteny search query (anchor path), specifically, the list of gene families present in the query track, the algorithm iterates the ordered list of gene families of each chromosome in the database. When a gene family that matches one of the query families is found a new candidate track is created. The candidate is grown by iteratively adding the next family in the list to the track until the number of families

added since the last match is greater than *intermediate*. The candidate track is then trimmed back to the last matched family and returned if the number of matched families it contains is greater than or equal to *matched*. The algorithm then continues iterating the ordered list of gene families at the family after the last family iterated by the previous candidate growth.

Since each chromosome in the database may have several thousand genes, the example implementation is optimized as follows: First, the algorithm leverages the indexing mechanisms of the underlying database to efficiently find all instances of the query gene families in the ordered list of gene families for each chromosome and memoizes each matched family's position in its corresponding list. Then, the iterative algorithm is applied to each chromosome's ordered list of matched gene families if the list contains at least *matched* number of families. Candidate tracks are grown as before, but the memoized position data is used to compute how many non-matched families lie between the last and next matched families, rather than iterating the non-matched families. A batch query is then performed to fetch all the non-matched families for the candidates that satisfy the *matched* parameter.

4.5.1.1 Time complexity In the worst case, the algorithm takes $O(L)$ time, where L is the total length (number of genes) of all the chromosomes in the database.

4.5.2 Merging alignments

The Repeat algorithm [32] is an extension of the Smith-Waterman local alignment algorithm [90]. Specifically, rather than finding the highest scoring local alignment, it finds all local alignments whose score is above a certain *threshold*. In this work, we extended the Repeat algorithm to identify inversions whose reversal in the containing sequence improves the sequence's alignment score.

In essence, the extension works by first aligning both the forward and reverse orientations of a sequence to the reference with the Repeat algorithm. All resulting forward alignments are then compared with all reverse alignments for shared gene content, which can be done efficiently with an interval tree. When a reverse alignment is found to have shared gene content with a forward alignment, memoized suffix scores from the alignments' traceback matrices are used to determine if replacing the inverted sequence in the forward alignment with the reverse alignment will improve the forward alignment's score, or vice versa. If so, the forward and reverse alignments are *merged* by replacing the inverted sequence and updating the memoized suffix scores and alignment score.

4.5.2.1 Time complexity Let n be the length of the sequences being aligned, F be the set of forward alignments, and R be the set of reverse alignments, where $|F|, |R| < n$ and $\sum_{f \in F} f, \sum_{r \in R} r \leq n$. We construct an interval tree for F and then search for overlaps with each interval in R, requiring $O(|F| \log |F|)$ and $O(|R| \log |F| + |R|m)$ time, respectively, where m is the numbers of intervals that overlap with any one interval in R, so $m \leq |F|$ but we expect it to be at most 1. Lastly, computing the score of an inversion and performing the inversion takes linearly time in the length of the overlapping segments from R. In the worst case, it will take $O(n)$ time to perform all inversions. Since $|F|, |R| < n$, the $O(n^2)$ complexity of the Repeat algorithm still dominates the run-time.

4.5.3 Alignment coordinates

An extension that we have applied to both the Repeat and Smith-Waterman alignment algorithms is the assignment of coordinates to the gene sequences based on their alignments. It works by *positioning* all resulting alignments relative to the reference sequence. Specifically, a matched character is given the position of the

character it matched in the reference sequence and inserted characters are given a position between that of the reference characters they were inserted between. For example, given gene family reference $\tau\alpha\kappa\gamma\gamma$ and sequence $\alpha\gamma\kappa\kappa\kappa\gamma$[8], the following hypothetical forward alignment would be positioned as:

position in reference:		0	1	2	3				4
alignment	reference:	τ	α	κ	γ	-	-	-	γ
	sequence:		α	-	γ	κ	κ	κ	γ
position in reference:			1		3	3.25	3.5	3.75	4

The sequence's forward alignment positioning indicates that it spans the character interval 1-4 in the reference.

By positioning all alignments relative to the reference sequence they are normalized to the same coordinate space. These normalized coordinates are what are used to position the tracks in the micro-synteny search visualization.

4.5.3.1 Time complexity Let n be the length of the sequences being aligned. The alignment coordinates could be computed during the alignment dynamic program by filling a coordinate matrix in conjunction with the score matrix. This would require $O(n^2)$ additional space and $O(1)$ additional time. Alternatively, the alignment coordinates could be computed by iterating the final alignment. The longest alignment possible has length $2n - 1$, meaning computing coordinates would require $O(n)$ additional time. Either way, the $O(n^2)$ complexity of the alignment algorithm still dominates the run-time.

[8]Greek characters are used only for example purposes. In practice the gene families are matched on their unique identifiers.

4.5.4 Track packing

GCV uses the extended Repeat algorithm to detect inversions in micro-synteny search results so that they may be explicitly drawn. In cases where more than one inversion has been found in a single alignment, all inversions are to be drawn as compactly as possible, that is, we want to pack as many inversions into as few visualization rows as possible without overlapping any of the inversions. Similarly, in the macro-synteny visualization we want to draw the synteny blocks from the same chromosome as compactly as possible.

This "Track Packing" problem is equivalent to the Interval Partitioning/Coloring (IP) problem for which there exists an optimal polynomial-time solution [60]. The solution works by iteratively applying the greedy polynomial-time solution for the Interval Scheduling (IS) problem [60] to a set of intervals until all the intervals have been scheduled. In the case of the micro-synteny alignments, the intervals are the position intervals described in Section 4.5.3. In the case of the macro-synteny blocks, the intervals are the blocks' genomic positions on the reference chromosome.

4.5.4.1 Time complexity

Let I be the total number of intervals to be packed. The IS problem requires that the intervals first be sorted by end time, taking $O(I \log I)$ time. It then iterates the intervals and greedily adds them to the schedule in $O(I)$ time. The IP problem repeats this iteration step until each interval has been added to a schedule. In the worst case, all the intervals will overlap and so this step will take I iterations, yielding a time complexity of $O(I^2)$. In our application, this worst case scenario is highly unlikely, so we expect the sorting to dominate the run-time.

4.6 Experimental Results

In this section we investigate the efficacy of using gene family assignments as a surrogate for direct pairwise sequence comparison by computing whole chromosome synteny blocks at the genic resolution. We then report on how the GCV is currently being used by the Legume Information System and Legume Federation projects.

4.6.1 Block resolution

DAGChainer [42] is an algorithm commonly used to compute "chains" of syntenic genes (synteny blocks) in complete genomes via dynamic programming. It takes as input a list of homologous gene pairs, typically computed by a pairwise sequence comparison method, such as the Basic Local Alignment Search Tool (BLAST) [2]. MCScanX [98] is an algorithm commonly used to identify colinear segments (syntenic blocks) in multiple genomes or highly redundant genomes, such as legumes. MCScanX works by computing pairwise segments of colinearity using DAGChainer and then combines them into larger segments by identifying overlapping related segments while resolving ambiguities caused by structural events, such as rearrangements, duplications, and inversions.

We compared the results of MCScanX on our gene family assignment derived homologous groups to the results of basic MCScanX on pairwise BLASTs of the coding DNA sequence (CDS) for the same genome. For a self-comparison of the *Glycine max* (Soybean) genome, the latter puts 66% of all gene models into 1110 blocks ranging from 5 to 1074 genes in extent. Using an unfiltered set of gene families, the comparable procedure yields 73% of genes in 2825 blocks ranging in size from 5 to 1102 genes. Inspection of the blocks produced with this approach suggested that many of the excess blocks were the results of genes from massively expanded families residing in

large clusters. Since our algorithms for finding similar regions require that the families matched be distinct, we repeated the comparison by filtering our families to exclude from consideration any that contained more than 20 members from Soybean. The results of this procedure were more similar to the BLAST-based approach with 61% of genes placed into 1319 blocks ranging in size from 6 to 983 genes. Most of these blocks are coequal in extent to the corresponding blocks from the BLAST-based procedure, but are missing some internal anchor genes due to the pre-filtering procedure employed in this context; in GCV's micro-synteny search implementation, these high copy gene families would still be scored as matches during the alignment procedure used on the resulting blocks. Furthermore, the macro-syntenic blocks produced by MCScanX on the gene family assignments are quite similar to those precomputed by a pipeline based on DAGChainer and used in the LIS implementation of the GCV; see Figure 4.4

4.6.2 LIS and LegFed integration

The Legume Information System[9] (LIS) [27] is an online platform for legume breeders and researchers that houses a variety of genetic and genomic data of model legume species relevant to industrial agriculture. The Legume Federation[10] (LegFed) is a consortium of legume researchers and groups, including LIS, with the objective of fostering the adoption of data standards, distributed development, and enabling comparative analyses. GCV was born from the needs of these projects and is integrated into the sites as follows.

LIS acquires annotated genomes from a variety of independently managed projects and primary data repositories. In accordance with the mission of the Legume

[9] `http://legumeinfo.org/lis_context_viewer`
[10] `http://legumefederation.org/lis_context_viewer`

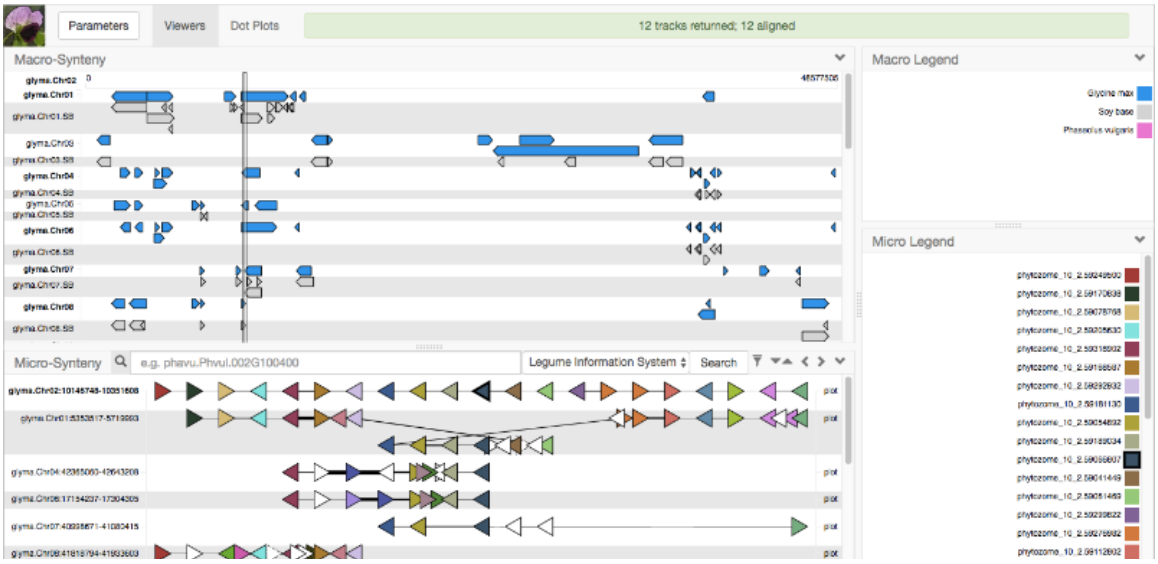

Figure 4.4: Comparison of results from MCScanX using gene family assignments (blue blocks) to those produced using DAGChainer on CDS pairwise matches (gray blocks). As can be seen, the results are generally similar, with the gene-family based use of MCScanX tending to coalesce into larger blocks some regions called as fragmentary by DAGChainer. It is also clear that the GCV algorithm for determining microsynteny to the region of soybean chromosome 2 used as the query is producing results of comparable sensitivity to those produced by MCScanX, as evidenced by the small blocks from chromosomes 4 and 6 found in both the MCScanX macro-synteny blocks and the GCV microsynteny representation, but missing from the DAGChainer-based results. On the other hand, MCScanX tends to ignore inverted segments that disrupt larger collinear blocks, as displayed in the focus region of the example and present in the corresponding DAGChainer blocks. See Figure 4.2 for the effects of this behavior in the context of the PGDD implementation of block search and display.

Federation, homology relationships among the genes annotated in these genomes are established by initial HMM-based assignment to the Phytozome Angiosperm-level gene families [40]; this in itself is sufficient for the purposes of making genomes available for use in GCV. Further steps of multiple sequence alignment and phylogenetic gene tree construction are used to produce interactive tree visualizations which interoperate with GCV in several ways. For example, a collection of genes can be specified from an arbitrary subtree and loaded into the basic view of GCV as the set of focus genes from which context tracks are derived. Alternatively, a leaf node of a phylogeny can be used as the focus gene of a query track in the search view of GCV. This linking can also be reversed, that is, in GCV the user may select a gene family and view the phylogenetic tree for that family with the members present in the linking GCV context highlighted.

Given the vast amount and different types of data housed by LIS, a heterogeneous collection of software is used to facilitate user exploration and knowledge discovery. As illustrated by the interoperability of the phylogenies and GCV, interlinking between these software is crucial to the utility of LIS. As such, GCV is configured to link to various LIS tools including Tripal gene pages [15] and InterMine reports [89]. Additionally, a service is used to link to external tools specific to the sources from which the various genomes were acquired, as described in Section 4.4.2.

4.7 Conclusion

The GCV is a powerful tool that is already being used to investigate functional contexts of interest, assess assembly/annotation quality, and perform analyses on geographically distributed data. As will be discussed in Chapter 5, the tool is ripe for integration with the algorithms discussed in Chapters 2 and 3.

4.8 Acknowledgements

The contents of this chapter was originally presented in [17, 27] and are reprinted here for the purpose of the author's educational theses.

CHAPTER FIVE

PROPOSAL

In Chaper 1, we presented the broad research question that the proposed dissertation would address:

1. What data structures, algorithms, and statistical methods can be used to perform bioinformatic analyses of pan-genomic data?

This question was broken down into three specific sub-questions:

2. Pan-genomic data are commonly represented in a graph structure; how can this structure be utilized to enable novel and efficient methods of bioinformatic analysis?

3. Given the potentially massive size of pan-genomic data, how can these data structures and algorithms be parallelized/distributed?

4. The interpretation of biological data and the results of their analysis are tasks performed by domain experts, often with the aid of visualization tools. How can pan-genomic data and the results of their analysis be visualized effectively for the purpose of interpretation?

In Chapters 2 through 4, we addressed the research questions presented in Chapter 1. Specifically, Chapters 2 and 3 addressed Questions 2 through 4, with an emphasis on Question 2, and Chapter 4 addressed Questions 3 and 4. In this chapter, we propose research tasks that are based on the contents of Chapters 2 through 4. These tasks will further address Question 3 and explore additional applications of the previously described methods, addressing the broader Question 1.

5.1 Task 1

As discussed in Chapter 4, the search view of the GCV can load pre-computed chromosome scale synteny blocks, allowing users to quickly assess if the genomic context they are viewing is part of a larger structure and enabling the rapid exploration of context space. Though this is a useful feature, this approach has its limitations. Specifically, since the blocks are precomputed, each time a new genome is added to a database that serves GCV, it must be run through a synteny pipeline and the results loaded into the database. This has the potential drawbacks of delay between genome acquisition and availability of corresponding chromosome scale synteny data, as well as the potential for inconsistencies between how syntenic blocks are computed across databases. Furthermore, a database can only serve blocks for chromosomes that it is aware of, which fundamentally undermines the GCV's distributed data model.

As we demonstrated in Chapter 4 with MCScanX [98] and DAGChainer) [42], genes described by their gene family assignments can serve as a surrogate for pairwise nucleotide comparison, providing comparable synteny blocks at a fraction of the computational cost.

Proposed contribution

We propose to implement an algorithm similar to MCScanX that can use gene family assignments to compute pairwise chromosome scale synteny blocks on demand, including for chromosomes that are not explicitly represented in the database.

Proposed methods

In order to achieve this, each database must be aware of the ordered gene family content of the query chromosome. Since the algorithm is operating at the genic level,

the "query" will be composed of tens of thousands of genes, rather than millions or billions of nucleotides, making it reasonable to send the query chromosome to each of the databases. This can be achieved by having the client send the query directly to each of the servers or by inter-server communication enabled via URI passing mediated by the client.

The tasks of search and alignment will be similar to that for performing genomic context searches, as described in Chapter 4, Section 4.5. The distinction here being that the alignment will need to be performed on the server due to the storage and computational limitations of the client. Synteny blocks will be computed for each chromosome in a database relative to the query. The algorithm will be a two step "islands and gaps" algorithm: 1) identify all forward and reverse contiguous sequences (islands) 2) use a dynamic program similar to that of MCScanX (DAGChainer) to "chain" islands together whose gaps are small enough on both the query and chromosome while resolving inconsistent orderings.

A further extension of this work would be to summarize these synteny blocks in the macro-synteny view of the GCV by computing consensus blocks for syntenic blocks that occur frequently. This could be done by treating each synteny block as an interval on the query chromosome and then finding intervals where many blocks overlap. This is effectively an instance of the AFS problem from Chapter 2 with the additional constraint that any anchor path must be a subpath of the query chromosome. This, combined with the relaxed column and row constraints, makes the problem tractable.

Intellectual merit

As will be discussed in Task 4, not only does this task provide a traditionally cumbersome analysis as an on demand service across multiple databases, it also serves as a gateway to a more efficient method of whole genome pairwise sequence alignment.

5.2 Task 2

As discussed in Chapter 4, the basic view of the GCV displays genomic contexts for a given set of genes. Though the gene set can be arbitrary, it can also contain homologous genes. For example, in LIS the set of genes may correspond to a particular phylogenetic tree, in which case there will likely be multiple syntenic genomic contexts present in the view. In this scenario, it would be useful to the user to group such contexts and align them as best as possible in order to emphasise the syntenic relationships, as is done in the search view.

Proposed contribution

We propose to implement an algorithm that clusters genome contexts with similar gene family content and then perform a multiple sequence alignment on the contexts within each cluster.

Proposed methods

Since the first goal of this task is to cluster the genomic contexts based on their shared gene family content, the FR algorithm from Chapter 3 is a good candidate clustering algorithm. The nuance here is that a gene family may participate in multiple distinct syntenic contexts. This could be problematic since the FR algorithm is agglomerative, so each family can only participate in a single root FR. The solution we propose is to apply the FR algorithm multiple times. After each application, the largest root FR will be kept and its supporting paths (genomic contexts) will be removed from the graph, but its gene families that participate in paths still in the graph will not be removed. The remaining FRs will then be discarded and the algorithm will be run again on the updated graph, freeing the gene families that were used by the selected FR to participate in other FRs.

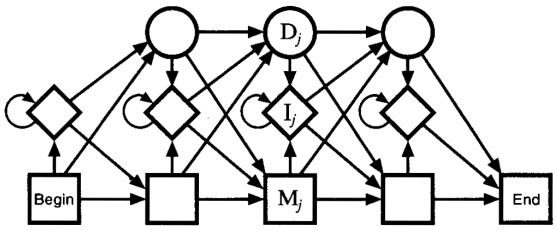

Figure 5.1: The topology of the profile HMM to be used for multiple sequence alignment [32]. The square nodes in the bottom row depict the match states. For example, the node with the label M_j is the j^{th} match state. The diamond nodes in the middle row depict the hidden insertion states, and the circular nodes in the top row depict the hidden deletion states. Similar to the labeled match state, the nodes labeled I_j and D_j depict the j^{th} insertion and deletion states, respectively. Each state can transition to each other type of state, and insertion states can also transition to themselves, enabling support for alignments containing arbitrarily long insertions.

Next is the goal of aligning the genomic contexts within each cluster. Unlike the search view, there is no "query" track to align against, so we must perform a multiple sequence alignment. This problem is known to be NP-complete [34, 56, 97], so heuristic and statistical methods are typically used. Though progressive approaches are generally more efficient [44, 46, 79], we propose a Hidden Markov Model (HMM) based approach to gain finer control over the alignment for visualization purposes, as we did in Chapter 4 with the Smith-Waterman and Repeat algorithms in the search view of the GCV.

The HMM alignment will be similar to that described in [32]. Specifically, the multiple sequence alignment will be computed by iteratively training a profile HMM. The topology of the model will be similar to that of [62], where, in addition to begin and end states, there is a visible match state and corresponding hidden delete state for each column in the alignment, and a hidden insertion state between each match/delete state pair; see Figure 5.1.

The initial HMM topology and transition/emission probabilities will be derived from the pairwise alignment of the two most similar genomic contexts in the cluster (FR), in terms of shared gene family content. Each remaining genomic context will then be added to the model using the Viterbi algorithm [96]. After a genomic context has been added to the HMM, model surgery will be performed. Specifically, since the purpose of constructing the HMM is to generate an alignment that is optimal for visualization purposes, we are interested in aligning structures that may be present in a subset of the sequences but not a majority. As such, the surgery will take insertion states that were traversed by the previous alignment and expand them into match states, creating new insertion and deletion states as needed.

Since the sole purpose of the model is the optimal visualization of the genomic contexts used to train the model, we are not concerned about overfitting the model to the training data.

Intellectual merit

This approach will allow users to better identify the syntenic relationships, if any, present in the basic view of the GCV, and thus, will better guide their interpretation of the data. Similar to Task 1, this task will also serve as a gateway to a more efficient method of whole genome multiple sequence alignment, as will be discussed in Task 4.

5.3 Task 3

In Chapter 3, we presented the FindFRs algorithm. The larger of the two data sets on which we evaluated it was comprised of 55 assemblies of 48 strains of yeast. Though this is a relatively small data set and the experiments were performed on a server with 4 Intel Xeon 2.2GHz 32 core processors and 1 TB of RAM, in the case of the $k = 25$ CDBG ($2,260,767$ nodes), the algorithm took several hours to run.

Proposed contribution

We propose to improve the run-time performance of the FR algorithm by revising the region merging step to enable further parallelization.

Proposed methods

The primary bottleneck that we aim to alleviate is the sequential merging of regions. Specifically, regions are greedily merged in most-support-first order, requiring that the support of the edges incident to the regions being merged be recomputed before the next edge is selected. To alleviate this bottleneck, we propose merging as many regions as possible at each iteration of the algorithm. A seemingly reasonable means of doing this is to compute a maximal weighted matching of the graph, where the weights are the edges' support. Each pair of regions covered by the matching will then be merged in the same iteration, which can easily be done in parallel since each region is only involved in one merge. The matching is maximal, rather than maximum, because we still want to select edges in greatest-support-first order while maximizing the number of edges in the matching. Computing such a matching in a single threaded program is trivial, and there exist parallel algorithms for computing such a matching in a shared memory computing environment [3, 13, 76, 82]. On average, this approach will reduce the number of edges in the graph by half at each iteration. Thus, on average, the FR hierarchy will be computed in $O(\log V)$ iterations, and $O(V)$ in the worst case, the latter being the number of iterations the algorithm currently takes. When done in parallel, this should greatly improve the run-time performance of the FR algorithm.

Intellectual merit

These improvements to the FR algorithm will make it practical for larger data sets and, thus, more applicable to the broader bioinformatics community. As will be discussed in Task 4, this will also make it more amenable to the task of multiple sequence alignment.

5.4 Task 4

As mentioned in Tasks 1 through 3, the algorithms discussed thus far may serve as fodder for novel methods of whole genome sequence alignment.

Proposed contribution

We propose to develop novel algorithms for whole genome pairwise and multiple sequence alignment based on the methods of Tasks 1 through 3.

Proposed methods

Specifically, the islands and gaps algorithm described in Task 1 would be applied to ordered sequences of CDBG nodes, rather than the raw nucleotide sequence. And the improvements to the FR algorithm described in Task 3 could be used to target much larger populations of genomes than can be handled by current multiple sequence alignment methods. Once the blocks and regions have been identified, respectively, traditional sequence alignment techniques could be used to generate the final nucleotide alignments, similar to the method proposed in Task 2. For populations that already have annotation data, the blocks and regions distinguishing the sequence to be aligned could be more efficiently computed by using gene family assignments as a surrogate, as was discussed in Chapter 4.

Intellectual merit

Although this task is included more as an exploration of the efficacy of this approach to whole genome alignment, it could potentially improve the existing state of the art, enabling the comparison of much larger, more complex populations.

5.5 Timeline

The following timeline depicts when each of the proposed research tasks will be carried out in the coming academic year.

	2017				2018			
	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Task 1	x	x	x					
Task 2			x	x	x			
Task 3					x	x	x	
Task 4							x	x

CHAPTER SIX

CONCLUSION

As the cost of sequencing DNA continues to drop the number of sequenced genomes continues to rapidly grow, both within and among related species. This necessitates the need for data structures, algorithms, and statistical methods to perform bioinformatic analyses of pan-genomic data. In Chapter 1, we characterized this need with a set of research questions. In Chapters 2 through 4, we addressed these questions with novel methods for mining synteny from pan-genomic data at the nucleotide and genic levels and explored a variety of novel applications of these methods. In Chapter 5, we proposed improvements upon these methods as well as additional applications. The existing methods presented are already being used by crop breeders and researchers to better understand and improve crops relevant to industrial agriculture. The proposed work will make these methods more broadly applicable to additional industries and areas of research.

Bibliography

[1] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. of VLDB*, volume 1215, pages 487–499, 1994.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[3] B. O. F. Auer and R. H. Bisseling. A gpu algorithm for greedy graph matching. In *Facing the Multicore-Challenge II*, pages 108–119. Springer, 2012.

[4] T. Beller and E. Ohlebusch. Efficient construction of a compressed de bruijn graph for pan-genome analysis. In *Combinatorial Pattern Matching*, pages 40–51. Springer, 2015.

[5] J. L. Bentley. Survey of techniques for fixed radius near neighbor searching. Technical report, Stanford Linear Accelerator Center, Calif.(USA), 1975.

[6] J. Berendzen, E. Cannon, A. Cleary, S. Dash, S. Hokin, W. Huang, A. Rice, P. Umale, N. Weeks, A. Wilkey, A. Farmer, S. Cannon, and M. Hadres. Legumeinfo.org: Legume research and trait data that is findable, accessible, interoperable and re-usable. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[7] V. Bewick, L. Cheek, and J. Ball. Statistics review 13: receiver operating characteristic curves. *Critical care*, 8(6):508, 2004.

[8] A. R. Borneman, B. A. Desany, D. Riches, J. P. Affourtit, A. H. Forgan, I. S. Pretorius, M. Egholm, and P. J. Chambers. Whole-genome comparison reveals novel genetic elements that characterize the genome of industrial strains of saccharomyces cerevisiae. *PLoS Genet*, 7(2):e1001287, 2011.

[9] K. R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):10, 2013.

[10] M. Bridges, E. A. Heron, C. O'Dushlaine, R. Segurado, D. Morris, A. Corvin, M. Gill, C. Pinto, I. S. Consortium, et al. Genetic classification of populations using supervised learning. *PloS one*, 6(5):e14802, 2011.

[11] J. Campbell, S. Dash, E. Cannon, A. Cleary, W. Huang, S. Kalberer, A. Rice, J. Singh, P. Umale, N. Weeks, A. Wilkey, J. Town, Christopher DeBerry, D. Fernandez-Baca, A. Farmer, and S. Cannon. What's new in the legume information system and the federated legume database initiative. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016.

[12] S. Cannon, A. Farmer, P. Umale, H. Lokhande, A. Cleary, N. Weeks, S. Kalberer, E. Cannon, S. Dash, D. Bitragunta, and J. Singh. The legume information system (legumeinfo. org) 2015. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015.

[13] A. Chan, F. Dehne, P. Bose, and M. Latzel. Coarse grained parallel algorithms for graph matching. *Parallel Computing*, 34(1):47–62, 2008.

[14] H. Cheng, P. S. Yu, and J. Han. Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *Proc. of ICDM'06*, pages 839–844, 2006.

[15] I.-H. Cho, M. Staton, T. Lee, D. Main, L.-A. Sanderson, K. E. Bett, S. Jung, C.-H. Cheng, and S. P. Ficklin. Tripal: a construction toolkit for online genome databases. 2012.

[16] M. Clamp, J. Cuff, S. M. Searle, and G. J. Barton. The jalview java alignment editor. *Bioinformatics*, 20(3):426–427, 2004.

[17] A. Cleary and A. Farmer. Genome context viewer: visual exploration of multiple annotated genomes using microsynteny. *Bioinformatics*, 2017.

[18] A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Mining frequent subpaths in pan-genomes. In *11th Annual Sequencing, Finishing, and Analysis in the Future Meeting (LANL SFAF)*, 2016.

[19] A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Approximate frequent subpath mining applied to pangenomics. In *International Conference on Bioinformatics and Computational Biology (BICoB)*, 2017.

[20] A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Exploring frequented regions in pan-genomic graphs. In *8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB)*, 2017.

[21] A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Investigating frequented regions (frs) in a yeast pan-genome. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[22] A. Cleary, T. Ramaraj, J. Mudge, and B. Mumey. Pangenomics: Persistent homology for pan-genome analysis. In *11th Annual Sequencing, Finishing, and Analysis in the Future Meeting (LANL SFAF)*, 2016.

[23] A. Cleary, L. Ren, S. Cannon, and A. Farmer. Tools for phylogenomic exploration within and among clade-oriented databases. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015.

[24] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] T. F. Cox and M. A. Cox. *Multidimensional scaling*. CRC press, 2000.

[26] S. Dash, J. Berendzen, J. Campbell, E. Cannon, S. Cannon, A. Cleary, A. Farmer, M. Hadres, S. Hokin, W. Huang, A. Rice, P. Umale, N. Weeks, and A. Wilkey. Customizing tripal sites with non-tripal components at the legume information system and peanutbase. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[27] S. Dash, J. D. Campbell, E. K. Cannon, A. M. Cleary, W. Huang, S. R. Kalberer, V. Karingula, A. G. Rice, J. Singh, P. E. Umale, et al. Legume information system (legumeinfo. org): a key component of a set of federated data resources for the legume family. *Nucleic acids research*, 44(D1):D1181–D1188, 2016.

[28] S. Dash, W. Huang, J. Campbell, J. Singh, A. Rice, P. Umale, A. Cleary, S. Kalberer, N. Weeks, V. Karingula, P. Gupta, S. Gunda, E. Cannon, A. Farmer, and S. Cannon. Lis (legume information system): A clade based web resource for legumes. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016.

[29] F. de Bruijn. A combinatorial problem. 1946.

[30] M. J. Dunham, H. Badrane, T. Ferea, J. Adams, P. O. Brown, F. Rosenzweig, and D. Botstein. Characteristic genome rearrangements in experimental evolution of saccharomyces cerevisiae. *Proc. of National Academy of Sciences*, 99(25):16144–16149, 2002.

[31] B. Dunn, C. Richter, D. J. Kvitek, T. Pugh, and G. Sherlock. Analysis of the saccharomyces cerevisiae pan-genome reveals a pool of copy number variants distributed in diverse yeast strains from differing industrial environments. *Genome research*, 22(5):908–924, 2012.

[32] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[33] R. C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[34] I. Elias. Settling the intractability of multiple alignment. *Journal of Computational Biology*, 13(7):1323–1339, 2006.

[35] A. Farmer, S. Cannon, S. Kalberer, J. Singh, E. Cannon, P. Umale, H. Lokhande, A. Cleary, N. Weeks, V. Karingula, and S. Dash. The legume information system (legumeinfo. org) 2015. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015.

[36] A. Farmer and A. Cleary. Genome context viewer: A user-friendly web application for visualizing and mining shared gene content among multiple genomes. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[37] A. Farmer, A. Cleary, S. Dash, A. Rice, J. Berendzen, S. Hokin, M. Hadres, P. Umale, J. Campbell, W. Huang, N. Weeks, A. Yadav, A. Wilkey, D. Grant, R. Nelson, K. Feeley, V. Carollo Blake, E. Cannon, V. Krishnakumar, S. Cannon, A. Chan, E. Lyons, C. Town, and D. Fernandez-Baca. The legume information system (http://legumeinfo.org) and the legume federation (http://legumefederation.org): Comparative genomics and genetics through cooperative resource development. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[38] A. Farmer, A. Cleary, A. Rice, P. Umale, S. Hokin, S. Dash, J. Campbell, W. Huang, N. Weeks, A. Wilkey, D. Grant, R. Nelson, K. Feeley, V. Krishnakumar, A. Yadav, J. DeBerry, D. Fernandez-Baca, E. Cannon, C. Town, and S. Cannon. The legume information system and the legume federation: Working together for the legume-fed world. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016.

[39] I. J. Good. Normal recurring decimals. *Journal of the London Mathematical Society*, 1(3):167–169, 1946.

[40] D. M. Goodstein, S. Shu, R. Howson, R. Neupane, R. D. Hayes, J. Fazo, T. Mitros, W. Dirks, U. Hellsten, N. Putnam, et al. Phytozome: a comparative platform for green plant genomics. *Nucleic acids research*, 40(D1):D1178–D1186, 2012.

[41] S. Guha. Efficiently mining frequent subpaths. In *Proc. of the Australasian Data Mining Conference*, pages 11–15, 2009.

[42] B. J. Haas, A. L. Delcher, J. R. Wortman, and S. L. Salzberg. Dagchainer: a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–3646, 2004.

[43] D. Haussler, S. J. O'Brien, O. A. Ryder, F. K. Barker, M. Clamp, A. J. Crawford, R. Hanner, O. Hanotte, W. E. Johnson, J. A. McGuire, et al. Genome 10k: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *Heredity*, 100(6):659–674, 2009.

[44] D. G. Higgins and P. M. Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[45] S. Hill, B. Srichandan, and R. Sunderraman. An iterative mapreduce approach to frequent subgraph mining in biological datasets. In *Proc. of ACM BCB*, pages 661–666, 2012.

[46] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of molecular evolution*, 20(2):175–186, 1984.

[47] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining protein family specific residue packing patterns from protein structure graphs. In *Proc. of RECOMB*, pages 308–315, 2004.

[48] H. L. Huang and M. C. Brandriss. The regulator of the yeast proline utilization pathway is differentially phosphorylated in response to the quality of the nitrogen source. *Molecular and cellular biology*, 20(3):892–899, 2000.

[49] W. Huang, S. Dash, A. Cleary, A. Rice, S. Hokin, J. Campbell, J. Berendzen, M. Hadres, P. Umale, N. Weeks, A. Wilkey, A. Farmer, S. Cannon, and E. Cannon. Peanutbase.org: A genomic and genetic data resource to support crop improvement in peanut. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[50] R. M. Idury and M. S. Waterman. A new algorithm for dna sequence assembly. *Journal of computational biology*, 2(2):291–306, 1995.

[51] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.

[52] N. H. G. R. Institute. Dna sequencing costs. `http://www.genome.gov/sequencingcosts/`, 2015. Accessed: 2015-10-20.

[53] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.

[54] I. Ishtar Snoek and H. Yde Steensma. Factors involved in anaerobic growth of saccharomyces cerevisiae. *Yeast*, 24(1):1–10, 2007.

[55] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(01):75–105, 2013.

[56] W. Just. Computational complexity of multiple sequence alignment with sp-score. *Journal of computational biology*, 8(6):615–623, 2001.

[57] K. C. Kao and G. Sherlock. Molecular characterization of clonal interference during adaptive evolution in asexual populations of saccharomyces cerevisiae. *Nature genetics*, 40(12):1499–1504, 2008.

[58] J. Kim, D. M. Larkin, Q. Cai, Y. Zhang, R.-L. Ge, L. Auvil, B. Capitanu, G. Zhang, H. A. Lewin, J. Ma, et al. Reference-assisted chromosome assembly. *Proc. of National Academy of Sciences*, 110(5):1785–1790, 2013.

[59] C. Kingsford, M. C. Schatz, and M. Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics*, 11(1):21, 2010.

[60] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.

[61] M. Koyutürk, A. Grama, and W. Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. *Bioinformatics*, 20(suppl 1):i200–i207, 2004.

[62] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994.

[63] T.-H. Lee, J. Kim, J. S. Robertson, and A. H. Paterson. Plant genome duplication database. *Plant Genomics Databases: Methods and Protocols*, pages 267–277, 2017.

[64] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[65] R. Li and W. Wang. Reafum: Representative approximate frequent subgraph mining. In *SIAM International Conference on Data Mining*, pages 2167–0099. SIAM, 2015.

[66] G. Liti, D. M. Carter, A. M. Moses, J. Warringer, L. Parts, S. A. James, R. P. Davey, I. N. Roberts, A. Burt, V. Koufopanou, et al. Population genomics of domestic and wild yeasts. *Nature*, 458(7236):337–341, 2009.

[67] J. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In *SDM*, volume 6, pages 405–416, 2006.

[68] M. D. Lopez and T. Samuelsson. egob: eukaryotic gene order browser. *Bioinformatics*, 27(8):1150–1151, 2011.

[69] A. Louis, N. T. T. Nguyen, M. Muffato, and H. R. Crollius. Genomicus update 2015: Karyoview and matrixview provide a genome-wide perspective to multispecies comparative genomics. *Nucleic acids research*, 43(D1):D682–D689, 2015.

[70] E. Lyons, B. Pedersen, J. Kane, M. Alam, R. Ming, H. Tang, X. Wang, J. Bowers, A. Paterson, D. Lisch, et al. Finding and comparing syntenic regions among arabidopsis and the outgroups papaya, poplar, and grape: Coge with rosids. *Plant physiology*, 148(4):1772–1781, 2008.

[71] S. Marcus, H. Lee, and M. C. Schatz. Splitmem: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics*, 30(24):3476–3483, 2014.

[72] T. Marschall, M. Marz, T. Abeel, L. Dijkstra, B. E. Dutilh, A. Ghaffaari, P. Kersey, W. Kloosterman, V. Makinen, A. Novak, et al. Computational pan-genomics: Status, promises and challenges. *bioRxiv*, page 043430, 2016.

[73] I. Minkin, A. Patel, M. Kolmogorov, N. Vyahhi, and S. Pham. Sibelia: a scalable and comprehensive synteny block generation tool for closely related microbial genomes. In *Algorithms in Bioinformatics*, pages 215–229. Springer, 2013.

[74] I. Minkin, S. Pham, and P. Medvedev. Twopaco: An efficient algorithm to build the compacted de bruijn graph from many complete genomes. *arXiv:1602.05856*, 2016.

[75] G. Moore. Cramming more components onto integrated circuits. *Readings in computer architecture*, page 56, 1965.

[76] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[77] C. J. Mungall, D. B. Emmert, F. Consortium, et al. A chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23(13):i337–i346, 2007.

[78] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[79] C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, 302(1):205–217, 2000.

[80] M. Novo, F. Bigey, E. Beyne, V. Galeote, F. Gavory, S. Mallet, B. Cambon, J.-L. Legras, P. Wincker, S. Casaregola, et al. Eukaryote-to-eukaryote gene transfer events revealed by the genome sequence of the wine yeast saccharomyces cerevisiae ec1118. *Proceedings of the National Academy of Sciences*, 106(38):16333–16338, 2009.

[81] I. Pagani, K. Liolios, J. Jansson, I.-M. A. Chen, T. Smirnova, B. Nosrat, V. M. Markowitz, and N. C. Kyrpides. The genomes online database (gold) v. 4: status of genomic and metagenomic projects and their associated metadata. *Nucleic acids research*, 40(D1):D571–D579, 2012.

[82] M. M. A. Patwary, R. H. Bisseling, and F. Manne. Parallel greedy graph matching using an edge partitioning approach. In *Proceedings of the fourth international workshop on High-level parallel programming and applications*, pages 45–54. ACM, 2010.

[83] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[84] S. K. Pham and P. A. Pevzner. Drimm-synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics*, 26(20):2509–2516, 2010.

[85] T. Ramaraj, A. Cleary, J. Mudge, and B. Mumey. Investigating frequented regions (frs) in a yeast pan-genome. In *Plant and Animal Genome XXV Conference (PAG)*, 2017.

[86] M. Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303, 2008.

[87] J. Shima, A. Ando, and H. Takagi. Possible roles of vacuolar h+-atpase and mitochondrial function in tolerance to air-drying stress revealed by genome-wide screening of saccharomyces cerevisiae deletion strains. *Yeast*, 25(3):179–190, 2008.

[88] F. Sigaux. Cancer genome or the development of molecular portraits of tumors. *Bulletin de l'Academie nationale de medecine*, 184(7):1441–7, 1999.

[89] R. N. Smith, J. Aleksic, D. Butano, A. Carr, S. Contrino, F. Hu, M. Lyne, R. Lyne, A. Kalderimis, K. Rutherford, et al. Intermine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data. *Bioinformatics*, 28(23):3163–3165, 2012.

[90] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[91] H. Takagi, F. Iwamoto, and S. Nakamori. Isolation of freeze-tolerant laboratory strains of saccharomyces cerevisiae from proline-analogue-resistant mutants. *Applied microbiology and biotechnology*, 47(4):405–411, 1997.

[92] H. Takagi, K. Sakai, K. Morida, and S. Nakamori. Proline accumulation by mutation or disruption of the proline oxidase gene improves resistance to freezing and desiccation stresses in saccharomyces cerevisiae. *FEMS microbiology letters*, 184(1):103–108, 2000.

[93] H. Takagi, M. Takaoka, A. Kawaguchi, and Y. Kubo. Effect of l-proline on sake brewing and ethanol stress in saccharomyces cerevisiae. *Applied and environmental microbiology*, 71(12):8656–8662, 2005.

[94] H. Tettelin, V. Masignani, M. J. Cieslewicz, C. Donati, D. Medini, N. L. Ward, S. V. Angiuoli, J. Crabtree, A. L. Jones, A. S. Durkin, et al. Genome analysis of multiple pathogenic isolates of streptococcus agalactiae: implications for the microbial pan-genome. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13950–13955, 2005.

[95] G. Vernikos, D. Medini, D. R. Riley, and H. Tettelin. Ten years of pan-genome analyses. *Current opinion in microbiology*, 23:148–154, 2015.

[96] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[97] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[98] Y. Wang, H. Tang, J. D. DeBarry, X. Tan, J. Li, X. Wang, T.-h. Lee, H. Jin, B. Marler, H. Guo, J. C. Kissinger, and A. H. Paterson. Mcscanx: a toolkit for detection and evolutionary analysis of gene synteny and collinearity. *Nucleic Acids Research*, 40(7):e49, 2012.

[99] J. W. Wenger, J. Piotrowski, S. Nagarajan, K. Chiotti, G. Sherlock, and F. Rosenzweig. Hunger artists: yeast adapted to carbon limitation show trade-offs under carbon sufficiency. *PLoS Genet.*, 7(8), 2011.

[100] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proceedings of ACM SIGKDD*, pages 194–203, 2001.

[101] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proc. of SIGKDD*, pages 344–353, 2004.

APPENDICES

APPENDIX A

PUBLICATIONS, PRESENTATIONS, AND POSTERS

A. Cleary and A. Farmer. Genome context viewer: visual exploration of multiple annotated genomes using microsynteny. *Bioinformatics*, 2017 [journal]

A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Exploring frequented regions in pan-genomic graphs. In *8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB)*, 2017 [conference]

A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Approximate frequent subpath mining applied to pangenomics. In *International Conference on Bioinformatics and Computational Biology (BICoB)*, 2017 [conference]

A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Investigating frequented regions (frs) in a yeast pan-genome. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [poster]

A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Mining frequent subpaths in pan-genomes. In *11th Annual Sequencing, Finishing, and Analysis in the Future Meeting (LANL SFAF)*, 2016 [presentation]

S. Dash, J. D. Campbell, E. K. Cannon, A. M. Cleary, W. Huang, S. R. Kalberer, V. Karingula, A. G. Rice, J. Singh, P. E. Umale, et al. Legume information system (legumeinfo. org): a key component of a set of federated data resources for the legume family. *Nucleic acids research*, 44(D1):D1181–D1188, 2016 [journal]

T. Ramaraj, A. Cleary, J. Mudge, and B. Mumey. Investigating frequented regions (frs) in a yeast pan-genome. In *Plant and Animal Genome XXV Conference (PAG)*,

2017 [poster]

W. Huang, S. Dash, A. Cleary, A. Rice, S. Hokin, J. Campbell, J. Berendzen, M. Hadres, P. Umale, N. Weeks, A. Wilkey, A. Farmer, S. Cannon, and E. Cannon. Peanutbase.org: A genomic and genetic data resource to support crop improvement in peanut. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [poster]

J. Berendzen, E. Cannon, A. Cleary, S. Dash, S. Hokin, W. Huang, A. Rice, P. Umale, N. Weeks, A. Wilkey, A. Farmer, S. Cannon, and M. Hadres. Legumeinfo.org: Legume research and trait data that is findable, accessible, interoperable and re-usable. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [poster]

A. Farmer, A. Cleary, S. Dash, A. Rice, J. Berendzen, S. Hokin, M. Hadres, P. Umale, J. Campbell, W. Huang, N. Weeks, A. Yadav, A. Wilkey, D. Grant, R. Nelson, K. Feeley, V. Carollo Blake, E. Cannon, V. Krishnakumar, S. Cannon, A. Chan, E. Lyons, C. Town, and D. Fernandez-Baca. The legume information system (http://legumeinfo.org) and the legume federation (http://legumefederation.org): Comparative genomics and genetics through cooperative resource development. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [presentation]

A. Farmer and A. Cleary. Genome context viewer: A user-friendly web application for visualizing and mining shared gene content among multiple genomes. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [poster]

S. Dash, J. Berendzen, J. Campbell, E. Cannon, S. Cannon, A. Cleary, A. Farmer, M. Hadres, S. Hokin, W. Huang, A. Rice, P. Umale, N. Weeks, and A. Wilkey.

Customizing tripal sites with non-tripal components at the legume information system and peanutbase. In *Plant and Animal Genome XXV Conference (PAG)*, 2017 [presentation]

A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge. Mining frequent subpaths in pan-genomes. In *11th Annual Sequencing, Finishing, and Analysis in the Future Meeting (LANL SFAF)*, 2016 [poster]

A. Cleary, T. Ramaraj, J. Mudge, and B. Mumey. Pangenomics: Persistent homology for pan-genome analysis. In *11th Annual Sequencing, Finishing, and Analysis in the Future Meeting (LANL SFAF)*, 2016 [poster]

J. Campbell, S. Dash, E. Cannon, A. Cleary, W. Huang, S. Kalberer, A. Rice, J. Singh, P. Umale, N. Weeks, A. Wilkey, J. Town, Christopher DeBerry, D. Fernandez-Baca, A. Farmer, and S. Cannon. What's new in the legume information system and the federated legume database initiative. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016 [presentation]

A. Farmer, A. Cleary, A. Rice, P. Umale, S. Hokin, S. Dash, J. Campbell, W. Huang, N. Weeks, A. Wilkey, D. Grant, R. Nelson, K. Feeley, V. Krishnakumar, A. Yadav, J. DeBerry, D. Fernandez-Baca, E. Cannon, C. Town, and S. Cannon. The legume information system and the legume federation: Working together for the legume-fed world. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016 [presentation]

S. Dash, W. Huang, J. Campbell, J. Singh, A. Rice, P. Umale, A. Cleary, S. Kalberer, N. Weeks, V. Karingula, P. Gupta, S. Gunda, E. Cannon, A. Farmer, and S. Cannon.

Lis (legume information system): A clade based web resource for legumes. In *Plant and Animal Genome XXIV Conference (PAG)*, 2016 [poster]

A. Cleary, L. Ren, S. Cannon, and A. Farmer. Tools for phylogenomic exploration within and among clade-oriented databases. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015 [poster]

S. Cannon, A. Farmer, P. Umale, H. Lokhande, A. Cleary, N. Weeks, S. Kalberer, E. Cannon, S. Dash, D. Bitragunta, and J. Singh. The legume information system (legumeinfo. org) 2015. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015 [poster]

A. Farmer, S. Cannon, S. Kalberer, J. Singh, E. Cannon, P. Umale, H. Lokhande, A. Cleary, N. Weeks, V. Karingula, and S. Dash. The legume information system (legumeinfo. org) 2015. In *Plant and Animal Genome XXIII Conference (PAG)*, 2015 [presentation]