Exploring Frequented Regions in Pan-Genomic Graphs

Alan Cleary, Thiruvarangan Ramaraj, Indika Kahanda, Joann Mudge, and Brendan Mumey,

Abstract—We consider 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). In this work, we formalize the FR problem and describe an efficient algorithm for finding FRs. Subsequently, we propose some 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* (bacterium) and *Saccharomyces cerevisiae* (yeast). We corroborate the biological relevance of FRs such as 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.

Index Terms—Pan-genomics, classification, visualization

1 Introduction

2 Introduction

With the unit cost of DNA sequencing continuing on a downward trajectory, research institutions and genome sequencing consortia have proposed and implemented strategies to sequence multiple genomes per species[1] [1]. This has resulted in a large volume of genome sequences of species that represent major phylogenetic clades [2]. As such, there is a need for novel methods of analysis that can scale to such unprecedented quantities of genomic data [3], [4].

A pan-genome represents the collective genomic information of multiple individuals or organisms from a related group or species [5]. Recently, pan-genomes have been represented using colored De Bruijn graphs [6], [7], [8], which can succinctly capture structural variation within a population by giving each genome's path through the graph a unique color [9].

An important problem in population genomics is the identification of synteny, that is, sequences that are (in)exactly preserved within the population. Though there exist a variety of whole genome pairwise and multiple sequence alignment approaches, these are generally not scalable. In this work we present the Frequented Regions (FR) problem, which mines regions of a graph that are frequently traversed by a set of paths in the graph. 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.

In Section 3 we discuss work related to the FR problem and in Section 4 we formalize the FR problem and discuss its computational complexity. In Section 5 we present an efficient algorithm for finding FRs, describe some applications in Section 6 and in Section 7 provide results on two pan-genome test cases. Lastly, in Section 8 we discuss future work and make closing remarks.

- *Indika Kahanda and Brendan Mumey are with the Gianforte School of Computing, Montana State University, Bozeman, MT, USA. E-mail: brendan.mumey@montana.edu*
- *Alan Cleary, Thiruvarangan Ramaraj and Joann Mudge are with the National Center for Genome Resources, Santa Fe, NM, USA.*

Manuscript received November 15, 2017; revised November 15, 2017.

1. See http://www.1000genomes.org/, http://www.1001genomes.org/, and http://www.100kgenome.vetmed.ucdavis.edu/.

3 Related Work

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 [10]. It works by performing an exhaustive search of the itemset space and has an exponential run-time complexity. There are variations of FI that are tolerant to noise in the data [11], [12], [13]. 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 [14]. It is a well studied problem [15] commonly applied to biological datasets [16], [17], [18]. 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 [19]. Yet another related problem is Frequent Subpath Mining, which is described in [20]. 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, the algorithm is exhaustive and has exponential run-time complexity. In [21] the authors formulate an approximate version of the Frequent Subpath Mining problem and present an efficient algorithm for mining syntenic regions from pan-genome graphs.

There exist a variety of tools for pan-genome analysis [22]. Unfortunately few of these are concerned with mining synteny [23] or scale beyond populations of microbial genomes, both of which are needs of the pan-genomics community [4]. One exception to the lack of synteny-based approaches is Sibelia [24], which determines syntenic regions from pan-genomes by iteratively eliminating bubbles in a De Bruijn graph with the sequence modification algorithm [25]. 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 7.

4 Problem Formulation

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 (compressed) De Bruijn graph (cDBG) of a pan-genome composed of multiple genomic sequences. Nodes in G represent specific k-mers (or $\geq k$-mers if the graph has been compressed). An edge (u, v) is present if and only if the last $k-1$ nucleotides of u match the first $k-1$ nucleotides of v. Each genomic 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 $p[i,j]$ in the De Bruijn graph.

Definition 4.1. 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) for a computational definition).
2) $p[i,j]$ is *maximal* in the sense that it cannot be extended to either the left or right, i.e. $\nexists$ (α, κ)-supporting subpath $p[i',j']$ s.t. $[i,j] \subsetneq [i',j']$.

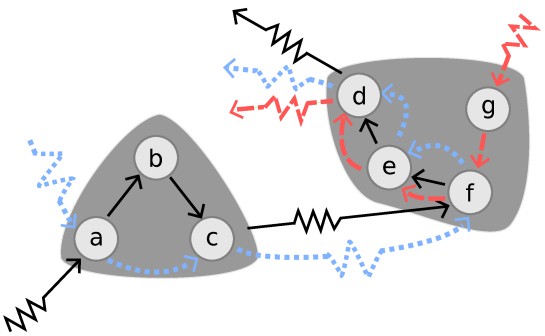


Fig. 1: Example FRs: Assuming $\alpha \leq \frac{2}{3}$, the left group of nodes $C_L = \{a, b, c\}$ forms a FR with support 2 (from the black and blue paths). The right group of nodes $C_R = \{d, e, f, g\}$ forms a FR with support 3 (from the back, blue and red paths). If C_L and C_R were merged, the 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.

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 implements this idea.

Definition 4.2. A *frequented region* (FR) is a tuple (C, S), where C is a set of De Bruijn 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

$$\text{support}(C, S) = |S| \tag{1}$$

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

$$\text{average-length(FR)} = \frac{\sum_{p[i,j] \in S} |\text{seq}(p[i,j])|}{|S|}. \tag{2}$$

Fig. 1 provides an example. The computational problem considered is to find FRs that have high support and high average length.

Computational complexity: Counting exact frequent itemsets is known to be #P-complete [26]. 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.

5 Finding Frequented Regions

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 bottom-up, agglomerative fashion, where each De Bruijn node starts in its own cluster and pairs of clusters are iteratively merged. When an FR is created for a set of nodes C, the corresponding set of supporting subpaths S must be found. This is

Algorithm 1 Compute supporting path segments

1: **function** COMPUTESUPPORT(C,p)
2: Let $S = \emptyset$
3: Let $m = [i : p[i] \in C]$
4: start $= 1$
5: **while** start $\leq |m|$ **do**
6: $i =$ start
7: **while** $i < |m|, \text{gap}(p, m, i) \leq \kappa$ **do**
8: $i = i + 1$
9: **end while**
10: **if** $(i - \text{start} + 1) \geq \alpha|C|$ **then**
11: $S = S \cup p[m[\text{start}], m[i]]$
12: **end if**
13: start $= i + 1$
14: **end while**
15: **return** S
16: **end function**

Algorithm 2 Evaluate FR merge

1: **function** EVALMERGE($(C_L, S_L), (C_R, S_R)$)
2: Let $C = C_L \cup C_R$.
3: Let $S = \emptyset$.
4: **for** $p \in P$ **do**
5: $S = S \cup$ COMPUTESUPPORT(C, p)
6: **end for**
7: **return** S
8: **end function**

accomplished by the COMPUTESUPPORT subroutine shown as Algorithm 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 $p[m[i], m[i+1]]$ matches a node in C, so the insert gap length is

$$\text{gap}(p, m, i) = \max(0, |\text{seq}(p[m[i], m[i+1]])| - |\text{seq}(p[m[i]])| - |\text{seq}(p[m[i+1]])|). \quad (3)$$

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. The procedure terminates when no new FRs with positive support can be found. Pseudocode of the procedure is provided in Algorithm 3. Line 10 can be computed efficiently using a priority queue of possible merges. When a new FR (C, S) is formed by merging (C_L, S_L) and (C_R, S_R), other potential merges involving either of these FRs remaining in the queue must be updated to be potential merges with the newly formed FR. After Algorithm 3 completes, the FRs will form a *hierarchical clustering* with one or more root FRs. 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 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 7 to such FRs.

Algorithm 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 $S =$ 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**

5.1 Reverse-complement support

Genes and other relevant sequence features can be encoded in either the forward or reverse-complement direction on a particular DNA sequence. We adopt a simple approach for detecting sequences that support a particular FR in the reverse-complement direction. For each sequence in the data set we generate a path p in the forward direction and a corresponding path $\bar{p}^r$ in the reverse-complement direction. This permits FRs to have supporting subpaths from either p or $\bar{p}^r$. Observe that for any FR (C, S), there is a reverse-complement version of it $(\bar{C}^r, \bar{S}^r)$, in which the FR nodes and supporting subpaths are reverse-complemented. To reduce the number of iFRs reported, we only report FRs with supporting path sets that are comprised of at least 50% paths in the forward direction. If an FR fails to be reported, there is a corresponding reverse-complemented FR that will be. This approach can be easily implemented by appending all reverse-complemented sequences from the original FASTA file to the end of the FASTA file, prior to constructing the compressed De Bruijn graph. Our `FindFRs` software implements this approach and allows users to specify if reverse-complemented paths should be considered.

5.2 Weighted Support

In Algorithm 3, FR merges are greedily performed in most-support-first order. We note that this metric can be weighted so that additional FR properties are emphasized during the selection process. For example, the total support of each potential merge can be weighted by the average penetrance of the supporting paths. This can prevent otherwise coherent FRs from being involved in tenuous merges due to lenient parameterization. Other weights could be based on insertion length, the size of the FRs being merged, or even classification labels associated with the sequences.

Algorithm 4 Report interesting FRs

```
1: procedure EXPLORE((C, S), m)
2:     if support(C, S) ≥ m then
3:         report (C, S).
4:     end if
5:     if |C| ≥ 1 then
6:         Let (C_L, S_L) = left-child(C, S)
7:         Let (C_R, S_R) = right-child(C, S)
8:         Let m' = max(m, support(C, S) + 1)
9:         EXPLORE((C_L, S_L), m')
10:        EXPLORE((C_R, S_R), m')
11:    end if
12: end procedure
```

Algorithm 5 Serial maximal weighted graph matching

```
1: procedure SERIALMATCHING(V, E)
2:     for v ∈ V do
3:         π(v) = ∞
4:     end for
5:     for {u, v} ∈ E in order of descending weight do
6:         if π(u) == ∞ and π(v) == ∞ then
7:             π(u) = v
8:             π(v) = u
9:         end if
10:    end for
11: end procedure
```

Algorithm 6 Parallel maximal weighted graph matching

```
1: procedure PARALLELMATCHING(V, E)
2:     while E ≠ ∅ do
3:         for v ∈ V do parallel
4:             π(v) = v.neighbors.first
5:         end for
6:         for v ∈ V do parallel
7:             u = π(v)
8:             if π(u) == v then
9:                 E = E \ {v, u}
10:                RemoveNode(E, v)
11:                RemoveNode(E, u)
12:            else
13:                π(v) = ∞
14:            end if
15:        end for
16:    end while
17: end procedure
18: procedure REMOVENODE(E, v)
19:    for u ∈ v.neighbors do
20:        u.neighbors = u.neighbors \ v
21:        E = E \ {v, u}
22:    end for
23: end procedure
```

5.3 Parallelization

The main loop in Algorithm 2 is trivially parallelizable, as computing the supporting subpaths for any path $p \in P$ is independent of the other paths. It is also possible to parallelize the main loop of Algorithm 3. We observe that merging FRs can be done consistently in parallel, provided no FR is involved in more than one merge. This is equivalent to the requirement that edges representing the FR pairs chosen for merging form a *matching*. To parallelize Algorithm 3, we first compute the score of each merger edge, using Algorithm 2, which can be done in parallel since each merger edge's score is independent of the others. We compute a *maximal weighted matching* which will represent the FR mergers to do in parallel. The matching found is *maximal* (no further edges can be added), rather than *maximum* (best possible), as matched edges are added greedily.

Finding maximal weighted matchings is a well-studied problem [27] for which there exists fast parallel [28] and distributed algorithms [29]. In this work, we consider two maximal matching algorithms: 1) A simple serial sorting based algorithm that greedily selects edges in greatest-weight-first order and achieves a $\frac{1}{2}$-approximation of the optimal maximum weight matching [30]; see Algorithm 5. (We use this as a baseline to measure the speedup of our parallel implementation.) And 2) a parallelized version of the serial algorithm based on locally dominant edges [31]; see Algorithm 6.

Both algorithms utilize a map π to report what vertex, if any, each vertex is matched to. Specifically, Algorithm 5 works by initializing each vertex's entry in π to infinity (Line 3). It then iterates the edges in greatest-weight-first order, adding edges to the matching only if both its vertices' π values are infinity (Lines 5-10). In order to achieve parallelism, Algorithm 6 assumes each vertex v maintains a list of its neighbors in greatest-weight-first order, denoted v.neighbors. The algorithm works by iterating until the edge set E is empty. Each iteration begins by setting each vertex's π value to its most weighted neighbor, v.neighbors.first, in parallel (Lines 3-5). It then iterates the vertices in parallel, checking if each vertex's π value's π value is equal to itself - a matching. If so, it removes the corresponding edge from the edge set and itself from its neighbors' neighbor lists, otherwise, the vertex's π value is set to infinity (Lines 6-15 and 18-23).

5.4 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 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 leaves. Determining the next FR merger (internal node) can be done efficiently using a priority queue. After each FR merger, the queue needs to be updated. Observe that $E = O(V)$, since at the start of the algorithm each vertex has $O(1)$ incident edges (the in degree and out degree are both bounded by 4 in any De Bruijn graph built on DNA sequences), and $|E|$ can only decrease as merges happen. This means the queue can be updated in $O(V + L + V \lg V)$ time, where $O(V + L)$ accounts for computing the support of all new potential FR mergers with the newly-created FR. The overall time complexity is thus, $O(LV + V^2 \lg V)$.

Algorithm 5 computes a matching in $O(E \lg E)$ time, since it requires the edges be sorted by greatest weight. On average, the number of FRs merged at each iteration is half, meaning it takes $O(\lg V)$ iterations to construct the FR hierarchy on average, and $O(v)$ iterations in the worst case. Accounting for the computation of support and the decreasing number of edges at each iteration, this results in an average run-time complexity of $O(L \lg V + V \lg V)$, and a worst case complexity $O(LV + V \lg V)$.

In Algorithm 6, if the edge weights are distributed randomly, then the main loop is expected to terminate after $O(\lg E)$ iterations, though the worst case is $O(E)$. By representing each vertex's neighbor list with an efficient data structure, such as a linked list, the time complexity of all the vertex operations is dominated by the initial sorting of the vertex neighbors. Specifically, the accumulated work performed on all vertices is $\sum_{v \in V} O(\delta(v) \lg \delta(v)) = O(E \lg \Delta)$, where $\delta(v)$ is the degree of vertex v and Δ is the maximum degree in the graph. Since all vertices are iterated at each iteration of the algorithm, this gives an expected time complexity of $O(V \lg E)$ and a worst case complexity of $O(VE)$ [31]. Thus, the overall expected time complexity is $O(L \lg V + V \lg^2 V)$, and the worst case complexity is $O(LV + V^2 \lg V)$.

6 Applications

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

6.1 Classification 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 7 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) [32] have been shown to be an effective approach to classifying genomes based on shared genetic features. Traditionally such distinctions are done with Genome Wide Association Studies (GWAS) and/or Principal Component Analysis (PCA) [33], however, SVMs have been shown to be more effective at this task than GWAS/PCA [34]. Therefore, in this work we use Support Vector Machines for classifying genomes within a pan-genome based on their FR content as described below.

Specifically, each example (i.e. genome) is represented as an n-dimensional vector (n is the total number of FRs used) in which each individual component of a vector corresponds to the number of times a certain FR occurs with that genome. We use these vectors as input to our SVM model [35] and evaluate its accuracy in predicting the correct industrial origin based on the FRs they are associated with (see Section 7.2.3 for results of this study).

However, depending on the parameters used and size of the pan-genome, a large number of iFRs may be identified by Algorithm 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

$$c_{ij} = \sum_{p \in G_i} \text{support}_p(\text{FR}_j), \tag{4}$$

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 $T_i = \sum_j c_{ij} + 1$ be the total count of iFRs found in group G_i (a pseudo-count of 1 is added to the count for each group) 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 $f_i = T_i/T$. The probability of observing the group counts for FR_j in a random FR is multinomially distributed with bin probabilities $\langle f_i \rangle$,

$$\Pr(\text{FR}_j) = \frac{n!}{c_{1j}! \cdots c_{kj}!} f_1^{c_{1j}} \cdots f_k^{c_{kj}}, \tag{5}$$

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.) The $\Pr(\text{FR}_j)$ values are used later to rank iFRs and only some number of the top-ranked iFRs are kept as features.

6.2 Graph simplification

Visualizing a pan-genome graph is a difficult task, especially when dealing with larger, complex genomes. It is relatively simpler 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 [6] 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. Fig. 4 shows an application of the approach to visualize paths in a yeast pan-genome.

7 Experimental Results

The algorithm was implemented in Java and is called `FindFRs`. We evaluated `FindFRs` on two datasets, *Staphylococcus aureus* and *Saccharomyces cerevisiae*. The *Staphylococcus*

aureus dataset was used to compare against Sibelia [24], 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 FindFRs and exhibit a variety of FR-based analyses.

We used [7] to construct the cDBGs for each dataset. Besides the main parameters α and κ, we also include two other parameters *minSup* and *minSize*, which serve to limit the amount of output generated; only iFRs whose support and size (number of cDBG nodes) are at least these minimums are reported.

In general, FindFRs parameters should be chosen relative to the desired level of conservation and prevalence of the synteny blocks to be mined. Furthermore, the user should be mindful of the interplay between these parameters and the quality of the results. For example, if $\alpha = 1$ and $\kappa = 0$, then only sequences that traverse the exact same sequence of cDBG nodes will be identified. Conversely, if α is small and κ is large, then the relationship among the sequences identified will be tenuous, at best. Although the legitimacy of the evolutionary relationships among the sequences identified should be determined by a domain expert, understanding that more lenient parameters increase the likelihood of false positives is important.

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.

7.1 *Staphylococcus aureus*

In this section, we compare FindFRs to Sibelia [24]. We selected Sibelia because it is the only other tool for finding synteny blocks in pan-genomic data, specifically from a De Bruijn graph. We use the same four *Staphylococcus aureus* strains (JH1, N315, TW20 and MSSA476) that were used in the Sibelia paper for comparison. Both programs were run with $k \in \{25, 100, 500, 1000\}$ values. Two versions of the FindFRs were used, one that performs merges based on FR support and one that performs merges based on support weighted by the average penetrance of the supporting paths, as described in Section 5.2. Sibelia was run with its default parameters. The FindFRs parameters were selected to be minsup $= 2$, because we were interested in identifying syntenic blocks present in two or more strains – the same as Sibelia; $\alpha = 0.5$, because it is ambiguous what sequence identity the Sibelia blocks would have, so 50% seemed neither too stringent or lenient; $\kappa = 0$, because Sibelia does not explicitly allow for insertions; and minsize $= 2$, because we did not want to identify trivial blocks (single de Bruijn nodes). The results are shown in Table 1, *iFRs and *iFR roots denote the results of FindFRs using weighted support.

As we can see, FindFRs reports a higher number of frequented regions with and without weighted support compared to Sibelias synteny blocks, indicating the finer scale of partitioning regions based on biological significance. For the non-weighted test case, we estimated percent overlap between regions reported by these two algorithms: For all k-mer values, approximately 98% of FindFRs FRs overlapped with Sibelias synteny blocks. Furthermore, we also computed overlap in terms of sequence percentage (i.e. what percentage of the Sibelia block sequences are contained in FindFRs FRs, and vice versa). For $k = 25$, alignments showed 91% of Sibelia Synteny block sequence base-pairs were contained within FindFRs frequented regions and only 82% of FindFRs frequented regions sequence base-pairs were contained within Sibelia Synteny blocks, indicating FindFRs FRs captured most of Sibelia Synteny blocks at the nucleotide base-pair level.

TABLE 1: Sibelia versus FindFRs comparison

S. aureus (4 Strains)	$k = 25$	$k = 100$	$k = 500$	$k = 1000$
Synteny blocks	142	143	132	132
iFRs	$10,740$	$1,334$	$2,011$	$1,351$
*iFRs	172	461	$1,064$	383
iFR roots	392	122	97	134
*iFR roots	50	22	180	101

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

Lastly, to compare the run times of both programs, we tested a slightly larger dataset consisting of 31 *Staphylococcus aureus* ($\approx$ 90 Mb) strains. For $k = 25$, Sibelia took 186 minutes, whereas FindFRs took only 106 seconds (no threading).

7.2 *Saccharomyces cerevisiae*

Yeast (*Saccharomyces cerevisiae*) is a well studied model system with some published comparative genomic and pan-genomic work [36], [37], allowing us to use the existing knowledge base to assess the 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 (http://www.yeastgenome.org/). The dataset consisted of 55 assemblies from 48 yeast strains over a wide range of industrial applications and geographic origins; see Table 2 and Table S1 in the supplementary materials. We constructed Yeast pan-genome cDBGs for $k \in \{25, 100, 500, 1000\}$. Again, we chose these values because it is still not well understood what k values are appropriate for pan-genome analysis. Here, no weighted support was used, and we varied the α and κ parameters, as indicated in the various experiments. As a sample run, we ran FindFRs with parameter values $\alpha = 0.7$, $\kappa \in \{0, 3000\}$, minsup $= 5$, and minsize $= 5$. Table 3 indicates the size of the cDBGs created and the number of iFRs found ($\kappa = 0$ only). Figure 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.

7.2.1 Parallelization Speedup

FindFRs implements both the serial and parallel maximal matching algorithms, Algorithm 5 and Algorithm 6, respectively. Since the number of paths in our datasets are relatively small, the computation of the maximal matchings dominates the run-time complexity. As such, here we

TABLE 2: The 48 yeast strains with usage and region listed where known.

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

TABLE 3: Number of cDBG nodes and iFRs ($\alpha = .0.7$,$\kappa = 0$, minSup $= 5$, minSize $= 5$)

k	cDBG nodes	iFRs
25	2,260,767	479,349
100	1,758,760	366,502
500	890,055	107,739
1000	443,764	31,500

compare the run-time of the serial and parallel versions of `FindFRs` in terms of the matching computed during the first iteration of the main loop in Algorithm 3, which is the largest graph for which a matching is computed during the run-time of `FindFRs`. These experiments were run with parameter values $\alpha = 0.7$, $\kappa = 1$, minSize $= 1$, and minSupport $= 2$. Since we are only measuring the run-times of the first graph matching computed for each dataset the size of the graphs and therefore the run-times are invariant with respect to the parameter values. The results are shown in Figure 3.

As we can see, when run with a single thread, Algorithm 6 has approximately the same run-time as Algorithm 5, incurring approximately 0.5 seconds of threading overhead on the smaller graphs ($k = 1000$ and $k = 500$).

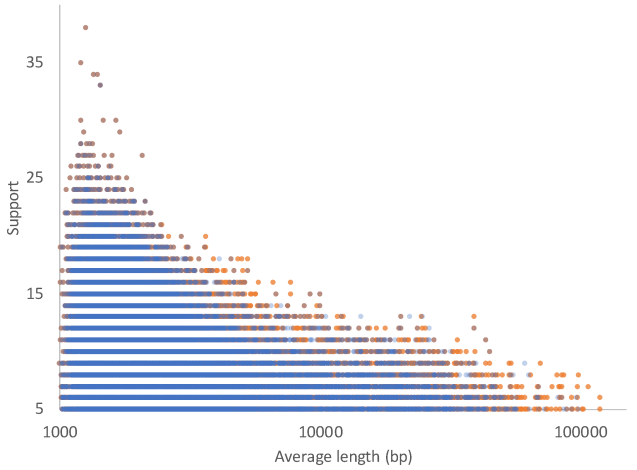


Fig. 2: Distribution of iFR support versus average length. Allowing insertions ($\kappa = 3000$) (orange points) creates some longer FRs vs. no insertions ($\kappa = 0$) (blue points). ($k = 1000$, $\alpha = 0.7$, minSup $= 5$, minSize $= 5$)

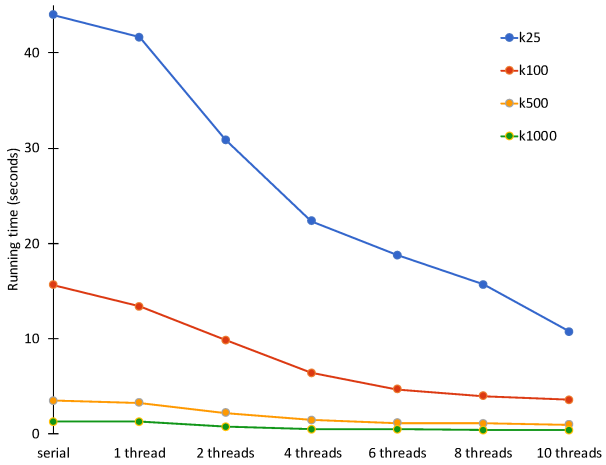


Fig. 3: Running times of the serial and parallel maximal matching algorithms - Algorithm 5 and Algorithm 6, respectively - on the *Saccharomyces cerevisiae* cDBGs (Table 3).

This indicates that it is indeed running in the expected amount of time, rather than the worst case. We also see that the total speed-up and the point at which the number of threads yields diminishing returns is proportional to the size of the graph. This suggests that our method can scale to much larger pan-genomic graphs than considered here.

7.2.2 Consistency with Yeast Biology

Because yeast is one of the simplest eukaryotes, it has been an important model system for genomic research [36], [37]. Yeast is also economically important. It is used industrially in bread, biofuels, and alcoholic beverages, including wine, sake, and ale. Further, yeast is important in other contexts, being found in natural environmental systems, including as pathogens of humans.

The ability to thrive in harsh environments, which is required for industrial specialization in yeast, is often ob-

tained by incorporating sets of genes from other organisms into their own genomes [36], [37]. To determine whether we could identify some of these foreign genes, we looked at novel genomic regions (insertions) that have been validated in the EC1118 yeast strain. These insertions contain genes that allow EC1118, which is used in wine-making, to grow in the presence of alcohol. EC1118 has three novel insertions compared to S288C [38]. While S228C is not in our dataset, we compared to laboratory strains that were descended from S288C (BY4741, BY4742, FY1679, X2180-1A, and YPH499). All three insertions were uncovered using our FR approach (Fig 4). In order to find FRs that would be large enough for paths to contain multiple genes but to allow for divergence between samples, we used a relatively large k value of 500 but a relatively lenient α value of 0.7.

The shortest insertion is a 17kb (kilobasepair or 1,000 base pair) novel insertion on chromosome XIV [38]. Five novel genes are inserted near the end of the chromosome between S288C genes YNL037C and YNL038W. This novel region was found on EC1118 sequence accession FN393084.1, which contained the insertion with more typical yeast DNA on either side (Fig. 4A). Multiple alcohol-related strains have versions of this insertion as indicated by the thicker green edges as well as green edges that follow alternate paths through the region. None of the laboratory strains, including the five S288C-derived strains, show support of any of the inserted FRs. Most of the nodes in the insertion occur only in alcohol-related strains (green) (Fig 4a), confirming that this region could be important in determining alcohol tolerance.

The 38kb insertion of foreign DNA in EC1118 near one end of chromosome VI [38] ($k = 500$, $\alpha = 0.7$, minSup $= 4$, minSize $= 4$) Fig 4B. This is a more complicated insertion that includes some deletions and rearrangements of chromosomal regions. Briefly, the tip of chromosome VI was deleted and replaced by a segment of DNA that moved over from chromosome VIII. The novel 38kb insertion was then stuck onto the very end of the chromosome.

Many of the EC1118 FRs that were moved to chromosome VI from chromosome VIII (12kb) are shared by the S288C-derived accessions (light blue). This is not surprising because the S288C-derived accessions have this region on chromosome VIII. None of the FRs in the 23kb 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 5kb region that was moved to chromosome X in EC1118 are contained in EC1118, either, even though this region occurs on chromosome X. 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 38kb from the end of chromosome VI. No FRs were found in the 38kb novel region, presumably because there wasn't enough support among yeast strains 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 insertion is a 65kb region that replaced the last 9.7kb of chromosome XV [38]. For simplicity, only EC1118 (green) and BY4741 (light blue) compared to all others (gray) are shown in Fig 4C and their chromosome XV paths have been highlighted ($k = 500$, $\alpha = 0.7$). The last gene shared by EC1118 and BY4741, YOR393W, has some

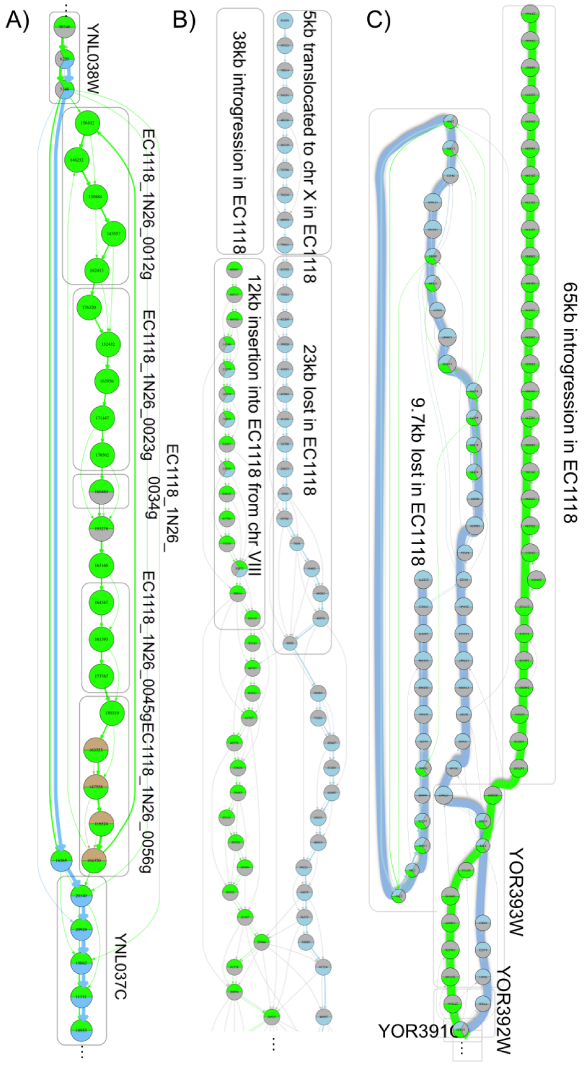


Fig. 4: The three novel insertion regions [38] showing paths through FRs from EC1118 (green) compared to S288C (represented by the S288C-derived strain BY4741, light blue). (A) 17kb insertion on chromosome XIV showing support from alcohol (green, including wine, sake, ale and bioethanol strains), laboratory (light blue), bakery (brown), and other (gray) strains. (B) A complex insertion event on chromosome VI shows that the tip of the chromosome was deleted and replaced by a DNA segment from chromosome VIII and the novel insertion. Strains included were EC1118 (green), S288C-derived (light blue, strains BY4741, BY4742, FY1679, X2180-1A, and YPH499), and other (gray). (C) A 65kb novel insertion replaced the end of chromosome XV in EC1118. Shown are EC1118 (green) and BY4741 (light-blue).

shared FRs and some FRs that have diverged. Thereafter, the paths diverge. The EC1118 65kb novel insertion, shared with other yeast strains, is highly conserved as there are not alternate paths through the region. The 9.7kb region in BY4741, which was deleted in EC1118 actually has FRs that are shared with EC1118 because these regions occur frequently near the ends of chromosomes.

Our pan-genomics approach allowed us to quickly go beyond confirming EC1118's inserted regions to exploring

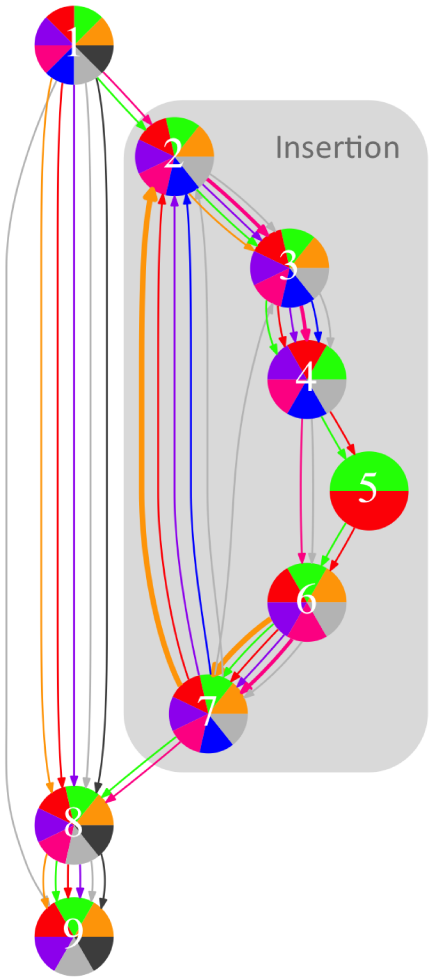


Fig. 5: The 17kb insertion that brought alcohol tolerance genes to EC1118 is shown. Yeast strains are highlighted that share four or more of the FRs (Lalvin QA23=pink, BC187=orange, L1528=purple, JAY291=red, and T73=blue). All of these strains are involved in alcohol production. The five S288C-derived yeast strains are represented in dark gray while all other strains are light gray. The number of copies of each path is represented by arrow weight. The novel inserted region is shaded.

these regions across multiple yeast genomes. As an example, we mined our graph for FRs from the 17kb EC1118 insertion in other yeast strains used in generating alcoholic beverages and biofuels ($k = 500$, $\alpha = 0.7$, $\kappa = 0$, minSup $= 2$, minSize $= 500$, reverse complement included). These parameters allowed us to collapse the insertion into fewer FRs but identification of the insertion was robust to the parameter change. Of 20 additional alcohol-related yeast strains, 9 shared FRs found in EC1118 in the 17kb insertion region. Five of these strains shared four or more FRs from the EC1118 17kb insertion and are highlighted, along with EC1118, in Fig 5.

Several interesting, biological observations are evident from the structure of the graph. None of the five S288C-derived laboratory lines (dark gray) have the inserted region on chromosome XIV nor in any other region of the genome. While the highlighted strains share many of the 17kb insertion FRs with EC1118, none share the same path. The most closely related path is Lalvin QA23 (pink), which not only shares all but one of the inserted FRs with EC1118 but also has these FRs integrated into the same region of the genome. This is evident by the connections between chromosome XIV nodes flanking the insertion (nodes 1 and 8) and the nodes on either end of the insertion (nodes 2 and 7, respectively). The fact that Lalvin QA23 has all of the novel nodes inserted into the same genomic region as EC1118 except one, suggests that they shared the same insertion event. While node 5 may have been lost from LalvinQA23, the fact that node 5 is so rare (present only in EC1118 (green) and JAY291 (red; bioethanol strain) but none of the other 46 yeast strains) suggests that it could represent another novel insertion.

While LalvinQA23 is the only strain with evidence suggesting that it shares an insertion site with EC1118, assembly fragmentation in some lines may prevent us from identifying their insertion site or confirming that their chr XIV does not have an insertion. Several lines, however, do show good evidence that the insertion occurred elsewhere. BC187 (wine; orange), L1528 (wine; purple) and JAY291 (biofuels; red) all have FR paths from node 1 to 8 to 9, just like the S288C-derived lines, indicating that this portion of chromosome XIV is intact and does not contain any insertions. Nevertheless, these strains clearly contain many of the novel FRs required for survival in high alcohol environments.

Indeed, some of these strains show multiple copies of these FRs, which may be important in increasing alcohol tolerance. BC187 has three genomic copies of the node 6 $\rightarrow$ node 7 $\rightarrow$ node 2 path that map to three different genomic regions. Intriguingly, Lalvin QA23, whose insertion so closely mirrors that of EC1118, appears to have two other genomic regions that contain subsets of the FR path on chromosome XIV. Finally, with the exception of the Lalvin QA23 (pink), there is rearrangement in the inserted regions compared to EC1118. Intriguingly, the four other strains (BC187, L1528, JAY291, and T73) all have node 7 leading into node 2. These two nodes make up opposite ends of the insertion in EC1118.

Our pan-genomics algorithm allowed us to quickly assay complex insertion events that are important for alcohol tolerance across multiple yeast strains. We were able to quickly identify yeast strains with similar regions inserted and hypothesize about which were independent insertion events. In addition to insertions, we identified complex rearrangements of the DNA that would be difficult to identify using standard DNA analysis techniques that work best when genes are in the same genomic location across yeast strains.

7.2.3 *Using FRs for classification*

In this Section, we explore the efficacy of classifying strains based on their mined FR content. As mentioned in Section 6.1, we use SVMs as our supervised learning model. Specifically, we applied two different one-against-rest multiclass SVM classifiers, one with linear kernels and one with degree 2 polynomial kernels, both with margin parameter $C = 10$. We chose one-against-rest due to its computational efficiency and interpretability. We normalized the data using L1 normalization and then used stratified cross-validation

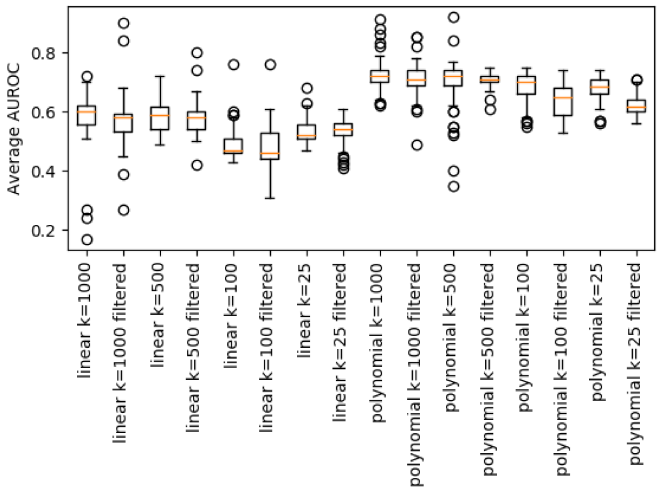


Fig. 6: Box plot of the AUROC curve results for both SVM configurations (linear vs degree 2 polynomial kernels) on unfiltered and filtered (top 1000, 500, and 250) *Saccharomyces cerevisiae* iFRs computed with `FindFrs` ($\alpha \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$, $\kappa = 0$, minSup $\in \{5, 10, 15, 20\}$, and minSize $\in \{5, 10, 25, 50, 100\}$).

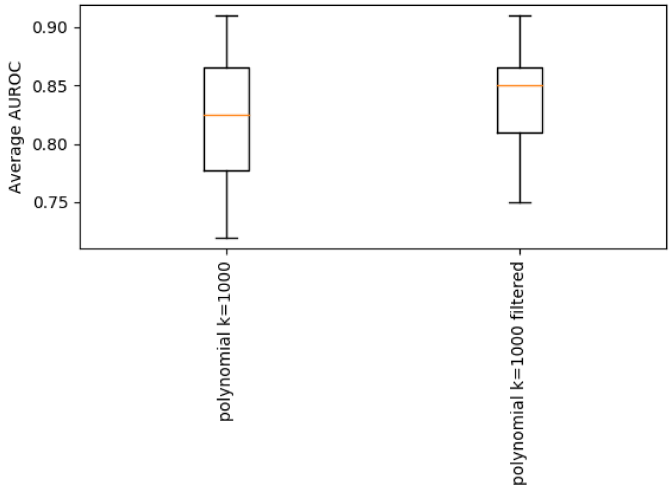


Fig. 7: Box plot of the AUROC curve results for the polynomial kernel SVM configuration on unfiltered and filtered (top 1000, 500, and 250) *Saccharomyces cerevisiae* iFRs computed with `FindFrs` ($\alpha \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$, $\kappa = 0$, minSup $\in \{5, 10\}$, and minSize $= 100$).

to evaluate the effectiveness of the model. In this work, we used these SVM configurations to predict the industrial origin for a given yeast strain.

We implemented the SVM models using the scikit-learn Python machine learning library[2]. We then used the models to classify the strains in the yeast dataset by their industrial application, or rather *source* from Table 2. The dataset is composed of 55 strains/examples annotated with nine distinct class labels, one label per example. Given the distribution of the source labels in the data set, we used two-fold stratified cross-validation. For $\alpha \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$, $\kappa = 0$, minSup $\in \{5, 10, 15, 20\}$, and minSize $\in \{5, 10, 25, 50, 100\}$, we computed the average Area Under Receiver Operating Characteristic (AUROC) curve [39] for 10 iterations of stratified cross-validated classifier training. We used such a broad set of parameters because it was unclear what parameters would yield FRs that are amenable to being used as classification features, if any. Additionally, we performed the same experiments while selecting the top 1000, 500, and 250 FRs (features) in the training examples using our multinomial-filter. The box plot of the SVMs' AUROC results is shown in Figure 6.

As we can see, both the linear and polynomial kernel classifiers generally perform better than random chance, though the polynomial kernel is consistently better than the linear. We also observe that the multinomial-filter appears to slightly decrease the classification power of the SVMs. An observation not apparent in Figure 6 is that the support and size of the FRs seems to directly affect the classification power of the polynomial kernel. Specifically, FRs with low support and large size tend to have much higher classification power across k values. For example, Figure 7 shows a box plot of the AUROC results for the polynomial kernel SVM on the $k = 1000$ data for only the low support (minsup $\in \{5, 10\}$) and large size (minsize $= 100$) datasets. As we can see, the outliers from Figure 6 are in fact these low support, large size datasets. We also observe that, in this case, the multinomial-filter appears to improve the classification power of the polynomial kernel SVM. This suggests that when mining FRs for classification purposes it is advised that `FindFRs` should be appropriately parameterized according to the kernel being used; iFRs with low support and large size in conjunction with the multinomial-filter appears to work well for the polynomial kernel.

7.2.4 *Visualizing pan-genomic space*

We also applied *multidimensional-scaling* [40], using the `isoMDS` method in *R*, to provide a visual representation of the industrial uses for yeast. The ranking method described in Section 6.1 was first used to determine the top 500 iFRs by p-value (eqn. 5) for discriminating the nine industrial uses from Table 2. Distances between usages were found using the Canberra method, an appropriate metric for count-based data, provided by R's `dist` function. As can be seen in Fig. 8, the plot provides a visual interpretation of yeast usages based entirely on the aggregate genomic content of each group.

8 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 [4].

FRs permit analyzing multiple related assemblies simultaneously without reference bias to find patterns both

2. http://scikit-learn.org/

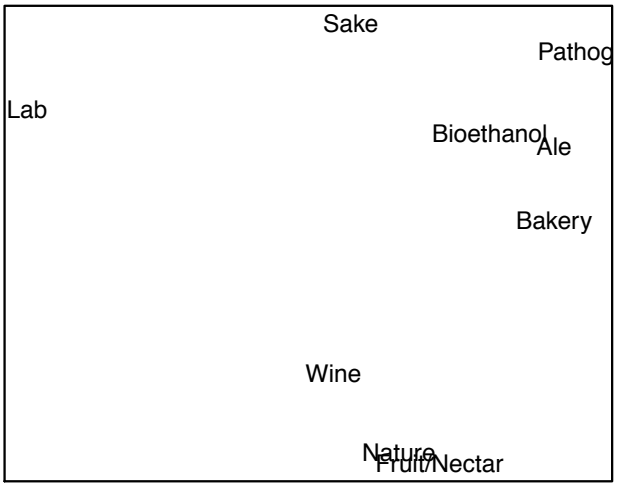


Fig. 8: Yeast industrial usage multidimensional-scaling (MDS) plot based on the top 500 discriminative iFRs found. MDS aims to place each object in a 2D box, such that the between-object distances are preserved as well as possible. The Canberra distance in R was used to compute distances from iFR frequency data for strains associated with each industrial usage and the MDS plot was generated using the `isoMDS` function. ($k = 500$, $\alpha = 0.7$, $\kappa = 0$)

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 (ie. chromosome or plasmid) level. Our FR approach would be best integrated into a visual tool to help researchers understand and explore pangenomic data, e.g. 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.

Software: github.com/abi-pangenomics/FindFRs

ACKNOWLEDGMENT

This work was supported by the National Science Foundation, under grant DBI-1542262.

REFERENCES

[1] 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," *Journal of Heredity*, vol. 100, no. 6, pp. 659–674, 2009.

[2] 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," *Proceedings of the National Academy of Sciences*, vol. 110, no. 5, pp. 1785–1790, 2013.

[3] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: astronomical or genomical?" *PLoS Biol*, vol. 13, no. 7, p. e1002195, 2015.

[4] T. C. P.-G. Consortium, "Computational pan-genomics: status, promises and challenges," *Briefings in Bioinformatics*, vol. 19, no. 1, pp. 118–135, 2018. [Online]. Available: http://dx.doi.org/10.1093/bib/bbw089

[5] 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*, vol. 102, no. 39, pp. 13 950–13 955, 2005.

[6] S. Marcus, H. Lee, and M. C. Schatz, "Splitmem: a graphical algorithm for pan-genome analysis with suffix skips," *Bioinformatics*, vol. 30, no. 24, pp. 3476–3483, 2014.

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

[8] 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.

[9] 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*, vol. 44, no. 2, pp. 226–232, 2012.

[10] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

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

[12] J. Liu, S. Paulsen, X. Sun, W. Wang, A. B. Nobel, and J. Prins, "Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis." in *SDM*, vol. 6, 2006, pp. 405–416.

[13] H. Cheng, P. S. Yu, and J. Han, "Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*, 2006, pp. 839–844.

[14] 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*. Springer, 2000, pp. 13–23.

[15] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *The Knowledge Engineering Review*, vol. 28, no. 01, pp. 75–105, 2013.

[16] M. Koyutürk, A. Grama, and W. Szpankowski, "An efficient algorithm for detecting frequent subgraphs in biological networks," *Bioinformatics*, vol. 20, no. suppl 1, pp. i200–i207, 2004.

[17] S. Hill, B. Srichandan, and R. Sunderraman, "An iterative mapreduce approach to frequent subgraph mining in biological datasets," in *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. ACM, 2012, pp. 661–666.

[18] 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 *Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*. ACM, 2004, pp. 308–315.

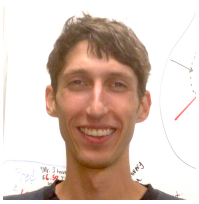
[19] R. Li and W. Wang, "Reafum: Representative approximate frequent subgraph mining," in *SIAM International Conference on Data Mining*. SIAM, 2015, pp. 2167–0099.

[20] S. Guha, "Efficiently mining frequent subpaths," in *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*, 2009, pp. 11–15.

[21] A. Cleary, B. Mumey, T. Ramaraj, and J. Mudge, "Approximate frequent subpath mining applied to pangenomics," in *BICoB*, 2017, pp. 59–65.

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

[23] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.
[24] 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*. Springer, 2013, pp. 215–229.
[25] S. K. Pham and P. A. Pevzner, "Drimm-synteny: decomposing genomes into evolutionary conserved segments," *Bioinformatics*, vol. 26, no. 20, pp. 2509–2516, 2010.
[26] G. Yang, "The complexity of mining maximal frequent itemsets and maximal frequent patterns," in *Proceedings of ACM SIGKDD*, 2004, pp. 344–353.
[27] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, no. 3, pp. 449–467, 1965.
[28] B. O. F. Auer and R. H. Bisseling, "A gpu algorithm for greedy graph matching," in *Facing the Multicore-Challenge II*. Springer, 2012, pp. 108–119.
[29] J.-H. Hoepman, "Simple distributed weighted matchings," *arXiv preprint cs/0410047*, 2004.
[30] R. Preis, "Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs," in *STACS*, vol. 99. Springer, 1999, pp. 259–269.
[31] F. Manne and R. H. Bisseling, "A parallel approximation algorithm for the weighted maximum matching problem," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 708–717.
[32] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[33] M. Ringnér, "What is principal component analysis?" *Nature biotechnology*, vol. 26, no. 3, p. 303, 2008.
[34] 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*, vol. 6, no. 5, p. e14802, 2011.
[35] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.
[36] 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*, vol. 22, no. 5, pp. 908–924, 2012.
[37] 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*, vol. 7, no. 2, p. e1001287, 2011.
[38] 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*, vol. 106, no. 38, pp. 16 333–16 338, 2009.
[39] V. Bewick, L. Cheek, and J. Ball, "Statistics review 13: receiver operating characteristic curves," *Critical care*, vol. 8, no. 6, p. 508, 2004.
[40] T. F. Cox and M. A. Cox, *Multidimensional scaling*. CRC press, 2000.



Alan Cleary is a Computational Research Scientist at the National Center for Genome Resources (NCGR) in Santa Fe, NM, USA. He earned his Ph.D in Computer Science at Montana State University in 2018. His research interests include algorithms and their application to computational biology, with an emphasis on pan/population genomics, big data, and high-performance computing.

Thiruvarangan Ramaraj is a Bioinformatics Research Scientist at the National Center for Genome Resources (NCGR) in Santa Fe, NM, USA. His research focus is mainly on Bioinformatics and Computational Biology. At NCGR, Dr. Ramaraj and his team primarily focus on developing workflows for large-scale biological sequence data analysis such as complex plant de novo genome/transcriptome assemblies. His current NSF funded projects include, (i) Investigation of genome-wide structural rearrangements of diploid and polyploid cotton genomes, (ii) Resequencing several accessions of cotton to address fundamental biological questions of polyploidy genomes and the genetic diversity of cotton, (iii) Investigating graph data structures to effectively represent pan-genomes and associated algorithms for analyzing pan-genomes and (iv) A systems-biology proposal on biofilms that started in August 2017.

Indika Kahanda is an Assistant Professor in the School of Computing at Montana State University, Bozeman, MT, USA. His research interests lie in the areas of Bioinformatics, Computational Biology, and Biomedical Natural Language Processing. In particular, he investigates approaches to (1) develop computational methods for functional genomics and (2) develop natural language processing tools for biomedical literature.

Joann Mudge is a Senior Research Scientist at the National Center for Genome Resources (NCGR) in Santa Fe, NM, USA. She focuses on plant genomics, including genome assembly, structural genomics, and comparative genomics. In addition to her pangenomics work, she works extensively on legumes and their symbionts. She is also involved in outreach, with an NIH-funded grant to teach elementary and secondary teachers about DNA and bioinformatics and help them develop their own DNA-based case studies.

Brendan Mumey is a Professor in the School of Computing at Montana State University. His research interests are broadly in algorithms and optimization, with most of his work stemming from applied problems in networking and computational biology. He earned his Ph.D. in Computer Science at the University of Washington in 1997 and joined the faculty at Montana State University in 1998. In 2011, he held a Visiting Fulbright-Aalto Distinguished Chair position at Aalto University in Finland.