Methods for Analysis of Prokaryotic Genome Architecture

Andrew S. Warren

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Lenwood S. Heath, Co-Chair
João C. Setubal, Co-Chair
Allan Dickerman
Iddo Friedberg
T. M. Murali
Liqing Zhang

May 5, 2017
Blacksburg, Virginia

Keywords: Ontology, Prokaryotes, Phylogenetic Profiling, Semantic Similarity, Genome Annotation, Gene Function, Pan-genome

Methods for Analysis of Prokaryotic Genome Architecture

Andrew S. Warren

(ABSTRACT)

Research in comparative microbial genomics has largely been organized around the concept of reference genomes. Reference genomes provide a useful comparative touchstone for closely related organisms. However, they do not necessarily represent the biological diversity in a *group* of genomes. Currently there are more than 96,000 bacterial genomes sequenced and this number is rapidly increasing. Some closely related groups have large numbers of genomes sequenced creating interesting comparative challenges: *E. coli* more than 5,400 isolates, *S. aureus* almost 9,000. As this sampling through sequencing becomes both deeper and broader, reference genome based methods become less effective at characterizing groups of organisms.

Functional motifs can help explain the organizing principles behind cellular systems in bacteria which have yet to be well understood. Currently there are relatively few bioinformatic tools for analyzing potential patterns at the level of genome organization that do not depend directly on sequence similarity. We present a framework for conducting genomic data mining to look for patterns that currently require human expert designation. We establish new computational methods for identifying patterns in prokaryotic genome construction through a mapping of genomic features, using semantic similarity, independent of a particular corpus to better approximate functional similarity.

We also present an algorithm for creating whole genome multiple sequence comparisons and a model for representing the similarities and differences among sequences as a graph of syntenic gene families. This effort touches on several different research fronts: graph representation of genomes and their alignments, synteny block analysis, whole genome sequence alignment, pan-genome analysis, multiple sequence alignment, and genome rearrangement analysis. Though our approach was originally developed from a pan-genome perspective for prokaryotes, the methods involved have the potential to speed up more expensive computation such as phylogenetic tree construction and SNP analysis. Novel elements include the contextualization of synteny analysis both between and within multi-contig genomes and an analytical framework for detecting genome level evolutionary events such as insertions, inversions, translocations, and fusions.

Methods for Analysis of Prokaryotic Genome Architecture

Andrew S. Warren

(GENRAL AUDIENCE ABSTRACT)

Research in comparative microbial genomics has largely been organized around the concept of reference genomes. Reference genomes provide a useful comparative touchstone for closely related organisms. However, they do not necessarily well represent the biological diversity in a group of genomes. As sampling through sequencing becomes both deeper and broader, reference genome based methods become less effective at characterizing groups of organisms. We present an algorithm for creating whole genome multiple sequence comparisons and a model for representing the similarities and differences among sequences as a graph of syntenic gene families called a pan-synteny graph.

As the evolutionary distance between organisms increase sequence similarity and homology detection tend to break down. However, similarities in the functional characteristics of certain genes and gene modules may persist or have converged over time. Detecting and defining patterns in these functional similarities, in relation to conserved gene order, is a largely unexplored problem. To create a model for representing the architectural similarity of functional modules, using ontologies and semantic similarity, we present a corpus independent semantic similarity method, and describe a computational framework for using semantic similarity and pan-synteny graphs.

Dedication

To my parents Sharon and Jerry for their unwaivering support.

To my brother Matthew and my sister Alyson for teaching me how to have fun.

To Coach Rhodes for sparking my interest in science.

Acknowledgments

Thank you to my adivsors Dr. Lenny Heath and Dr. João Setubal for their advice and encouragment. I also thank my committee Dr. Allan Dickerman, Dr. Iddo Friedberg, Dr. T. M. Murali, and Dr. Liqing Zhang for their suggestions and direction. Thanks also to Dr. Rebecca Wattam for her invaluable insight into the biology of *Brucella*. Thank you to my friends and co-workers at the Biocomplexity Institute for providing an excellent research environment. I would also like to thank my friends of the Executive Lunch for their humor and encouragement. Finally I would like to thank my family, for giving me perspective. The dude abides.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Current best estimates place the number of microbial species on Earth to be as many as 1 trillion [41]. Such a large reservoir of uncharacterized diversity combined with the rapidly increasing number of publicly available bacterial genomes necessitate new methods for managing and visualizing data so that the variations among bacterial populations can be understood. Bacterial sequencing projects encompass a range of sample types from entire environmental samples to multiple isolates for a particular species or strain. As such any method that seeks to further structure the broad sampling represented by the aggregation of bacterial genome sequencing must be able to handle a range of diversity and conditions.

The prokaryotic cell can be considered a complex machine consisting of multiple systems (i) a metabolic system that carries out physiological tasks (ii) a regulatory system that determines what conditions promote these tasks and (iii) a transduction system that transmits signals across different regulatory elements [34]. These systems make repeated use of modules with similar biochemical and genetic properties at different hierarchical levels [37]. A **gene module** is a set of genes grouped together based on specific conditions which are context-dependent. These conditions vary depending on the application such that modules can be cliques of interacting proteins, clusters of genes based on expression profile, or members of pathways in signaling, regulatory, or metabolic networks [73]. The conservation of such modules forms the basis of the analysis methods presented here.

Prokaryotic organisms represent one of the largest biomasses on the planet and have adapted to a wide variety of living conditions with an even wider variety of cellular mechanisms. Biologists seek to understand these solutions by organizing genes into higher order structures, such as gene modules, that carry out a common cellular function. Due to improvements in throughput and reductions in cost, a much larger sampling of this diversity is taking place through whole genome sequencing. With this increase in genome sequence data for a wide variety of organisms it is now possible to look for patterns in cellular systems, at an inter-organismal level. Finding patterns across modules that use distinct non-homologous genes to fulfill similar cellular roles across multiple organisms is possible [47]. We present work

that organizes prokaryotic genome structure into a computational framework and methods for determining functional similarity that allows for the systematic exploration of these patterns. We refer to these patterns as functional motifs since they describe the common characteristics for performing some biological task.

Functional motifs can help explain the organizing principles behind cellular systems in bacteria which have yet to be well understood [79]. For example, chitin metabolism has been found to have several fundamental functional characteristics despite being carried out by various non-homologous genes in different organisms [80]. Currently there are relatively few bioinformatic tools for analyzing potential patterns at the level of genome organization that do not depend directly on sequence similarity. Resulting patterns can be used to populate databases that are currently driven by manual curation, such as SEED [47], and can potentially make significant contributions to the design of functional units in synthetic biology [4]. Here I establish new computational methods for identifying patterns in prokaryotic genome construction through a mapping of genomic features using semantic similarity and conserved gene neighborhood. As the number of prokaryotic genomic sequences skyrockets we risk losing perspective on how the mass of sequences fit into a coherent biological framework. A high level description of this data is necessary if we are going to traverse the massive repositories of sequence information and continue studying the complex evolutionary relationships that have given rise to a formidable level of biodiversity. The work presented here seeks to characterize the commonalities and differences among prokaryotes and to look for patterns that increase the understanding of their biology and evolution.

Functional modules occur across multiple organisms according to orthologous relationships between genes. A functional module in one organism can be said to exist in a second organism if all the gene members of that module have an ortholog in the second organism. For example, a module that helps regulate chemotaxis in *E. coli* can also be found in *Shigella flexneri* because each gene member of that module has a corresponding ortholog. We refer to the occurrence of a module in a particular organism as an *instance* of that module. In this work, the term *functional module* implies the collection of all instances of a module unless otherwise specified. Because modules often have more than one instance, particularly in a set of closely related organisms, they can be thought of as a set of orthologous families.

The properties of functional modules can be described using the annotations of the genes that comprise them. In this work I provide an initial framework for finding patterns across the properties of multiple, distinct functional modules whose respective instances carry out equivalent roles in different organisms. We refer to these patterns as functional motifs. A **functional motif** is a collection of common biological properties found in two or more distinct functional modules. A motif can be seen as a grouping of more than one functional module based on the common functional properties of the orthologous families within each module. For this work these properties are represented by Gene Ontology annotations. From a computational perspective a motif is a data structure that creates a mapping between functional modules (gene sets) based on the ontology terms associated with the orthologous families within those modules. This mapping has a substructure in which genes (or COGs

in the general case) with similar functional properties are grouped together across modules resulting in **functional components**. Functional components represent the conserved characteristics of genes between modules and are based on at most one gene from each module. The result is a representation of gene properties that are commonly found to co-occur in distinct modules that fulfill some common biological purpose. A motif can represent different levels of organization with respect to microbial gene groups e.g. operons, regulons, modulons [23]. The underlying principle is that different organisms have acquired or evolved similar solutions to similar problems, be it through transfer, inheritance, divergence, or convergence. This work seeks to enable the characterization of these solutions with functional semantics, at a systems level.

The methods presented: a graph model to organize similarities and differences based on a gene modules; and a corpus independent information content that can be used to compare gene function, serve as preliminaries for a system to find patterns of biological attributes. In this case, functions are represented by ontology gene annotations, that occur across various levels of organization in prokaryotes. Such a system is intended to establish patterns that distinguish groups of organisms and establish association rules which indicate the constituent components necessary to create certain biological constructs. Put simply, given a collection of gene modules, and their constituent genes and annotations, the objective of such a system is to detect and describe significant patterns in the annotations that span multiple modules.

Traditional gene module analysis has been focused on predicting occurrences of gene modules but little has been done to analyze their architecture [79]. As the evolutionary distance between organisms increase sequence similarity and homology detection tend to break down. However, similarities in the functional characteristics of certain genes and gene modules may persist or have converged over time. Detecting and defining patterns in these functional similarities, at the module level, is a largely unexplored problem. To create a model for representing the architectural similarity of functional modules using ontologies and semantic similarity we present a corpus independent semantic similarity method.

In recent years, the concept of a pan-genome has been put forward as one possible method for viewing, organizing, and understanding this body of data. A pan-genome can be regarded as the core and variable features found in a given set of bacterial genomes. Beyond this, the research community has yet to reach an exact specification for what a pan-genome should contain, but previous work often includes: orthologous gene groups from among the genome set and a notion of the level of agreement using gene conservation, synteny, or average nucleotide identity.

Most instantiations of pan-genomes thus far exist in a somewhat inaccessible state, either as abstractions tied to a particular resource, which cannot be used for further independent computation, or as a collection of statistics and low-level evidence, which are not inherently useful beyond the scope of the study that generated them. This lack of a standard representational form, the pressing need for new constructs to manage the overwhelming number of bacterial sequencing projects, and the desire to create a computable and usable pan-genome

motivates our work on: pan-synteny graphs, a formalism to represent the core and variable features found in a set of bacterial genomes; an algorithm for creating them; and multiple visualization and computational techniques for leveraging them.

Chapter 2

Pan-synteny graphs

2.1 Motivation

Whole-genome alignment and pan-genome analysis are useful tools in understanding the similarities and differences of many genomes in an evolutionary context. Here, I introduce the concept of pan-synteny graphs, an analysis method that combines elements of both to represent conservation and change of multiple prokaryotic genomes at an architectural level. Pan-synteny graphs represent a reference free based approach for the comparison of many genomes and allows for the identification of synteny, insertion, deletion, replacement, inversion, recombination, missed assembly joins, evolutionary hotspots, and reference based scaffolding.

I present an algorithm for creating whole genome multiple sequence comparisons and a model for representing the similarities and differences among sequences as a graph of syntenic gene families. As part of the pan-synteny graph creation, I first create a de Bruijn graph. Instead of the alphabet of nucleotides commonly used in genome assembly, I use an alphabet of gene families. This de Bruijn graph is then processed to create the pan-synteny graph. My approach is novel in that it explicitly controls how regions from the same sequence and genome are aligned, identifies genome level evolutionary events such as inversions, and it generates a graph in which all sequences are fully represented as paths. This approach demonstrates a way to harness the previous computation involved in protein family calculation to speed up repeat comparative analysis of annotated genomes. I develop the software suite Panaconda for the calculation of pan-synteny graphs given annotation input and an implementation of methods for their layout and visualization.

The main motivation is to create a new standard of information exchange, processing, and visualization for high level bacterial comparative genomics using the concept of pan-synteny graphs to enable the better understanding of evolutionary relationships. Specifically this work:

- Formalizes the pan-synteny graph concept and the implementation of a fast, scalable method for its creation.
- Establishes file format(s) to transmit the pan-synteny graphs in an extensible way that maximizes its utility on multiple compute platforms.
- Creates visualization tools that can be used to understand and explore new relationships between large groups of bacteria.
- Establishes the utility of pan-synteny graphs by demonstrating its use in identifying evolutionary events among genomes.

By establishing the utility of the pan-genome concept I address two pressing needs of the bacterial research community: a reduction in the cost of processing a large number of sequencing projects for comparative analyis and using consistency of annotation to inform computation.

2.2 Background

Research in comparative microbial genomics has largely been organized around the concept of reference genomes [68]. Reference genomes provide a useful comparative touchstone for closely related organisms. However, they do not necessarily well represent the biological diversity in a group of genomes. Currently, there are more than 96,000 bacterial genomes sequenced [75] and this number is rapidly increasing. Some closely related groups have large numbers of genomes sequenced creating interesting comparative challenges: *E. coli* has more than 5,400 isolates, while *S. aureus* has almost 9,000. As this sampling through sequencing becomes both deeper and broader, reference genome based methods become less effective at characterizing groups of organisms.

Pan-genomes represent a method to analyze sets of genomes as a whole. One basic representation is the unique and core genes of a set [71]. This method provides a general characterization of content but does not capture the inter-relations among members of a group nor does it model the genomic context in which the gene sets occur. Other more detailed methods create a pan-genome representation at the nucleotide level. One such method, SplitMEM [43], creates a compressed de Bruijn graph for representing shared variation and single nucleotide polymorphisms. Strategies are still evolving to fully leverage this complex representation. The pan-synteny model targets genome architecture analysis and thus can be placed somewhere on the meso-scale of pan-genome analysis methods, being more detailed than a simple categorization of gene types and more abstracted than a SNP level analysis.

Whole genome alignment methods also take on the challenge of producing a detailed comparison at the nucleotide level. These methods can assist greatly when performing SNP and phylogenetic analysis. Many methods require the specification of a reference genome,

relatively high memory, and that genomes be closely related. One example of this, Harvest, is a tool for establishing core genome alignments among multiple genomes that recruits highly related genomes using genome distance metric MUMi [17]. There are a number of methods available for performing whole genome alignment [18], many of which are useful for the creation of SNP trees and analysis for comparing closely related genomes at the DNA level. Lee *et al.* [36] introduce the Partial Order Alignment (POA) graph for representing alignments and calls into question the practice of representing multiple sequence alignments as grids, since they are not well adapted to representing complex rearrangements and gaps accurately. Raphael *et al.* [58] follows this up by introducing the ABA graph constructed to align proteins given initial BLAST seed alignments. The method detailed here also performs a whole genome multiple sequence alignment at the gene order level and encodes it as a pan-synteny graph.

Comparative methods that focus on synteny traditionally focus on finding syntenic blocks. As Ghiurcuta *et al.* [22] points out, homology forms the basis for the construction of most synteny block analysis and the creation of blocks usually involves the process of "extending homologies among markers to homologies among blocks." A syntenic block is typically thought of as a conserved ordering of homologous genes between two or more genomes. DRIMM-Synteny [55] is a method for finding synteny blocks in multiple mammals. DRIMM uses the creation of an A-Bruijn graph and its subsequent processing to approximate the minimum number of edits necessary to convert from one sequence to another in the pursuit of synteny block construction. This is done by constructing or finding non-branching paths in an A-Bruijn graph with multiplicity greater than one.

Sibelia [46] is a follow on project to DRIMM and also uses A-Bruijn graphs to construct synteny blocks. In this work, the authors assert, that when analyzing bacteria, using an alphabet of nucleotides is necessary. While we find the iterative refinement approach based on different window sizes to be novel, we also find that Panaconda's abstraction away from a fixed relation for the "glue" [53] that constitutes a homologous relationship can yield insightful results in prokaryotes. Both the Sibelia and DRIMM approaches edit sequences to remove loops (also known as whirls) and bulges in a de Bruijn graph to achieve non-overlapping synteny decompositions. In doing so, they destroy sequence continuity and limit the discovery of potentially complex evolutionary relationships. The pan-synteny graph approach we outline here bypasses the need to remove these anomalies and preserves sequence continuity.

The de Bruijn graph has been widely used as a memory efficient construct for performing genome assembly [54, 81]. It should be noted that the de Bruijn graphs commonly referred to in describing modern assembly algorithms are not precisely the same graph as first described by Nicolaas de Bruijn [12]. The original de Bruijn graph, $B = (V, E)$, for a given length k and alphabet Σ, is created such that *every possible* $k-1$ lengthed sequence is assigned to a node $v \in V$, and for every pair of nodes (v_1, v_2) a directed edge $e \in E$ connects v_1 to v_2 if there exists some k-mer $\in \Sigma^k$ whose prefix and suffix are assigned to v_1 and v_2 respectively. The resulting graph B, can be used to find a Eulerian cycle, which gives the shortest circular

superstring which contains all possible substrings of length k. When describing assembly algorithms it is common for the term de Bruijn graph to imply a restiction on the edge set E such that only k-mers generated from a source string, e.g. sequencing reads, are represented. All subsequent use of the term 'de Bruijn graph' invokes this convention.

The pan-synteny graph problem is similar in nature to both the multiple global sequence alignment and multiple local sequene alignment. A multiple global alignment of p strings, where $p > 2$, $S = S_1, S_2, ..., S_p$. In the multiple alignment problem spaces are inserted into, or at the end of, sequences such that the resulting strings are the same length. Each string can be placed on one of p rows and arranged such that each character occupies a unique column [26]. A local multiple alignment of S involves selecting one substring from each $S_i \in S$ and then globally aligning those substrings. Both the multiple sequence alignment and pan-synteny graph alignment seek to partition a set of strings into groups of substrings in such a way that substrings within a group are more likely to share a common evolutionary history than substrings in different groups. The pan-synteny graph problem does this by initialy grouping k-mers into the same rf-node and traversing over those nodes in such a way that substrings from the genome set G are grouped together on a letter-by-letter basis. There is an inherent difference in output; instead of a consensus sequence or a grid of sequences the pan-synteny graph problem represents the grouping of subsequences as an annotated graph.

In constructing multiple sequence alignments, many approaches use stacked rows with column assignments indicating conserved characters in the alignment. Lee *et al.* [36] initially pointed out that when representing many diverse sequences, this layout can prove problematic as the constraints created by a grid like format are not capable of representing all rearrangements and gaps accurately. This is especially true when trying to align whole genomes. When representing synteny, dot-plots tend to work well for inversions and rearrangements in the comparison of two genomes but does not scale well beyond his number. A dot-plot is a plot of similarity between two sequences s_1 and s_2 arranged on a rectilinear grid such that one sequence is represented by the x-axis, say s_1, and the other is represented by the y-axis, s_2. For a given position i on s_1 and j on s_2 if the sequences are similar a dot is placed at coordinates representing (i, j).

The multiple sequence alignment problem as it is traditionally dealt with in biological sequence analysis has an alphabet of relatively small size, *four nucleotides or twenty amino acids*, in comparison to the pan-synteny graph problem which can use thousands or potentially millions of distinct characters from the alphabet of feature families. This larger alphabet precludes the use methods that require a limited combinatorial set of transitions from one character to the next, such as suffix tree. The creation of an de Bruijn-like graph and DFS expansion into a pan-synteny graph is similar to the "repeated-motif" class of methods for constructing multiple sequence alignments. Repeated-motif methods for finding multiple alignments first find a motif, *i.e. anchor*, common to many sequences. The number of sequences a motif occurs in is referred to as its "multiplicity". Many methods require that a motif or anchor point have a certian multiplicity value. Sequences are then "shifted" so that occurrences of the motif are aligned with one another. Subsequences flanking the motif

go through a similar process until no motif of sufficient multiplicity is found. Sequences that did not contain the original motif are aligned separately and merged [26]. We highlight two such methods that have similarities to pan-synteny graph creation.

One repeated-motif method for finding matches and errors in multiple sequences, by Leung *et al.* [38], employs several conventions also used in pan-synteny graph creation. First, sequences are concatenated and k-mers are replaced by a sequence of successive integers. Each k-mer is connected to all of its occurrences by constructing a linked list. These "core blocks" are extended using flanking blocks to form long matches allowing for some errors. Pan-genome graph construction uses a similar convention in that it processes, in base coordinate sorted order, the annotations of each genome by creating k-mers, giving the occurrence of each k-mer a label based on an increasing integer count, and assigning all occurrences of a specific k-mer to a single rf-graph node. Instead of processing these blocks into matches, pan-synteny graph construction processes the resulting de Bruijn like graph into a graph respresentation of conserved syntenny. Instead of allowing for errors, pan-synteny graph explicitly represents any variation encountered as an alternative path in the pan-synteny graph.

Sagot *et al.* [61] also present a method for identifying blocks of similarity among multiple sequences. This procedure looks for a substring of greatest possible length which have q occurrences, referred to as the quorum constraint, among a set of the input sequences. This is done by first identifying all occurrences of substrings of length k, with greater than q occurrences, and extending these "anchors" via DFS adding one character at a time. The algorithm for creating a pan-synteny graph proceeds in a similar fashion by creating an rf-graph and establishing anchor nodes from which to launch a DFS exploration of the graph. Moving from one node to the next in an rf-graph, also represents the addition of a single character. Instead of returing a particular subsequence, the pan-synteny graph procedure gives the graph as output. Because pan-synteny graph creation may be targeted at representing a particular type of synteny, the creation algorithm communicates information about the progression of integer labels, as discussed in the synopsis of the method by Leung *et al.*, in both the descent and ascent phases of DFS, so that it can partition the instances of k-mers into different segments of the pan-synteny graph to best represent the target synteny type.

2.3 Definitions

We introduce the novel concept of the pan-synteny graph. To provide a clear definition of what a pan-synteny graph is, we first make explicit concepts and notation used in defining the graph and describing its creation. A *contig* is a string of nucleotides represented by the alphabet $\{A, C, T, G\}$ that represents a contiguous region of DNA. Let G represent the set of all genomes for which a pan-synteny graph is to be defined, and let P be the set of all contigs such that a genome, $g \in G$, is comprised of contigs sequenced from the genomic DNA of g. Each $g \in G$ can be represented as a set of at least one contig. A genomic

feature is a nucleotide interval with specific coordinates in a $p \in P$ which has been assigned some particular meaning. A genomic feature can be any commonly annotated aspect of bacterial genomes, *e.g., the coding sequence for a protein coding gene, an RNA gene, or a transcriptional start site.* Let F be the set of all annotated features from P.

While most data in many pan-genome analysis focuses on protein families, there is no restriction in this analysis to protein coding genes or protein families. This form of analysis only requires that the units in question be given as contiguous intervals that occur on a sequence and that those units have coordinates for the sequence and genome where they occur as well as some sort of group designation, e.g., a protein family, or an orthologous group. A feature family, d, is a set of features, usually from more than one genome, that creates a set of related features, a example being protein familes, which commonly relates multiple proteins based on the concept of orthology or functional equivalence. A feature, f, is said to be annotated with a feature family d, if it is a member of that family $f \in d$. For the purposes of describing a pan-synteny graph it is assumed that each feature $f \in F$ is annotated with a feature family.

The pan-synteny graph problem has a nested structure with respect to G. Let D be the set of unique feature families in G such that every $f \in F$ has an assignment $d \in D$. In many computational biology problems genomes are represented as a string using an alphabet of nucleotide letters $\Sigma = \{A, C, T, G\}$. To simplify the problem of summarizing similarities among many bacterial genomes an alphabet consisting of feature families Σ_D is employed. Any sequence $p \in P$ with n features can be defined as a sequence of features $f_1, f_2 ... f_n$ and mapped to a sequence of feature families $d^1, d^2 ... d^x$ given $d \in \Sigma_D$ where n is the number of genomic features in a contig $p \in P$. For a genome g_i, let $\pi_P(g_i)$ represent the contigs of g_i. Similarly for a contig p_i let $\pi_F(p_i)$ represent the ordered series of features underlying p_i. For the purposes of pan-synteny graph creation it can be assumed that for an arbitrary feature at position j of a contig $p \in P$, f_j, there exists an assignment to a feature family d from Σ_D that represents f_j. Because each family can be assigned to more than one feature in a single sequence, any sequence of feature families is potentially repetitive if a feature family occurs more than once in p.

To illustrate the hierarchical nature of the pan-synteny input we give an example. Let G be a genome group consisting of two genomes, $G = \{g_1, g_2\}$ and let each genome be composed of one contig, such that the contig for g_1 is represented as $\pi_P(g_1) = \{p_1\}$ and the contig for g_2 is $\pi_P(g_2) = \{p_2\}$. Then for the graph G, the set of all contigs for G, P, can be given as $P = \{p_1, p_2\}$. Let each contig be composed of n features such that the features of p_1 are represented as $\pi_F(p_1) = f_1, f_2 ... f_n$ and the features of p_2 are $\pi_F(p_2) = f'_1, f'_2, ... f'_n$. The set of all features F for G is $F = \{\pi_F(p_1) \bigcup \pi_F(p_2)\}$. For this example let the first feature in each contig be assigned a feature family d^1 and let all other features in both contigs be assigned a feature family of d^2. In this example the resulting alphabet for pan-synteny graph construction is $\Sigma_D = \{d^1, d^2\}$.

Synteny is defined as conserved gene order [8] and therefore can only be assigned when

considering the *conservation* of more than one genomic feature in more than one location. A set of contiguous features that are conserved across multiple organisms is referred to as a *synteny block* [33, 52]. To enable the creation of pan-synteny graphs, we substitute the typical nucleotide alphabet for one of feature families. By doing so, we are leveraging the previous computation used in calculating feature families to make the creation of a pan-synteny graph cheap and fast. In the context of the pan-synteny graph problem, two features are defined as being conserved relative to one another if they are both assigned to the same family. It follows that identical sequences of feature families among two or more locations suggests synteny.

A pan-synteny graph, $R(V, E)$, is defined to be a node set V and edge set E, such that each node $v \in V$ represents a subset of features from a single feature family and each edge $e \in E$ connects two sets of features from v_1 and v_2 which are syntenic, in the contigs annotated to $(v_1,\ v_2)$. Put another way every edge $e \in E$ and its incident nodes v_1 and v_2 represent a partitioning of the genomic feature space F such that each feature pair annotated to $(v_1,\ v_2)$ have conserved, contiguous feature family assignments in the contigs in which those features are found, and are flanked by at least $k-2$ features meeting these same conditions. The nodes and edges of a pan-synteny graph are referred to as ps-nodes and ps-edges. Given a ps-node $v \in V$ and a feature $f \in F$, let $v \leftarrow f$ represent annotating v with f, and let $F(v)$ represent all the features annotated to v. For the set of sequences P included in a pan-synteny graph R, each sequence p of length n is represented as a path in the pan-synteny graph, where each feature with a family assignment is represented by a node along that path. More formally, if a genome sequence $p \in P$ is represented by a sequence of features $f_1, f_2, ..., f_n$, and these features label a set of nodes $v_1, v_2, ..., v_n$ in the pan-syteny graph such that there is a bijection from features to nodes representing p, then every pair of consecutive labels f_i and f_{i+1} will label adjacent vertices in the pan-synteny graph such that a path is constructed. We refer to the expression of each sequence in P as a path in R as the full-path property.

2.4 Algorithm

A pan-synteny graph is constructed to represent syntenically conserved genomic features across multiple genomes. To establish synteny in the context of a pan-synteny graph construction, two parameters are defined: a window size $k >= 3$, and a minimum genome count m. To be considered a synteny block, a sequence of genomic features must be of at least length k and must represent a sequence of contiguous feature family assignments that exists in m or more locations in G. Here the term 'location' is used loosely and will be further refined as the categories of synteny are made explicit. The parameters k and m, significantly affect the structure and meaning of a pan-synteny graph for a given set of genomes.

Previous comparative genomic analysis of synteny has focused on determining synteny between genomes, between contigs in the same genome, within contigs, or some combination

thereof [2, 3, 7, 15, 22, 40, 46, 49, 55]. In pan-synteny graph construction the syntenic context defines the relationships of interest that should underpin a synteny block. If the context is at the genome level, C_G, then a synteny block is defined by similarity *between genomes* and not within them. If the target context, C, is between-contigs, C_P, a synteny block is defined by similarity between genomes, between contigs, and not within contigs. Finally, if the target context is within-contigs, represented as $C_\varnothing$, then synteny blocks represent similarity within contigs, between contigs, and between genomes.

Similar to DRIMM and Sibelia the Panaconda method creates a de Bruijn graph as part of the conserved gene order analysis. Unlike these two approaches, Panaconda processes the de Bruijn graph so that sequence continuity is preserved in the resulting simplified pan-synteny graph (ps-graph). Panaconda takes as input an annotation, ordered by lower base coordinate, of genome features with some family designation. These annotations are processed into k-mers of family assignments and are used to construct a labeled de Bruijn graph that we designate an rf-graph. An rf-graph is a form of double stranded de Bruijn graph [40] $DB^*(G, k)$ which encodes both the forward and reverse strand of a sequence on the same node and maintains four edge types to model the transition from one kmer orientation to another.

Panaconda takes a number of parameters that govern how the ps-graph is constructed. Let m be the minimum multiplicity of sequences that must be incident on a node or edge for it to be shown in the ps-graph. Let k be the k-mer size given to the de Bruijn graph construction which partitions each sequence of size n into $n - k + 1$ overlapping k-mers. The k-size can be viewed as the minimum size of conserved neighborhood to constitute a syntenic block.

The pan-synteny graph problem assumes as input both a set of contigs P, each represented by a series of features, and a set of input annotations which maps every feature in P to a feature family: for every $f \in F$ there exists a $d \in D$ such that d is assigned to f. Thus every $p \in P$ can be represented as a sequence of features or a sequence of feature families. Let d_i and d_j give the feature family assignments of a features f_i at position i and f_j at position j in a sequence $p \in P$. In this case d_i and d_j may or may not represent the same character from Σ_D. Further, let $D(p)$ give the translation of a sequence of features in p as the corresponding sequence of feature families. For a given sequence p' of length n, $p' = f_1, f_2, ..., f_n$, $D(p)$ gives a sequence, $d_1, d_2, ..., d_n$, using the alphabet of Σ_D. A feature f_i is said to be an *instance* of a feature family d at position i, in sequence p, if f_i maps to d in the set of input annotations for G. For any $p \in P$, let $F^k(p)$ represent the series of overlapping k-mers found in p, where $F^k(p) = (f_1, f_2, ...f_k), (f_2, f_3, ...f_{k+1}), ...(f_{n-k}, f_{n+1-k}, ...f_n)$ where n is the length of $p \in P$. Let $F^k(p_i)$ represent the k-mer starting at position i in p, and $F^k(p_i)_j$ represent the feature in position j of $F^k(p_i)$. Similarly let $D^k(p)$ give the translated feature family k-mers of sequence p, such that $D^k(p) = \{(d_1, d_2, ...d_k), (d_2, d_3, ...d_{k+1}), ...(d_{n-k}, d_{n+1-k}, ...d_n)\}$. For any set of sequences S given by G, let $D^k(S)$ be union of all k-mers given by $D^k(s)$ for $s \in S$.

2.4.1 RF-graph creation

The pan-synteny graph is created using an annotated double stranded de Bruijn graph [40], that we introduce as a reverse fragment graph (rf-graph) $R^k = (V^k, E^k)$. An rf-graph is similar to a de Bruijn graph except that both the forward and reverse of a given subsequence are assigned to the same node and edges are labeled to indicate the orientation of k-mer instances in their sequence of origin. Nodes and edges of the rf-graph are referred to as rf-nodes and rf-edges. In pan-synteny graph construction, the vertex set V^k for an rf-graph represents the k-mers of $D^k(P)$. Let ϕ be a k-mer such that $\phi \in D^k(P)$ and let $-\phi$ represent the reverse of ϕ and let $(\phi, -\phi)$ be known as a *fragment pair*. A single rf-node, $v^k \in V^k$, represents a single fragment pair. Let H be a hash function that identifies whether a k-mer instance, ϕ, is the forward or reverse instance of any fragment pair such that $H(\phi) \mapsto v^k \in V^k$ for all $\phi \in D^k(P)$.

Each rf-node is annotated with all the instances of it's underlying fragment pair found in G. As in all de Bruijn graphs, adjacent nodes are connected by edges if they occur as overlapping subsequences with a $k-1$ overlap in some contig $p \in P$. Let (v_1^k, v_2^k) be a pair of rf-nodes connected by an rf-edge. An edge is created between v_1^k and v_2^k if and only if there is a prefix or suffix of length $k-1$ in the underlying k-mers of v_1^k and v_2^k representing at least one instance of a $k-1$ substring in P at the same position, in the same contig, p. To retain the information about which portion of the k-mers have this overlap, every rf-edge is given a direction, category, and feature annotation which allows this information to be recovered.

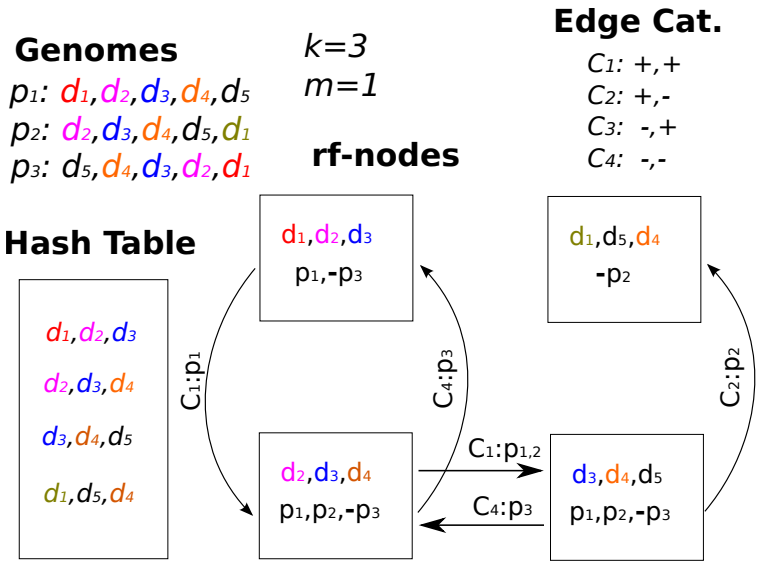

Figure 2.1: Creation of rf-graph

The rf-graph and its subsequent processing leverages the edge types found in a double stranded de Bruijn graph and maintains the multigraph view of an A-Bruijn graph. Specifi-

cally, the view that the features $f_1, f_2, ..., f_n$ in a given $p \in P$ label a set of directed edges in the rf-graph and that those edges have more than one label from one or more sequence. As in the A-Bruijn graph [53] we refer to the number of features that label a given node as its multiplicity. As part of the rf-node creation process, a hash function is applied to k-mers, which performs a lexicographic comparison of the beginning and ending characters of the k-mer. If the character at the end is less than the beginning, the k-mer is flipped. If the beginning and end character are equivalent then the adjacent characters are compared. This continues until an inequality is found that determines the orientation or the middle of the k-mer is reached in which case the orientation remains unchanged from its source genome (Figure 2.2).

The hashing algorithm (Figure 2.2) enforces a consistent ordering of k-mer and its reverse and insures that a k-mer and its reverse are hashed to the same bin in the hash table. This enables genomes to be placed in correct syntenic context within the rf-graph regardless of the direction of sequencing, and subsequent ordering of feature families. Because we want to maintain sequence continuity in processing the rf-graph and the downstream ps-graph, each of the $n - k$ transitions between k-mers of a given sequence $p \in P$ of length n is modeled as a distinct set of $n - k$ edges or edge labels. This means that for a given sequence the transition from one position to the next can be specifically referenced when traversing the graph.

```
1:  function H(φ)
    ▷ INPUT: A k-mer φ in alphabet Σ_D
    ▷ OUTPUT: Hash key for identifying rf-nodes, boolean whether key is reverse of k-mer,
    boolean whether the k-mer is a palindrome
2:      palindrome ≔ 0
3:      reversed ≔ 0
4:      key ≔ φ
5:      if φ = reverse(φ) then
6:          palindrome ≔ 1
7:      else
8:          for i ∈ [0, k − 1] do
9:              if φ_i < φ_{k−1−i} then
10:                 break loop
11:             else if φ_i > φ_(k − 1 − i) then
12:                 key ≔ reverse(φ)
13:                 reversed ≔ 1
14:                 break loop
15:     return key, reversed, palindrome
```

Figure 2.2: The procedure for hashing a k-mer and it's reverse to a single node.

Each bin in the resulting hash instantiates an rf-node. Edges in the rf-graph, an rf-edge, are created between rf-nodes that represent neighboring k-mers in the sequence from a source

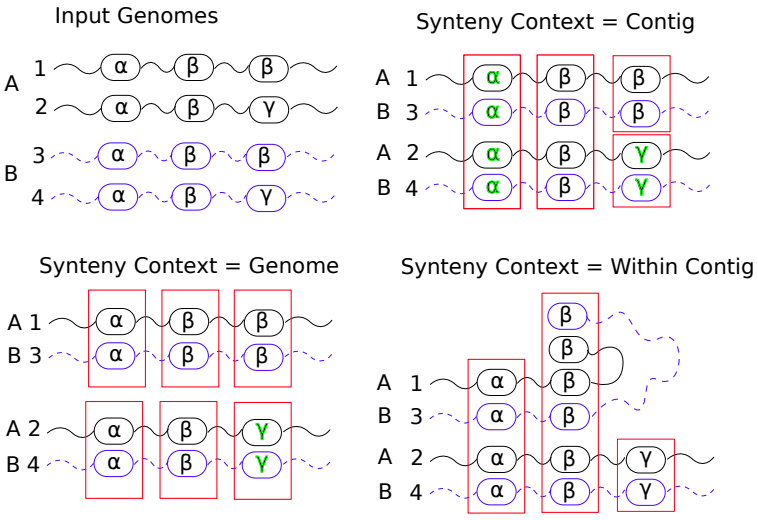

Figure 2.3: The synteny context is given as input to the TFS-Alignment algorithm and dictates how alignments are started. Here the red boxes show the default alignments in each case and correspond to the ps-nodes to be created.

genome (Figure 2.4). The rf-edges are directional, taking on the direction from the source k-mer to the next one in sequence. Each rf-node is annotated with the features $f \in F$ from the genome(s) of origin and nucleotide coordinates of those features such that the original sequence of features or feature families can be recovered for any given $p \in P$.

To account for the reversal of some k-mers, rf-edges are assigned categories to distinguish the orientation of its incident k-mers in the original sequence. A kmer's reversal relative to the original genome sequence is represented with a $-$ and the absence of reversal with $+$. The rf-edge categories can then be given as directed edges in the form $\{(+,+),(+,-),(-,+),(-,-)\}$. Due to the constraints of the reversing hash function and directional edges, only 1 category can exist per directed edge. When traversing from one rf-node to another rf-edges can be used to determine the orientation of the next k-mer and which side of the $k-1$ ovelap will be appended in the next rf-node. The labels and their categories are used in subsequent transformation from an rf-graph to a ps-graph.

In order to create syntenic blocks in accordance with the syntenic context, pan-synteny graph construction uses *sequence bins* and *context bins* to detect duplication during rf-graph construction. A *sequence bin* is a subset of sequences within a genome and a *context bin* is the set of k-mers from those sequences that are used to determine if a k-mer is a duplicate. When building an rf-graph both the sequence bin and context bin are populated per genome or per sequence depending on the synteny context. If the synteny context is at the genome level, C_G, then a sequence bin, S, is populated by all sequences for a given genome g, $S = \pi_P(g)$. If the synteny context is at the contig level, C_P, then a sequence bin, S, is populated by the contig p currently being processed. If the synteny context is at the feature level, $C_\varnothing$ then the sequence bin S is empty. A *context bin* B, is populated by the union of all k-mers that

occur in S, $D^k(S)$, according to the reversal hash function H, $B = \bigcup_{\phi}^{D^k(S)} H(\phi)$. In rf-graph construction (Figure 2.5), as each sequence $p \in P$ is processed, context bins B are created and populated with k-mers, to determine if a k-mer occurs more than once in a sequence bin. The number of context bins will depend on the target context: if C_G there will be $|G|$ unique context bins; if C_P there will be $|P|$ uniqe context bins; and if $C_\varnothing$ then the context bin will remain empty effectively resulting in no duplicate detection.

The choice of this *syntenic context* impacts the nature and number of sytennic blocks that will exist in a given genome group, G, and the amount disambiguation that must occur to represent that context. If a k-mer has more than one instance *within* a sequence bin it is defined to be a duplicate. We define a *μ-block*, μ, to be a subset of features $\mu \subset F$, assigned to a ps-node, that fixes a single position on at most k k-mers, such that no $p \in S$ is represented more than once. A μ-block can be thought of as a subset of the instances of a particular family $d \in D$ defined according to the synteny context and corresponds to the red blocks shown in in Figure 2.3. A μ-block can be thought of as a subunit of a synteny block for a given syntenic context C and represents a grouping of features that will be used to annotate a node in the ps-graph.

If a k-mer occurs more than once in a sequence bin, then some disambguation must occur to group the features in such a way that a μ-block is created using only one instance of d from the sequence bin S for the duplicate k-mer in question. The importance of this can be highlighted by an example; if $C = G$ and more than one feature from a given sequence bin is included in a μ-block, then the block would represent both within genome synteny and between genome synteny, making it an ambiguous representation of C. An rf-node annotated with k-mer instances stemming from m or more context bins contains k or more μ-blocks. Distinguishing multiple μ-blocks assigned to the same rf-node will be described as part of ps-graph construction.

The context given as an input parameter to Panaconda governs which nodes in the rf-graph are used as anchors to begin an alignment. If a k-mer occurs more than once within any context bin in then its corresponding rf-node is marked as a *duplicate*. If a k-mer is a palindrome then its corresponding rf-node is also labeled as such. All rf-nodes that are neither duplicates nor palindromes are *anchor* nodes (see Figure 2.4). *Anchor* nodes serve as the "gluing" agent to begin an alignment. Alignments are created according to the following constraints:

Constraint 1 All features underlying the same character position in an rf-node are assigned to the same ps-node if their coincident annotation does not violate the syntenic context as defined by Constraint 2

Constraint 2 Two or more features from a single character position in an rf-node, that stem from a single context bin, should be assigned to separate ps-nodes unless their coincident annotations are specifically permitted by Provision 1.

Constraint 3 Each feature found in G should be assigned to only a single ps-node.

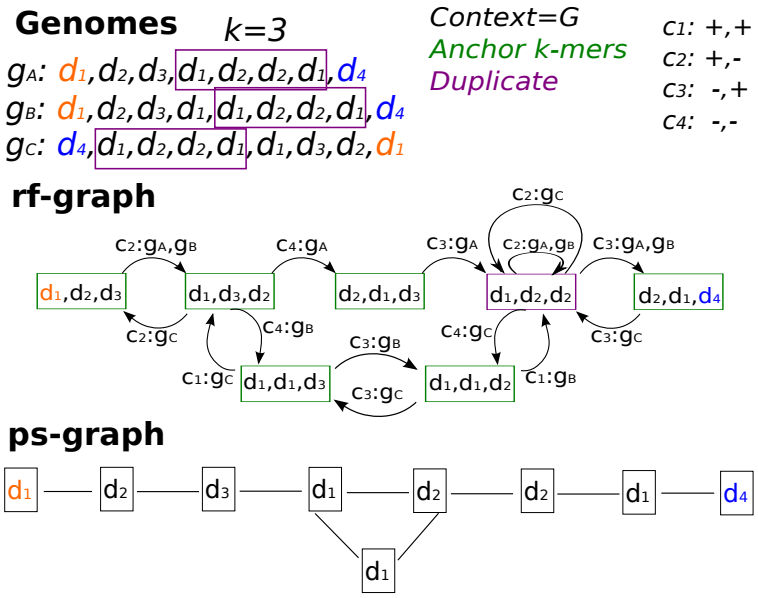

Figure 2.4: The rf-graph and resulting ps-graph given three sequences, $k = 3$, and C_G. Although not shown explicitly here the rf-edges can be travelled in reverse. In this case the sequence is effectively reversed and the compliment of the edge type is used.

Provision 1 Paralogs (2 or more members of a single feature family belonging to the same genome), assigned to the same character position in a k-mer may co-annotate a ps-node if this co-annotation is suggested by two different anchor-nodes and the paralogs do not come from the same sequence. Such a co-annotation may be labeled as a violation of the syntenic context.

Constraint 2, and the notion of syntenic context, is defined in part to deal with k-mers that are duplicated within a certain context bin. If k-mers did not repeat within a genome these notions would not be necessary to represent synteny. Because k-mers are duplicated they can be ambiguous in terms of making a definite assignment in satisfying the constraints. As such the partitioning, or grouping, of features by way of ps-node assignment is determined using information from rf-nodes which are not ambiguous. In these cases, without some provision, Constraints 1-3 could be met trivially by assigning all features to separate ps-nodes. This would not represent synteny, as no features would be coincident, and defeat the purpose of the graph. This leads to Provision 1.

Table 2.1: Label categories applied to the rf-edges for rf-nodes(v_1, v_2)

C1	$+1$	$+1$	Both v_1, v_2 in original order in the context of $p \in P$
C2	$+1$	-1	Second node, v_2, from the pair (v_1, v_2) has been reversed
C3	-1	$+1$	First node, v_1, from the pair v_1, v_2 representing e has been reversed
C4	-1	-1	Both nodes v_1 and v_2 have k-mers that are reversed

1: **function** InitData(C,$D^k(P)$)
▷ INPUT: C the context for the synteny alignments $\{G, P\}$
▷ INPUT: $D^k(P)$ the set of k-mer series for all $p \in P$ using Σ_D ordered by $g \in G$
▷ OUTPUT: $R = (V^k, E^k)$ an rf-graph
▷ Let $V^k[key]$ return a $v^k \in V^k$ if one exists, or create a new one at key if none exists
▷ Let $G(\phi)$ give an ID for the genome g that contains ϕ
▷ Let $P(\phi)$ give an ID for the sequence p that contains ϕ
2: $n \coloneqq 0$ ▷ Counter that uniquely identifies every feature
3: $B \coloneqq \emptyset$ ▷ Context bin
4: $g' \coloneqq \emptyset$
5: $p' \coloneqq \emptyset$
6: **for** $\phi \in D^k(P)$ **do**
7: **if** $C = G \wedge g' \neq G(\phi)$ **then**
8: $B \coloneqq \emptyset$
9: **else if** $C = P \wedge p' \neq P(\phi)$ **then**
10: $B \coloneqq \emptyset$
11: $(key, rev, pal) \coloneqq \mathrm{H}(\phi)$
12: $v^k \coloneqq V^k[key]$
13: **if** pal **then**
14: $v^k \leftarrow palindrome$
15: **if** $key \in B$ **then**
16: $v^k \leftarrow duplicate$
17: $v^k \leftarrow [n, n + k - 1]$ ▷ annotate node with next k features
18: $B \coloneqq B \bigcup \{key\}$
19: **if** $p' = P(\phi)$ **then** AddEdge($v^k, v^{k'}, rev, rev'$)
20: $rev' \coloneqq rev$
21: $v^{k'} \coloneqq v^k$
22: $p' \coloneqq P(\phi)$
23: $n \coloneqq n + k$
24: $g' \coloneqq G(\phi)$
25: **return** V^k, E^k

Figure 2.5: Procedure for creating an rf-graph

2.4.2 PS-Graph Creation

TFS-Traversal

Starting from an anchor node in the rf-graph we employ a search strategy, which we refer to as thread first search (TFS). The thread first search algorithm assumes a multi-graph $M(V, E)$ view of an rf-graph labeled with sequences from a set P where a given $p \in P$ of length n corresponds to n labels $f_1, f_2, , f_n$ such that each is assigned to a node in M. In this case every node in M is labeled with at least one label from a $p \in P$. For a sequence of features $f_1, f_2, ..., f_n$ let the pair f_i and f_{i+1} represent adjancent features in the sequence for $1 \leq i \leq n$. Following the ordered labels of each element of a sequence $p \in P$ corresponds to a walk of the graph. We refer to the set of edges that correspond to a walk of a given sequence as a thread. Anchor nodes are labeled according to the context bins previously defined.

In the TFS algorithm nodes are queued for visiting based on the feature labels and their corresponding threads. On visiting an anchor node, all threads with features present in that node are *activated*. If a thread in a node is *active* then it's edges incident to the current node are used to queue adjacent nodes. For a given visit, nodes can only be queued once, though more than one active thread may induce its queuing. For a queued node v_q all threads that induce the queueing of v_q are grouped together and passed on to the visit of v_q. We call this grouping of threads a *bundle*. Bundles are necessary because in TFS traversal non-anchor nodes do not activate threads. Thus only threads that are pre-activated by way of a bundle are queued on a visit to a non-anchor node.

Starting at an anchor node TFS *activates* all threads present in that node, and adds to the visit-queue those adjacent nodes which have connecting edges from active threads. Because a visit to a non-anchor node only queues those adjacent nodes whose threads are already active in the incoming bundle, an anchor node only queues adjacent nodes once and a non-anchor node may be visited and queue adjacent node visits multiple times.The addition of a thread to a bundle via a feature f_i is encoded in a two step process. First the next feature f_{i+k-1} or f_{i-k-1} that is to be added at one end of the $k-1$ overlapping characters is projected according to one of the four rf-edge types. The projected feature is then added to a feature packet which is passed forward to the visit of v_q.

For a node v, k-mer index x (where $1 \leq x \leq k$), let Fv_x be all features at position x in the k-mer represented by v. For a feature f_i at position i of a sequence p let $V^{+1}(f_i, v)$ give the adjacent feature $\hat{f}_r$, and corresponding node $\hat{v}_r$, that shifts the reading frame for v in the sequence p, containing f, to the right. Likewise let $V^{-1}(f_i, v)$ give the adjacent feature $\hat{f}_l$ and corresponding node $\hat{v}_l$ that shifts the reading frame to the left. For a given rf-graph, we give the following detailed procedure for thread first search.

1: **function** TFS(M, v, v_p, T)
▷ INPUT: M an rf-graph with anchor, duplicate, and palindrome labels
▷ INPUT: Node currently being visted $v \in V$ and thread bundle T
▷ For a node v being visited let v_p be the previous node.
2: Queue $= \emptyset$
3: **if** v is an anchor node **then**
4: $F' = F(v_k) \bigcup F(v_1)$
5: **else**
6: $F' = T$
7: **for** Features $f_i \in F'$) **do**
8: $\hat{f}_r, \hat{v_r} = V^{+1}(f_i, v)$
9: $\hat{f}_l, \hat{v_l} = V^{-1}(f_i, v)$
10: **if** $\hat{v_l} \neq v_p$ and f_i not exhausted in v **then**
11: **if** $\hat{v_l} \notin$ Queue **then**
12: $T' = \{f_l\}$
13: Append $(\hat{v_l}, T')$ to Queue
14: **else**
15: Update Queue$(\hat{v_l}, T')$ by adding f_l to T'
16: **if** $\hat{v_r} \neq v_p$ **then**
17: **if** $\hat{v_r} \notin$ Queue **then**
18: $T' = \{f_r\}$
19: Append $(\hat{v_r}, T')$ to Queue
20: **else**
21: Update Queue$(\hat{v_r}, T')$ by adding f_r to T'
22: Mark f_i as exhausted in v
23: **while** Queue $\neq \emptyset$ **do**
24: $(v_n, T') =$ Pop Queue
25: TFS(M, v_n, v, T')

TFS-Alignment

To create a pan synteny graph Panaconda runs a process designated TFS-alignment. In the manner of the ABA-alignment algorithm [53] and the POA algorithm [36], the TFS-alignment process "glues" features together by emitting a new node in the ps-graph with annotations that indicate the features being aligned in that node. Given an rf-graph for a set of genomes G, the TFS-alignment visits each node according to TFS-traversal, and emits connected nodes in the ps-graph. Depending on the context C, and the resulting anchor nodes, alignments will be started between sequences in different genomes (C_G); between sequences in both the same genome and different genomes (C_P); or between and within sequences in the same or different genomes ($C_\varnothing$) (see Figure 2.3).

TFS-alignment is performed such that by default all features underlying the same character position in an rf-node are assigned to the same ps-node. As we shall make clear subsequently

it is not always possible to do this and maintian the full-path property of the ps-graph. Another property of the ps-graph is that each feature $f \in F$ given by G is only annotated to a single node in the ps-graph. The TFS-Alignment algorithm can be started on any available anchor node. In Panaconda, the TFS-alignment is started iteratively on anchor nodes in decreasing order of thread multiplicity. We refer to the process of visiting an rf-node and creating ps-nodes as *expansion*. In the course of an rf-node expansion anywhere from 1 to $k + r$ ps-nodes will be created, where r is a number of extra nodes emitted. By assigning features to the same ps-node we say they are *glued* together. In the default case, the first rf-node visited is expanded to k ps-nodes. For a given traversal, after the first node only the kth family of an adjacent rf-node outside the $k-1$ overlap need be expanded unless new threads are activated. We refer to this position in the k-mer as the *target position*. Which side of a k-mer the target position occurs on can be determined by the rf-edge class traversed. The target position is always used to determine which nodes to queue next.

As seen in the TFS-traversal procedure a new thread is activated at an anchor node when it is incident on that node but not part of in the incoming bundle. When a new thread is activated all k features in that thread are assigned to ps-nodes, $k-1$ of those to existing ps-nodes for the $k-1$ pre-processed positions. In this case both the target position and the position on the other side of the k-mer are used to queue nodes for subsequent visits. If the node to be queued represents the previous node visited then the bundle information is returned in the unwinding phase, similar to that of depth first search, of the TFS-traversal. Figure 2.4 gives an example of the TFS-alignment conversion from an rf-graph to a ps-graph. An rf-graph can be viewed as a specially labeled double stranded de Bruijn graph, $DB*(G, k)$ where $k \geq 3$, with labels of k characters, in this case feature families, on the nodes and the transition of 1 character in some set of set of sequences from P labeling directed edges. The TFS-alignment can then be viewed as a conversion, under certain constraints, to a graph with a single character labeling nodes and an edge representing a 1 character transition in a block of at least k characters (or nodes) with multiplicity of m or higher.

In order to construct a pan-synteny graph an rf-graph is traversed and each rf-node, v^k, is processed so that each of the k feature families in the k-mer, are represented by a distinct ps-node. There are two forms of low complexity which present a challenge in this process. One is k-mers which are palindromes, defined as a k-mer that reads the same both backwards and forwards. Some assignment procedures of features to families can lead to repetitive, i.e. low complexity regions, of sequence. For example if every protein of unknown function is assigned to the same hypothetical family then there may be stretches of the same protein family assignment in a poorly annotated region of the genome. This can lead to k-mers that are palindromes. Because the rf-graph hashes k-mers which are identical in the forward and reverse to the same rf-node, palindromic k-mers inhibit definite ordering and annotation of rf-nodes with features. The other form of ambiguity stems from duplicate k-mers which occur more than once within a context bin. Duplicate k-mers create ambiguity depending on the syntenic context represented by the pan-synteny graph. In order to disambiguate these cases an rf-node $v^k \in V^k$ is marked as a *duplicate node*. In either case the definite

ordering and assignment of features to ps-nodes using rf-nodes requires disambiguation using additional information encoded in non-ambiguous nodes.

Collisions

As part of the alignment process ps-nodes are created and annotated with the feature, sequence, and genome given as input to Panaconda. Due to complex patterns of conservation and shuffling it sometimes happens that TFS-alignment, in accordance with the bundling information generated at anchor nodes, will attempt to assign the same genome, or sequence in violation of the given context C_G or C_S. In essence the alignment instructions generated at two different anchor nodes conflict with each other under the constraints of the context C. We refer to this conflict in alignment information as a *collision*. We show here that collisions are a valuable source of information for the recognition of evolutionary relationships. Unless otherwise stated it is safe to assume for the purposes of this explanation that the context is set to be "between genomes" C_G.

An anchor-node is an rf-node which is not marked as a duplicate or palindrome and serve as starting points for the transformation of a rf-graph to a pg-graph. In order to construct a pan-synteny graph an rf-graph is traversed and each rf-node, v^k, is processed such that the instances $f \in F$ annotated to each position j in a k-mer $D^k(v^k)_j$, are assigned to ps-nodes. A μ-block is created according to bundling instructions received at an *anchor-node*. Feature annotations are transferred from rf-nodes to ps-nodes in μ-blocks. From a given anchor node, v^k, a μ-block is instantiated or expanded according to the constraints of the synteny context. If two features $(f_i, f_j) \in F$ are both annotated to a single ps-node $v \in V$, then there exists some anchor node $v^k \in V^k$ that establishes a bundle which suggests the creation of a μ-block with (f_i, f_j). Anchor nodes define a grouping of features, a partition of F. As is demonstrated in Figure 2.6, the variable and repetative nature of biological sequences can lead to a situation where two different anchor nodes suggest two different groupings for a particular feature or suggest the creation of a μ-block containing two different positions from a single sequence bin (Figure 2.7).

In the default case features sharing the same position in an anchor k-mer are assigned to the same ps-node. Due to short repeat sequences it is not always possible to both group all features sharing a position in an anchor k-mer, and maintain the full-path property of the ps-graph. We refer to this situation as the "shift" problem. The shift problem is caused by repeat characters and the solution is analogous to creating a gap in the alignment representation of a sequence to accommodate an extra repeat character in another.

As seen in Figure 2.6, the third character in sequence p_1 (reading left to right) can be "shifted" to the right in the alignment if the rf-node corresponding to $\alpha_3, \alpha_4, \alpha_5$ is processed first. Shifts are often detected when activating a new thread at an anchor node. As a consequence of the tfs-traversal algorithm passing forward only bundle information that is relevant in the adjacent node, threads that have already been processed can be "rediscovered" when visiting

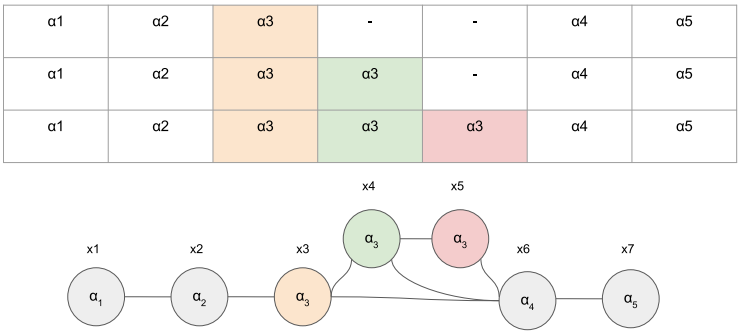

Figure 2.6: Due to repeat sequences alignments can be shifted giving variable annotation to resulting ps-nodes. The top shows the given sequences in a traditional multiple sequence alignment arrangement (from which the configuration of the rf-graph can be determined). The bottom is a possible ps-graph if $k = 3$.

an anchor node further in the traversal process. In this case a rediscovered thread and it's corresponding features may have already been partially assigned. The TFS-alignment algorithm treats these features and the corresponding ps-nodes as available positions for alignment for the features of all activated threads. Continuing with the example in Figure 2.6, if the path in the rf-graph that corresponds to creating the ps-edge $(x3, x6)$ is taken first, then the threads for p_2 and p_3 are rediscovered. When this happens the tfs-alignment algorithm will investigate an alignment to a previous instance of the repeat character in the same sequence. In such cases the context constraint determines if this is viewed as a collision. In the case where the path corresponding to p_2 is taken first, a shift will still occur relative to p_3. In both these cases the TFS-alignment emits an extra node to represent the extra character and preserve the full-path property. If the highest repeat path is traversed first then three nodes are emitted with no special exception needed and only the direction of approach determines the resulting shift. We note that based on varying traversal order, the aligning and emitting of nodes may result in alignment graphs that are not isomorphic. When a set of multiple ps-nodes are available for alignment at a fixed position in an anchor k-mer, they are all assigned an identical "alternate ID" in the output expressing this group relationship. We leave the possibility of treating these groups as a kind of hypernode in the pursuit of isomorphic alignments as a subject for future development.

The shift problem and resulting extra nodes turns out to have large downstream effects. As previously described, when new threads are activated a portion of the k-mer at the rf-node will represent pre-processed positions and will have corresponding ps-nodes at each position. Because repeats can "stack up" at a given location in the alignment a choice arises as to which ps-node to assign an activated feature to. For this purpose we introduce the concept of an *instance key*. For a given feature f an instance key $I_{key}(f)$ has two components. The first component we refer to as the *i-component* and is a set representation of all rf-nodes (k-mers)

in which a feature occurs. Depending on its position in it's source sequence a feature f may have an i-component from size 1 to k. Let all rf-nodes V in an rf-graph $M(V, E)$ with n number of nodes $|V| = n$, be assigned an integer ID value 1 to n. Let f_t be a feature $f_t \in F$ from a sequence $p \in P$ that occurs in rf-nodes with ID's h, i, j; then the i-component of it's instance key can be represented as $\{h, i, j\}$.

The second component of an instance key, the r-component, is the length of the character repeat, if any, in which the feature occurs in p. The r-component is encoded in such a way that its value is disjoint from the set of possible values for the i-component. Let the same feature f_t occur in a repeat of size y in sequence p, and it's r-component be represented as $Rt(y)$. Given a subsequence $f_1, f_2, f_3, f_4, f_5, f_6$ of $p \in P$ that corresponds to a character representation $d^1, d^2, d^2, d^2, d^3, d^4$, such that the ith position in one corresponds to the ith position in the other: we give the r-component value of feature f_2 as $Rt(f_2) = 3_{repeat}$. An instance key is the union set of these two components such that for f_t, $I_{key}(f_t) = \{h, i, j, Rt(y)\}$.

For a visit to an rf-node v in which new threads are activated, let $\hat{F}(x, v)$ be those features at a fixed position x, $1 \leq x \leq k$, in v that are available for ps-node assignment. Also for a given visit to an rf-node v during TFS-alignment: let $R(V', E')$ represent the ps-graph constructed so far; let $\bar{F}(x, v)$ be those features that are already assigned to a ps-node and implicated for alignment in the current bundle; let $I_{key}(\bar{F}(x, v))$ give the instance keys of those features that are already assigned; and finally let $V'(f)$ give the ps-node to which a feature has been assigned. If at position x there is more than one feature, $\bar{F}(x, v)$, assigned to more than one ps-node, $|\{V'(f) \mid f \in \bar{F}(x, v)\}| > 1$, then ps-node assignments for each $f \in \hat{F}(x, v)$ are determined by their respective instance key. In this case the instance keys of the unassigned features are compared to the instance keys of the assigned features.

Let f be an unassigned feature $f \in \hat{F}(x, v)$ that we wish to assign, and i be its instance key $\hat{i} = I_{key}(f)$; the assignment of f is determined by finding an instance key $z \in I_{key}(\bar{F}(x, v))$ that has the largest intersection with $\hat{i}$. If there is a tie between instance keys, then the one representing the ps-node with the most assignments is chosen. If when attempting the assignment of f a shift collision occurs, a new node in the ps-graph is created to which f is assigned. The r-component of the instance key helps to discriminate between available assignments when long stretches of repeat characters are present. Edges in the ps-graph are created between ps-nodes that have feature assignments which are consecutive in some $p \in P$. Through this procedure all featuress with the same instance key are guaranteed to be assigned to the same rf-node, and the full-path property of the ps-graph is maintained.

Shift collisions occur when bundling information attempts the alignment, in context C_G or C_S, of two parts of the same sequence due to repeat characters. In the case of C_G when two bundles suggest the alignment of two sequences from the same genome it can be indicative of a relative rearrangement that may be a fusion, fission, translocation, or misassembly. When a collision happens at the ends of two sequences from the same genome it can be indicative of a missed join. In terms of the TFS-alignment procedure when collisions occur between two different sequences in the same genome no structural change is made to the

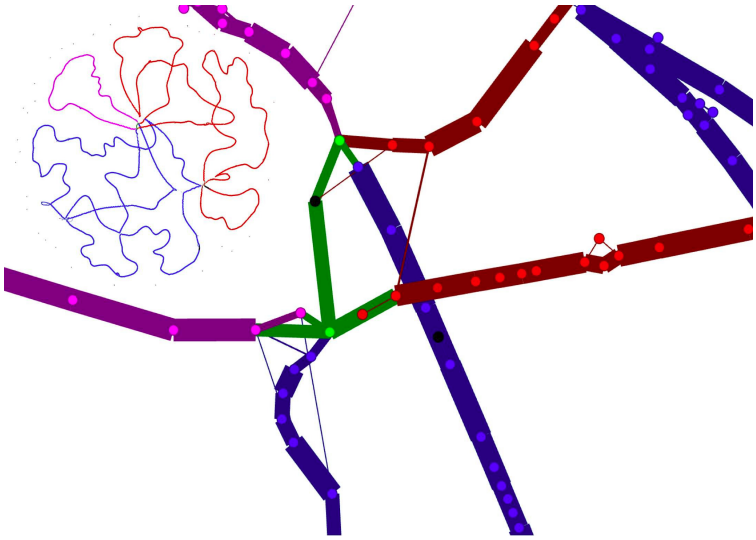

Figure 2.7: The region shown in fuchsia is typically found on chromosome 2 of *Brucella suis*. In the example shown a translocation has occurred in strains 1330 and VBI22 placing the region on chromosome 1. A zoomed out view is shown in the top left of the image.

graph. Panaconda creates a record of all such collisions and the ps-nodes are marked as being involved in a collision. When collisions occur the type of event it implies will often depend on the "threading" pattern that results. In Figure 2.7 a graph of 42 *Brucella suis* is shown with a large translocation between replicons in two of the organisms. In this case the two collision points annex a region that has moved from one chromosome to another.

In some cases a collision can occur that tries to align a sequence *to itself* which does not stem from repeat sequences. When this occurs it is because at least two genomes in G each join two regions in the other that are not contiguous. This relative rearrangement is the result of an inversion. In Figure 2.8 we show an inversion in a graph of 2 *Brucella melitensis* strains, 1 *Brucella microti*, and 1 *Brucella abortus*. The inversion is known in *Brucella abortus* bv. 1 str. 9-941 [70] and occurs on NC_003318, at between BMEII1009 hypothetical protein and BMEII0292 a putative Heme-regulated two-component response regulator.

2.5 Results

The panaconda software generates the pan-synteny graph, and optionally the rf-graph, in GEXF format. This makes the resulting alignment portable and usable in an number of other programs including: Gephi [6], NetworkX [62], and Cytoscape. Each node is labeled with the genome, sequence, and feature IDs assigned as part of the alignment process. Because the number of genomes represented in a graph (hundreds possibly thousands of

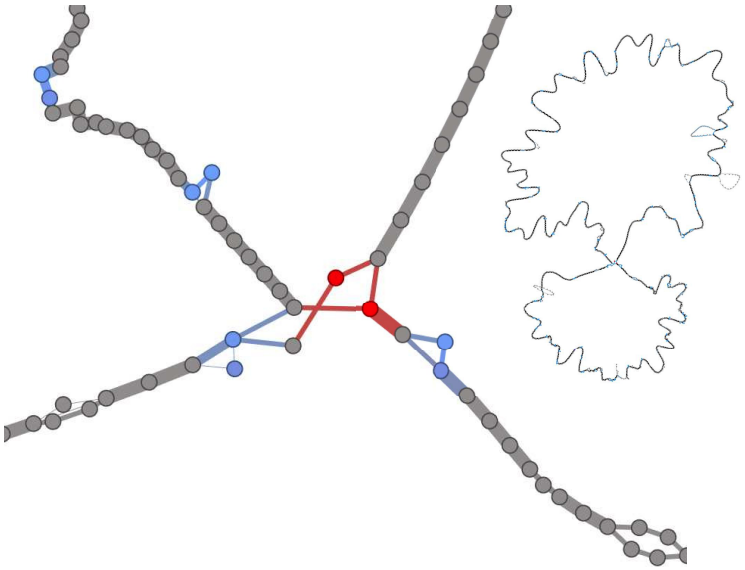

Figure 2.8: A zoomed (left) and high level perspective (right) on an inversion in *Brucella abortus*. The inversion collision is shown in red and shift collisions are shown in blue.

genomes) the amount of metadata associated with the graph can be quite large. For pan-genome graphs in general a index lookup scheme using a BGZF compressed backing store may improve portability. Panaconda's graph creation component is implemented in python using NetworkX. The component for laying out the graph is written in Java and is based on the Gephi toolkit. The visualization component is based on the Gexf-JS software, a browser based interactive Gexf viewer, and is written in Javascript and HTML.

2.5.1 Behavior

A pan-synteny graph models the conservation in a set of strings using a neighborhood size k. The higher the value of k the more stringent the requirement of conservation amongst sequences in P for merging into a single pan-node. A larger k will lead to an increase in unique rf-nodes but, for any m value greater than 1, a smaller proportion of rf-nodes will satisfy the minimum genome requirement and create ps-nodes. It also likely that an increase in k will lead to fewer hub nodes. When connecting very diverse regions from genomes in P, hub nodes can be problematic in using the pg-graph to visually and algorithmically understand global syntenic relationships. A higher k should lead to fewer rf-nodes meeting the requirement m.

To determine whether the pan-synteny graph is modelling the similarities and differences among a group of genomes in a resonable way I compare it with two different similarity metrics for whole genomes, MUMi a score based on maximal unique matches and a diversity metric δ. Given a group of genomes P, and an Order level phylogenetic tree T, containing

all genomes in P, $\delta = \sum_{p_i \in P} T_{max}(p_i)$ where $T_{max}(p_i)$ is the maximum distance from p_i to every other $p \in P$ (Figure 2.10). Distances are measured in nucleotide substitutions pers site. Similarly the $MUMi_{max}$ score is based on the largest MUMi value for all pairs of genomes in P. To measure the complexity modeled in a pan-synteny graph (V, E) we compute the node increase over the highest number of families, d_p, for any $p \in P$: $n^+ = (|V| - max_{p \in P}(d_p))/|V|$. This comparison is done using 10 different genome groups, with 10 genomes each, representing a range of diversity. As seen in Figure there is a strong correlation between the node increase n^+ and both the MUMi and diversity metric. In the MUMi plot (Figure 2.9), data points colored in red represent groups whose maximum MUMi score comes from pairs that have an unequal number of replicons.

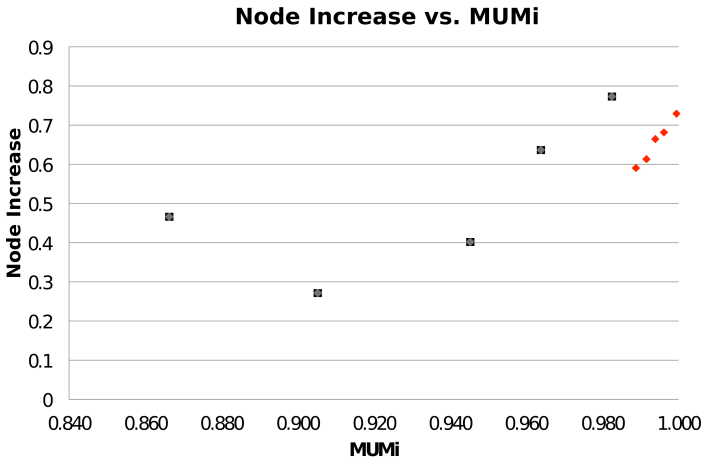

Figure 2.9: Scatter plot of node increase vs. MUMi

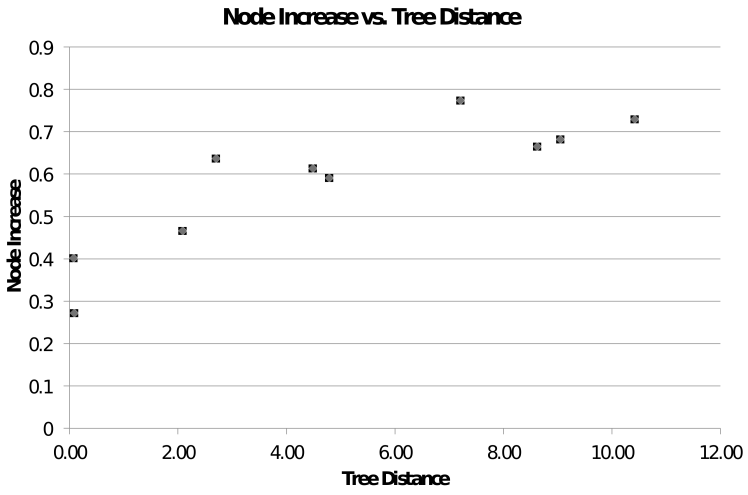

Figure 2.10: Scatter plot of node increase vs. Tree Distance

2.5.2 Performance

Because Panaconda takes advantage of previous computation invested in designating feature family assignments it runs fairly fast, taking under 3 minutes to align all available 565 sequences of *Brucella* from PATRIC (see Figure 2.11). Each visit to an rf-node in the TFS-alignment algorithm is guaranteed to align at least one feature $f \in F$ giving an upper bound of $\mathcal{O}(F)$. Intuitively, genomes that are more closely related are more likely to have more features co-incident on a given anchor node and thus have more features exhausted per visit than distantly related genomes. For example, aligning 100 *Brucella* genomes, a relatively well conserved genus, took 31 seconds and generated a ps-graph with 5,681 nodes compared to aligning 100 *Bacillus* genomes, a more divergent genus, which took 95 seconds and generated a graph with 116,289 nodes. All times were generated on a laptop with a 2.3 GHz Intel Core i7 processor and 16Gb of RAM.

2.5.3 Visualization

Multiple sequence alignment is a well researched and difficult problem [15, 10, 19]. Especially when dealing with diverse sequences that have complex evolutionary relationships [46]. Similarly synteny has been explored often using dot-plots [3, 1]. In both types of analysis the traditional grid based layout does not scale well with increasing numbers of sequences [40]. By using graphs we are able to represent complex rearrangements, and hotspots of evolutionary change without sacrificing detail. The default Gexf-JS shows node attributes when clicking on a node and shows node labels when zoomed in. We have modified the base

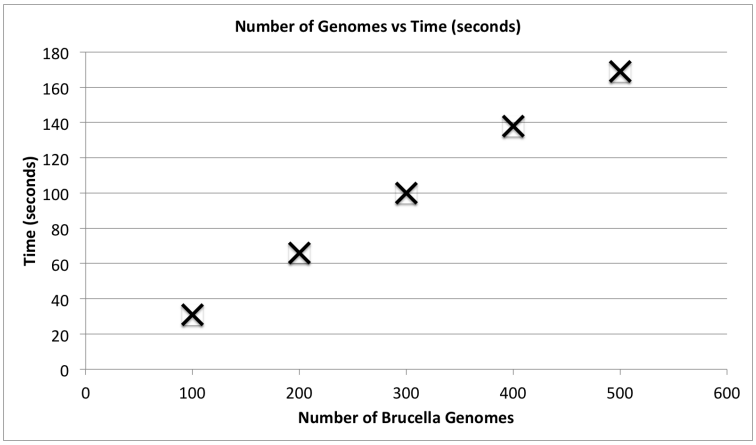

Figure 2.11: Run time of Panaconda in seconds on increasing numbers of *Brucella* genomes

Gexf-JS viewer to expect and display a table of genome, sequence and feature IDs. We have also added the ability of dynamic path highlighting for sequences and genomes.

As seen in Figure 2.12 when clicking in the node attribute table on a genome or sequence ID all nodes and edges corresponding to that ID are highlighted. In this case it allows us to see a suggested scaffolding for the unfinished genome *Mycobacterium tuberculosis* SP21 against *Mycobacterium tuberculosis* H37Rv. We have also configured Panaconda to calculate a diversity quotient based on the genus or species listed in the input annotaion. Based on the selected taxon level and the resulting diversity quotient, nodes are colored from yellow to red from a low to high. The edge weight is set to be the fraction of genomes incident on that edge. With this difference in visual encoding it is possible to see regions where certain species or genera are overrepresented. A command line option is provided to automatically layout the pan-synteny graph that is generated. After trying a number of combinations, we have found that the multilevel agglomerative edge bundling [21] implemented in older versions of Gephi works best. This is followed by Force Atlas 2 [28].

Evolutionary hotspots relative to the genomes in G can be recognized by a high degree of splitting of a strongly conserved block. Figure 2.13 shows several incidents of lateral transfer have occurred, in the genus *Brucella*. Nineteen genes essential to the synthesis of lipopolysaccharide (LPS) and necessary to produce the smooth phenotype have been identified in the classically known *Brucella* [24, 25, 77]. A study of atypical members of this genus [76], and recent studies of *Brucella* isolated from amphibians [67, 14] have shown that phylogenetically more ancient members of the clade have different genes, and build a different LPS. It has been shown that all *Brucella* share flanking regions and a tRNA gene where the inserts of these genes have occurred. The panaconda visualization built on the strains with the traditional *Brucella* O-antigen (*B. abortus* 2308, *B. abortus* 9-941, *B. melitensis* 16M, *B. microti* CCM 4915, *B. inopinata* BO1) and some strains isolated from amphibians (*Brucella* sp. 09RB8471, *Brucella* sp. 10RB9215 and *Brucella* sp. 09RB8910) show that there have been at least three separate insertions into this hotspot.

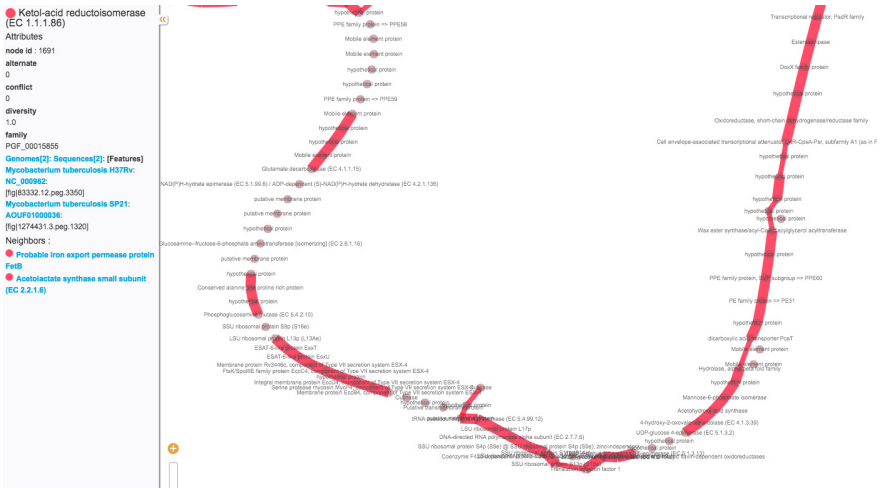

Figure 2.12: A potential scaffolding shown with two *Tuberculosis* strains H37Rv and SP21.

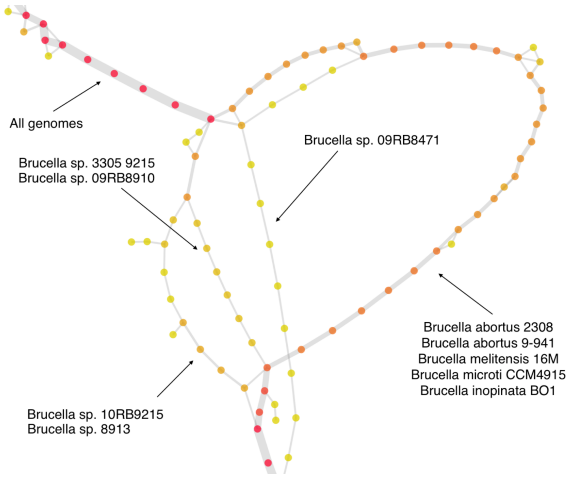

Figure 2.13: A highly divergent region in strains of *Brucella*

2.6 Discussion

It is worth pointing out that pan-synteny graphs generated by Panaconda are closely related to the POA graphs introduced by [36] in that they both: model which positions in multiple sequences are aligned to one another; maintain the ordering of the positions within those sequences; fuse conserved letters into nodes as part of the alignment; and store sequence positions on the generated nodes. However, the approach of these two methods is somewhat different in that Panaconda creates a de Bruijn graph and uses it's traversal to create the ps-graph from sequence conservation in any orientation, whereas POA requires that blocks of alignment occur in the same orieintation. In their work on performing multiple sequence alignments on sequences with repeated and shuffled elements [58] also use de Bruijn graphs to construct an alignment graph called an ABA graph. Besides the obvious difference in alphabet pan-synteny graphs are different from ABA graphs in that the former does not contain cycles when following edge labels corresponding to the original sequence.

The panaconda algorithm to uses multiple conserved k-mers of feature family annotations as the basis of its synteny recognition (multiplicity in the ps-graph greater than one). Like many synteny block finding algorithms our approach requires a form of anchors. Unlike some previous methods [49] there is no requirement that a single anchor be present in all genomes. Unlike many synteny analysis algorithms [55, 46] the TFS-alignment method does not focus on alignment disruption due to loops or bulges [49]. This can be attributed to the abundance of anchor k-mers owing to the large alphabet employed. As a result no editing operations on sequences are needed and the alignment is able to generate a graph where the full-path property is maintained.

We note that the default setting for Panaconda is a multiplicity count of one. In this default case Panaconda creates and processes an rf-graph into a pg-graph which itself represents both syntenic blocks, in graph components with high multiplicity, and simple gene neighborhood relationships for those regions with only a single genome present. We assert that bacterial evolutionary relationships can be better understood when both synteny relationships and unique regions are analyzed together. Because of the complex rearrangements that can take place between and within bacterial genomes explanatory methods that seek to characterize their relationships should be robust to large differences. We demonstrate that the pan-synteny graph is capable of locating and representing a wide range of evolutionary phenomena. By decoupling the comparison of genomes from strict nucleotide and amino acid sequence analysis we are able to broaden or narrow the scope of the analysis depending on the unit abstraction chosen to create the alphabet. With our analysis we supply examples which have three different types of alphabet provided by the PATRIC resource [75]: figfams [44], pg-fams, and pl-fams [16].

Chapter 3

Entropy Estimates for DAG-Based Ontologies

3.1 Motivation

Entropy measurements on hierarchical structures have been used in information retrieval and natural language modeling. Here I explore its application to semantic similarity. By finding shared ontology terms, semantic similarity can be established between annotated genes. A common procedure for establishing semantic similarity is to calculate the descriptiveness (information content) of ontology terms and use these values to determine the similarity of annotations. Most often information content is calculated for an ontology term by analyzing its frequency in an annotation corpus. The inherent problems in using these values to model functional similarity motivates this work.

I present a novel calculation for establishing the entropy of a DAG-based ontology, which can be used in an alternative method for establishing the information content (IC) of its terms. I also compare our IC metric to two other IC metrics using semantic and sequence similarity.

3.2 Introduction

The work detailed here represents a preliminary to further investigation of functional patterns in prokaryotes such that functional conservation can be judged independently of the frequency that terms occur in an annotation corpus. This work focuses on an abstraction of a gene modules known as **functional module** which is defined as a group of genes whose products contribute to the same functional process in the cell [64, 9]. This process can occur at any level of organization in the cellular systems listed above. Because functional modules

are an organizing construct of biologists, the "contribution" of a gene to a functional process is context-dependent. A gene may contribute by providing a structural component to the synthesis of a molecule, regulating expression of an essential enzyme, or by some other means; as long as the conditions for creating the module are meaningful. For example, Wu *et al.* [78] predict a functional module for the signaling of chemotaxis in *E. coli* that consists of four regulatory genes (**cheR, cheA, cheW, cheB**) using a combination of GO similarity, and phylogenetic and neighborhood profiling. The STRING database [30], which incorporates a more diverse set of input data including expression analysis and literature mining, predicts a similar module for *E. coli* but also includes a methyl-accepting chemotaxis protein (**tap**) and a protein phosphatase gene (**cheZ**). In the STRING database this module expands to include additional genes as the scoring threshold for incorporating evidence is lowered. Each of these modules represents a different subset of genes for chemotaxis created under a specific set of conditions. Unless otherwise stated any "module" referred to in this proposal is a functional module.

The popularity of using ontologies in the analysis of biological data has grown rapidly [29] since the introduction of the Gene Ontology [5]. The Gene Ontology is a controlled vocabulary represented by three categories, Biological Process, Cellular Component, and Molecular Function, each represented as a Directed Acyclic Graph (DAG). Assignment of GO terms from a particular GO category to a gene, referred to as an annotation, can be said to describe a property of a gene within the subject defined by that GO category. For example assignment of term GO:0004673 describes a gene as having the histidine kinase property when addressing the subject of its molecular function because it is from the Molecular Function category of GO. The GO graph is constructed such that nodes are controlled vocabulary terms and edges are relationships between thos terms. The edges in the Gene Ontology have a variety of types. For the purposes of this work I use only two, those designating the 'is_a' and 'part_of' relationships. These edges create a subsumption heirarchy such that assignment of a GO term x to a gene implies the assignment of all GO terms on the path from x to the root node of the GO category from which x is taken.

3.3 Background

One application of ontology annotations is to determine the semantic similarity between two or more entities, using ontology terms, to convey the extent of functional similarity. For the purposes of this work, semantic similarity is a measure that quantifies the relatedness of two genes based on their ontology annotations. There are a wide variety of methods for determining semantic similarity based on ontologies [65, 63, 39, 72, 59, 60, 31, 42, 57, 51]. Many of these methods estimate the information content (IC) of ontology terms. The information content of an ontology term is a measure of how specific and how much information that term conveys. The most widely used method for estimating information content is to calculate suprisal [59] which, for an ontology term t with probability $p(t)$, is

given as

$$rIC(t) = -\log p(t) \tag{3.1}$$

. Here $p(t)$ is usually estimated by the frequency that the term occurs in a set of annotated genes referred to as an annotation corpus. I use rIC to distinguish this from other methods used to calculate information content. This method brings with it some inherent problems. Basing IC, and thus semantic similarity, on a specific corpus runs the risk of biasing on research topics that have been more thoroughly investigated and propagated by electronic annotations. A recent paper on behalf of the Gene Ontology consortium [69] highlights the drawbacks of making biological inferences using metrics based on a specific annotation corpus. It suggests that such metrics are subject to the "open world assumption", i.e. that our complete biological knowledge is not represented in gene annotations and if a gene lacks a particular GO annotation it does not mean that the gene lacks that function. Similarly, the frequencies of annotations in a particular corpus are subject to the focus of analyses that have been performed on its constituent genes. Depending on the corpus, semantic similarity calculations using term frequency can give little weight to GO terms that convey a very specific function and occur often, or give significant weight to GO terms that are shallow and seldom used. Because gene annotations are in flux, basing the value of terms in a semantic similarity calculation on the current state of annotation makes the resulting values vulnerable to the open world assumption. Semantic similarity measures that use corpus-based probability estimates make sense when comparing or grouping genes relative to the body of knowledge in a corpus, but, because they do not directly model the conceptual explicitness of a term, they may lose resolution when it comes to relating how functionally similar genes are.

In the case of microbial annotation analysis the construction of a corpus for instance-based calculation of semantic similarity would be problematic. Determining information content of a term from an "annotation corpus" conditions on the bias found in both manual and automatic annotations [51]. Manual annotations can be biased towards ontology terms from a curators area of expertise, or because of a particular research goal, and are limited in availability. Automatic annotations are more widely available but are subject to erroneous assignment via sequence similarity propagation. Also, these instance-based measures will produce varying results based on the corpus (annotation) chosen [72] which adds a degree of inconsistency. Because this project aims to investigate functional patterns based on the occurrence (instances) of various annotations across multiple organisms an instance based method for calculating semantic similarity would likely introduce a confounding factor, bias the result, and potentially limit the sensitivity of the method. Furthermore, the variability among bacterial annotation quality, method of construction, and rate of turnover would have a negative impact on the reproducibility of the overall method.

Because gene annotations represent an ongoing process they must be treated as an incomplete characterization of a gene's properties. Many semantic similarity concepts developed for ontologies average or normalize their similarity metrics in such a way that additional information on a particular gene product actually decreases the similarity to another gene

product. Those methods that only consider the IC of the MICA [60, 39, 31] in determining term-term similarity don't account for the shared information represented by all the DCA. To address this Couto *et al.* [13] developed a method that averages the IC of all DCA in calculating term-term similarity. In effect this uses what should be a positive semantic signal in a negative manner. If two terms share more than one DCA the similarity of those terms will be smaller relative to a similar pair that share only the MICA. In order to treat the semantic signal in a consistently positive way Popescu *et al.* [57] adopted the use of a Sugeno fuzzy measure to determine gene-gene similarity. However, the adaptations made by Popescu *et al.* lead to inconsistent and overabundant assignment of maximum similarity, resulting in a lack of resolution in partitioning the annotation space. In establishing semantic gene similarity many methods take the average term similarity [51] of annotation term pairs. In doing so they treat the semantic signal of annotations negatively by decreasing similarity when additional shared information is present beyond the most specific shared annotation. We maintain that the partial contribution of additional information contributed by less specific terms should add to the similarity not decrease it.

An alternative to estimating the IC of ontology terms from a corpus is using the structure of the ontology itself. Seco *et al.* [63] present a method for determining the IC of terms in WordNet taxonomy [45], a text analysis resource, based on the number of descendants. They argue that the more descendants a concept has the less information it expresses, otherwise there would be no need to further differentiate it. They set the information content of a term t to be

$$sIC(t) = 1 - \frac{\log(\Delta_t + 1)}{\log(|N|)} \tag{3.2}$$

, where $|N|$ is the number of terms in WordNet and Δ_t gives the number of descendants for term t. Though this metric relieves any dependency on using a corpus, it only values the potential for further refinement without considering the number of ancestor terms that have been specified to define a term. As a result, all leaf nodes have the highest possible IC value regardless of their depth in the taxonomy. This too suffers from a kind of "open world assumption" in that leaf terms may be created at any depth within the ontology structure at any time. Intuitively, this can lead to overestimation of how informative a term is. A term may be a leaf simply because it implies a subject that has not been extensively developed in the ontology.
The issues with existing IC estimation and its use in semantic similarity motivate our method for calculating the entropy of a DAG-based ontology and using it to derive a new information content metric that is independent of individual corpus characteristics.

3.4 Entropy Of Ontology

In this paper, entropy refers to Shannon entropy, which is a measure of uncertainty associated with a message, i.e. a random variable, given a message source. In this case the message

source is an ontology and the message is an annotation. I use "information content" to refer to the amount, measured in bits, that an ontology term contributes to the uncertainty that a particular annotation will be made. Given an ontology $M = (N, E)$, let N represent the terms of the ontology, E represent the edges, and r represent the root node. For a term t let let Δ_t represent the descendants of t and Π_t represent the ancestors. To estimate the entropy of a DAG-based ontology I calculate the joint entropy of selecting a pair of random terms from the ontology. In other words, I calculate the uncertainty associated with generating a two-term annotation where X and Y are random variables that represent the first and second term respectively. The entropy of the ontology M is given as

$$H(M) = H(X, Y_x) = H(X) + H(Y_x \mid X) =$$
$$-\sum_{x \in X} p(x) \log p(x) - \sum_{x \in X} \sum_{y \in Y_x} p(x)p(y \mid x) \log p(y \mid x) \tag{3.3}$$

. Where

$$Y_x = \{(M \setminus (\Delta_x \bigcup \Pi_x)) \bigcup r\} \tag{3.4}$$

It is assumed in this case that the ontology given, as with GO, represents a subsumption hierarchy and that any term used in an annotation implies its ancestors. Rather than use the frequency of terms in an annotation corpus, I use a maximum entropy estimate where the probability that the first term is selected for annotation is $p(x) = 1/|N|$ and the second term is $p(y \mid x) = 1/|Y_x|$. For a particular annotation this evaluates the probability of selecting an additional term given that the first term selected precludes its ancestors, descendants, and itself, from being selected as the second term. It is possible to use term frequency in a corpus to estimate the probability as part of this calculation but I do not explore that here. Figure 3.1 illustrates the entropies assigned to various DAG configurations using this method.

3.5 Entropy To Information

In information theory the amount of information transmitted by a signal can be computed as the decrease in uncertainty at a receiver and is given as $R = H(x) - H(y \mid x)$ where $H(x)$ is the uncertainty as to what signal x will be sent and $H(y \mid x)$ is the remaining uncertainty sent after x is received [56]. Although the surprisal of a word is usually taken as its information content, it has been suggested that the reduction in uncertainty after processing a word can be taken as the amount of information it conveys [20, 27]. To explore this idea I propose and evaluate two information content metrics for DAG-based ontologies that use the approximate level of uncertainty in the DAG before and after a term is assigned. The first [74], Formula 3.7, evaluates only the reduction in entropy when considering the elimination of a term's ancestors as a source of further annotation. The second, Formula X, considers reduction in entropy due to the elimination of both ancestors and descendants.

Given the entropy value for a particular ontology I calculate information content for a term z by estimating the conditional joint entropy supposing that terms previous assignment, then taking the difference in the original and conditional entropies. The information content of the term is given as the decrease in entropy created by excluding z and its ancestors.

$$gIC(z) = \frac{H(X, Y_x) - H(X_z, Y_{xz} \mid z)}{H(X, Y_x)} \tag{3.5}$$

$$X_z = \{(M \setminus \Pi_z) \bigcup r\} \tag{3.6}$$

$$Y_{xz} = \{(M \setminus (\Delta_x \bigcup \Pi_x \bigcup \Pi_z)) \bigcup r\} \tag{3.7}$$

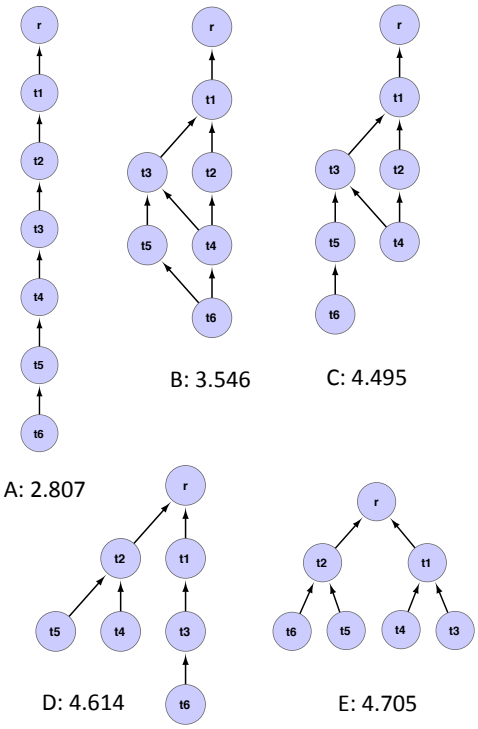

Figure 3.1: Entropy values of various DAG configurations.

For any given term our information content measure reflects the information transmitted in its assignment relative to the uncertainty in the ontology. While this metric may be influenced by a particular concept being underdeveloped, it takes into account both the number of ancestors, the number of descendants, and the overall structure of the graph.

3.6 Semantic Similarity Benchmark

To determine how well our metric models the biological specificity of ontology terms relative to the other metrics available, I compute and compare semantic similarity values for Gene Ontology annotations using rIC, gIC, and sIC as input. The performance of semantic similarity is difficult to evaluate because it attempts to quantify the relatedness of concepts that are typically interpreted by humans. As Pesquita *et al.* [51] point out there can be no way to determine the true functional similarity between two gene products. "If there were, there would be no need to apply semantic similarity in the first place." Because of this, methods are usually qualified by examining their behavior relative to a particular application, e.g. clustering, function prediction, cellular location prediction, and protein-protein interaction prediction.

To analyze performance I compare the correlation between the average semantic similarities and sequence similarity scores [42]. In order to closely tie the resulting semantic similarity value to the value of the information content I use Resnik's maximally informative common ancestor method (SimMax) [59]. To measure sequence similarity I used the relative reciprocal BLAST score (RRBS) [50]. Given two proteins A and B this is given as

$$RRBS = \frac{Bitscore(A,B) + Bitscore(B,A)}{Bitscore(A,A) + Bitscore(B,B)} \tag{3.8}$$

To conduct our testing I randomly selected 1,000,000 protein sequences with GO Molecular Function annotations from the UniprotKB proteome. GO annotations were obtained from Uniprot-GOA. Annotations, sequences, and the GO hierarchy were all obtained on July 2011. A BLASTp search was conducted using an all against all approach. Because I wish to measure correlation with sequence similarity and not the effects of shallow annotations, I filtered annotations so that only terms with a minimum edge depth of two or more remained. This resulted in 3,118,974 unique protein pairs with semantic and sequence similarity values. To make the results easier to compare all information content values were normalized using their respective maximum values before determining semantic similarity.

3.6.1 Benchmark Analysis

To account for the variability in annotation, protein sequence, and biological function I calculate average values for semantic and sequence similarity over fixed intervals. I binned protein pairs into groups by sorting according to increasing RRBS, then by increasing semantic similarity value, and finally creating bins at every 1,000 data points. For each bin the mean RRBS and SimMax value was calculated. Figure 3.2 shows the relationship between semantic and sequence similarity using all three IC metrics.

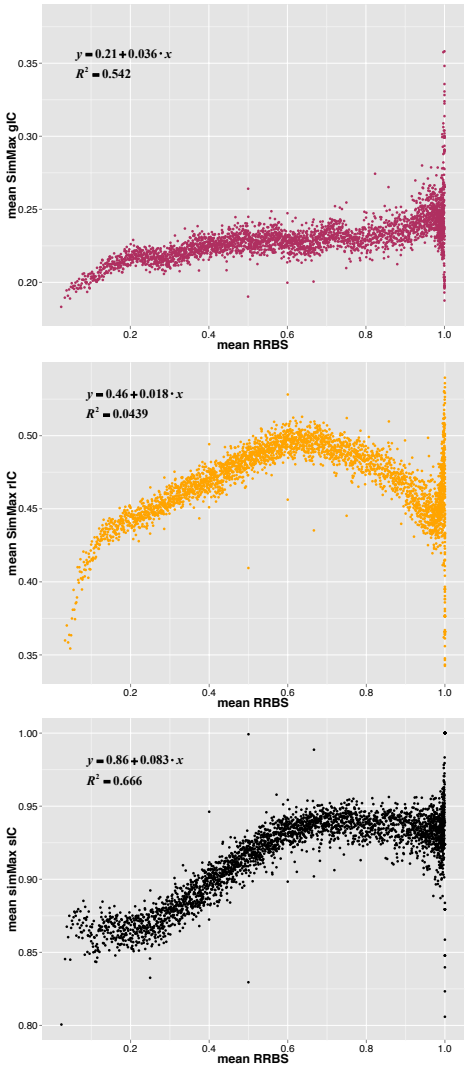

Figure 3.2: SimMax gIC, rIC, and sIC values vs. RRBS.

In our sample 6% of the protein pairs (197,391) were identical, i.e. had an RRBS value of 1. Protein pairs at this level of similarity showed the widest range of SimMax values for all three information content metrics. In Figure 3.2 all three plots are zoomed in on a particular y-value range and exclude some data points with RRBS=1.

As seen in Figure 3.2 only the SimMax value based on gIC shows a general increase across the entire range of sequence similarity. For both sIC and rIC, the semantic similarity values peak at an approximate RRBS value of 0.60. In the case of rIC the value substantially decreases after this point. This is caused by shared Molecular Function terms occurring at an increasingly high frequency in the annotation corpus for many of the protein pairs with RRBS on the interval (0.60, 1.0]. As seen in Figure 3.3 the protein pairs randomly selected have a high degree of similar proteins. This will increase the chances the likelihood that

functions will happen at a higher frequency and thus decrease the surprisal value. Although not tested explicitly here, it stands to reason that pairs of sequences with similarity values above 0.60 are more likely to have their annotations electronically transferred than those below. This can serve as a confounding factor by spreading informative ontology terms to multiple proteins and has been shown to influence semantic similarity results [50].
The SimMax values based on sIC tend to level off above an RRBS value of 0.60. In this case the number of descendants does not provide sufficient resolution to distinguish the shared functions as increasingly specific.
To further characterize the results I also compute the range of semantic similarity values and the R^2 for simple linear regression (Table 3.1). The range is calculated as the difference between the maximum and minimum averaged SimMax values taken at the maximum and minimum RRBS values respectively. Some of these values are excluded from Figure 3.2 due to the zoomed nature of the graph. To account for oversampling I excluded all data points with $RRBS = 1$ from linear regression. The results based on sIC show a slightly better R^2 than those based on gIC. However, the range of sIC values is limited which may inhibit resolution of semantic similarity beyond the scale captured by our sequence similarity values.

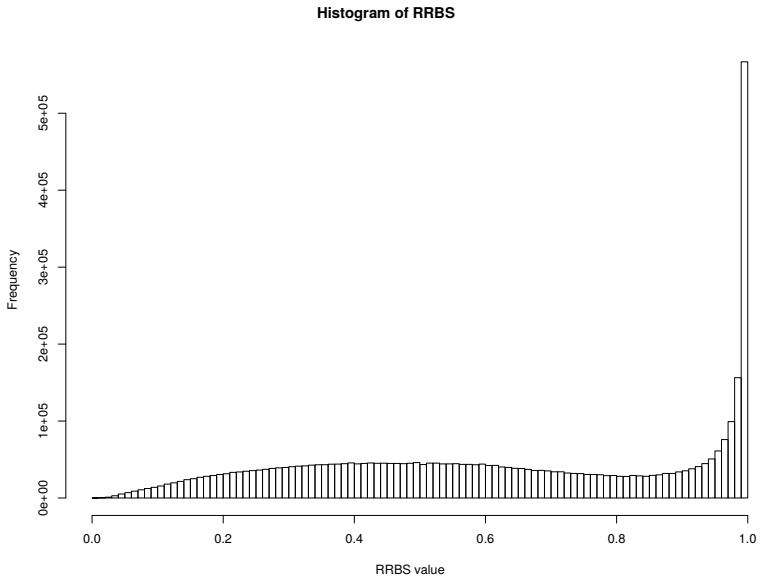

Figure 3.3: Distribution of RRBS values among randomly selected protein pairs

Table 3.1: Benchmark results of SimMax for different IC input

Input	Range	Min	Max	R^2
gIC	0.668	0.183	0.851	0.542
rIC	0.460	0.340	0.800	0.044
sIC	0.199	0.801	1.0	0.666

3.7 Discussion

Determining information content of a term from an annotation corpus conditions on the bias found in both manual and automatic annotations. The variability in annotation quality, method of assignment, and rate of turnover can influence the content of that corpus. I present a method for determining information content independent of annotation trends. This makes our metric more suitable for comparing results across corpora and potentially more reliable in data mining applications because it avoids circularity between the semantic similarity calculation and the annotation corpus being analyzed. In comparison to the corpus-based metric, our information content metric has a higher correlation with sequence similarity and a broader range of values for distinguishing between protein pairs when using Resnik's most informative common ancestor method. This indicates a better representation of functional similarity and shows its potential for enhancing existing analysis methods based on semantic similarity.

Chapter 4

Conclusion

Though synteny-block analysis has previously leveraged de Bruijn graphs for their determination this work promotes the shift to a graphical model for the final representation of synteny relationships. Further we believe that such a shift combined with the abstract use of feature families not only adds utility to pre-existing family databases, e.g. COGs and others, but can serve as a framework to speed up normally expensive operations such as phylogenetic tree construction and SNP analysis. This abstraction not only serves to make the resulting model approachable in terms of human cognition but also makes its construction more resilient to distant and complex evolutionary relationships.

The purpose of a pan-synteny graph is to encode synteny information among genomes, using ps-nodes and ps-edges that represent co-conserved feature family assignments, in order to identify meaningful relationships between, and potentially within, those genomes. In this work, we focus on analysis of prokaryotic genomes, but it is possible to extend the methods presented here to eukaryotes. This effort touches on several different research fronts: graph representation of genomes and their alignments, synteny block analysis, whole genome sequence alignment, pan-genome analysis, multiple sequence alignment, and genome rearrangement analysis. Though this approach was originally developed from a pan-genome perspective for prokaryotes the methods involved have applicability to a wide range of topics. Elements of this work can be found in all topics described above but we believe this approach represents a unique combination. Novel elements include the contextualization of synteny analysis both between and within multi-contig genomes. We also believe the algorithmic approach for discovering collision points has great value in the recognition of evolutionary relationships between a group of genomes.

To make pan-synteny graphs accessible we have a well-defined procedure for their creation that can be performed quickly. We have created pan-synteny graphs using 400 genomes in under a minute. Enabling the creation of pan-genome graph's in a short amount of time enables researchers to explore many interesting biological questions using comparative genomics that would not normally be possible. Typically to perform a comparative genomics

analysis a single reference genome must be chosen. This limits the amount of information that can be conveyed concerning the evolution, similarities, and differences of a group of organisms. At a glance our pan-synteny graph visualization allows a researcher to see: the core conserved region amongst a group of genomes, regions of similarity between replicons, potential genomic islands or large deletions that make a genome or set of genomes unique, potential inversions, and potential assembly errors. While this may seem to be a large number of structural components to include in a single visualization, these factors are often confounding in the assembly and annotation of bacterial genomes and visualizing them all at once gives new insight into the composite nature of bacterial groups.

As part of the prototype for creating, transmitting, and viewing pan-synteny graphs we have grounded development in several well-established communities for scientific calculation and communication. The source code for creating pan-synteny graphs is written in Python and leverages NetworkX [62], a package for the creation, manipulation, and study of complex graphs. Currently, the graph itself is written in Graph Exchange XML Format (GEXF), which can be read by various stand-alone and web-based graph manipulation software including NetworkX, Gephi [6], and Cytoscape [66]. The graph is laid out using a protocol that incorporates two different algorithms Yifan Hus Multilevel [21] and Force Atlas 2 [28] implemented in Gephi. An topic of future work will be considering how best to encode metadata associated with the features, families, and genomes represented by pan-synteny graph without disrupting the standards for graph exchange. It is likely that a companion format will be necessary so that this metadata can be dynamically transmitted independent of the core graph structure. As the number of genomes included scales up, this will enable the core graph structure to be encoded in a form to be suitable for web transmission and allow initial rendering with dynamic detail much like Google Maps.

The accumulation of bacterial sequences is transforming the provisioning of large-scale bioinformatics resources into a big data challenge. Many traditional strategies for applying bioinformatics methods were not designed to operate at the petabyte scale. The pan-genome graph concept we introduce represents a new data type for collapsing many thousands of genome sequences into a data object, which represents their similarities and differences, and can be used to minimize the compute cycles necessary for traditionally expensive calculations. The benefit in using pan-synteny graphs as an organizational cornerstone for tackling large-scale computation, on bacterial genome sequences, is that the graph size scales in proportion to the number of new feature families and in response to variations in genomic content, not the amount of genomic sequence it represents. As the number of bacterial genome sequences reaches into the millions it is not inconceivable that the number of genomes will exceed the number of feature families; making the pan-synteny graph an even stronger organizing concept.

The level of detail that a pan-synteny graph captures can be controlled by selecting the type of feature families included in its construction, the minimum agreement necessary to create a pan-synteny segment, and the diversity of organisms used as input. Its ability to record similarities and differences in a group context allow it to serve as a shortcut in traditionally

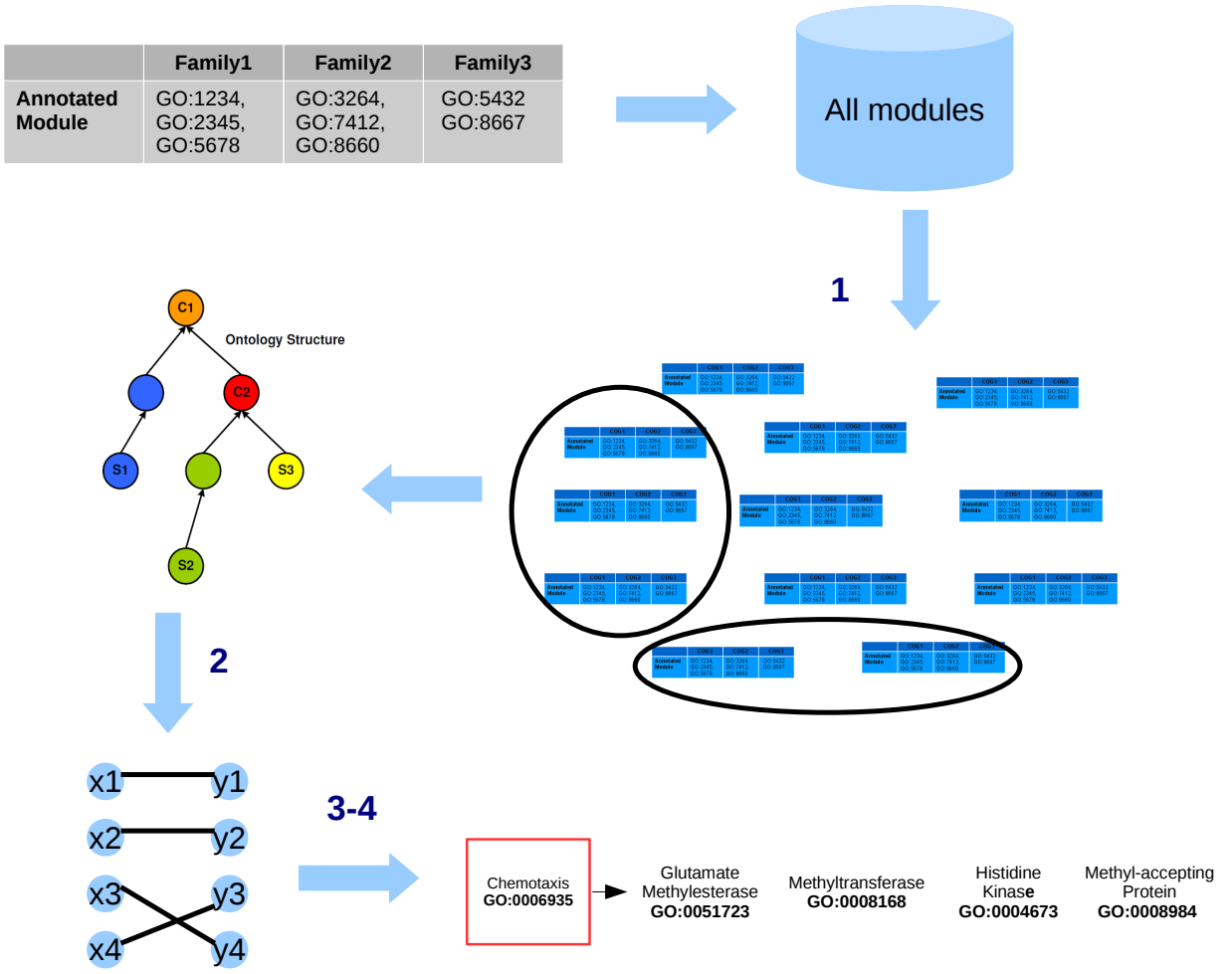

Figure 4.1: Procedure for establishing functional motifs. Step 1 uses the organization provided by pan-synteny graphs to relate gene modules. Step 2 matches the graph to an annotation corpus. Steps 3 and 4 relate modules and establish conservation of functional motifs.

high-cost computations. For example, SNP calculations involving bacterial sequences traditionally require pre-designation of target areas to be used for comparing nucleotide sequence. By using a pan-synteny graph these regions can be determined dynamically and do not need to be completely conserved in all genomes being typed but can be chosen based on their ability to discriminate in areas where uncertainty remains. Similarly by determining unique and conserved regions in the context of a genome group a phylogenetic tree calculation can be simplified by using the pan-synteny graph to identify regions, which provide distinct evolutionary signal. Phylogenetic tree calculations enabled by a pan-synteny graph have the potential to skip expensive context finding alignments since the coordinates for equivalent regions are already embedded in the graph.

A functional motif can be viewed as a description of two or more functional modules. A functional module is a grouping of genes that fulfill some common purpose in an organism. While there are examples of creating gene modules in large databases *e.g. KEGG pathways [32], SEED [47], MetaCyc [35]*, and existing methods for finding co-occurring genes [48, 78, 11], there is no method that looks for motifs on an ontological level. From these prior works it is well established that these functional module themes exist. But there is no formal computational system for finding and characterizing them. The closest concept to the result we will create is probably the subsystems of the SEED project. To describe subsystems Overbeek et al. [47] use an analogy of a spreadsheet. In this analogy the spreadsheet represents a subsystem, each column represents a functional role that executes a task in that subsystem, and each row is a genome. Genes from a genome that carry out a functional role populate the cells. In the SEED model genes that fulfill the same role (in the same column) in different genomes (different rows) need not be homologous. A functional motif is a description of gene properties that commonly contribute to a particular process, where as a subsystem is a predefined set of roles that are filled in by different genes. While a subsystem may contain a functional motif, as we have defined it, the pattern it represents is not derived from data but is specified manually based on systems of interest whose properties come from existing biological knowledge.

We propose to create several new methods, metrics, and data abstractions motivated by finding and characterizing functional motifs across multiple prokaryotic genomes. We propose a new information content metric for ontology terms based on Shannon information theory. This metric is arguably more resilient to the open world problem associated with the active development of an annotation corpus and responsive to changes in ontology structure. In accounting for this we found that our metric more closely approximates functional similarity than the commonly used suprisal-based information content metric when applied with the commonly used SimMax semantic similarity method. This work addresses both the modeling and analysis of gene modules through a graph based alignment method and the approximation of functional similarity through semantic similarity using ontology annotations. It suggests an initial framework for the identification of functional motifs in the annotations of these modules. Figure 4.1 shows a process whereby gene modules can be related to each other through their semantic similarity, and patterns can be established through strong conservation of functional patterns using genomic data mining. In this framework both pan-synteny graphs, and corpus-independent semantic similarity, establish the search space and the scoring system, respectively, and form the basis of a genome mining system for establishing functional motifs.

Bibliography

[1] M. A. Alekseyev and P. A. Pevzner, *Breakpoint graphs and ancestral genome reconstructions*, Genome Research, 19 (2009), pp. 943–957.

[2] S. V. Angiuoli and S. L. Salzberg, *Mugsy: fast multiple alignment of closely related whole genomes*, Bioinformatics, 27 (2011), pp. 334–342.

[3] J. A. Arjona-Medina and O. Trelles, *Computational Synteny Block: A Framework to Identify Evolutionary Events*, Ieee Transactions on Nanobioscience, 15 (2016), pp. 343–353. WOS:000381445900006.

[4] A. Arkin, *Setting the standard in synthetic biology*, Nat Biotech, 26 (2008), pp. 771–774.

[5] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock, *Gene ontology: tool for the unification of biology. the gene ontology consortium*, Nature Genetics, 25 (2000), pp. 25–29. PMID: 10802651.

[6] M. Bastian, S. Heymann, and M. Jacomy, *Gephi: An open source software for exploring and manipulating networks*, (2009).

[7] M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. A. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller, *Aligning multiple genomic sequences with the threaded blockset aligner*, Genome Research, 14 (2004), pp. 708–715.

[8] T. A. Brown, *Genomes 3*, Garland Science, 2007.

[9] M. Campillos, C. von Mering, L. J. Jensen, and P. Bork, *Identification and analysis of evolutionarily cohesive functional modules in protein networks*, Genome Research, 16 (2006), pp. 374–382.

[10] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson, *Multiple sequence alignment with the Clustal series of programs*, Nucleic Acids Research, 31 (2003), pp. 3497–3500.

[11] S. Cokus, S. Mizutani, and M. Pellegrini, *An improved method for identifying functionally linked proteins using phylogenetic profiles*, BMC Bioinformatics, 8 (2007), p. S7.

[12] P. E. C. Compeau, P. A. Pevzner, and G. Tesler, *How to apply de bruijn graphs to genome assembly*, Nature Biotechnology, 29 (2011), pp. 987–991.

[13] F. M. Couto, M. J. Silva, and P. M. Coutinho, *Semantic similarity over the gene ontology: family correlation and selecting disjunctive ancestors*, in Proceedings of the 14th ACM international conference on Information and knowledge management, Bremen, Germany, 2005, ACM, pp. 343–344.

[14] S. A. Dahouk, S. Khler, A. Occhialini, M. P. J. d. Bags, J. A. Hammerl, T. Eisenberg, G. Vergnaud, A. Cloeckaert, M. S. Zygmunt, A. M. Whatmore, F. Melzer, K. P. Drees, J. T. Foster, A. R. Wattam, and H. C. Scholz, *Brucella spp. of amphibians comprise genomically diverse motile strains competent for replication in macrophages and survival in mammalian hosts*, Scientific Reports, 7 (2017), p. 44420.

[15] A. C. E. Darling, B. Mau, F. R. Blattner, and N. T. Perna, *Mauve: Multiple alignment of conserved genomic sequence with rearrangements*, Genome Research, 14 (2004), pp. 1394–1403.

[16] J. J. Davis, S. Gerdes, G. J. Olsen, R. Olson, G. D. Pusch, M. Shukla, V. Vonstein, A. R. Wattam, and H. Yoo, *PATtyFams: Protein Families for the Microbial Genomes in the PATRIC Database*, Frontiers in Microbiology, 7 (2016).

[17] M. Deloger, M. El Karoui, and M.-A. Petit, *A genomic distance based on MUM indicates discontinuity between most bacterial species and genera*, J Bacteriol, 191 (2009), pp. 91–99.

[18] D. Earl, N. Nguyen, G. Hickey, R. S. Harris, S. Fitzgerald, K. Beal, I. Seledtsov, V. Molodtsov, B. J. Raney, H. Clawson, J. Kim, C. Kemena, J.-M. Chang, I. Erb, A. Poliakov, M. Hou, J. Herrero, W. J. Kent, V. Solovyev, A. E. Darling, J. Ma, C. Notredame, M. Brudno, I. Dubchak, D. Haussler, and B. Paten, *Alignathon: a competitive assessment of whole-genome alignment methods*, Genome Research, 24 (2014), pp. 2077–2089.

[19] R. C. Edgar, *MUSCLE: multiple sequence alignment with high accuracy and high throughput*, Nucleic Acids Research, 32 (2004), pp. 1792–1797.

[20] S. L. Frank, *Uncertainty reduction as a measure of cognitive processing effort*, ACL 2010, (2010), p. 81.

[21] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger, *Multilevel agglomerative edge bundling for visualizing large graphs*, IEEE, 2011, pp. 187–194.

[22] C. G. Ghiurcuta and B. M. E. Moret, *Evaluating synteny for improved comparative studies*, Bioinformatics, 30 (2014), pp. i9–i18.

[23] E. Gilles, *Signal transduction and regulation in bacteria*, Computers & Chemical Engineering, 30 (2006), pp. 1687–1699.

[24] F. Godfroid, A. Cloeckaert, B. Taminiau, I. Danese, A. Tibor, X. de Bolle, P. Mertens, and J. J. Letesson, *Genetic organisation of the lipopolysaccharide O-antigen biosynthesis region of brucella melitensis 16m (wbk)*, Research in Microbiology, 151 (2000), pp. 655–668.

[25] D. Gonzlez, M.-J. Grill, M.-J. D. Miguel, T. Ali, V. Arce-Gorvel, R.-M. Delrue, R. Conde-lvarez, P. Muoz, I. Lpez-Goi, M. Iriarte, C.-M. Marn, A. Weintraub, G. Widmalm, M. Zygmunt, J.-J. Letesson, J.-P. Gorvel, J.-M. Blasco, and I. Moriyn, *Brucellosis Vaccines: Assessment of Brucella melitensis Lipopolysaccharide Rough Mutants Defective in Core and O-Polysaccharide Synthesis and Export*, PLOS ONE, 3 (2008), p. e2760.

[26] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1 ed., May 1997.

[27] J. Hale, *Uncertainty about the rest of the sentence*, Cognitive Science, 30 (2006), pp. 643–672.

[28] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, *ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software*, PloS one, 9 (2014), p. e98679.

[29] L. J. Jensen and P. Bork, *Ontologies in quantitative biology: A basis for comparison, integration, and discovery*, PLoS Biol, 8 (2010), p. e1000374.

[30] L. J. Jensen, M. Kuhn, M. Stark, S. Chaffron, C. Creevey, J. Muller, T. Doerks, P. Julien, A. Roth, M. Simonovic, P. Bork, and C. von Mering, *STRING 8–a global view on proteins and their functional interactions in 630 organisms*, Nucl. Acids Res., 37 (2009), pp. D412–416.

[31] J. J. Jiang and D. W. Conrath, *Semantic similarity based on corpus statistics and lexical taxonomy*, Sept. 1997, p. 9008.

[32] M. Kanehisa, S. Goto, M. Hattori, K. F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa, *From genomics to chemical genomics: new developments in KEGG*, Nucl. Acids Res., 34 (2006), pp. D354–357.

[33] W. J. Kent, R. Baertsch, A. Hinrichs, W. Miller, and D. Haussler, *Evolution's cauldron: Duplication, deletion, and rearrangement in the mouse and human genomes*, Proceedings of the National Academy of Sciences, 100 (2003), pp. 11484–11489.

[34] A. Kremling, K. Jahreis, J. W. Lengeler, and E. D. Gilles, *The organization of metabolic reaction networks: A Signal-Oriented approach to cellular models*, Metabolic Engineering, 2 (2000), pp. 190–200.

[35] C. J. Krieger, P. Zhang, L. A. Mueller, A. Wang, S. Paley, M. Arnaud, J. Pick, S. Y. Rhee, and P. D. Karp, *MetaCyc: a multiorganism database of metabolic pathways and enzymes*, Nucl. Acids Res., 32 (2004), pp. D438–442.

[36] C. Lee, C. Grasso, and M. F. Sharlow, *Multiple sequence alignment using partial order graphs*, Bioinformatics, 18 (2002), pp. 452–464.

[37] J. W. Lengeler, G. Drews, and H. G. Schlegel, *Biology of the prokaryotes*, Georg Thieme Verlag, 1999.

[38] M.-Y. Leung, B. E. Blaisdell, C. Burge, and S. Karlin, *An efficient algorithm for identifying matches with errors in multiple long molecular sequences*, Journal of molecular biology, 221 (1991), pp. 1367–1378.

[39] D. Lin, *An Information-Theoretic definition of similarity*, in Machine learning: proceedings of the fifteenth international conference (ICML'98), 1998, p. 296.

[40] Y. Lin, S. Nurk, and P. A. Pevzner, *What is the difference between the breakpoint graph and the de Bruijn graph?*, BMC Genomics, 15 (2014), p. S6.

[41] K. J. Locey and J. T. Lennon, *Scaling laws predict global microbial diversity*, Proceedings of the National Academy of Sciences, 113 (2016), pp. 5970–5975.

[42] P. W. Lord, R. D. Stevens, A. Brass, and C. A. Goble, *Investigating semantic similarity measures across the gene ontology: the relationship between sequence and annotation*, Bioinformatics, 19 (2003), pp. 1275–1283.

[43] S. Marcus, H. Lee, and M. C. Schatz, *SplitMEM: a graphical algorithm for pangenome analysis with suffix skips*, Bioinformatics, 30 (2014), pp. 3476–3483.

[44] F. Meyer, R. Overbeek, and A. Rodriguez, *FIGfams: yet another set of protein families*, Nucleic Acids Research, 37 (2009), pp. 6643–54.

[45] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, *Introduction to WordNet: an on-line lexical database**, Int J Lexicography, 3 (1990), pp. 235–244.

[46] 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*, arXiv:1307.7941 [q-bio], (2013). arXiv: 1307.7941.

[47] R. Overbeek, T. Begley, R. M. Butler, J. V. Choudhuri, H. Chuang, M. Cohoon, V. de Crecy-Lagard, N. Diaz, T. Disz, R. Edwards, M. Fonstein, E. D. Frank, S. Gerdes, E. M. Glass, A. Goesmann, A. Hanson, D. Iwata-Reuyl, R. Jensen, N. Jamshidi, L. Krause, M. Kubal, N. Larsen, B. Linke, A. C. McHardy, F. Meyer, H. Neuweger, G. Olsen, R. Olson, A. Osterman, V. Portnoy, G. D. Pusch, D. A. Rodionov, C. Ruckert, J. Steiner, R. Stevens, I. Thiele, O. Vassieva, Y. Ye, O. Zagnitko, and V. Vonstein, *The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes*, Nucl. Acids Res., 33 (2005), pp. 5691–5702.

[48] M. Pellegrini, E. M. Marcotte, M. J. Thompson, D. Eisenberg, and T. O. Yeates, *Assigning protein functions by comparative genome analysis: Protein phylogenetic profiles*, Proceedings of the National Academy of Sciences of the United States of America, 96 (1999), pp. 4285–4288.

[49] Q. Peng, M. A. Alekseyev, G. Tesler, and P. A. Pevzner, *Decoding Synteny Blocks and Large-Scale Duplications in Mammalian and Plant Genomes*, in Algorithms in Bioinformatics, Springer, Berlin, Heidelberg, Sept. 2009, pp. 220–232.

[50] C. Pesquita, D. Faria, H. Bastos, A. Ferreira, A. Falcao, and F. Couto, *Metrics for GO based protein semantic similarity: a systematic evaluation*, BMC Bioinformatics, 9 (2008), p. S4.

[51] C. Pesquita, D. Faria, A. O. Falco, P. Lord, and F. M. Couto, *Semantic similarity in biomedical ontologies*, 5 (2009). PMID: 19649320 PMCID: 2712090.

[52] P. Pevzner and G. Tesler, *Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes*, Genome Research, 13 (2003), pp. 37–45.

[53] P. A. Pevzner, H. Tang, and G. Tesler, *De Novo Repeat Classification and Fragment Assembly*, Genome Research, 14 (2004), pp. 1786–1796.

[54] P. A. Pevzner, H. Tang, and M. S. Waterman, *An eulerian path approach to DNA fragment assembly*, Proceedings of the National Academy of Sciences of the United States of America, 98 (2001), pp. 9748–9753.

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

[56] J. R. Pierce, *An introduction to information theory: symbols, signals & noise*, Courier Dover Publications, 1980.

[57] M. Popescu, J. M. Keller, and J. A. Mitchell, *Fuzzy measures on the gene ontology for gene product similarity*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 3 (2006), pp. 263–274.

[58] B. Raphael, D. Zhi, H. Tang, and P. Pevzner, *A novel method for multiple alignment of sequences with repeated and shuffled elements*, Genome Research, 14 (2004), pp. 2336–2346.

[59] P. Resnik, *Using information content to evaluate semantic similarity in a taxonomy*, Proceedings of the 14th International Joint Conference on Artificial Intelligence, (1995). Proceedings of the 14th International Joint Conference on Artificial Intelligence http://arxiv.org/abs/cmp-lg/9511007.

[60] P. Resnik, *Semantic similarity in a taxonomy: An Information-Based measure and its application to problems of ambiguity in natural language*, Journal of Artificial IntelligenceResearch, 11 (1999), p. 95130.

[61] M.-F. Sagot, A. Viari, and H. Soldano, *Multiple sequence comparison a peptide matching approach*, Theoretical Computer Science, 180 (1997), pp. 115–137.

[62] D. A. Schult and P. Swart, *Exploring network structure, dynamics, and function using NetworkX*, in Proceedings of the 7th Python in Science Conferences (SciPy 2008), vol. 2008, 2008, pp. 11–16.

[63] N. Seco, T. Veale, and J. Hayes, *An intrinsic information content metric for semantic similarity in WordNet*, in ECAI, vol. 16, 2004, p. 1089.

[64] R. Sharan, I. Ulitsky, and R. Shamir, *Network-based prediction of protein function*, Mol Syst Biol, 3 (2007).

[65] B. Sheehan, A. Quigley, B. Gaudin, and S. Dobson, *A relation based measure of semantic similarity for gene ontology annotations*, BMC Bioinformatics, 9 (2008), p. 468.

[66] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker, *Cytoscape 2.8: new features for data integration and network visualization*, Bioinformatics, 27 (2011), pp. 431–432.

[67] P. F. Soler-Llorns, C. R. Quance, S. D. Lawhon, T. P. Stuber, J. F. Edwards, T. A. Ficht, S. Robbe-Austerman, D. O'Callaghan, and A. Keriel, *A Brucella spp. Isolate from a Pac-Man Frog (Ceratophrys ornata) Reveals Characteristics Departing from Classical Brucellae*, Frontiers in Cellular and Infection Microbiology, 6 (2016).

[68] T. Tatusova, S. Ciufo, B. Fedorov, K. ONeill, and I. Tolstoy, *RefSeq microbial genomes database: new representation and annotation strategy*, Nucleic Acids Research, 42 (2014), pp. D553–D559.

[69] P. D. Thomas, V. Wood, C. J. Mungall, S. E. Lewis, J. A. Blake, and on behalf of the Gene Ontology Consortium, *On the use of gene ontology annotations to assess functional similarity among orthologs and paralogs: A short report*, PLoS Comput Biol, 8 (2012), p. e1002386.

[70] G. Tsoktouridis, C. A. Merz, S. P. Manning, R. Giovagnoli-Kurtz, L. E. Williams, C. V. Mujer, S. Hagius, P. Elzer, R. J. Redkar, G. Patra, and V. G. DelVecchio, *Molecular characterization of Brucella abortus chromosome II recombination*, Journal of Bacteriology, 185 (2003), pp. 6130–6136.

[71] G. Vernikos, D. Medini, D. R. Riley, and H. Tettelin, *Ten years of pan-genome analyses*, Current Opinion in Microbiology, 23 (2015), pp. 148–154.

[72] J. Z. Wang, Z. Du, R. Payattakool, P. S. Yu, and C. Chen, *A new method to measure the semantic similarity of GO terms*, Bioinformatics, 23 (2007), pp. 1274–1281.

[73] X. Wang, E. Dalkic, M. Wu, and C. Chan, *Gene-module level analysis: Identification to networks and dynamics*, Current opinion in biotechnology, 19 (2008), pp. 482–491. PMID: 18725293 PMCID: 2615490.

[74] A. Warren and J. C. Setubal, *Using entropy estimates for DAG-based ontologies*, in ISMB Bio-Ontologies 2012, Oct. 2012.

[75] A. R. Wattam, J. J. Davis, R. Assaf, S. Boisvert, T. Brettin, C. Bun, N. Conrad, E. M. Dietrich, T. Disz, J. L. Gabbard, S. Gerdes, C. S. Henry, R. W. Kenyon, D. Machi, C. Mao, E. K. Nordberg, G. J. Olsen, D. E. Murphy-Olson, R. Olson, R. Overbeek, B. Parrello, G. D. Pusch, M. Shukla, V. Vonstein, A. Warren, F. Xia, H. Yoo, and R. L. Stevens, *Improvements to PATRIC, the all-bacterial Bioinformatics Database and Analysis Resource Center*, Nucleic Acids Research, (2016), p. gkw1017.

[76] A. R. Wattam, T. J. Inzana, K. P. Williams, S. P. Mane, M. Shukla, N. F. Almeida, A. W. Dickerman, S. Mason, I. Moriyn, D. OCallaghan, A. M. Whatmore, B. W. Sobral, R. V. Tiller, A. R. Hoffmaster, M. A. Frace, C. D. Castro, A. Molinaro, S. M. Boyle, B. K. De, and J. C. Setubal, *Comparative Genomics of Early-Diverging Brucella Strains Reveals a Novel Lipopolysaccharide Biosynthesis Pathway*, mBio, 3 (2012), pp. e00246–12.

[77] A. R. Wattam, K. P. Williams, E. E. Snyder, N. F. Almeida, M. Shukla, A. W. Dickerman, O. R. Crasta, R. Kenyon, J. Lu, J. M. Shallom, H. Yoo, T. A. Ficht, R. M. Tsolis, C. Munk, R. Tapia, C. S. Han, J. C. Detter,

D. Bruce, T. S. Brettin, B. W. Sobral, S. M. Boyle, and J. C. Setubal, *Analysis of ten Brucella genomes reveals evidence for horizontal gene transfer despite a preferred intracellular lifestyle*, Journal of Bacteriology, 191 (2009), pp. 3569–3579.

[78] H. Wu, Z. Su, F. Mao, V. Olman, and Y. Xu, *Prediction of functional modules based on comparative genome analysis and gene ontology application*, Nucl. Acids Res., 33 (2005), pp. 2822–2837.

[79] Y. Xu, *Computational challenges in deciphering genomic structures of bacteria*, Journal of Computer Science and Technology, 25 (2010), pp. 53–70.

[80] C. Yang, D. A. Rodionov, X. Li, O. N. Laikova, M. S. Gelfand, O. P. Zagnitko, M. F. Romine, A. Y. Obraztsova, K. H. Nealson, and A. L. Osterman, *Comparative genomics and experimental characterization of N-Acetylglucosamine utilization pathway of shewanella oneidensis*, Journal of Biological Chemistry, 281 (2006), pp. 29872–29885.

[81] D. R. Zerbino and E. Birney, *Velvet: Algorithms for de novo short read assembly using de bruijn graphs*, Genome Research, 18 (2008), pp. 821–829.