R-FAP: Rapid Functional Annotation of Prokaryotes Using Taxon-specific Pan-genomes and 10-mer Peptides

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Jordan Matthew Utley
May 2014

Copyright © 2014 by Jordan M. Utley
All rights reserved.

Acknowledgements and Dedication

There are a great many people to whom I owe my appreciation for being able to complete this thesis. First, my advisor, Dr. Loren Hauser, has been an incomparable source of insight regarding both this thesis as well as the field of biology as a whole.

Secondly, Dr. Harry Richards, who originally recruited me to the University of Tennessee, has been an unwavering source of guidance and encouragement during my time here.

Thirdly, my committee members, Drs. Brian O'Meara, Elizabeth Fozo, and Chongle Pan have been indispensable with their guidance, support, and patience throughout this process.

Additionally, Miriam Land, J.J. Chai, Dylan Storey, and many other colleagues in the Scalable Computing a Leading-Edge Innovative Technologies (SCALE-IT) fellowship, the Genome Science and Technology (GST) program, and Oak Ridge National Laboratory have all been wonderful sources of both practical as well as conceptual advice.

Denise Koessler—as a role model, a colleague, and a close personal friend—has helped shape my understanding and love of interdisciplinary collaboration through partnerships on multiple projects during my time at the University of Tennessee.

My friends Patrick Taylor, Lane Carasik, Sara Konsavage, Jonathan Hopper, Jacob Hassman, Coleman Garner, and numerous others have sustained me on this path in countless intangible but nonetheless invaluable ways.

And finally, my loving partner, Noah Mayhew, has been my rock through my time in graduate school, sharing in the joys and supporting me through the frustrations.

This thesis is gratefully dedicated to all of these individuals who have had a hand in my success.

Abstract

The growing implementation of next-generation sequencing technologies presents numerous fields with the opportunity to identify bacteria in near real-time. Fields such as counter-terrorism, forensics, medicine, and even microbial ecology are positioned to benefit from such advances and implementation. However, with the ability to rapidly produce high-quality sequence data comes the need to interpret this data as quickly as it is produced. While gene prediction algorithms have kept pace, functional prediction methods have not.

To bypass the need for large-scale queries to multiple databases for each newly-sequenced genome, the project detailed herein seeks to identify the genes shared within a taxonomic group using the *pan-genome* for that group. Doing so allows the pan-genome to be queried against this set of databases a single time, then rapidly searched with new genomes using k-mer peptide matching to make functional predictions.

Thirty-one strains from *Salmonella enterica* subsp. *enterica* were used to build the pan-genome for this taxon as a test model. Proteins in a new genome could then be matched with complete consistence to the resulting database in a matter of seconds (per genome) using a k-mer peptide search algorithm. This represents a major advancement in annotation speed over existing pipelines.

Table of Contents

Chapter 1 Introduction and General Information 1
Background: Addressing New Threats 1
Challenges 2
Solutions 2
Chapter 2 Literature Review 4
Genome Annotation: Genes and Functions 5
Gene prediction. 5
Function prediction. 6
Pan-genomes 7
Average Nucleotide Identity (ANI) and phylogenetic breadth. 7
Protein families: orthologs vs. paralogs. 9
Salmonella enterica: A Test-bed 10
Classification and Identification via *k*-mer Matching 10
Chapter 3 Materials and Methods 13
Building the Pan-genome of *Salmonella enterica* subsp. *enterica* 13
Determining phylogenetic breadth of representative strains. 13
Determining protein families using OrthoMCL. 14
Building consensus sequences with Muscle and most-common residues. 15
Eliminating redundant sequences and annotating the database. 16
Searching the Pan-genome with Query Protein Sequences Using K-mer Peptides 17
Identifying protein-coding, tRNA, and rRNA genes in novel query genomes. 17
Using Simrank to match query proteins to database sequences. 17
Identifying and annotating novel proteins in query genomes. 18
Testing R-FAP against existing pipelines: speed and accuracy 18
Chapter 4 Results, Discussion, and Recommendations 20
The Pan-genome Assembly Process 20
R-FAP as a Tool for Function Calling and Annotation 21
Discussion and Recommendations 22
Thresholds and accuracy. 22
Implications. 23
Implementation. 24
List of References 26
Appendices 35
Appendix A: Tables 36
Appendix B: Figures 38
Appendix C: The R-FAP Pan-genome Assembly Source Code 43
Appendix D: The R-FAP Annotation Tool Source Code 50
Vita 54

List of Tables

Table 1. Comparison of annotation pipelines. 36
Table 2. Strains included in the pan-genome database. 37

List of Figures

Figure 1. Building a hypothetical pan-genome in concept. 38
Figure 2. Minimizing the false positive count and the false negative rate. 39
Figure 3. Outline of the pan-genome assembly workflow..... 40
Figure 4. The phylogenetic tree of the 31 S. enterica strains..... 41
Figure 5. Functional prediction of proteins using R-FAP. 42

Chapter 1

Introduction and General Information

Background: Addressing New Threats

The threats of epidemics and pathogen-based bioweapon terrorist attacks have been brought to the forefront of the public's attention several times over the last decade. Outbreaks of Severe Acute Respiratory Syndrome (SARS) (Lee, et al., 2003; Marra, et al., 2003) and various strains of influenza (The WHO Consultation on Human Influenza A/H5, 2005; Smith, et al., 2009), reports of increasing antibiotic resistance in various bacterial pathogens (Goldstein, et al., 2012; Ohnishi, et al., 2011), and the mail-borne anthrax attacks following the events of 9/11 have all demonstrated our vulnerability to such threats and the need for a rapid response involving the emerging field of microbial forensics (Budowle, Murch, & Chakrabort, 2005). The first step in such a response is the identification of the responsible pathogen, and by necessity this step must be rapid.

To this end, the advent of ever-faster sequencing technologies now allows a novel prokaryotic genome to be fully sequenced at 100x coverage and assembled in a matter of a few hours (Loman, et al., 2012). Furthermore, gene calling algorithms now take mere minutes per genome (Hyatt, et al., 2010), thus keeping pace with sequencing needs and technology. This technology is currently being put to the test at the University of California-Davis, where Dr. Bart Weimer and colleagues are funded by the US Food and Drug Administration to sequence the genomes of 100,000 foodborne pathogenic strains over the course of four years.

Challenges

However, while sequencing and gene-calling technologies are now fast enough to meet this challenge, current pipelines for functional annotation are not, as they rely on search to multiple database such as Pfam (Finn, et al., 2014), InterPro (Hunter, et al., 2012), and the NCBI's non-redundant (NR) protein database Even worse, some are hosted on remote servers and run in batches, which can delay results by weeks. Indeed, the Rapid Annotations using Subsystems Technology (RAST) system, as well as the internal ORNL pipeline, can annotate a new genome in 12-24 hours (Aziz, et al., 2008), whereas the DOE/JGI annotation pipeline relies on submissions to the Integrated Microbial Genomes (IMG) system (Markowitz, et al., 2006; Markowitz, et al., 2010), which are only processed every 2-3 weeks (Mavromatis, et al., 2009). Furthermore, none of these systems is designed to handle the dataflow from a project, which at full capacity will need annotations for veritable hundreds of genomes every week. On the other hand, since such a project will focus on strains of known species, existing annotations from such pipelines could be used in annotating new strains due to the small phylogenetic distance between them.

Solutions

The focus of this thesis, then, is a solution to this problem. The approach is based on the concept of pan-genomes, the collection of all genes/proteins at a given phyla level (species, subspecies, etc.). Such a collection must be built from a given set of representative genomes, accounting for orthologs and paralogs, as well as proteins unique to a single strain. Once such a set of proteins is amassed, they can be submitted for

annotation to an existing pipeline, a process which would need occur only once. Subsequently, this database of proteins can be rapidly searched with a newly-sequenced genome using the recently-recognized (Aziz, et al., 2008) method of k-mer peptide matching, which avoids the need for time-consuming alignments, instead counting the number of identical peptides of length *k* between database sequences and query sequences (Berendzen, et al., 2012). All that is needed after this is to apply annotation information to the query proteins from their respective matches in the database.

Chapter 2

Literature Review

In annotating any newly-sequenced prokaryotic genome, there are two primary steps which must be completed. First, the coding sequences for proteins and the various RNA molecules must be identified. Although prokaryotic genomes range from 140 kilobases (kb) to 17 megabases (Mb), most fall in the 3-5 Mb range (Land, Details of a survey of sequenced bacterial genomes, 2014) and as such, identification must necessarily be accomplished *in silico*. This has been the focus of a great deal of research and several algorithms have been produced to address this problem. The second step of genome annotation is the process of determining what the newly-identified sequences actually *do*. While methods exist for identifying prominent RNA-encoding genes, such as tRNAs and rRNAs, in relatively short order, protein-coding genes are a different matter. To date, the process by which this process has been accomplished is to search the putative products of such genes against a series of curated databases, looking for similarities between these products and previously-characterized sequences. However, such searches frequently rely on time- and computationally-expensive processes such as sequence alignments and hidden Markov model searches in batch runs with other submissions. A prospective solution to this problem has been found in the area of k-mer peptide searches, which run rapidly and would allow for a single database (the *pan-genome*) to be given a consensus annotation from numerous sources a single time, then searched with a new set of sequences any number of times.

Genome Annotation: Genes and Functions

Gene prediction. The advent of complete genome sequencing, and particularly sequencing *en masse* with the advent of high-throughput technologies, has necessitated the development of tools to identify the coding regions of a novel genome. Early methods for this process coalesced as early as 1986 with the work of Mark Borodovsky in Markov chain models, culminating in his group's Genmark software (Borodovsky & McIninch, 1993), having since been updated to Genemark.hmm, which uses hidden Markov models (Lukashin & Borodovsky, 1998). Drawing on this work but instead using *interpolated* Markov models, the most commonly-used software for prokaryotic gene prediction, GLIMMER, was subsequently developed by the Salzberg research group (Salzberg, Delcher, Kasif, & White, 1998; Delcher, Bratke, Powers, & Salzberg, 2007).

However, while both Genmark and especially GLIMMER remain widely-used, their accuracies are sensitive to the %GC content the input genome, particularly above 60% (McHardy, Goesmann, Pühler, & Meyer, 2004). As %GC content increases, the thymines and adenines needed for standard stop codons (TAG, TAA, and TGA) become increasingly rare, resulting in a larger number of long, non-coding open reading frames (ORFs) which GLIMMER and Genmark/Genemark.hmm are prone to incorrectly labeling as genes. In contrast, the more recently-developed PRODIGAL—which self trains on each genome, and uses a dynamic programming algorithm to call genes based primarily on hexamer (di-codon) frequency and ribosome binding site (RBS) strength and location relative to the start codon—is significantly less vulnerable to extremes in %GC content and is also reliably more accurate with regards to start calls (Hyatt, et al., 2010).

Because of these advantages, combined with the great speed with which PRODIGAL runs (generally 1-2 minutes per genome), this was used to identify genes for this project. (This was also done for the sake of consistency, since these strains were previously annotated using many different methods; as such, some genes and proteins used in this work differ from those found for these strains GenBank.)

Function prediction. The process of identifying the function of protein-coding genes depends on the principle that homology (similarity of sequence due to a common ancestry) implies similarity of function. While there are certain acknowledged issues with this principle (Galperin & Koonin, 2012), it does largely hold up within the well-studied phyla of prokaryotes, where there has been a great deal of experimental verification (Koonin & Galperin, 2003). The application of this principle has primarily taken the form of searchable repositories of functional information gained from experimental data. Some of these are general-purpose databases, such as KEGG (Ogata, Goto, Fujibuchi, & Kanehisa, 1998), COG (Tatusov, Koonin, & Lipman, 1997; Kristensen, et al., 2010), the multi-database system of InterPro (Apweiler, et al., 2001; Hunter, et al., 2012) and the NCBI's GenBank and protein domain database combined with the BLAST algorithm (Altschul, Gish, Miller, Myers, & Lipman, 1990; Benson, et al., 2014). Others, such as Pfam (Sonnhammer, Eddy, & Durbin, 1997; Finn, et al., 2014) act as domain-identification tools, or serve to identify specific types of proteins, such as enzymes in the case of PRIAM (Claudel-Renard, Chevalet, Faraut, & Kahn, 2003).

While all of these databases provide functional information for a set of query gene products, the desire to achieve a comprehensive understanding of a given genome, as

well as to avoid bias, has led to the development of annotation *pipelines*. These pipelines take advantage of these and other databases using large-scale searches to align individual domains and whole proteins to database sequences and predict respective functions by homology. While there are numerous such pipelines, the details of three notable pipelines are outlined in Table 1 (note that all tables and figures are found in the appendices): the Rapid Annotations using Subsystems Technology (RAST) pipeline; the Department of Energy/Joint Genome Institute pipeline; and the internal pipeline used at Oak Ridge National Labority, developed from the DOE/JGI pipeline.

Pan-genomes

A pan-genome, as the name would suggest, is a "genome" which represents all genes, both shared and unique, within a given taxonomic group, generally the species (Medini, Donati, Tettelin, Masignani, & Rappuoli, 2005). Building a pan-genome thus requires that the shared genes, or protein families, be identified and included. This process can be thought of as a "layering" of genes onto a conceptual genomic 'scaffold' (see Figure 1). Furthermore, in taxa with numerous (>75) members, going through the process of building an initial pan-genome using all members is not practical due to large computational costs, and a representative subset must be chosen instead; in this instance, it is important to ensure that this subset represents the full phylogenetic breadth of the taxon in question.

Average Nucleotide Identity (ANI) and phylogenetic breadth. As mentioned previously, it is necessary to represent the phylogenetic diversity of the taxon for which the pan-genome is being constructed. Thus, in a species containing numerous strains, it

becomes necessary to select a sufficiently-diverse set of these strains such that the resulting pan-genome is adequately representative of the species.

Numerous methods have been used in the past to determine this phylogenetic diversity in prokaryotes. The advent of first-generation sequencing technology allowed comparison of conserved sequences like the 16s rRNA gene, or a conventional set of "core" proteins in the genome. More recently, however, the arrival of next-generation sequencers combined with ever-increasing computational power has allowed phylogenetics to be conducted on the genomic scale (Zhi, Zhao, Li, & Zhao, 2012). Though not without its own challenges, this approach overcomes the low resolving power of the relatively conserved 16s rRNA gene and the confounding myriad of potential issues brought on by lateral gene transfer in a small set of protein-coding genes (Sentausa & Fournier, 2013).

To take advantage of these developments in determining the phylogenetic diversity of the chosen strains, this project will establish the average nucleotide identity (ANI) of each strain relative to every other strain. There are several methods by which to accomplish this task, but the most efficient and straightforward is to produce alignments of uniform genome fragments from the different strains and calculate the ANI, which in reality is merely a percent similarity, based on the best of these alignments. The JSpecies software (Richter & Rossello-Mora, 2009) incorporates three separate algorithms for this purpose (BLAST, MUMer, and Tetra), which can be used together or separately and run in parallel. JSpecies creates a reciprocal similarity matrix from each algorithm which, with minor modifications, can be quickly imported to the APE phylogenetics package for

R (Paradis, Claude, & Strimmer, 2004). These matrices can then be used to build phylogenetic trees using several different algorithms, such as neighbor-joining (NJ), which in turn can be visualized and further manipulated in the Molecular Evolutionary Genetics Analysis (MEGA) package (Tamura, Stecher, Peterson, Filipski, & Kumar, 2013).

Protein families: orthologs vs. paralogs. Once a representative set of strains has been chosen, it is necessary to identify from amongst this set, those genes which are equivalent from one genome to another. This requires distinguishing between orthologs, which are the true inter-genome equivalents, and paralogs, which represent intra-genome duplication events in ancestor species which, relatively unconstrained by evolutionary forces, frequently differ in function (Altenhoff & Dessimoz, 2012). This in turn presents unique challenges due to the large number and close relation of the considered strains. There are two primary computational approaches to solving this problem: tree reconciliation methods such as RIO (Zmasek & Eddy, 2002) and OrthoStrapper (Storm & Sonnhammer, 2002), or graph-based methods such as COG (Tatusov, Koonin, & Lipman, 1997) and OrthoMCL (Li, Stoeckert, & Roos, 2003). Although primarily designed for analysis amoung eurkaroytes, OrthoMCL's unique Markov clustering algorithm (van Dongen, 2000) combined with its parallelized nature makes it well-suited to the this project and is thus the method by which protein families will be determined for the pan-genome.

Salmonella enterica: A Test-bed

It was necessary to choose a taxon in which to test the pan-genome building as well as function-prediction algorithms, which were the focus of this thesis. There were two overriding justifications for selecting *Salmonella enterica*, a gram-negative, rod-shaped species which annually causes millions of cases of typhoid fever (Crump & Mintz, 2010) and non-typhoid infection (Chen, Wang, Su, & Chiu, 2013) such as poultry-borne food poisoning (Cox, Cason, & Richardson, 2011); first, the deliverables of this thesis are primarily targeted to large sequencing projects such as Dr. Bart Weimer's "100,000 genomes" project, which is focused on food-borne pathogens such as *S. enterica*. Secondly, the pan-genome of *S. enterica* has already been investigated previously (Jacobsen, Hendriksen, Aaresturp, Ussery, & Friis, 2011), which will allow for validation of the pan-genome building algorithm.

Classification and Identification via *k*-mer Matching

As mentioned previously, the rate-limiting step in prokaryotic genome annotation is functional prediction of protein-coding genes. While some pipelines' reliance on batch runs of numerous genomes every few days or weeks is a contributing factor, the primary delay is caused by querying one or more databases with each gene. This is because database queries necessitate creating alignments or performing Hidden Markov Model searches between query and database sequences; while such processes may need only a 30 seconds or a minute per gene, this becomes time-consuming even for the fastest algorithms when faced with the need to identify several thousand genes per genome and, for projects such as Dr. Weimer's, hundreds or thousands of genomes at a time. Recently,

however, it has been recognized that sequence similarity can be determined more rapidly by avoiding alignments completely and instead matching peptides, or substrings, of length *k* between query and subject sequences, be they whole genomes (Larsen, et al., 2014) or individual genes and proteins. Percent similarity between two sequences is calculated by counting the number of Boolean matches and normalizing for the total length of the sequence. This has already been implemented for numerous purposes, including phylogenetics, alignment algorithms such as BLAST (Altschul, Gish, Miller, Myers, & Lipman, 1990), and metagenomics studies (Edwards, et al., 2012). Even more relevantly, the developers of the RAST pipeline have recently updated their methodology to include a k-mer matching step (Overbeek, et al., 2014).

However, unlike with the BLAST algorithm, in determining sequence relatedness by *k*-mer matching alone, *k* is not defined dynamically but is instead set by the user. Several groups have made the case for different values of *k*, but the developers of the Sequedex *k*-mer metagenomics tool have demonstrated that $k=10$ is the most sensible value (Berendzen, et al., 2012) for peptides, balancing sensitivity with exclusivity. Nevertheless, although Sequedex can process an entire genome in mere minutes, its current implementation focuses on functional and phylogenetic categorization of metagenomic data, making it poorly-suited both to detailed functional predictions in an assembled genome, as well as to integration into a larger pipeline. Therefore, the work in this thesis uses the Simrank algorithm (DeSantis, et al., 2011) for the ultimate purpose of matching query protein sequences to their corresponding sequences in the pan-genome. The Simrank algorithm is implemented as a Perl module, where it is first used to create a

binary database file of k-mer peptides of user-specified length from the initial collection of pan-genome protein sequences. Simrank then takes each query sequence and calculates matches by dividing the total number of matched peptides by the total length of the shorter of the two sequences. In all, once the binary file is in place, the runtime for Simrank on an average-sized *S. enterica* genome is measured not in minutes, but in seconds.

Chapter 3

Materials and Methods

Building the Pan-genome of *Salmonella enterica* subsp. *enterica*

Determining phylogenetic breadth of representative strains. In order to ensure that the pan-genome database sufficiently represented the phylogenetic diversity of *S. enterica*, the first step in building such a database was to build a phylogenetic tree with the prospective strains and compare it to existing phylogenies of *S. enterica*. For the reasons explained previously, the Average Nucleotide Identity (ANI) of the chosen strains relative to each other was used to create the distance matrix from which this tree was built. Using JSpecies v1.2.1, the complete genome sequence of each strain was first broken into non-overlapping fragments of 1020 nucleotides, and a reciprocal BLAST search of each fragment against every other fragment was run in parallel on 16 processors on the Asp cluster at ORNL. This process takes approximately 8-9 hours, and the resulting output is a matrix of the percent identities of each strain to every other strain.

This output forms the basis for a distance matrix. However, because this is a reciprocal search, there are two percentages for each pair of strains (that is to say, ‘Strain 1’ has a certain percent identity to ‘Strain 2’ but ‘Strain 2’ can possess a different percent identity to ‘Strain 1’); this is inherent in the BLAST alignment process, as the resulting score is dependent on which sequence is considered the subject and which is considered the query. Nevertheless, there is a direct correlation between the proportional difference in sequence length and the difference in calculated similarity, meaning that significant discrepancies between corresponding scores generally arise only in the case of a

significant proportional difference in sequence length (which results in differing query coverage). In a species such as *S. enterica*, large differences in genome length do not exist and as such, each pair of corresponding scores was considered close enough in value (±1.5%) that merely averaging the two numbers to produce a single triangular matrix was determined to be acceptable. Subsequently, this was transformed into a true distance matrix by subtracting each of these percentages from 100, thus converting each number from a percent similarity to a percent difference, or distance.

Finally, the actual tree-building step was performed with the Ape phylogenetics package for R, using the neighbor-joining algorithm and writing to file using Netwick format. The tree was subsequently visualized using MEGA, which was also used to set the out-group, *S. enterica* subsp. *arizonae*, as the root for the tree.

Determining protein families using OrthoMCL. Once a sufficiently broad phylogenetic range of strains had been chosen (for a complete list, see Table 2), the next step was to identify the protein families present in these strains. Therefore, in order to provide consistency and increased accuracy, all protein sequences were identified from the genome sequences using Prodigal and concatenated into a single list, which produced a total of 137,718 protein sequences. This file was subsequently used as the input for OrthoMCL, with the initial all-vs.-all BLAST search being run on 36 processors. In all, the OrthoMCL run took approximately 18 hours.

While OrthoMCL is both fast (considering the input size) and reliable, it is not perfect; approximately 5% of output clusters either represent multiple protein families grouped into a “super-cluster” or simply include one or more proteins which do not

belong. To correct for this, a file was created which displayed the BLAST bit scores, normalized to the sequence length, for each pair of proteins in a prospective cluster from OrthoMCL as a series of matrices. Using this file, the two scores for any two pairs of proteins were then averaged (as with the tree-building process mentioned previously), and if this average score was equal to or greater than the weighted average score of the cluster, the two proteins were considered a match. By iterating through each gene in a given prospective cluster and applying this threshold, an initial list of protein families was generated.

Building consensus sequences with Muscle and most-common residues. The next step in building a pan-genome database is to determine the consensus sequence for each protein family. To this end, the sequences for all proteins in a newly-determined cluster were placed in a file, one file per cluster. The alignment for each cluster was then created using the Muscle algorithm (Edgar, 2004), distributed using a Python wrapper script provided by JJ Chai at ORNL. This script used the standard number of iterations for each alignment and output the resulting alignments in ClustalW format. In all, this process completed in approximately 20 minutes when run on 6 CPUs.

Next, these alignments were concatenated into a single file and each alignment processed to produce a single consensus sequence for the protein family, by iterating through each position in the alignment. Whenever a given position was universally conserved, the corresponding residue was added to the end of the consensus sequence; in cases where a given position lacked universal consensus in the alignment, the most common residue at that position was determined and added to the consensus sequence; in

the case where a gap was the most common at a given non-conserved position, nothing was added to the consensus sequence. All protein family consensus sequences were ultimately printed to a FASTA-formatted file, and this process took approximately 30-45 seconds in total.

Eliminating redundant sequences and annotating the database. A checkpoint step in building the pan-genome as a searchable database is to ensure that no sequence is represented in the database more than once; this serves to eliminate unnecessary search space, as well as to prevent any mis-annotations due to spurious matches to query proteins. To accomplish this, the initial database was sorted from longest sequence to shortest, then split into five smaller files. Each file was used as the input to a Perl script, which used the Simrank Perl module to incrementally build and re-format a new database file. The first sequence was added to a blank database file, then the next-largest sequence was searched against this file using the Simrank matching algorithm; if the query protein matched below 50% similarity, it was added to the database file and this file was then re-formatted into the necessary binary file. This process was repeated for each protein in the subset until a database of unique sequences had been created. This process was conducted in parallel for all five database-subsets. Then, database 2 was searched against database 1, and all proteins matching below 50% similarity were added to database 1, and this was repeated in series for databases 3-5 until a complete database of protein families was created. In all, this process ran in approximately 3 hours (the serial version of this process was also tested, but took over 3 days to complete). This entire process was then repeated for the list of proteins, which appear in the genome of only one representative strain.

Once a database of unique "single-occurrence" genes was created, it was concatenated with the database of unique protein families.

Finally, this complete set of proteins was annotated using the existing ORNL bacterial annotation pipeline.

Searching the Pan-genome with Query Protein Sequences Using K-mer Peptides

Identifying protein-coding, tRNA, and rRNA genes in novel query genomes. A new genome to be annotated was first input to Prodigal (Hyatt, et al., 2010) to produce a list of protein-coding gene coordinates in the genome. The sequences from these coordinates were then translated to protein sequences and collected into a single FASTA-formatted file. tRNA and rRNA sequences were identified using tRNAScan (Fichant & Burks, 1991) and RNAmmer (Lagesen, et al., 2007), respectively.

Using Simrank to match query proteins to database sequences. In order to implement rapid protein matching between those in a novel genome and those in the annotated database, a k-mer search was implemented using the Simrank module for Perl. The database file was imported and formatted to a binary file of the same name, using k=10 and minimum length of 10. Next, the FASTA-formatted file of proteins from the query genome was used as the argument to the Simrank search function, with the function set to return only top match above a certain minimum threshold. This threshold was determined dynamically for each genome by minimizing the number of false positives (i.e., the number of secondary matches above the threshold being tested) and the 'false negative' rate (i.e., the percent of the total proteins *not* matched in the database at the threshold being tested) with respect to each other. This process is graphically

represented in Figure 2. The annotation for this matching protein was then copied to the output file of the pipeline.

Identifying and annotating novel proteins in query genomes. To identify novel proteins in the query genome, the protein-matching Perl script iterated through each query protein's entry in data structure returned by the Simrank search function. If a given protein did not match to any entry in the database at or above the threshold, its entry in this data structure was blank, and its locus tag and sequence were then added to a new data structure, and subsequently all un-matched proteins were printed to a separated FASTA-formatted file. This file was then used as input to the existing ORNL annotation pipeline. This annotation was then added to the output for the query genome.

Testing R-FAP against existing pipelines: speed and accuracy. Once the new annotation pipeline was in place, it was necessary to demonstrate it as a marked improvement over existing tools in the field. Since the original inspiration for the development of the tool, as well as one of its primary intended uses, has been the large-scale sequencing projects such as that at UC-Davis, a test of the tool's speed was conducted by running it on a total protein FASTA files from one hundred *S. enterica* strains. This served not only to determine a statistically-significant mean for run-time on a single genome, but also to test its ability to handle large number of genomes at a time.

To test the consistency and accuracy of the tool, it was run on the total protein sequences from a separately-annotated strain with a static threshold of 5% similarity. *S. enterica* genome. *S. Cubana str. CFSAN002050* was annotated using both the internal ORNL pipeline and the R-FAP tool, and the resulting annotations for each protein were

then compared. The accuracy was then calculated as the percentage of proteins which showed consistency between the two sets of annotation.

Chapter 4

Results, Discussion, and Recommendations

The Pan-genome Assembly Process

The workflow of the pan-genome assembly tool is outlined in Figure 3. In short, ANI for the 31 *S. enterica* strains was determined using Jspecies, which took approximately 6 hours to run. From this process, the BioNJ tree-building algorithm was used to produce the tree found in Figure 4, which shows that although most included strains represent minimal phylogenetic diversity, the inclusion of *S. enterica* subsp. *arizonae* increases the phylogenetic breadth to nearly 6%, comparable to existing trees for the species (National Center for Biotechnology Information, 2014).

Using OrthoMCL to determine protein families took approximately 18 hours and produced a total of 7,330 initial protein clusters. Post-processing of these clusters using *assemble_clusters.pl* took less than two hours and resulted in a total of 9,028 clusters in all. The sequences of these clusters were then aligned using Muscle, which in parallelized form ran in approximately 25 minutes, and *get_consensus.pl* was then used to build a consensus sequence from each of these alignments. Finally, using the Simrank algorithm to remove redundant sequences, a final list of protein families was generated, totaling 8,031 non-unique protein families (meaning, families found in two or more genomes) and 3,395 “single-occurrence” families (meaning those proteins found in only one genome), for a total of 11,426 sequences in all.

R-FAP as a Tool for Function Calling and Annotation

The final workflow for protein function prediction is laid out in Figure 5. Briefly, the newly-predicted proteins in a novel strain of *S. enterica* are input to the Simrank algorithm, which splits them into 10-*mer* peptides and searches them against the annotated pan-genome database. Each protein which matches a database protein above the selected threshold is then given the annotation of that match. Each protein for which no match is found in the database must be annotated separately; it is intended that this will be done using the internal ORNL pipeline, which will produce annotations for the few unmatched proteins in short order. These annotations will then be output along with those found using R-FAP.

In terms of speed, R-FAP processed 100 *S. enterica* genomes using a dynamic threshold in 4 hours, 33 minutes, 39 seconds, for an average of 2 minutes and 44 seconds per genome. This version settled on a threshold of 70 or 75% in all cases. For comparison, a static-threshold (50% similarity) version of R-FAP was run on the same set of genomes and ran in just under 53 minutes, for an average of 31.8 seconds per genome. This speed-up, combined with few genes missed, was found to be preferable to a dynamic threshold (the discussion below explains why minimizing "false positive" secondary matches is of negligible significance to minimizing the number of genes missed)

Regarding accuracy, R-FAP matched 4,468 proteins out of 4,674 to the pan-genome database at 5% similarity or above. Of these, 4,166 (93.5%) were given the exact same annotation by both tools. However, while the annotations for the pan-genome database in their current form are "single-entry" (meaning one piece of annotation per

protein family), those from a full genome annotated by the ORNL pipeline receive information for each queried database; deeper analysis and comparison of this information revealed of the 6.5% (292 proteins) which initially appeared inconsistent, 3.9% (173 proteins) were actually correct and 2.6% (118 proteins) were mis-annotated due to their corresponding database sequences having been inconsistently labeled during the post-processing step which followed their original annotation. Ultimately, this analysis showed that only a single protein (representing just 0.02% of the 4,674 proteins) was incorrectly annotated by the R-FAP tool.

Discussion and Recommendations

Thresholds and accuracy. The process of determining the minimum match threshold in real-time was implemented for two primary reasons; first, to eliminate 'false positive' secondary matches from the Simrank algorithm; secondly, to ensure accurate annotations for all allowed matches. However, avoiding these so-called 'false positives' have little bearing on the function of R-FAP, as the top match is the only match considered regardless of whether there is a secondary match. Furthermore, testing demonstrated that the primary matches were accurate 99.8% of the time all the way down to 5% similarity, and 100% of the time above 30% similarity. Since such a threshold results in an average of <5% of the total proteins being missed by R-FAP (which would then have to be annotated using the existing ORNL pipeline), the process of dynamically setting the match threshold for each new genome does not appear to be worth the time expenditure, and will likely be abandoned in the future.

Implications. The issues which confounded testing of accuracy, such as *prima facie* differences in annotation that further analysis showed to be the same, highlight the importance of the annotations applied to pan-genome sequences. Thus, future work will likely involve manual curation of the database and the annotations which it contains to ensure the best possible accuracy. On the other hand, it is also possible that the search/matching algorithm itself might be improved. Currently, Simrank calculates the similarity between two sequences by counting the number of matching k-mers and dividing by the length of the *shorter* of the two sequences. This means that a sequence which represents only a single domain could match at a very high percent to a much larger sequence which happens to contain said domain; this could also cause problems in cases where pseudo-genes have been called as real genes. The single incorrectly-annotated sequence is a prime example of both of these possibilities. R-FAP labeled it a 2'-5' RNA ligase, but a BLAST search of the sequence indicated that it only covered ~46% of 2'-5' RNA ligase and is likely a pseudo-gene; nevertheless, because the percent similarity was calculated based on the query sequence rather than the database sequence, it matched at 28% similarity. It is possible, therefore, that in the future it might be better to use the *longer* of the two sequences, or perhaps to simply use the length of the database sequence in all cases.

Nonetheless, the primary impact of this work is as a proof-of-concept for the use of a pan-genome based k-mer matching approach to functional prediction in genome annotation. In point of fact, the RAST pipeline is perhaps the biggest prospective competitor to R-FAP due to its recently update to take advantage of k-mer peptide

matching, in place of alignments, to its FIGfam database (Overbeek, et al., 2014). However, RAST's use of Glimmer to predict genes requires several iterations to achieve any level of accuracy and, combined with the massive search space inherent in the FIGfam database, this puts R-FAP at a distinct advantage. Indeed, whereas a pipeline such as that at ORNL spends approximately 90% of its 6-12 hours (~100 CPU hours) of runtime on functional prediction (Land, Details of the ORNL Microbial Annotation Pipeline, 2014), k-mer matching to a pan-genome database of proteins allows R-FAP to do the same process in seconds to minutes. In a hypothetical pipeline, therefore, 90% of the original CPU time would now be reduced by a factor of the genes annotated by R-FAP as a proportion of the total number of genes. Put another way, if R-FAP identified 95% of the total number of protein-coding genes in a genome, only 5% would need to be annotated using the traditional database search method. Thus, the CPU time would be reduced from ~100 hours to ~14.5 (~10 hours [gene prediction] + ~30 sec. [95% of functional prediction] + 4.5 hours [remaining 5%]). Thus, if 100 CPU hours represent ~8 hours of runtime, runtime would be reduced to just over an hour. This alone represents a major advancement, but even more significant is the reduction in sheer computing power required to accomplish the task of annotation.

Implementation. Two primary possibilities for implementation of R-FAP present themselves readily. The first, already discussed, is the large sequencing project, such as Dr. Weimer's, which seeks to study pathogens and other organisms on a vast scale to gain insight into their phylogenetic and functional variability. Such projects will require

high-throughput annotation mechanisms in order to keep pace and allow researchers to interpret data in any semblance of real-time.

The second possible implementation of a pipeline which uses R-FAP for protein function prediction is a rapid diagnostic tool for medical applications. As an increasing number of hospitals and clinics acquire next-generation sequencing technology to use in the real-time identification of pathogens, there will be an increasing need for rapid annotation of the output of these sequencers. A set of pan-genome databases for the most common human pathogenic species, combined with a pipeline based on R-FAP, would allow healthcare professionals to quickly identify not only a pathogen, but also the specific treatments to which it is resistant or susceptible. This in turn would allow the ability to make treatment decisions in a matter of hours rather than days, and could potentially save countless lives.

List of References

Altenhoff, A. M., & Dessimoz, C. (2012). Inferring Orthology and Paralogy. In M. Anisimova (Ed.), *Evolutionary Genomics: Statistical and Computational Methods* (Vol. 1, pp. 259-279). Humana Press. doi:10.1007/978-1-61779-582-4_9

Altschul, S. F., Gish, W. R., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology, 215*(3), 403-410.

Apweiler, R., Attwood, T. K., Bairoch, A., Bateman, A., Birney, E., Biswas, M., . . . Zdobnov, E. M. (2001). The InterPro database, an integrated documentation resource for protein families, domains and functional sites. *Nucleic Acids Research, 29*(1), 37–40.

Aziz, R. K., Bartels, D., Best, A. A., DeJongh, M., Disz, T., Edwards, R. A., . . . McNeil, L. K. (2008). The RAST Server: Rapid Annotations using Subsystems Technology. *BMC Genomics, 9*(75).

Benson, D. A., Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Sayers, E. W. (2014). GenBank. *Nucleic Acids Research, 42*(Database issue), D32–D37. doi:10.1093/nar/gkt1030

Berendzen, J., Bruno, W. J., Cohn, J. D., Hengartner, N. W., Kuske, C. R., McMahon, B. H., . . . Xie, G. (2012). Rapid phylogenetic and functional classification of short genomic fragments with signature peptides. *BMC Research Notes, 5*(460).

Borodovsky, M., & McIninch, J. (1993). Genmark: Parallel gene recognition for both DNA strands. *Computers & Chemistry, 17*(2), 123-133.

Budowle, B., Murch, R., & Chakrabort, R. (2005). Microbial forensics: the next forensic challenge. *International Journal of Legal Medicine, 119*(6), 317–30.

Chen, H. M., Wang, Y., Su, L. H., & Chiu, C. H. (2013). Nontyphoid Salmonella Infection: Microbiology, Clinical Features, and Antimicrobial Therapy. *Pediatrics & Neonatology, 54*(3), 147-152.

Claudel-Renard, C., Chevalet, C., Faraut, T., & Kahn, D. (2003). Enzyme-specific profiles for genome annotation: PRIAM. *Nucleic Acids Research, 31*(22), 6633-6639.

Cox, N., Cason, J., & Richardson, L. (2011). Minimization of Salmonella Contamination on Raw Poultry. *Annual Review of Food Science and Technology, 2*, 75-95.

Crump, J. A., & Mintz, E. D. (2010). Global trends in typhoid and paratyphoid fever. *Clinical Infectious Diseases, 50*(2), 241–246.

Delcher, A. L., Bratke, K. A., Powers, E. C., & Salzberg, S. L. (2007). Identifying bacterial genes and endosymbiont DNA with Glimmer. *Bioinformatics, 23*(6), 673–679. doi:10.1093/bioinformatics/btm009

DeSantis, T. Z., Keller, K., Karaoz, U., Alekseyenko, A. V., Singh, N. N., Brodie, E. L., . . . Larsen, N. (2011). Simrank: Rapid and sensitive general-purpose k-mer search tool. *BMC Ecology, 11*(1), 1-8. doi:10.1186/1472-6785-11-11

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research, 32*(5), 1792-1797.

Edwards, R. A., Olson, R., Disz, T., Pusch, G. D., Vonstein, V., Stevens, R., & Overbeek, R. (2012). Real time metagenomics: using k-mers to annotate metagenomes. *Bioinformatics, 28*(24), 3316–3317.

Fichant, G. A., & Burks, C. (1991). Identifying potential tRNA genes in genomic DNA sequences. *Journal of Molecular Biology, 220*(3), 659–671.

Finn, R. D., Bateman, A., Clements, J., Coggill, P., Eberhardt, R. Y., Eddy, S. R., . . . Punta, M. (2014). Pfam: the protein families database. *Nucleic Acids Research, 42*(Database issue), D222–D230. doi:10.1093/nar/gkt1223

Galperin, M. Y., & Koonin, E. V. (2012). Divergence and Convergence in Enzyme Evolution. *Journal of Biological Chemistry, 287*(1), 21-28.

Goldstein, E., Kirkcaldy, R. D., Reshef, D., Berman, S., Weinstock, H., Sabeti, P., . . . Lipsitch, M. (2012). Factors Related to Increasing Prevalence of Resistance to Ciprofloxacin and Other Antimicrobial Drugs in Neisseria gonorrhoeae. *Emerging Infectious Diseases, 18*(8), 1290-1297.

Hunter, S., Jones, P., Mitchell, A., Apweiler, R., Attwood, T. K., Bateman, A., . . . Yong, S.-Y. (2012). InterPro in 2011: new developments in the family and domain prediction database. *Nucleic Acids Research, 40*(Database issue), D306–D312.

Hyatt, D., Chen, G.-L., LoCascio, P. F., Land, M. L., Larimer, F. W., & Hauser, L. J. (2010). Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics, 11*, 119-130.

Jacobsen, A., Hendriksen, R. S., Aaresturp, F. M., Ussery, D. W., & Friis, C. (2011). The Salmonella enterica Pan-genome. *Microbial Ecology, 62*(3), 487–504.

Koonin, E. V., & Galperin, M. Y. (2003). Genome Annotation and Analysis. In E. V. Koonin, & M. Y. Galperin, *Sequence - Evolution - Function: Computational*

Approaches in Comparative Genomics (pp. 193-226). Boston: Kluwer Academic. Retrieved from http://www.ncbi.nlm.nih.gov/books/NBK20253/

Kristensen, D. M., Kannan, L., Coleman, M. K., Wolf, Y. I., Sorokin, A., Koonin, E. V., & Mushegian, A. (2010). A low-polynomial algorithm for assembling clusters of orthologous groups from intergenomic symmetric best matches. *Bioinformatics, 26*(12), 1481–1487.

Lagesen, K., Hallin, P., Rødland, E. A., Stærfeldt, H.-H., Rognes, T., & Ussery, D. W. (2007). RNAmmer: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Research, 35*(9), 3100–3108.

Land, M. (2014, March 28). Details of a survey of sequenced bacterial genomes. (J. M. Utley, Interviewer)

Land, M. (2014, March 31). Details of the ORNL Microbial Annotation Pipeline. (J. M. Utley, Interviewer)

Larsen, M. V., Cosentino, S., Lukjancenko, O., Saputra, D., Rasmussen, S., Hasman, H., . . . Lund, O. (2014). Benchmarking of Methods for Genomic Taxonomy. *Journal of Clinical Microbiology, Publshed online ahead of print*. doi:10.1128/JCM.02981-13

Lee, N., Hui, D., Wu, A., Chan, P., Cameron, P., Joynt, G. M., . . . Sung, J. J. (2003). A Major Outbreak of Severe Acute Respiratory Syndrome in Hong Kong. *The New England Journal of Medicine, 20*(348), 1986-94.

Li, L., Stoeckert, C. J., & Roos, D. S. (2003). OrthoMCL: Identification of Ortholog Groups for Eukaryotic Genomes. *Genome Research, 13*, 2178-2189.

Loman, N. J., Constantinidou, C., Chan, J. Z., Halachev, M., Sergeant, M., Penn, C. W., . . . Pallen, M. J. (2012). High-throughput bacterial genome sequencing: an embarrassment of choice, a world of opportunity. *Nature Reviews Microbiology, 10,* 599-606.

Lukashin, A. V., & Borodovsky, M. (1998). GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Research, 26*(4), 1107–1115.

Markowitz, V. M., Chen, I.-M. A., Palaniappan, K., Chu, K., Szeto, E., Grechkin, Y., . . . Kyrpides, N. C. (2010). The integrated microbial genomes system: an expanding comparative analysis resource. *Nucleic Acids Research, 38*, D382–D390.

Markowitz, V. M., Korzeniewski, F., Palaniappan, K., Szeto, E., Werner, G., Padki, A., . . . Kyrpides, N. C. (2006). The integrated microbial genomes (IMG) system. *Nucleic Acids Research, 34*, D344–D348.

Marra, M. A., Jones, S. J., Astell, C. R., Holt, R. A., Brooks-Wilson, A., Butterfield, Y. S., . . . Roper, R. L. (2003). The Genome Sequence of the SARS-Associated Coronavirus. *Science, 300*(5624), 1399-1404.

Mavromatis, K., Ivanova, N. N., Chen, I.-M. A., Szeto, E., Markowitz, V. M., & Kyrpides, N. C. (2009). The DOE-JGI Standard Operating Procedure for the Annotations of Microbial Genomes. *Standards in Genomic Science, 1*, 63-67.

McHardy, A. C., Goesmann, A., Pühler, A., & Meyer, F. (2004). Development of joint application strategies for two microbial gene finders. *Bioinformatics, 20*(10), 1622–1631.

Medini, D., Donati, C., Tettelin, H., Masignani, V., & Rappuoli, R. (2005). The microbial pan-genome. *Current Opinion in Genetics & Development, 15*, 589–594.

National Center for Biotechnology Information. (2014, March 27). *Salmonella Enterica*. Retrieved from NCBI Genome: http://www.ncbi.nlm.nih.gov/genome/?term=salmonella+enterica

Ogata, H., Goto, S., Fujibuchi, W., & Kanehisa, M. (1998). Computation with the KEGG pathway database. *Biosystems, 47*(1-2), 119-128.

Ohnishi, M., Golparian, D., Shimuta, K., Saika, T., Hoshina, S., Iwasaku, K., . . . Unemo, M. (2011). Is Neisseria gonorrhoeae Initiating a Future Era of Untreatable Gonorrhea?: Detailed Characterization of the First Strain with High-Level Resistance to Ceftriaxone. *Antimicrobial Agents and Chemotherapy, 55*(7), 3538–3545.

Overbeek, R., Olson, R., Pusch, G. D., Olsen, G. J., Davis, J. J., Disz, T., . . . Stevens, R. (2014). The SEED and the Rapid Annotation of microbial genomes using Subsystems Technology (RAST). *Nucleic Acids Research, 42*(Database issue), D206–D214. doi:10.1093/nar/gkt1226

Paradis, E., Claude, J., & Strimmer, K. (2004). APE: Analyses of Phylogenetics and Evolution in R language. *Bioinformatics, 20*(2), 289–290.

Richter, M., & Rossello-Mora, R. (2009). Shifting the genomic gold standard for the prokaryotic species definition. *Proceedings of the National Academy of Sciences, 106*(45), 19126-19131.

Salzberg, S. L., Delcher, A. L., Kasif, S., & White, O. (1998). Microbial gene identification using interpolated Markov models. *Nucleic Acids Research, 26*(2), 544–548.

Sentausa, E., & Fournier, P.-E. (2013). Advantages and limitations of genomics in prokaryotic taxonomy. *Clinical Microbiology and Infection, 19*(9), 790–795.

Smith, G. J., Vijaykrishna, D., Bahl, J., Lycett, S. J., Worobey, M., Pybus, O. G., . . . Rambaut, A. (2009). Origins and evolutionary genomics of the 2009 swine-origin H1N1 influenza A epidemic. *Nature, 459*, 1122-1126.

Sonnhammer, E. L., Eddy, S. R., & Durbin, R. (1997). Pfam: A Comprehensive Database of Protein Domain Families Based on SeedAlignments. *Proteins: Structure, Function, and Bioinformatics, 28*(3), 405–420.

Storm, C. E., & Sonnhammer, E. L. (2002). Automated ortholog inference from phylogenetic trees and calculation of orthology reliability. *Bioinformatics, 18*(1), 92–99.

Tamura, K., Stecher, G., Peterson, D., Filipski, A., & Kumar, S. (2013). MEGA6: Molecular Evolutionary Genetics Analysis Version 6.0. *Molecular Biology and Evolution, 30*(12), 2725–2729.

Tatusov, R. L., Koonin, E. V., & Lipman, D. J. (1997). A genomic perspective on protein families. *Science, 278*(5338), 631-637.

The Writing Committee of the World Health Organization (WHO) Consultation on Human Influenza A/H5. (2005). Avian Influenza A (H5N1) Infection in Humans. *The New England Journal of Medicine, 353*, 1374-1385.

van Dongen, S. M. (2000). *Graph Clustering by Flow Simulation.* (Unpublished doctoral dissertation). University of Utrecht, Utrecht, Netherlands.

Zhi, X. Y., Zhao, W., Li, W. J., & Zhao, G. P. (2012). Prokaryotic systematics in the genomics era. *Antonie van Leeuwenhoek, 101*(1), 21–34.

Zmasek, C. M., & Eddy, S. R. (2002). RIO: Analyzing proteomes by automated phylogenomics using resampled inference of orthologs. *BMC BIoinformatics, 3*(1), 14.

Appendices

Appendix A: Tables

Table 1. Comparison of annotation pipelines.

Pipeline	***Gene-prediction software***	***Number of database searches***	***Approximate run-time***
DOE/JGI	Prodigal	6 (sequential)	Up to 2 weeks (batch, server)
RAST	Glimmer	1	12-24 hours (server)
ORNL	Prodigal	4	6-12 hours or ~100 CPU hours

Table 2. Strains included in the pan-genome database.

Strain	***Genome size (Mb)***	***Protein-coding genes***
S. Paratyphi A str. AKU_12601	4.58	4,351
S. Paratyphi A str. ATCC 9150	4.59	4,348
S. 4,[5],12:i:- str. CVM23701	4.90	4,694
S. Heidelberg str. SL476	4.89	4,680
S. Heidelberg str. SL476	4.73	4,432
S. Saintpaul str. SARA23	4.72	4,350
S. Saintpaul str. SARA29	4.93	4,757
S. Javiana str. GA_MM040414	4.55	4,221
S. Schwarzengrund str. CVM196333	4.71	4,551
S. Schwarzengrund str. SL480	4.76	4,547
S. Typhi str. CT18	4.81	5,065
S. Typhi str. Ty2	4.79	4,632
S. Tennessee str. CDC07-0191	4.79	4,546
S. Agona str. SL483	4.84	4,508
S. Kentucky str. CDC 191	4.70	4,383
S. Kentucky str. CBM29188	4.79	4,745
S. Weltevreden str. HI N05-537	5.05	4,784
S. Virchow str. SL491	4.88	4,596
S. Paratyphi C str. RKS4594	4.83	4,690
S. Chorleraesuis str. SC-B67	4.76	4,792
S. Enteritidis str. P125109	4.69	4,363
S. Gallinarum str. 287/91	4.66	4,466
S. Dublin str. CT_02021853	4.84	4,682
S. Typhimurium str. D23580	4.88	4,804
S. Typhimurium str. 14028S	4.76	4,653
S. Typhimurium str. LT2	4.86	4,635
S. Paratyphi B str. SPB7	4.86	4,555
S. Newport str. SL317	4.95	4,720
S. Newport str. SL254	4.83	4,710
S. Hadar str. RI_05P066	4.79	4,487
S. arizonae str RSK2980	4.60	4,278

Appendix B: Figures

A.

Strain 1	Strain 2	Strain 3	Core genome	Pan-genome
		acA	*aaA*	*acA*
abA	*abA*		*aaB*	*abA*
aaA	*aaA*	*aaA*	*aaC*	*aaA*
aaB	*aaB*	*aaB*	*baA*	*aaB*
aaC	*aaC*	*aaC*	*caA*	*aaC*
baA	*baA*	*baA*	*daA*	*baA*
caA	*caA*	*caA*		*caA*
daA	*daA*	*daA*		*daA*
	eaA	*eaA*		*eaA*

B.

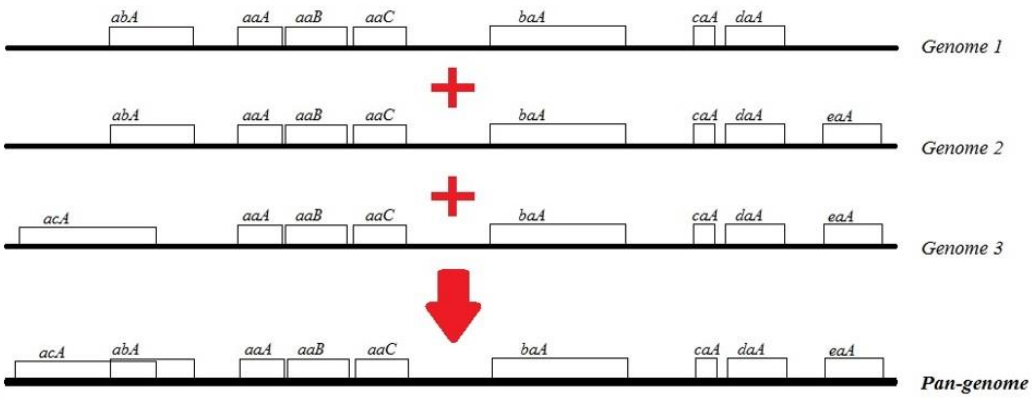


Figure 1. Building a hypothetical pan-genome in concept. Part **(A.)** illustrates the difference between a pan-genome and a core genome, which consists of all genes from every genome. Part **(B.)** demonstrates how a pan-genome is a sort of "layering" of the genes from each genome (note that this 'genomic scaffold' is purely conceptual; in the end, the pan-genome is merely the list of genes and their products).

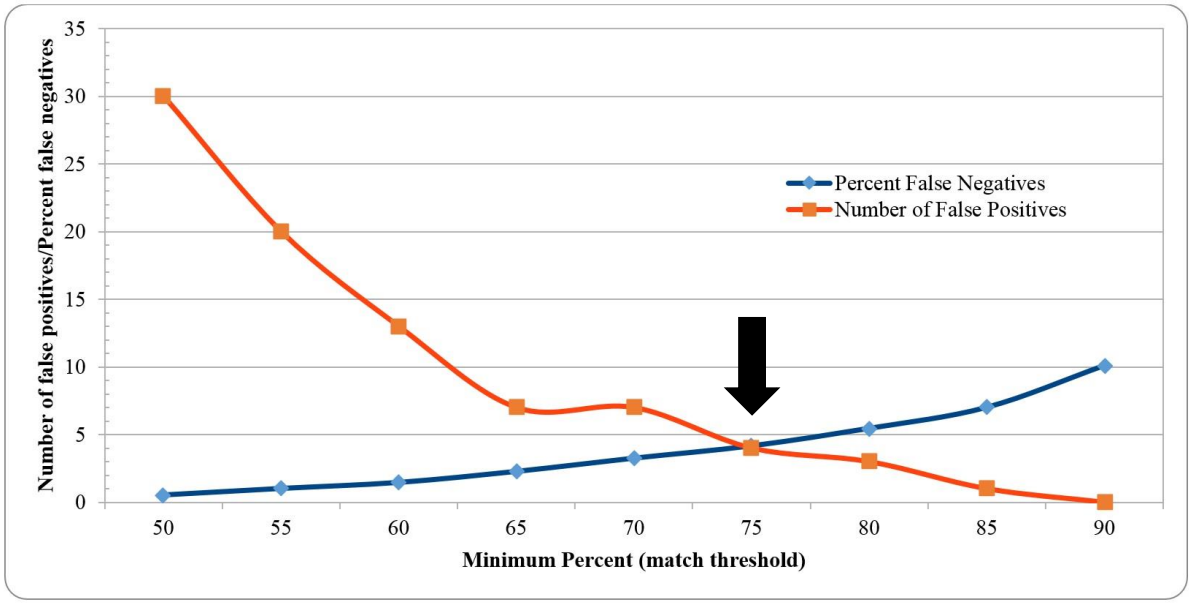


Figure 2. Minimizing the false positive count and the false negative rate. As seen here, the number of false positives (defined as secondary matches which fall above the minimum threshold) decreases as the minimum threshold is decreases, whereas the 'false negative rate' (i.e., as the number of proteins *not* matched in the database search divided by the total number of query proteins) increases he ideal threshold. Thus, the best threshold is that where the two intersect and are therefore minimized with respect to each other (as denoted by the black arrow).

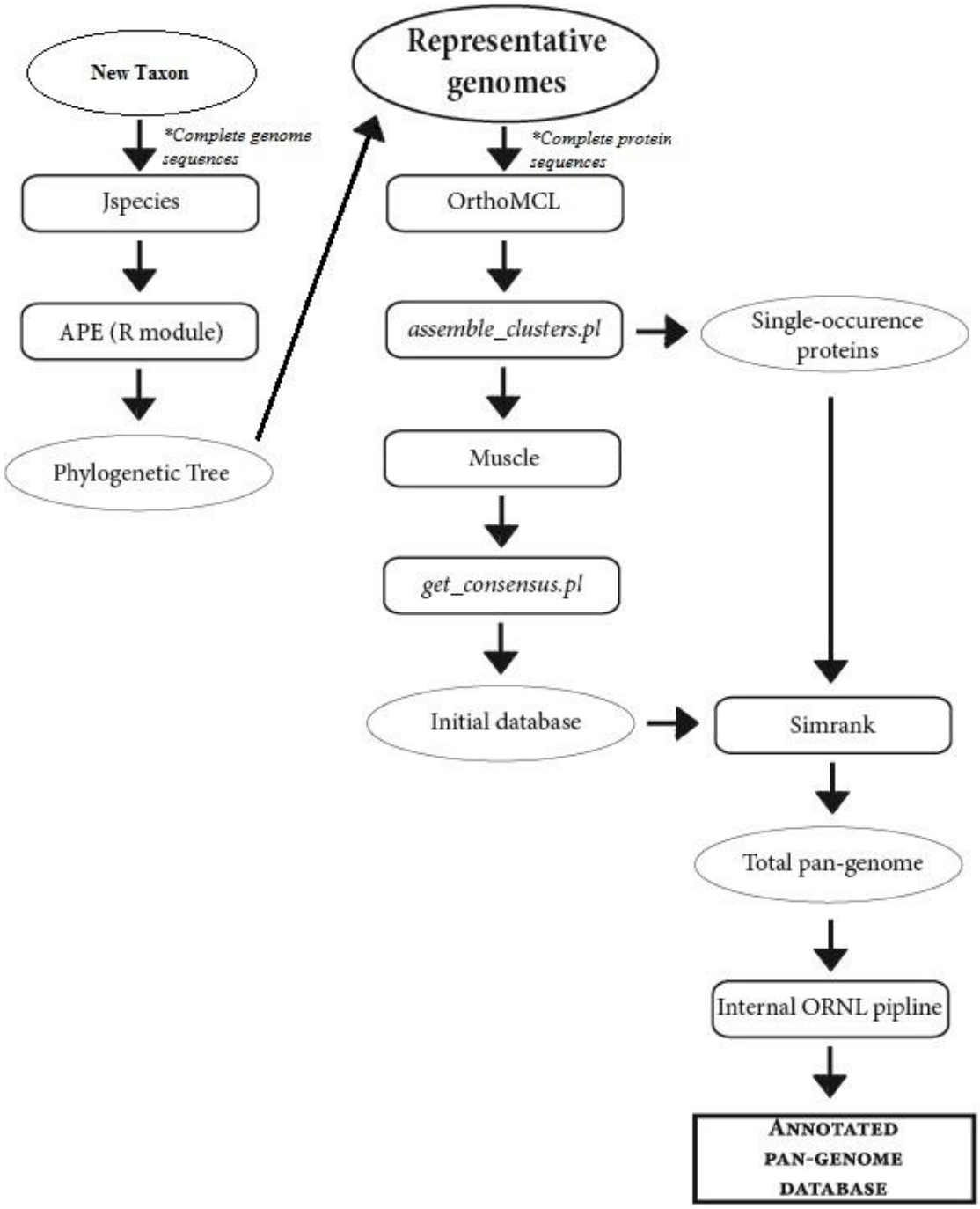


Figure 3. Outline of the pan-genome assembly workflow. Shown here is the step-by-step workflow for producing the pan-genome of *S. enterica*, and specifically, its annotated database of proteins.

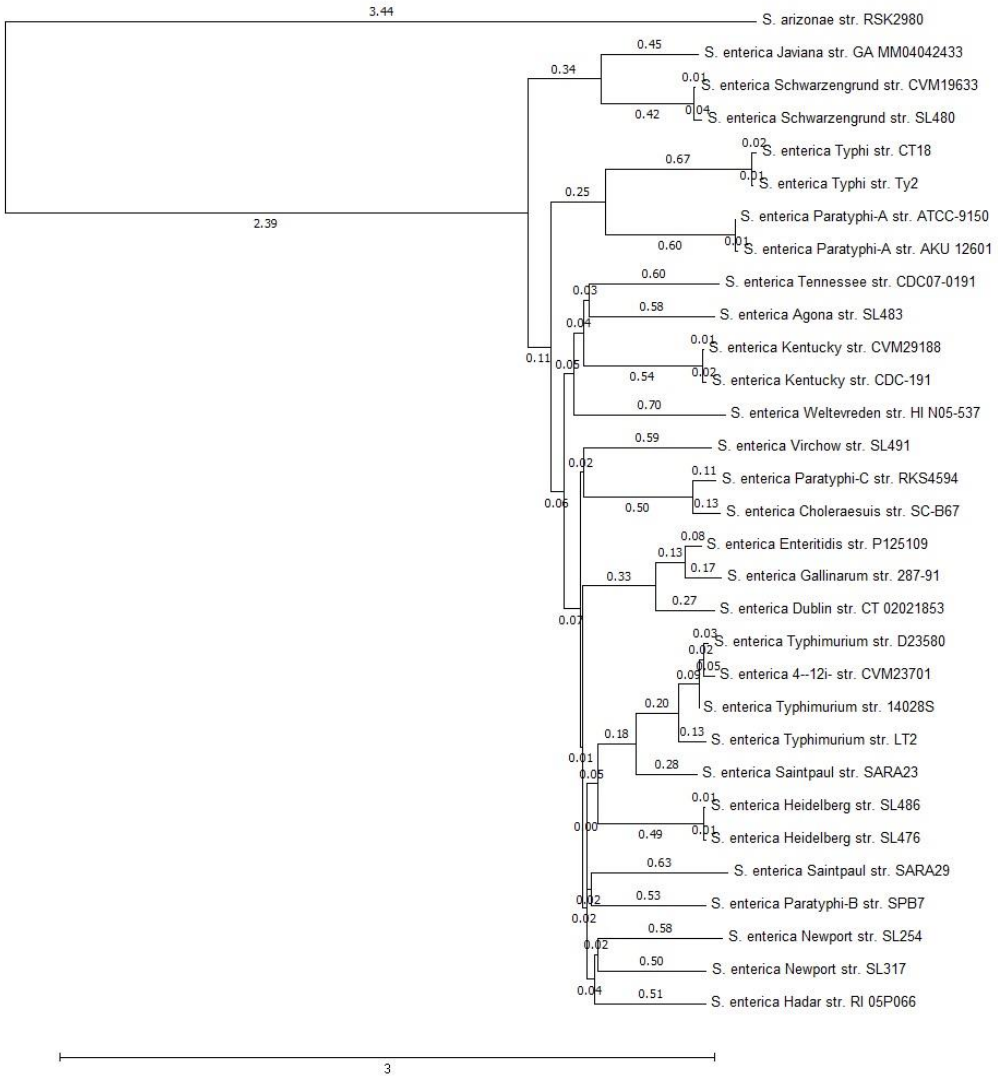


Figure 4. The phylogenetic tree of the 31 S. enterica *strains.* Shown here is the tree built using ANI in Jspecies and the BioNJ algorithm in APE package for R, which demonstrates tight grouping in all *S. enterica* subsp. *enterica* strains and *S. enterica* subsp. *arizonae* as a clear out-group, with a relative distance of several times that of any other two strains.

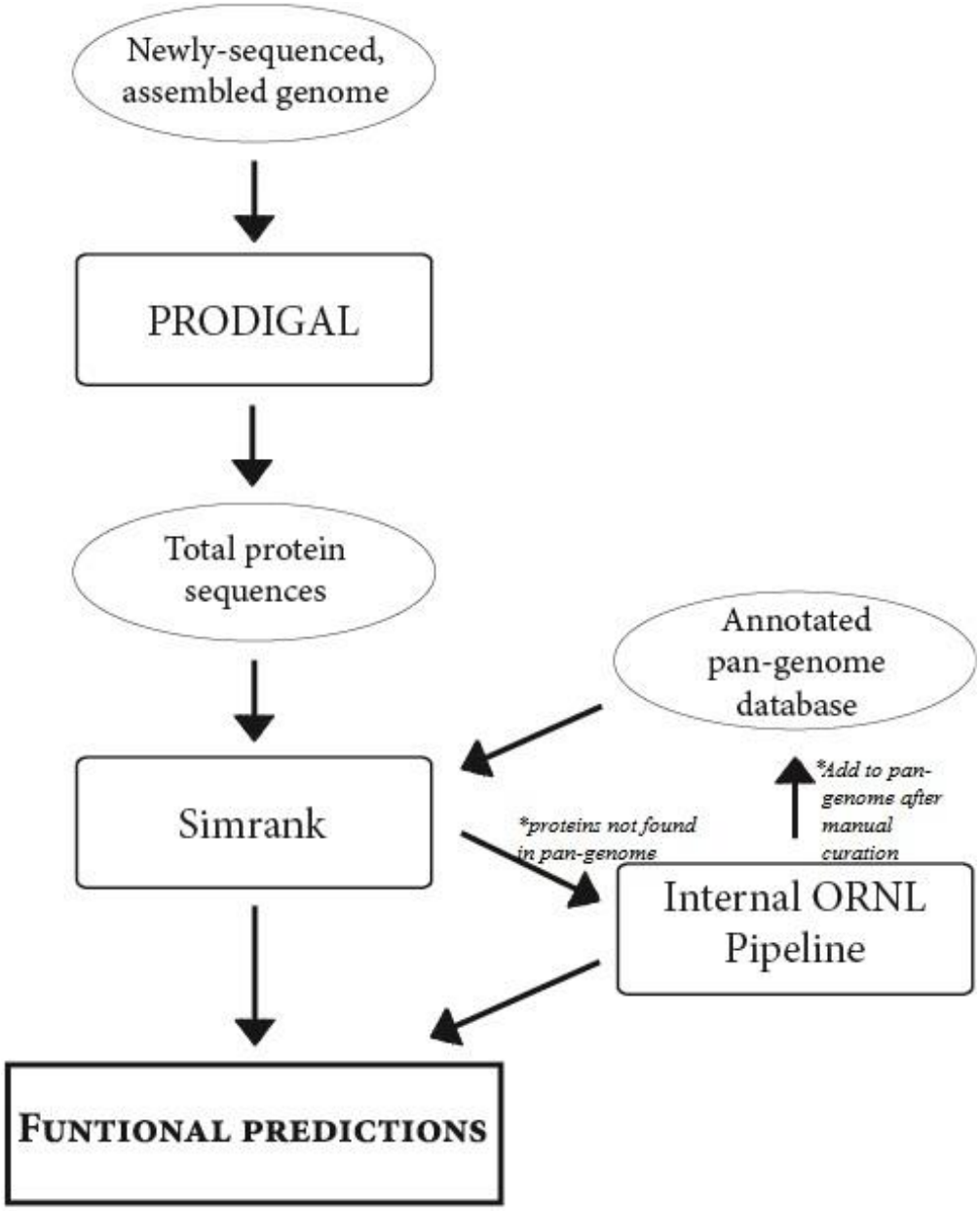


Figure 5. Functional prediction of proteins using R-FAP. The basic workflow of R-FAP: protein sequences, translated from gene predictions made in PRODIGAL, are searched against the pan-genome database using Simrank. All genes found in this database are given the annotation of their match in the database, while all proteins *not* found in the database are collected and sent to the internal ORNL annotation pipeline for annotation. Upon manual curation, these proteins and their annotations are added to the pan-genome database to help expedite future runs.

Appendix C: The R-FAP Pan-genome Assembly Source Code

```
#!/usr/bin/perl


use File::Path qw(make_path remove_tree);
use Getopt::Long;
use lib 'lib';
use String::Simrank;

$start_time = `date`;
$start_seconds = time;

print "\n\nStarted at: $start_time\n\n";


print "\n--------------------------------------------------------------------------------\n";
print "Welcome to the Rapid Functional Annotation of Prokaryotes (R-FAP) pan-genome\n";
print "database builder. This software takes the complete proteomes from a set of\n";
print "strains which you select, build the protein families using OrthoMCL, then define\n";
print "a consensus sequence for each protein family with the Muscle alignment software.\n";
print "--------------------------------------------------------------------------------\n\n\n";

print "Please ensure that the OrthoMCL directory is your current working directory,\n";
print "otherwise the OrthoMCL command will fail and the process will come to a halt.\n";
print "\nPlease also specify the path to OrthoMCL (or leave blank if running at ORNL):\n";

$PATH_TO_ORTHOMCL = <STDIN>;

print "\nAnd please also specify the pathway to the Muscle alignment software:\n";

$PATH_TO_MUSCLE =<STDIN>;



if($PATH_TO_ORTHOMCL !~ /\w/)
{
    $PATH_TO_ORTHOMCL = "/auto/database/orthomcl/"; #this will act as a default directory when the code
is run on ORNL computers
}
elsif($PATH_TO_ORTHOMCL!~ /^\// || $PATH_TO_ORTHOMCL!~ /\/$/)
{#this elsif serves in case the user forgets to put a '/' at the front or back of the directory name
    if($PATH_TO_ORTHOMCL!~ /^\//)
    {
        substr($PATH_TO_ORTHOMCL,0,0) = "/";
    }

    if($PATH_TO_ORTHOMCL!~ /\/$/)
    {
        $PATH_TO_ORTHOMCL .= "/";
    }
}

if($PATH_TO_MUSCLE !~ /\w/)
{
    #No reason to launch a 15-hour OrthoMCL run if you don't have the needed software on the receiving
end.
        die "Muscle location not specified\n\n";
}
elsif($PATH_TO_MUSCLE!~ /^\// || $PATH_TO_MUSCLE!~ /\/$/)
{
    if($PATH_TO_MUSCLE!~ /^\//)
    {
        substr($PATH_TO_MUSCLE,0,0) = "/";
    }

    if($PATH_TO_MUSCLE!~ /\/$/)
    {
        $PATH_TO_MUSCLE .= "/";
    }
}
```

```
print "As required by most implementations of OrthoMCL, please ensure that each strain\n";
print "is represented by its complete set of protein sequences in a separate FASTA file\n";
print "with the name ending in '.fa' and placed into the OrthoMCL data directory.\n\n";

print "Please specify these files now, each separated by a comma with no spaces:\n\n";

$file_names= <STDIN>;


# The following sets of 'if' statements attempt to ensure that the user has formatted the
# input correctly; it will not catch every error (OrthoMCL is surprisingly particular in
# this regard) but anything not caught here will be caught by OrthoMCL itself.
if ($file_names !~ /\.fa/ || $file_names =~ /(\.fasta|\.FASTA|\.faa)/)
{
        die "\n\nFile names must be formated 'name.fa'\n\n";
}

if($file_names !~ /\,+/)
{
    die "\n\nYou must input at least two files.n\n";
}

chomp $file_names;
$file_names =~ s/ //;

print "\n\nAlso, since the SearchIO feature of BioPerl is a requirement for OrthoMCL\n";
print "implementation, please specify the path to BioPerl with '\/' at front and back\n\n";

$bioperl_location = <STDIN>;
chomp $bioperl_location;

if($bioperl_location !~ /^\//)
{
    substr($bioperl_location,0,0) = "/";
}

if($bioperl_location !~ /\/$/)
{
    $bioperl_location .= "/";
}

print "\n\n\n";

# In order, the features of the OrthoMCL command as used for the purpose of creating a pan-genome are:
#   -I${bioperl_location}: informs OrthoMCL of the BioPerl file location.
#   orthomcl.pl: the central Perl wrapper script which calls the various algorithms.
#   --mode 1: OrthoMCL has four different modes; building new protein families requires Mode 1.
#   --fa_files ${file_names}: tells the OrthoMCL script what to look for in the data directory

`perl -I${bioperl_location} orthomcl.pl --mode 1 --fa_files ${file_names}`; || die "\nCan't run without
OrthoMCL!\n\n";

$date = `date`;

#the following uses the same method as OrthoMCL to create and name the working directory for a given
OrthoMCL run
$ORTHOMCL_WORKING_DIR = $PATH_TO_ORTHOMCL.(split(" ",$date))[1]."_".(split(" ",$date))[2]."/";

print "The path to the working directory is: $ORTHOMCL_WORKING_DIR\n\n\n";

make_path("$ORTHOMCL_WORKING_DIR/R-FAP/"); #create a new directory in the given working directory for
files related to R-FAP


#the following variables contain the paths to various OrthoMCL output files which are needed for the
next step
$all_fa = $ORTHOMCL_WORKING_DIR."tmp/all.fa";
$all_blast = $ORTHOMCL_WORKING_DIR."tmp/all.blast";
$all_blast_bbh = $ORTHOMCL_WORKING_DIR."all_blast.bbh";
```

```
$all_blast_score = $ORTHOMCL_WORKING_DIR."R-FAP/all_blast_score.bbh";
$blast_error = $ORTHOMCL_WORKING_DIR."R-FAP/blast_error.txt";
$all_orthomcl = $ORTHOMCL_WORKING_DIR."all_orthomcl.out";
$orthomcl_score_matrices = $ORTHOMCL_WORKING_DIR."R-FAP/orthomcl_score_matrices.txt";

# The following command calls the script which creates the reciprocal score matrices for each
# cluster created in OrthoMCL. It starts with the raw blast scores and creates a set of normalized
# scores (raw score divided by sequence length) which are then organized into a series of matrices.

$output = `perl ./Create_OrthoMCL_matrices.pl $all_fa $all_blast $all_blast_bbh $all_blast_score
$blast_error $all_blast_score $all_orthomcl $orthomcl_score_matrices`;
print "$output\n";

$final_cluster_lists = $ORTHOMCL_WORKING_DIR."R-FAP/final_cluster_lists.txt";

# The following command takes the matrices which were just created and uses them to tease apart those
# clusters which either contain multiple protein families or which contain wrongly-included proteins.
# This script calculates the threshold as the average of all scores in the matrix, and then iterates
# through each position of the matrix (and its corresponding position on the other side) and compares
# these scores to the threshold. If they are less than 80% of the threshold, the two corresponding
# proteins are not considered a match. Clusters are thus teased apart and printed to a list.

`perl ./Split_OrthoMCL_clusters.pl $orthomcl_score_matrices $final_cluster_lists`;

$alignments_directory = $ORTHOMCL_WORKING_DIR."R-FAP/alignments/"; #this will be used for 'make_path'
$single_occurence_proteins = $ORTHOMCL_WORKING_DIR."R-FAP/single_occurence_proteins.fasta";

# The following command takes the list file created above and pulls the sequences from the complete
# sequence 'all.fa' file created by OrthoMCL. These are organized in to a set of FASTA files, one
# for each cluster.
`perl ./Collect_sequences.pl $all_fa $final_cluster_lists $alignments_directory
$single_occurence_proteins`;

make_path($alignments_directory."muscle_output/";) #for organization sake, create a new subdirectory
for all the alignment files
$muscle_output = $alignments_directory."muscle_output/";

# The following command  calls a script (notice it calls a Python script instead of a Perl script)
# written by a colleague at ORNL (JJ Chai) which runs the Muscle alignment algorithm on multiple
# processors in parallel. It has been modified to allow the user to tell it the location of the Muscle
# software. For these purposes, it is hard-coded to output in Clustal format (".aln").

`python ./Parallelized_muscle.py -d $alignments_directory -o $muscle_output -m $PATH_TO_MUSCLE -p 4`;

$pre_cat_alignments = $muscle_output."*.aln";
$concatenated_alignments = $muscle_output."all_alignments.txt";

# The following is a bash command which takes all of the alignment files just created and joins them.
`cat $pre_cat_alignments > $concatenated_alignments`;

$initial_families = $ORTHOMCL_WORKING_DIR."R-FAP/initial_protein_families.fasta";

# The following takes the joint file just created and processes each alignment to produce a consensus
# sequence to represent each alignement, which in turn represent each protein family with at least two
# members in the pan-genome. The method for determing consensus is to iterate through every position
# in a given alignment and determine the most-common residue at every point which isn't universally
# conserved. The resulting sequences are then printed to a file as the initial pan-genome database.

`perl ./Create_consensus_sequences.pl $concatenated_alignments $initial_families`;



#-----------------------------------------------------------------------------------------------

# Everything which follows is the final post-processing of the protein families and single-occurence
# proteins to ensure that every sequence included in the pan-genome database is sufficiently unique
# so as to minimize spurious matches and thereby, spurious annotation. The method by which this is
# accomplished is to use the Simrank algorithm to determine the primary and secondary matches of each
# sequence at 50% similarity or higher; these secondary matches, where they exist, represent two
# sequences which are too similar to be included in the same database. For the sake of maximum number
# of possible k-mers in the final database, the method eliminates the shorter of the two sequences.
```

```
#
# This method is first applied to the initial set of protein families, using the file as both subject
# and querey, and then applied to the set of single-occurence proteins.

open(SEQ,$initial_families) || die "Did not open $sequence_filename.\n\n";

$i=0;
$sequence_name = '';

#first, open the initial protein family FASTA file and create a hash of its sequences...
while($buf = <SEQ>)
{
    chomp $buf;

    if($buf =~ /\>/)
    {
         if($i>0)
         {
                 $Lengths{$sequence_name} = length($Sequences{$sequence_name});
         }

         $sequence_name = substr $buf, 1;


         $sequence = '';
         $length = 0;
    }
    elsif($buf !~ /\>/ && $buf =~ /\w/)
    {
         $Sequences{$sequence_name} .= $buf;
    }
    else
    {
    }

    $i++;
}

$length_before = scalar (keys %Sequences);
print "\n\nThe total number of sequences in the initial database is: $length_before.\n\n";

$eliminate_redundant_clusters = new String::Simrank ({ data => $initial_families });

$eliminate_redundant_clusters->formatdb({  wordlen => 10,
                                           minlen  => 10,
                                           });

#...then use Simrank to match the initial families against themselves...
$matches = $eliminate_redundant_clusters->match_oligos( {
                          query => $initial_families, #the file of query sequences
                          outlen => 2, #the number of matches to return for a query sequence
                          minpct => 50, #the minimum percent similarity to consider a match
                          silent => true,
                          valid_chars => 'ABCDEFGHIJKLMNOPQRSTUVWYXZ',
                         });
#...then look at the resulting matches.
foreach $key (keys %{$matches})
{

    $k = 0;
    foreach $hit ( @{ $matches->{$key} } )
    {
        $gene_name = $key;
        $match = $hit->[0];
        $match_percent = $hit->[1];

        # it is reasonable to assume that in almost all cases, the top match for
        # any sequence will be itself so we're interested in the other match
        if($key ne $match)
        {
            if($Lengths{$key}<=$Lengths{$match})
                {
                #delete the shorter one
```

```
                delete $Sequences{$key};
                }
        }
    }
}

$length_after = scalar(keys %Sequences);
print "The total number of protein families after removing redunant sequences: $length_after.\n\n";

$complete_clusters = $ORTHOMCL_WORKING_DIR."R-FAP/final_protein_families.fasta";
open(PROC, ">$complete_clusters");

foreach $protein (keys %Sequences)
{
    print PROC ">$protein\n$Sequences{$protein}";
}

close PROC;
undef %Sequences;
undef %Lengths;

# Now that redundant sequences have been eliminated in the protein families, this must also be
# done in the so-called single-occurence proteins. This uses the exact same approach as before.

open(SEQ,$single_occurence_proteins) || die "Did not open $sequence_filename.\n\n";

$i=0;
$sequence_name = '';

#first, open the initial single-occurence protein FASTA file and create a hash of its sequences...
while($buf = <SEQ>)
{
    chomp $buf;

    if($buf =~ /\>/)
    {
        if($i>0)
        {
                $Lengths{$sequence_name} = length($Sequences{$sequence_name});
        }

        $sequence_name = substr $buf, 1;


        $sequence = '';
        $length = 0;
    }
    elsif($buf !~ /\>/ && $buf =~ /\w/)
    {
        $Sequences{$sequence_name} .= $buf;
    }
    else
    {
    }

    $i++;
}

$length_before = scalar (keys %Sequences);
print "\n\nThe total number of sequences in the initial database is: $length_before.\n\n";

$reduce_singles = new String::Simrank ({ data => $single_occurence_proteins });

$reduce_clusters->formatdb({ wordlen => 10,
                minlen  => 10,
               });

#...then use Simrank to match the initial singles against themselves...
$reduced_matches = $reduce_singles->match_oligos( {
                        query => $initial_families,
                        outlen => 2,
                        minpct => 50,
```

```
                        silent => true,
                        valid_chars => 'ABCDEFGHIJKLMNOPQRSTUVWYXZ',
                      });
#...then look at the resulting matches.
foreach $key (keys %{$reduced_matches})
{

    $k = 0;
    foreach $hit ( @{ $matches->{$key} } )
    {
        $gene_name = $key;
        $match = $hit->[0];
        $match_percent = $hit->[1];

        # it is reasonable to assume that in most cases, the top match for
        # any sequence will be itself so we're interested in the other match
        if($key ne $reduced_match)
        {
            if($Lengths{$key}<=$Lengths{$match})
                {
                #delete the shorter one
                delete $Sequences{$key};
                }
        }
    }
}

$length_after = scalar(keys %Sequences);
print "Total number of single-occurence proteins after removing redunant sequences:
$length_after.\n\n";

$reduced_singles = $ORTHOMCL_WORKING_DIR."R-FAP/reduced_singles.fasta";
open(RED, ">$reduced_singles");

foreach $protein (keys %Sequences)
{
    print RED ">$protein\n$Sequences{$protein}";
}

close RED;
undef %Sequences;
undef %Lengths;


# Now that we've eliminated the redundant sequences in the protein families, we must do the same for
# the single-occurence proteins to ensure that we do not include any sequences which were incorrectly
# exclused from one of the protein families. This is much more conceptually simple: if it matches a
# protein family consensus sequence above 50% similarity, it is eliminated.

open(SINGLE, $reduced_singles);

while($buf=<SINGLE>)
{
    chomp $buf;

    if($buf =~ /\>/)
    {

           $sequence_name = substr $buf, 1;

         $sequence = '';
          $length = 0;
    }
    elsif($buf !~ /\>/ && $buf =~ /\w/)
    {
        $Sequences{$sequence_name} .= $buf;
    }
    else
    {
    }
}

$eliminate_redundant_singles = new String::Simrank ({ data => $complete_clusters });
```

```
$eliminate_redundant_singles->formatdb({   wordlen => 10,
                                           minlen  => 10,
                                            });

$single_matches = $eliminate_redundant_singles->match_oligos( {
                           query => $reduced_singles,
                           outlen => 1,
                           minpct => 50,
                           silent => true,
                           valid_chars => 'ABCDEFGHIJKLMNOPQRSTUVWYXZ',
                           });

foreach $key (keys %{$single_matches})
{
    if( @{ $matches->{$key} } != '')
    {
    delete $Sequences{$key};
    }

}

open(FINAL,">>$complete_clusters");

foreach $sequence (keys %Sequences)
{
    # Append the final set of single-occurence proteins to the file already containing the final set
    # of protein families and this becomes our final pan-genome database. It is now ready to be
    # implemented as the user sees fit.
    print FINAL ">$sequence\n$Sequences{Sequence}";
}

close FINAL;


$end_time = `date`;
$end_seconds = time;
$run_seconds = $end_seconds-$start_seconds;
$run_hours = $run_seconds/3600;

print "\n\nStarted at: $start_time\nEnded at: $end_time\nTotal run time (hours): $run_hours\n\n";
```

Appendix D: The R-FAP Annotation Tool Source Code

```
#!/usr/bin/perl

$start_time = localtime(time);
print "Started at $start_time\n\n";

use Getopt::Long;
use lib 'lib';
use String::Simrank;
use autodie;
use File::Path qw(make_path remove_tree);

print "\n-------------------------------------------------------------------------------\n";
print "Welcome to the main Rapid Functional Annotation of Prokaryotes (R-FAP) tool.\n";
print "database builder. This software will take the complete gene translations from a\n";
print "set of selected strains and provide annotations for them in under a minute.\n";
print "-------------------------------------------------------------------------------\n\n\n";

print "Please provide the full path to the directory of the .faa files to be annotated:\n";

$operating_directory = <STDIN>;

if($operating_directory !~ /\w/) #exit if user provides no directory
{
        die "You must specify where to find the input files!\n\n";
}
elsif($operating_directory!~ /^\// || $operating_directory!~ /\/$/)
{#in case user forgets to include the '/' at the beginning or end
    if($operating_directory !~ /^\//)
    {
        substr($operating_directory,0,0) = "/";
    }

    if($operating_directory !~ /\/$/)
    {
        $operating_directory .= "/";
    }
}

#these two 'make_path' commands create the subdirectories for the output files
make_path($operating_directory."annotation_files/");
make_path($operating_directory."unmatched_sequence_files");

print "Next, please enter the full path to the FASTA-formatted pan-genome database\n";
print "which will be used to match query proteins to their appropriate annotations:\n\n";

$pangenome_filepath = <STDIN>;

if($pangenome_filepath !~ /\w/)
{
        die "You must specify where to find the database files!\n\n";
}
elsif($pangenome_filepath !~ /^\//)
{
    substr($pangenome_filepath,0,0) = "/";
}

print "\nNow, please specify the full path to the three-column annotation source\n";
print "file. (It would probably be best to make sure its located in the operating\n";
print "directory, but in either case you need to enter the full path to it here.)\n\n";

$annotation_filename = <STDIN>;

if($annotation_filename !~ /\w/)
{
    die "You must provide a source of annotations!\n\n";
}
elsif($annotation_filename !~ /^\//)
{
    substr($annotation_filename,0,0) = "/";
}

open(ANNO,$annotation_filename) || die "Could not open $annotation_filename\n\n";
$anno_iterator = 0;
```

```
# This loop iterates through the annotation file, which is a three-column, tab-delimited
# file of basic annotation for each gene and the tool used to determine it

while($buf = <ANNO>)
{
    chomp $buf;
    if($anno_iterator > 0)
    {
         ($protein,$id_tool,$annotation) = split(/\t/, $buf);

         if($id_tool !~ /\w/)
         {
                 $id_tool = "";
         }

         $Annotation_hash{$protein} = "$id_tool\t$annotation";
    }
    $anno_iterator++;
}

# Below are the commands which declare a new Simrank object and create a new binary database file
$sr = new String::Simrank ({ data => $pangenome_filepath });
if ($cl_args->{"rebuild"} || !$sr->{binary_ready} )
{
         $sr->formatdb({  wordlen => 10,
                          minlen  => 10,
                          });
}

$i=0;


# This loop will iterate through all of the '.faa' files in the specified directory and annotate them

while (defined($sequence_filename = glob $operating_directory.'*.faa'))
{

    open(SEQ,$sequence_filename) || die "Did not open $sequence_filename.\n\n";

    $sequence_name = '';

    while($buf = <SEQ>)
    {
         chomp $buf;

         if($buf =~ /\>/)
         {
                 $sequence_name = substr $buf, 1;
                 $sequence = '';
                 $length = 0;
         }
         elsif($buf !~ /\>/ && $buf =~ /\w/)
         {
                 $Sequences{$sequence_name} .= $buf;
         }
         else
         {
         }
    }
    close SEQ;

    $number_of_query_genes = scalar(keys %Sequences);
    $current_time = localtime(time);
    print "\nStarted at $start_time\nCurrent time is $current_time\n\n";
    print "Number of genes in $sequence_filename = $number_of_query_genes\n";

    #30% appears to be the lowest point at which accuracy is still maintained at 100%
    $minimum_match_percentage = 30;

    #this is actually just a count of the number of secondary matches found by Simrank
    $number_of_false_positives = 0;

    #this is actually just a percentage of the total proteins in a genome which Simrank doesn't find
    $false_negative_rate = 0;

    $j=0;
    $number_of_false_positives = 0;
```

```
    undef %Match_list;
    undef @putative_new_genes;

    # The following command calculates the match between each query protein and each sequence
    # in the search database; these matches are sorted most- to least-similar and retunred as
    # a hash (specifically, a hash of arrays)
    $matches = $sr->match_oligos( { query => $sequence_filename,
                    outlen => 2,
                    minpct => $minimum_match_percentage,
                    silent => true,
                    valid_chars => 'ABCDEFGHIJKLMNOPQRSTUVWYXZ',
                  });

    # Once this hash has been created, one must step through it protein at a time; each query
    # protein has an entry in the hash, even if it didn't match anything in the database.
    foreach $key (keys %{$matches})
    {
    #this 'if' tests to see if a protein's entry in the hash is blank; if it is, then it
    #didn't match to anything in the database above the specified similarity threshold.
    if( @{ $matches->{$key} } == '')
    {
        push(@putative_new_genes, $key);
    }

    $k = 0;
    foreach $hit ( @{ $matches->{$key} } )
    {
        $gene_name = $key;
        $match = $hit->[0];
        $match_percent = $hit->[1];

        if($k==0) #this indicates the first (and therefore best) match
        {
                $Match_list{$gene_name} .= "$gene_name\t$Annotation_hash{$match}";
        }
        elsif($k>0) #even if this is ever true, it represents a spurious match
        {
                $number_of_false_positives++
        }

        $k++;
    }
    $j++;
    }

    $number_of_matched_genes = scalar(keys %Match_list);

    #the following calculates the 'miss rate' by dividing the number which ACTUALLY matched
    #at or above the given threshold from the total number of proteins searched.
    $false_negative_rate = 100*(1-($number_of_matched_genes/$number_of_query_genes));
    $i++;
    print "Currently on genome $i of 100\n\tMissed rate = $false_negative_rate\n";
    print "\tNumber of secondary matches = $number_of_false_positives\n";

    $short_input_filename_start = rindex($sequence_filename,"/");
    $short_input_filename = substr($sequence_filename,$short_input_filename_start);

    while($short_input_filename =~ /\./)
    {
         chop $short_input_filename;
    }

    #create the output file from the input file name
    $final_output_filename =
$operating_directory."annotation_files/".$short_input_filename."_annotation.txt";
    open(OUT, ">$final_output_filename") || die "Could not create and/or open
$final_output_filename.\n\n";

    #create the name for the FASTA file of unmatched proteins
    $unmatched_genes_filename =
$operating_directory."unmatched_sequence_files/".$short_input_filename."_unmatched.fasta";
    open(UNMATCHED, ">$unmatched_genes_filename") || die "Could not create and/or open
$unmatched_genes_filename.\n\n";

    print OUT "Protein\tID_tool\tAnnotation\n";
    foreach $gene (keys %Match_list)
    {
```

```
            print OUT "$Match_list{$gene}\n"; #note that this is formatted like the annotation file
        }

        close OUT;


        for($k=0;$k<scalar(@putative_new_genes);$k++)
        {
            print UNMATCHED ">$putative_new_genes[$k]\n$Sequences{$putative_new_genes[$k]}\n";
        }

        close UNMATCHED;

        #it's important to 'undef' (erase) these data structures each time
        undef @putative_new_genes;
        undef %Match_list;
        undef %Sequences;

}

$current_time = localtime(time);
print "\n\nStarted at $start_time\nFinished at $current_time\n\n\n";
```

Vita

Jordan M. Utley attended Trevecca Nazarene University, where he received a Bachelor of Science, *magnum cum laude,* in biology with minors in chemistry and history in May of 2010. He was subsequently admitted to the University of Tennessee/Oak Ridge National Laboratory Graduate School of Genome Science and Technology in August of 2010.