8-10-2012

Assembly, annotation, and polymorphic characterization of the Erysiphe necator transcriptome

Jason Myers

R·I·T

Assembly, Annotation, and Polymorphic Characterization of the *Erysiphe necator* Transcriptome

by

Jason Myers

Submitted in partial fulfillment of the requirements for the Master of Science degree in Bioinformatics at Rochester Institute of Technology.

Department of Biological Sciences

School of Life Sciences

Rochester Institute of Technology

Rochester, NY

August 10, 2012

Committee:

Dr. Gary Skuse
Associate Head of the School Life Sciences/Professor/Committee Member

Dr. Michael Osier
Bioinformatics Program Head/Program Advisor/Associate Professor/Committee Member

Dr. Dina Newman
Thesis Advisor/Associate Professor

Dr. Lance Cadle-Davidson
Associate Thesis Advisor/Plant Pathologist USDA-ARS

Dr. Angela Baldo
Thesis Project Advisor/Computational Biologist USDA-ARS

Abstract:

The objectives of this study were to develop a transcriptomic reference resource and to characterize polymorphism between isolates of *Erysiphe necator* (syn. *Uncinula necator*), grape powdery mildew. The wine and fresh fruit markets are economically vital to many countries worldwide, and *E. necator* infection can cause severe crop damage and subsequent financial loss. Most of the publicly available sequence data for Erysiphales are from research done on *Blumeria graminis* f. sp. *hordei*, barley powdery mildew, which occupies a distinct clade within the Erysiphales. We obtained 641,601 sequencing reads from a Roche 454-FLX next-generation sequencer (RNA-Seq) and performed multiple assemblies using the Mira assembly software. The best assembly was *de novo* and yielded 39,686 contiguous sequences. The reference was then ordered based on similarity to *B. graminis* genes and annotated based on sequence similarity to known proteins. 11,605 SNPs and 5,248 INDELs were called against the reference using RNA-Seq data from 55 additional geographically and phenotypically distinct isolates of *E. necator*. The reference transcriptome, annotations, and polymorphic characterization collections from this project represent a vast resource for *E. necator* and should allow for future research of this organism and other Erysiphales. Our results illustrate that RNA-Seq is a valid alternative to whole-genome sequencing for genetic characterization of non-model organisms.

Table of Contents

1. Introduction:

The purpose of the research described here is to create transcriptomic resources to support study of *Erysiphe necator* (syn. *Uncinula necator*), the grape powdery mildew fungus. *E. necator* is an obligate biotroph of the genus *Vitis* (grapevine), meaning that it only grows and reproduces on living plant tissue [1]. The fungus is believed to have originated in eastern North America, was introduced to Europe around 1845, and spread to all grape-producing regions in the world, likely as a result of the trading of grapevines [2]. For most of its life cycle this fungus exists as a haploid ascomycete reproducing asexually by spores known as conidia, but when both mating types are present, sexual reproduction is possible, resulting in the production of ascospore-containing cleistothecia [2]. Most of the publicly available data for Erysiphales are from research done on *Blumeria graminis* f. sp. *hordei* (*Bgh*), barley powdery mildew, which occupies a distinct clade within the Erysiphales (Fig. 1) [1, 3, 4].

Figure 1 Phylogenetic analysis of the ITS region for 45 powdery mildews plus two outgroup species (*Phialocephala* and *Phialophora*). The tree is a phylogram of one of the 40 most parsimonious trees, which was found using a heuristic search employing the random stepwise addition option of PAUP. The tree also has the highest likelihood of the 40 most parsimonious trees, which was found by determining the log likelihood of all 40 trees. Horizontal branch lengths are proportional to the number of nucleotide substitutions that were inferred to have occurred along a particular branch of the tree. Branch support was determined by 1000 bootstrapped data sets, shown on the tree as numbers above the supported branches. Branches = 95% are strongly supported and are shown in bold

(Felsenstein 1985), bootstrap values below 50% are not shown. Bremer support is shown below the branches (Bremer 1988). The consistency index (CI) is 0.574, the retention index (RI) is 0.809, and the rescaled consistency index (RC) is 0.464 (Farris 1989). Conidium and conidiophore morphology in relation to phylogeny inferred from ITS variation is mapped onto the tree. Conidiophore types: A, D, and E, spores formed singly; B, C, F, and G, spores formed in chains. A, B, C, F, and G all belong to the mitosporic genus *Oidium*; D is the mitosporic genus *Ovulariopsis*, and E is the mitosporic genus *Oidiopsis*. The genus *Oidium* is further subdivided into six subgenera based on conidial surface patterns. *Oidium* subgenus *Pseudoidium* is represented as A; *Oidium* subgenus *Striatoidium* is represented as B; *Oidium* subgenus *Reticuloidium* is represented as C; *Oidium* subgenera *Fibroidium* and *Setoidium* is represented as F; and *Oidium* subgenus *Oidium* is represented as G. The numbers adjacent to the right of the tree correspond to the six major clades of powdery mildews, which are strongly correlated with the six different mitosporic types.

Source: [4]

E. necator infects host cells in much the same way as *Bgh* [5]. The mode of infection after a conidium lands and adheres on the host involves germination of the conidium, appressorium attachment to the cuticle, and formation of a penetration peg to pierce the host cuticle and cell wall [3, 5]. Successful infection results in the formation of a haustorium, an extension of a penetration peg that forms within a single host epidermal cell (Fig. 2) [6].

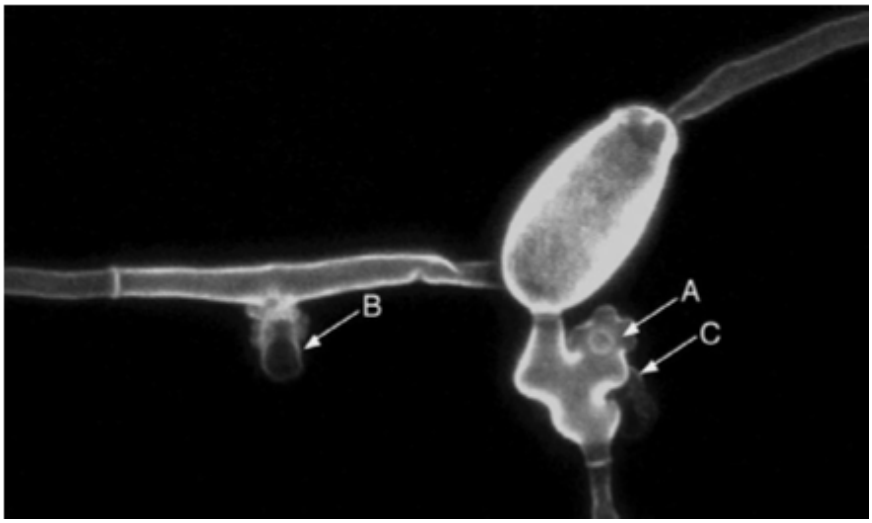

Figure 2 Laser scanning confocal micrograph of *Erysiphe necator* on an ontogenically susceptible leaf of *Vitis vinifera* at 72 h post-inoculation, stained with wheat germ agglutinin conjugated with Alexafluor-488, showing the multilobed primary appressorium and penetration pore (A) and secondary germ tube with appressorium (B). The globose haustorium (C) is faintly and partially visible at the lower right, beneath the primary appressorium.

Source: [6]

The haustorium is the only cell of the fungus that interacts directly with a host cell. It secretes proteins that suppress host defenses and shuttle nutrients from the host to the pathogen, determining whether the interaction between host and parasite will be compatible or incompatible [3, 5]. One of the goals of transcriptomic analysis is to identify candidate secreted proteins, as they are likely to be under significant evolutionary pressure to overcome the host response (apoptosis) to their presence [3].

Research of the grape powdery-mildew fungus, *E. necator,* is motivated by its economic impact on the grape industry everywhere grapes are grown [7]. The wine and fresh fruit markets are economically vital to many countries worldwide, and *E. necator* infection can cause severe crop damage and subsequent financial loss. There are also financial and environmental issues with over-use of fungicides to combat the disease because little is understood about severity

thresholds that have a significant effect on quality and yield [7]. The lack of knowledge causes undue financial strain on grape growers by indirectly advocating excessive application of fungicide [7]. While genomic characterization of *E. necator* is desirable for the above reasons, only 19 ESTs have been deposited at the National Center for Biotechnology Information's (NCBI) GenBank for *E. necator* currently [8]. This fact is most likely due to the inability to culture *E. necator* axenically (i.e. it must be cultured on living plant tissue) [5]. *Bgh* has been researched much more than any other powdery mildew species and there are significant genomic resources available for comparison of *E. necator* ESTs to *Bgh* genes [1]. An extensive EST collection for *E. necator* is important because it will increase the amount of information available for comparison of adaptive variation amongst all the Erysiphales. In an attempt to increase knowledge about *E. necator,* the goals of this study include the assembly of EST sequences into a reference transcriptome, the annotation of this reference, and the characterization of polymorphism between the reference and 55 phenotypically and/or geographically distinct isolates of *E. necator*.

Transcriptomics as a field of study within biology began in the mid 1990's with the advent of microarrays [9] and has progressed through advancements in high-throughput sequencing technologies [10] and the development of the RNA-Seq method [11]. While the term transcriptome can encompass the RNA complement of a cell, including mRNA, tRNA, rRNA, and non-coding RNA for a particular physiological condition or developmental stage of a cell [12], the use of the term in this thesis will focus on the mRNA complement of an organism. The study of the transcriptome is essential for characterization of development, disease, functional elements of a genome, and/or genetic diversity within a species, allowing unprecedented views into the complexity of genetic expression in nearly any organism [13]. Several approaches have

been developed to facilitate researchers in the study of transcriptomes including hybridization-based and sequencing-based methods, but for either approach, genes must first be identified in a reference sequence database [12]. The objectives of this study were to develop a transcriptomic reference resource and to characterize polymorphism between isolates of *E. necator*. Figure 3 shows a graphical representation of the many steps undertaken here towards these objectives.

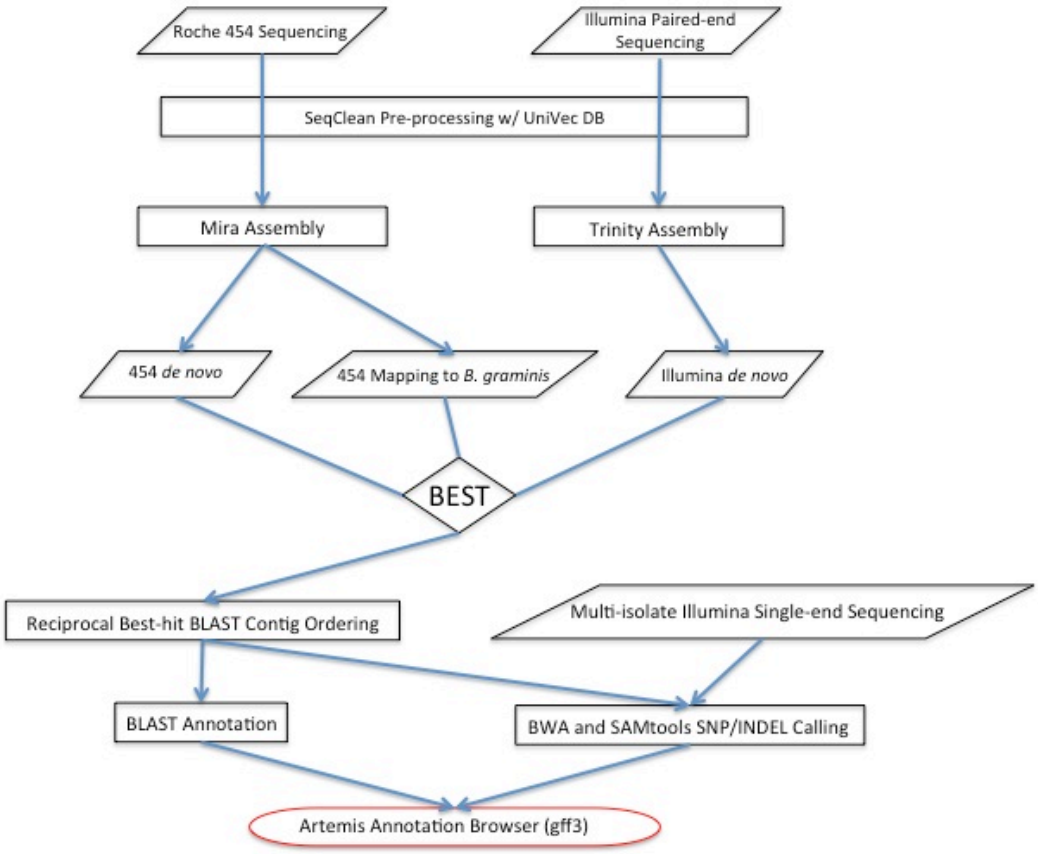

Figure 3 Graphic representation of the work proposed and completed during the course of this project. The Roche 454-FLX and Illumina HiSeq next generation sequencing technologies were used to generate RNA-Seq data for the reference strain (G14), and Illumina HiSeq was used to generate RNA-Seq data for 55 additional isolates of *Erysiphe necator*. Mira [14, 15] and Trinity [16] are both computer software packages capable of assembling RNA-Seq data *de novo* and were used on the reference strain data to generate the mRNA contiguous sequences (contigs). The best assembly was chosen, then the contigs were ordered based on their *in silico*-translated similarity to proteins of the closest related

> organism with a publicly available, ordered genome (*Blumeria graminis* f. sp. *hordei*). The *E. necator* contigs were then annotated based on similarity to the National Center for Biotechnology Information's non-redundant database and the SwissProt database. SNPs and INDELs between the reference isolate and 55 additional isolates were then called using BWA [17] and Samtools [18] software. All of the annotation and SNP/INDEL information attained during the above steps were then compiled in the gff3 file-format for viewing within the Artemis Annotation Browser [19].

Sequencing-based methods of transcriptome study offer a way of quantifying gene expression through hard counts of the transcripts present in a sample, but more importantly for this study, a way to obtain the sequence of those transcripts. During the last few years, Sanger sequencing of cDNA or EST libraries has been supplanted with high-throughput tag-based methods for serial analysis of gene expression (SAGE), cap analysis of gene expression (CAGE), and massively parallel signature sequencing (MPSS) [12]. Tag-based methods allow one to quantify genetic expression, but they require a significant amount of genetic annotation of the transcriptome in order to make sense of the resulting data [12]. Lacking resources for whole genome sequencing, transcriptomics through next-generation sequencing of ESTs, known as RNA-Seq, offers an efficient means to attain genomic level data for non-model organisms, such as *E. necator* [20].

The RNA-Seq method involves converting a population of RNA fragments, generally transcripts that have been post-transcriptionally modified with a poly (A) tail, to a library of cDNA sequences with adaptors attached that allow for amplification via PCR [12]. The PCR products are then sequenced in a high-throughput manner using one of the next-generation sequencing technologies. The Roche 454 Life Science, Illumina GA, and Applied Biosystems SOLiD technologies have been used in previous studies; however, any high-throughput sequencing method is conceivably applicable for use in RNA- Seq experiments [12]. The reads

generated from RNA-Seq runs are generally between 30-400bp in length depending on the version of the sequencing technology used [12]. One of the advantages of RNA- Seq exploited in this study is that genomic reference sequence is not necessary for the assembly of reads. *De novo* assembly is possible if sufficiently high depth of coverage is attained per transcript [12, 20]. Another advantage is the relatively small amount of sequencing necessary for characterizing the transcriptome of a eukaryotic organism as opposed to sequencing the genome of the same organism. This is due to the fact that most eukaryotic genomes are comprised of large amounts of non-coding DNA in the form of introns and intragenic regions [20]. While the non-coding regions of DNA contain transcription factor binding sites and are important for discovering how and why gene expression differs within a species, they do not generally apply to the study of how transcripts and their protein products differ between individuals. EST sequences lack intragenic DNA and in most cases lack introns (though alternative splicing of a transcript can result in the inclusion of intron sequence), which means less sequencing is needed than for whole genome sequencing [20].

There are significant computational challenges to dealing with RNA-Seq data. Storage and manipulation of the large quantity of data generated by the next-generation sequencing technology is not trivial, easily ranging from tens to hundreds of gigabytes per run [12]. The number and small lengths of the fragments reduces cost and time required for transcriptome sequencing but increases the complexity of computational analysis. Assembling the small reads generated during RNA-Seq in a genome-independent manner is particularly difficult computationally; fortunately, many programs have been developed to facilitate researchers use of such data [21]. Programs capable of this type of assembly, including Velvet [22], TransABySS [23], and MIRA [14, 15], produce consensus transcripts (contigs) that can be aligned to

nucleotide or protein databases for annotation of function via sequence similarity [21]. Many more assembly programs (Cufflinks, Scripture, etc.) are available for assembly of RNA-Seq data using genomic sequence as a template, called a mapping assembly [21]. MIRA is an example of a program that can perform *de novo* and mapping assemblies, which makes it attractive to use for comparisons between methods [14]. EST sequences from one species can also be used as a reference for a mapping assembly of RNA-Seq data from a related species, if the two species are closely related [20].

It is difficult to annotate a RNA-Seq transcriptome without a reference genomic sequence to alignment. Many methods have been developed to utilize our current understanding of genetic features such as open reading frames (ORFs), intron splicing (and alternative splicing), pseudogenes, transposons, similarity and protein folding to assist with assigning reading frames, location of translation complex binding regions, and most importantly function [24]. However, even with the tools available, it is difficult to annotate RNA-Seq without genomic sequence because there is no reliable way to determine if the contigs generated in an assembly program are complete (the full EST was sequenced) or if multiple contigs resulted from the same transcript due to sequencing error. With that in mind, one must be careful in choosing parameters for the various stages of assembly, annotation, and polymorphic characterization to ensure that the results are as accurate as possible (generally by increasing significance thresholds). The large-scale nature of RNA-Seq experiments does however bring the level of annotation possible for non-model organisms up to what was previously only possible for model organisms, even if at the expense of some certainty in those annotations.

Characterizing polymorphism involves the determination of single nucleotide polymorphisms (SNPs) and insertion/deletions (INDELs) within a population. SNP and INDEL

calling is classically accomplished by aligning ESTs to a genomic reference sequence but can be accomplished using a transcriptome as the reference [20]. The Computational Biology Service Unit (CBSU) at Cornell University has developed a pipeline for SNP/INDEL calling against a reference transcriptome using the alignment program Burrows-Wheeler Aligner (BWA) [17] and Samtools, a group of alignment manipulation tools [18]. SNP/INDEL data are attractive information because they can be used to determine haplotypes, to identify different isolates within the same species, and/or to infer the effects of non-synonymous mutation on the efficiency of a given protein [20]. Inferring the effect of non-synonymous mutation from SNP/INDEL data however, is entirely dependent on the level of phenotypic characterization of the samples used during the SNP/INDEL calling as well as the level of certainty in annotations of the transcriptome.

2. Methods and Materials:

2.1 Sequencing: All powdery mildew isolates were grown on grape leaves. Fungal tissue was collected using nail polish prior to RNA isolation [5]. A cDNA library of the reference isolate g14 collected from grape hybrid Rosette in Geneva, NY was normalized and sequenced using a 454 GS FLX genomic sequencer (Roche, Inc.). The reference isolate was also sequenced using an Illumina GA HiSeq sequencer with paired-end reads. RNA from 55 additional isolates were barcoded with 5 base-pairs during the creation of the cDNA libraries, allowing for pooled, single-end re-sequencing using an Illumina GA sequencer for comparison with the deep-sequenced reference isolate. The Illumina re-sequencing runs consisted of one run where 7 isolates were sequenced in a single lane and one run where the other 48 isolates were sequenced together. For each of the runs, the FastX-toolkit barcode splitter [25] was used to separate the reads by the 5 base-pair barcode indicating to which isolate the sequencing read belonged.

Screening and trimming of sequencing reads was accomplished using the SeqClean tool [26] developed by the Dana-Farber Cancer Institute at Harvard School of Public Health with contamination defined by NCBI's UniVec database [27], poly (A) tails, low quality regions, and low complexity regions. The 55 isolates to be characterized for polymorphism were obtained from different regions of the United States, 18 isolates from the Northeastern US, 19 isolates from the Southeastern US, 7 isolates from the Central US, 3 isolates from the Western US, and 8 isolates from Chile (Table 1) [2]. As seen in table 1, the reference isolate and 55 additional isolates were obtained from the leaves of 5 *Vitis* species and 12 cultivars of *V. vinifera* and interspecific hybrids [2].

Table 1 Information about the source and sequencing depth of 56 Isolates of *Erysiphe necator* including Reference Strain 'G14'.

Isolate Name	Reads	Source Location	Region	Wild/Cultivated	Source Species
G14 (ref)	(454) - 641601 (Illumina) - 329494032	Geneva, NY	NE	c	hybrid 'Rosette'
ANYP	2355909	Athens, NY	NE	c	hybrid 'Pinard'
Bcomb	1530728	Monterrey Cty, CA	W	c	*V. vinifera* 'Chardonnay'
BenS2	3076003	Graham, NC	SE	c	hybrid 'Seyval Blanc'
BILTCY	2280271	Asheville, NC	SE	c	*V. vinifera* 'Chardonnay'
BLMT1S	3171007	Blood Mountain, GA	SE	w	*V. aestivalis*
BLMT2	2929141	Blood Mountain, GA	SE	w	*V. aestivalis*
CH10	2848566	Chile	N/A	Unknown	*V. vinifera*
CH13	2154049	Chile	N/A	Unknown	*V. vinifera*
CH15	1730490	Chile	N/A	Unknown	*V. vinifera*
CH18	2901042	Chile	N/A	Unknown	*V. vinifera*
CH19	1712158	Chile	N/A	Unknown	*V. vinifera*
CH33	3060507	Chile	N/A	Unknown	*V. vinifera*
CH36	2390968	Chile	N/A	Unknown	*V. vinifera*
CH8	2925124	Chile	N/A	Unknown	*V. vinifera*
DRESDEN2	2698121	Dresden, NY	NE	c	*V. vinifera* 'Chardonnay'
EBRUNJ	1379940	East Brunswick, NJ	NE	w	*V. labrusca*
FCON2	2636391	Fredonia, NY	NE	c	labrusca hybrid 'Concord'
G9	2968103	Geneva, NY	NE	c	hybrid 'Rosette'
GAMER1	1719418	Blairsville, GA	SE	c	*V. vinifera* 'Merlot'
GAMUS	998566	DeSoto Falls, GA	SE	w	*V. rotundifolia*
HIGHLAND	2434631	Highland, NY	NE	w	*V. riparia*
HVLCY	2284503	Highland, NY	NE	c	*V. vinifera* 'Chardonnay'
ITH8	2463313	Lansing, NY	NE	w	*V. riparia*
KAN1	2397932	Lawrence, KS	C	w	*V. riparia*
KSFRAN	1511666	Emporia, KS	C	c	hybrid 'Frontenac'
LICY	2532703	Riverhead, NY	NE	c	*V. vinifera* 'Chardonnay'
LNYM	2443312	Lockport, NY	NE	c	*V. vinifera* 'Merlot'
LNYN	2524487	Lockport, NY	NE	c	labrusca hybrid 'Niagara'
LW1	2140790	Watkins Glen, NY	NE	c	labrusca hybrid 'Niagara'
NCAes6	124649	Pisgah Forest, NC	SE	w	*V. aestivalis*
NCLAB4	3071817	Blue Ridge, NC	SE	w	*V. labrusca*
NY19	2591475	Burdett, NY	NE	c	*V. vinifera* 'Chardonnay'
NY63	2607958	Burdett, NY	NE	c	*V. vinifera* 'Chardonnay'
NY90	2754809	Burdett, NY	NE	c	*V. vinifera* 'Chardonnay'
pcf3t	1544710	Panther Creek falls GA	SE	w	*V. rotundifolia*
PCTHT	2629283	Panther Creek falls GA	SE	w	*V. rotundifolia*
PUMOCNH	2066696	Purdy, MO	C	c	hybrid 'Chambourcin
PumoComb	20331077	Purdy, MO	C	c	hybrid 'Chambourcin
RAYLEN2	2378631	Mocksville, NC	SE	c	*V. vinifera* 'Cabernet Sauvignon'
rhnc3	2550849	Tryon, NC	SE	c	*V. vinifera* 'Chardonnay'
RoAcl	4254181	Hurdle Mills, NC	SE	c	hybrid 'Chardonel'
ROACS	2393708	Hurdle Mills, NC	SE	c	*V. vinifera* 'Cabernet Sauvignon'
RoaMus3	2728482	Hurdle Mills, NC	SE	w	*V. rotundifolia*
ROAMUS4	2398750	Hurdle Mills, NC	SE	c	*V. rotundifolia*
SHNC1	2379342	Pittsboro, NC	SE	c	hybrid 'Chambourcin
SJMONT	2382662	St. James, MO	C	c	hybrid 'Norton'
TXCS1	2871374	Comanche, TX	C	c	*V. vinifera* 'Cabernet Sauvignon'
VA2	2187275	Winchester, VA	SE	c	*V. vinifera* 'Chardonnay'
VA36	2403721	Winchester, VA	SE	c	*V. vinifera* 'Chardonnay'
VA67	2615019	Winchester, VA	SE	c	*V. vinifera* 'Chardonnay'
VA94	2833196	Winchester, VA	SE	c	*V. vinifera* 'Chardonnay'
Watertown	3702769	Watertown, NY	NE	w	*V. riparia*
WEG1	2181257	Watkins Glen, NY	NE	w	*V. aestivalis*
WEG4	3058859	Watkins Glen, NY	NE	w	*V. aestivalis*
WVMONT	2501498	Waverly, MO	C	c	hybrid 'Norton'

2.2 Computational Resources: The following machines were used to perform all computations: 16-core Linux machine with 80 gigabytes of RAM, 24-core Linux machine with 32 gigabytes of RAM, and a 48-core Linux machine with 512 gigabytes of RAM.

2.3 Assembly: Raw 454 sequencing reads in SFF format from the reference isolate were initially converted to the FASTA and QUAL formats needed for assembly using the Python script SFF_extract [28] with the end-clipping option enabled. Multiple assemblies were performed using the MIRA software [14, 15] the default EST specific parameters for 454 data and the following combinations of sequencing reads to determine which resulted in the best assembly: 1) 454 reads alone in a *de novo* assembly; 2) 454 reads combined with Illumina paired-end reads in a hybrid *de novo* assembly; and 3) 454 reads mapped to *Bgh* mRNA sequences. Each assembly was performed using the two quality settings: "normal" and "accurate". Each assembly was also compared via BLASTx (e-value cutoff of $1xe^{-10}$) against the set of *Bgh* proteins and the set of core eukaryotic genes defined for *Saccharomyces cerevisiae* (CEGMA) [29] to assess their relative composition. Trinity assembly software [16] was used to attain an Illumina-only *de novo* assembly of the G14 reference strain. The four Illumina-only assemblies consist of the first 5.75 million pairs of reads with- and without- the Jaccard_clip option and the first 11.5 million pairs of reads with- and without- the Jaccard_clip option. The Jaccard_clip option is designed to minimize fusion transcripts for paired-end RNA-Seq data from compact, gene dense, fungal genomes by clipping contigs apart where there is low pairing support [16].

2.4 Ordering Contigs: Initial ordering of the reference transcriptome contigs was based on the detection of orthologs with *Bgh* proteins, through reciprocal best hits BLAST analysis, using the order of the proteins within the *Bgh* genome [30, 31]. Reciprocal BLAST searches were performed between the reference transcriptome contigs and *Bgh* proteins with the BlastAll

program and the soft filtering option (-F "m S") per the recommendation of a previous ortholog–matching study [30]. A series of Perl programs (Appendix 1) were written for the determination of orthologs between the *E. necator* reference transcriptome and *Bgh* proteins based on the BLAST results. Contigs that were not deemed orthologous with *Bgh* proteins were subsequently ordered in descending fashion by bit score from protein BLASTx against the non-redundant GenBank database at NCBI with a Perl script (Appendix 2). Reference transcriptome contigs with neither orthologs to *Bgh* nor hits to the non-redundant GenBank database were ordered by size in descending fashion via a Perl script (Appendix 3). The ordering of the contigs was undertaken to facilitate viewing of the transcriptome using the genome-browsing software Artemis [19]. The final ordering script (Appendix 4) uses all of the ordering information to create a reference multi-FASTA file (contigs are separate FASTA's), a reference FASTA file (contigs separated by 10 N's), and a reference GFF3 file.

2.5 Annotation: Sequence similarity-based functional annotations were derived from BLAST searches (BLASTx) against the NCBI's non-redundant database of protein sequences, and the SwissProt database of protein sequences with e-value thresholds of $1xe^{-10}$. Blast2GO was used to determine the top-hit species during the NCBI's NR protein BLAST with an e-value threshold of $1xe^{-3}$ [32]. The bioPerl program, bp_search2gff.pl was used to convert the BLAST annotations to the GFF format [33]. A Perl script (Appendix 5) was written to convert BLAST results in the GFF format to GFF3 format for viewing annotations in the context of the reference transcriptome using Artemis [19]. The following annotation files were created from BLAST searches to allow the user to choose the level of annotation wanted: topSwiss.gff3 (the highest scoring hit to SwissProt only), topBG.gff3 (the highest scoring hit to a *Bgh* protein only), allBG.gff3 (all hits to *Bgh* proteins), topViralNR.gff3 (the highest scoring hit to Viral proteins from NR only),

fungalNR.gff3 (all hits to Viral proteins in NR), top5_30NR.gff3 (the top 5 hits to the NR database with an e-value of e^{-30} or less), and topNR.gff3 (all hits to the NR database). Each annotation file in GFF3 format was color-coded to provide a visible distinction between annotations within Artemis. Additionally each BLAST hit was colored based on the bit score and the color assigned to the database from which it was derived.

2.6 SNP/INDEL Calling: SNP and INDEL calling among the 55 isolates of *E. necator* and the reference transcriptome was based on the CBSU pipeline using BWA [17] to align the sequencing reads of the individual isolates to the reference transcriptome separately. The alignments were then combined using the Samtools [18] mpileup program to call SNPs and INDELs based on all of the sequencing reads from the 55 isolates that aligned to any loci of the reference transcriptome. A series of Perl scripts (Appendix 6) were written to filter the resulting polymorphic data in Variant Call Format (VCF) and to convert the polymorphism data to Comma Separated Values (CSV) format and FASTA format for further study. More specifically, the program PGDSpider was used to convert the SNPs in VCF format initially to FASTA format and then the Perl scripts in the appendix were used [34]. Annotation of the reference transcriptome with the SNP/INDEL information for viewing with Artemis was accomplished by writing a Perl script (Appendix 7) capable of converting the CSV files of SNPs and INDELs of the 55 isolates of *E. necator* to the GFF3 format. The filters applied to the polymorphic data include removal of SNP/INDEL calls with overall quality scores and mapping quality scores below 20, calls made with coverage less than 5 (not enough coverage to be significant), calls where the reference sequence or alternative sequence at the potential polymorphic loci is non ATGC (indicative of indecisive base calls), and INDEL calls were there are multiple alternative sequences because the Samtools programs do not report accurate quality for these types of calls

[18]. Another Perl script (Appendix 8) was written and used to categorize SNPs as synonymous or non-synonymous mutations based on their location within the predicted protein-coding regions from BLASTx and BLASTp runs. The BLAST hit with the highest bit score spanning the position of each individual SNP was used to set the frame of the potential protein-coding region. A final Perl script was written to compile all of the annotation and SNP/INDEL information into one easy to use tabular file (Appendix 9).

3. Results:

3.1 Mira Assembly: The vast majority (90%) of the Roche 454 reads for reference isolate G14 were validated by SeqClean analysis, passing in terms of both quality and complexity (Fig. 4). The passed reads were subjected to end trimming based on quality and presence of poly A/T sequence while still being at least 25 bp in length. Of the 66,265 failed reads, the reasons for failure included: shorter than 25 bp before trimming (7%), vector contamination (2%), and shorter than 25 bp after quality and poly A/T trimming (1%). Less than 1% of the failures were due to either low quality or low complexity (Dust) of the sequence. Pre-processing results for the Illumina data sets were similar, with a majority of sequences being of sufficient length, quality, and complexity to proceed with the project using the NGS data obtained (data not shown). Vector contamination was less than 5% of the total reads for all samples combined but for some isolates it was the most prevalent reason sequences were determined to be invalid.

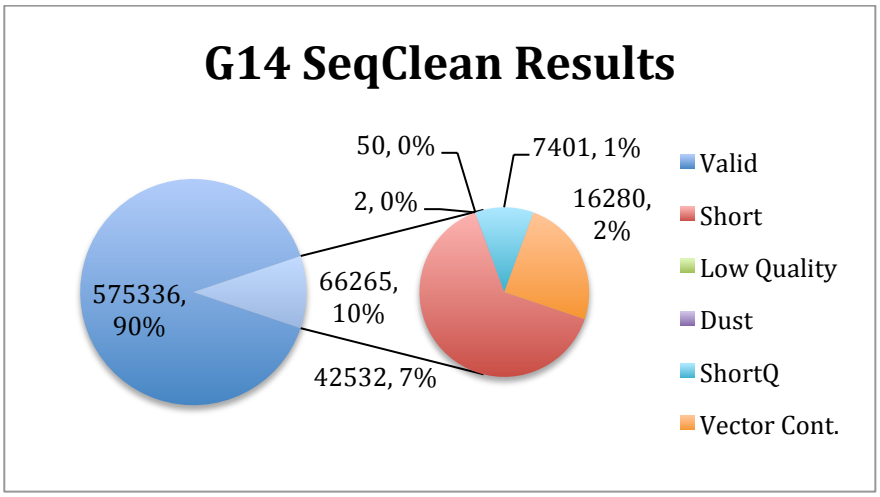

Figure 4 The Results of the SeqClean + UniVec Database Pre-processing of the Roche 454 Generated Reference Data (G14). The secondary pie chart shows the reason each of the 66,265 (10%) sequences was deemed invalid. A sequence was determined to be of Low Quality when more than 3% of bases in a sequence were undetermined (2). Dust refers to the low complexity program and was applied when less than 40nt left unmasked (50).

After pre-processing, the Roche 454 data were assembled into contigs using Mira assembly software [14,15]. The first two assemblies were performed to determine which quality level ("accurate" or "normal") to select and the third assembly was performed to compare mapping to the pathogen related to *E. necator* (*Bgh*) versus the previous two *de novo* assemblies. The three methods produced similar results, with the two *de novo* approaches resulting in nearly identical data (Table 2). For example, the average contig lengths for the *de novo* assemblies are the same (647bp) and only 12bp shorter than the mapping + *de novo* assembly. Similarly, the total sum of the contigs in base pairs varies by less than 1% between *de novo* and mapping assemblies, and the number of contigs varies by less than 3%.

Table 2 Summary Data for the Three Different Mira Assemblies.

A		*B*		*C*	
Mean	647	Mean	647	Mean	659
Standard Error	3	Standard Error	3	Standard Error	3
Median	412	Median	411	Median	417
Mode	254	Mode	255	Mode	250
Standard Deviation	628	Standard Deviation	628	Standard Deviation	637
Range	7,149	Range	7,221	Range	7,149
Minimum	101	Minimum	101	Minimum	101
Maximum	7,250	Maximum	7,322	Maximum	7,250
Sum in bp	25,682,765	Sum in bp	25,682,934	Sum in bp	25,468,671
Contig Count	39,686	Contig Count	39,696	Contig Count	38,642

A – "Accurate" de novo Mira Assembly
B – "Normal" *de novo* Mira Assembly
C - Mapping to the *Blumeria graminis* genome

Thus, additional information was needed to determine which Mira assembly to use for annotation and SNP/INDEL calling. Each of the assemblies was queried against the set of *Bgh* proteins (BLASTx e-value cutoff of $1xe^{-10}$) to ascertain which assembly resulted in the most contigs with significant similarity (Table 3).

Table 3 Summarized Results of BLASTx Analysis of the Three Different Mira Assemblies Queried Against *Bgh* Proteins.

	BLAST Score	*A*	*B*	*C*
E. necator Contigs	>479	1526	1516	1511
	>287	2996	2987	2977
Total	>0	11188	11210	10964
B. graminis Proteins	>479	1506	1499	1500
	>287	2718	2711	2708
Total	>0	4834	4833	4838

A – "Accurate" *de novo* Mira Assembly
B – "Normal" *de novo* Mira Assembly
C - Mapping to the *Blumeria graminis* genome

* "Contigs" refers to the *Erysiphe necator* reference contigs created during the Trinity assembly and "Proteins" refers to the *Blumeria graminis* proteins that were hit from this one BLAST run. The BLAST information summarized in this table is split up into categories of blast score, greater than 479, greater than 287,

and a combined total of all hits with an e-value greater than $1xe^{-10}$ in an effort to visualize relatively long and medium length hits.

A BLAST score reflects the identity between two sequences as opposed to the e-value associated with each hit, which reflects the probability of a hit occurring by random chance. While the statistical information in table 2 for each assembly did not provide an adequate basis for delineating between them, the "accurate" *de novo* assembly had 1,526 contigs matching *Bgh* proteins at a score exceeding 480 while the "normal" *de novo* assembly had 1,516 contigs and the mapping assembly had 1,511 at the same score threshold. Additionally, the "accurate" *de novo* assembly had 2,996 contigs matching *Bgh* proteins with scores greater than 287, while the "normal" *de novo* assembly had 2,987 and the mapping assembly had 2,977. Table 3 also shows the same general pattern of in regards to the *Bgh* proteins that were hit when each assembly was queried against them. The "accurate" de novo assembly had the most long and medium length hit proteins, 1,506 and 2,718 respectively. The information in table 3 shows that the Mira assembled *E. necator* contigs may represent incomplete gene assemblies that could be further assembled with additional sequence, but that a vast majority of the orthologs have been sequenced in *E. necator*. For example, in the "accurate" assembly approximately 11,181 *E. necator* contigs match approximately 4,834 (84.6%)of the 5,717 *Bgh* proteins.

Because the 454 assembly was based on a normalized cDNA library, the number of reads per contig should have a linear relationship with contig length, and the number of reads per kb should be nearly equal across contigs. A strong, positive linear correlation ($R^2 = 0.74$) was observed between the number of reads per contigs and the length of the contigs (Fig. 5). However, several outlier contigs had 10-fold more reads than expected, suggesting that the reads were from homeostatic transcripts or the contigs were the result of merged transcripts.

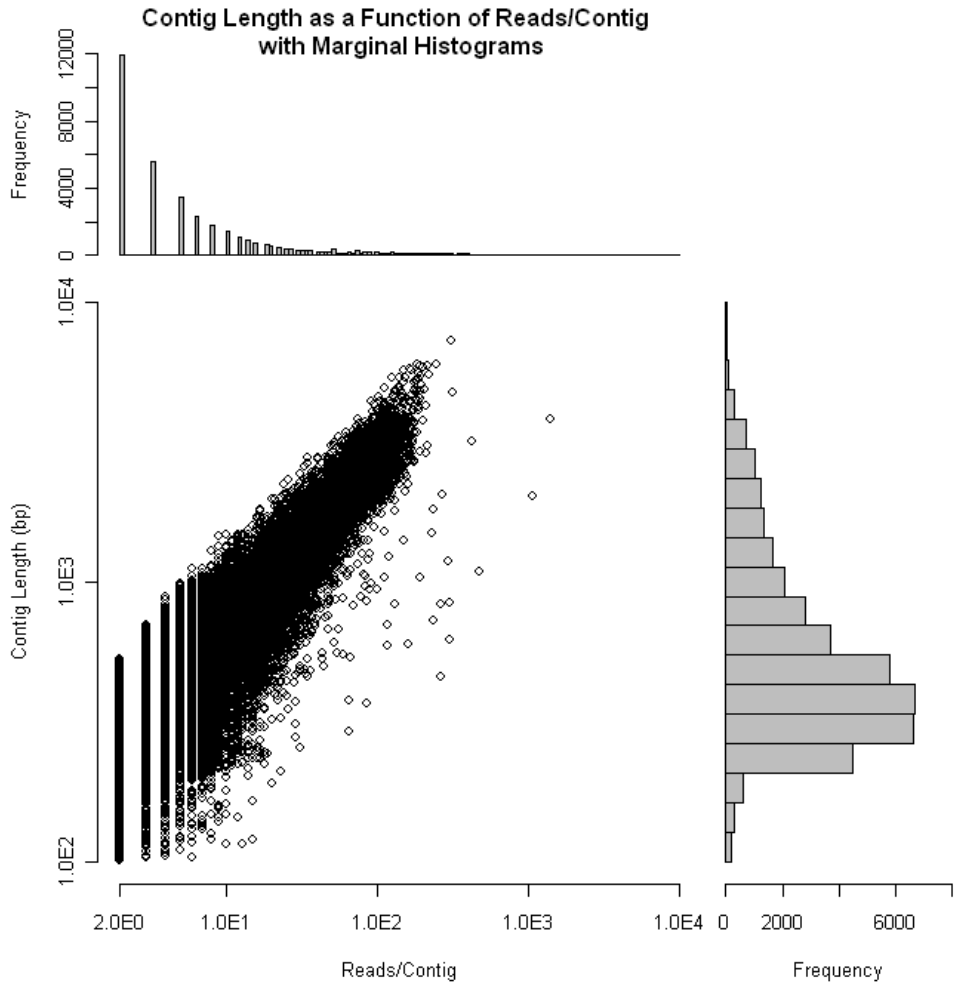

Figure 5 Scatter plot of Comparison between Sequence Length and Number of Reads per Contig of the "Accurate" *de novo* 454 Mira Assembly with Marginal Histograms. The x and y-axis and x-axis of each histogram have a logarithmic scale. The majority of sequences are composed of only 2 reads. Note that the longer contigs incorporate more reads than short contigs.

The assertion that the best assembly should be the data set containing more long (>= 1,000bp) contigs was also checked for validity (Fig. 6). The majority of *E. necator* reference contigs (28,498) do not match any *Bgh* proteins and are shown to reference how few contigs were hit. From figure 6, it is clear that longer *E. necator* contigs have higher scoring BLAST hits to the *Bgh* proteins.

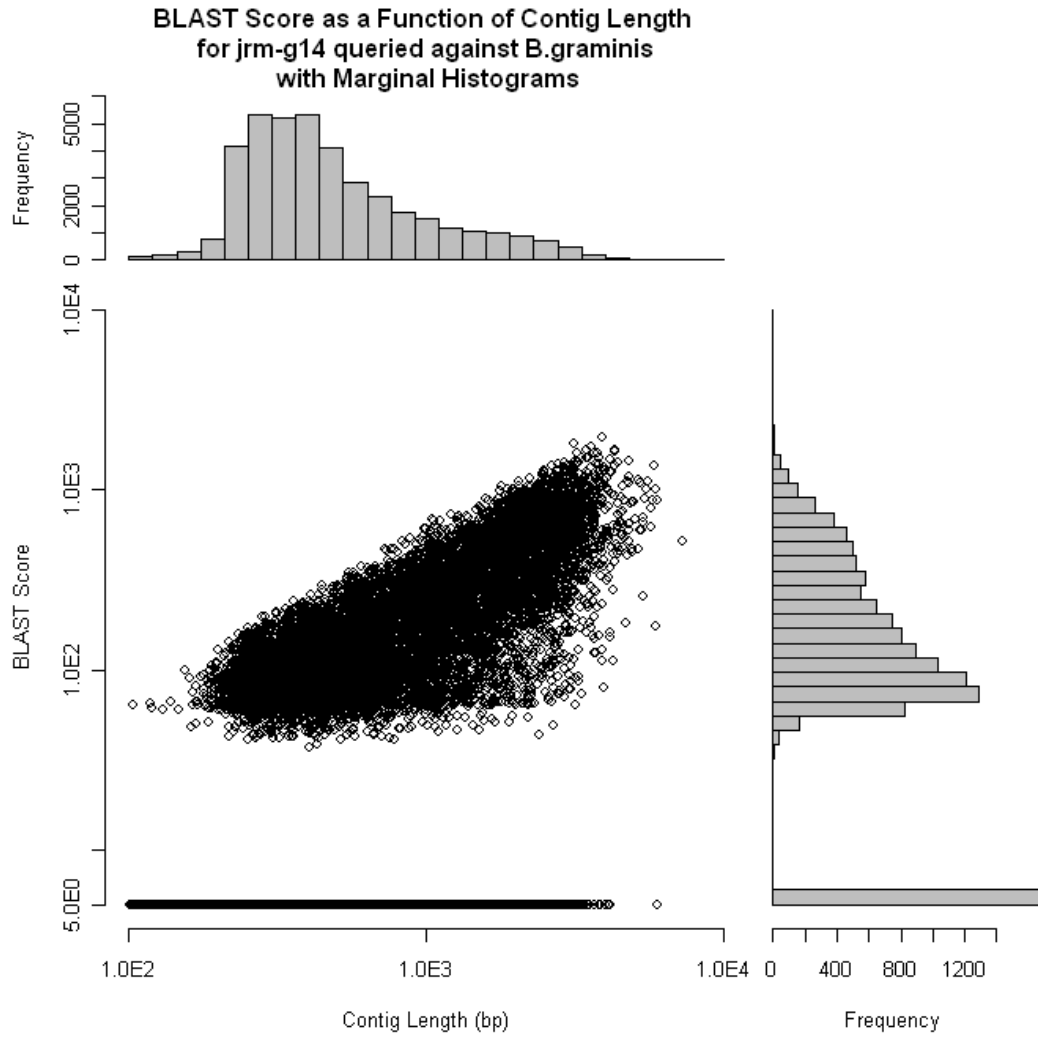

Figure 6 Scatter plot of Contig Length vs. BLAST score of the "Accurate" *de novo* 454 Mira Assembly Queried against the *Blumeria graminis* proteins with Marginal Histograms. The x-axis and y-axis of the scatter plot and the x-axis of each histogram have a logarithmic scale. Note that the general trend of the data set suggests that longer contigs have higher scoring blast hits. Non-matching contigs (28,498) were given a BLAST score of 5 to improve the right hand marginal histogram and show the entire reference data set.

The "accurate" *de novo* Mira assembly was then checked for potential contamination from human sequences from sample processing and plant sequences from fungal culturing. The easiest way to check for this sort of contamination was to BLAST the G14 contigs against the NCBI's chloroplast protein and mitochondrial protein components of the non-redundant database. The

results of the BLAST check for plant contamination with an e-value cutoff of $1xe^{-10}$ can be seen in figure 7.

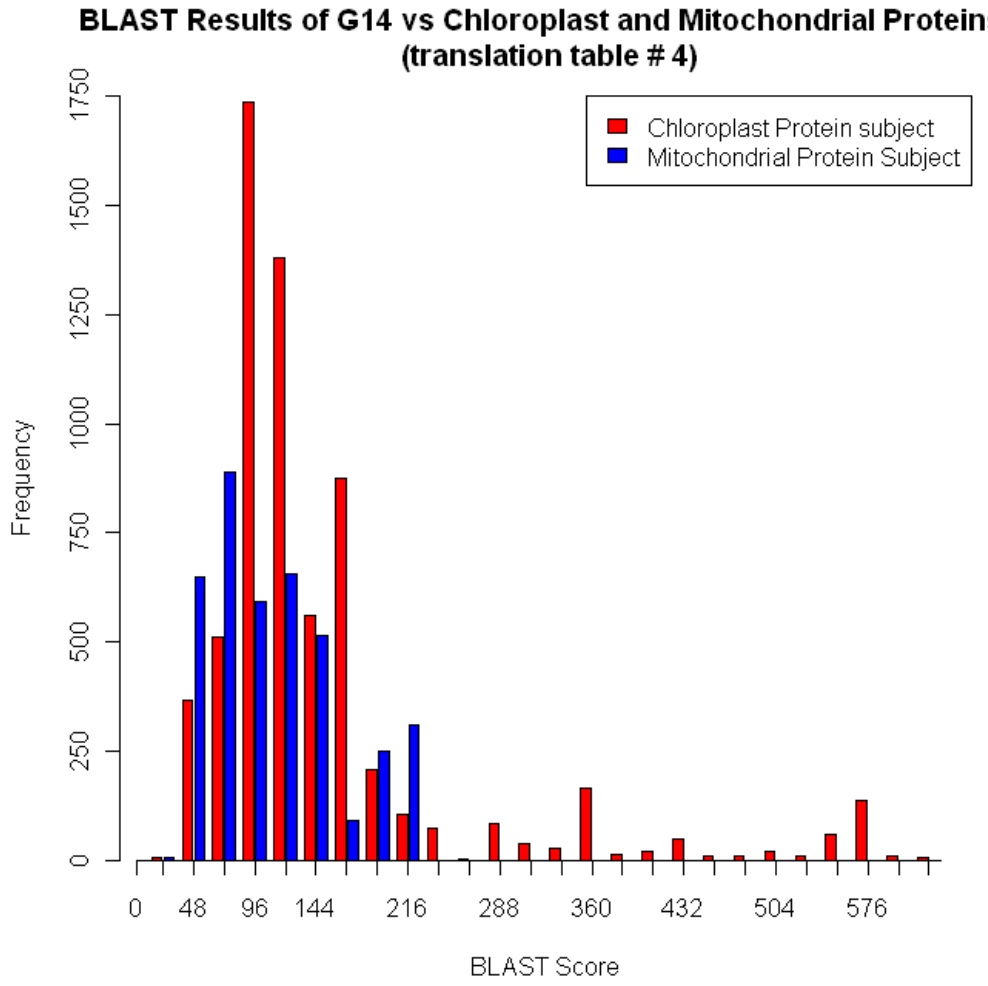

Figure 7 BLAST Results of the G14 Reference Transcriptome Queried Against Chloroplast and Mitochondrial Proteins from the NCBIs NR database. Each BLASTx run had an e-value cutoff of $1xe^{-10}$ and the mitochondrial BLAST used the #4 translation table (mitochondrial specific) for the *Erysiphe necator* translations. Frequency refers to the number of chloroplast and mitochondrial proteins that had similarity to *E. necator* contigs. Many chloroplast and mitochondrial proteins had similarity to a very small number of *E. necator* contigs.

Only 15 *E. necator* contigs had hits to mitochondrial proteins, all of which had higher scoring hits to *Bgh* proteins. Additionally, 114 unique *E. necator* contigs had hits to chloroplast

proteins and only 13 of those had high scoring BLAST hits (>287). Those 13 contigs were also found to have higher scoring hits to *Bgh* proteins. Combined, 129 unique *E. necator* contigs were hit with this test, which is just 0.3% of the total 39,686 contigs. Alternatively, 3,175 unique chloroplast proteins and 2665 unique mitochondrial proteins had hits. Of the total 39,686 *E. necator* reference contigs, few (129) had low scoring similarity with many chloroplast (3,175) and mitochondrial (2,665) proteins. Given that only a small number showed similarity to these two databases suggests that there is little to no contamination of our reference transcriptome from the plant it was taken from.

In additional to the contamination checks performed on the 454 reference transcriptome we also tested it for the amount of core eukaryotic genes that were incorporated from the CEGMA database (Fig. 8). 452 of the 458 CEGMA proteins (99%) had hits to the reference transcriptome at an e-value greater than $1xe^{-10}$. However, only 448 reference contigs matched the 452 proteins, which is 97.8% of the total 458 CEGMA proteins. The majority of CEGMA proteins were present in the initial reference transcriptome assembly for *E. necator* (Fig. 8).

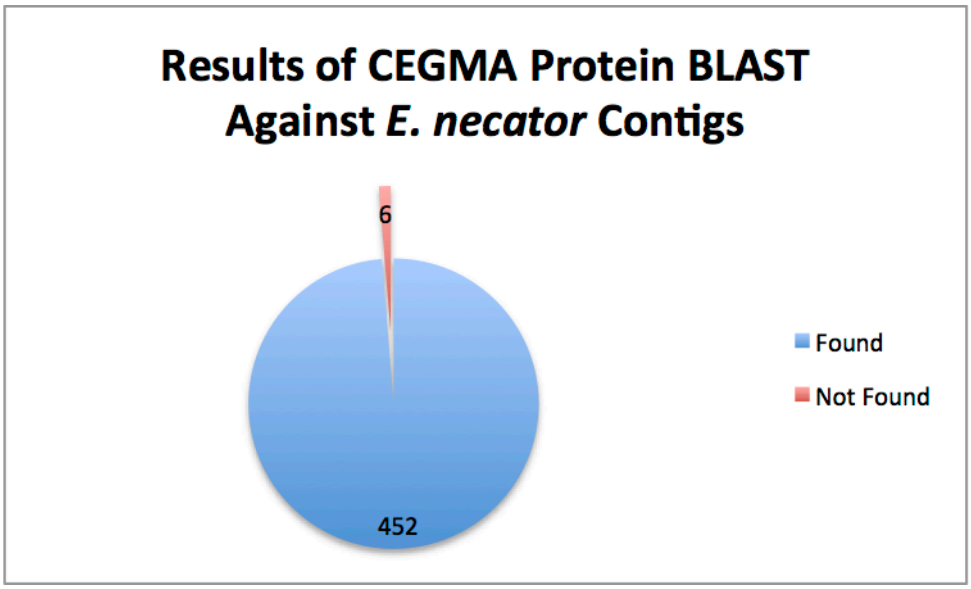

Figure 8 Results of CEGMA Protein BLASTx Against *Erysiphe necator* Contigs using tBLASTn. The CEGMA protein database containing 458 proteins was queried against the translated nucleotides of the *E. necator* reference transcriptome.

3.2 Contig Ordering: Since there is no publicly available whole-genome sequence available for *E. necator*, contig ordering was based on gene order in the publicly available *Bgh* genome [1]. In determining how to perform the reciprocal best hits BLAST ordering, we compared results of: a) translating both the *E. necator* reference sequences and the *Bgh* CDS (tBLASTx), versus b) translating the *E. necator* reference sequences to compare with annotated *Bgh* protein sequences (BLASTx; Fig. 9).

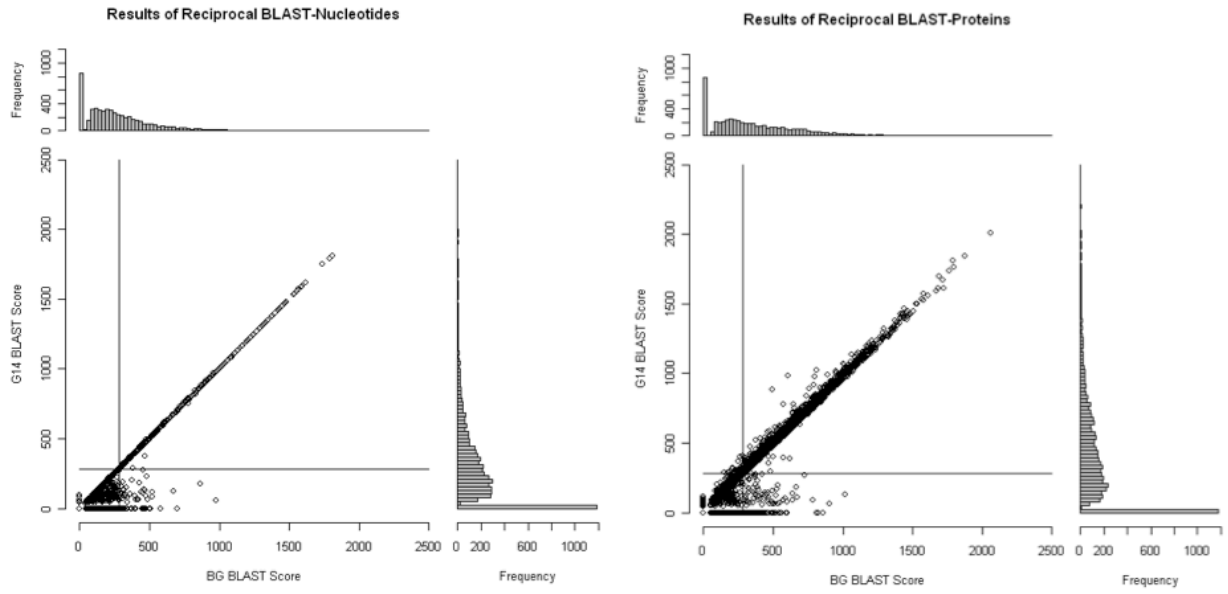

Figure 9 Scatter plots with Marginal Histograms of tBLASTx and BLASTx Reciprocal BLAST Results between the 454 Mira "Accurate" *de novo* and *Blumeria graminis* CDS/Proteins. The x-axis of each figure shows the BLAST scores when *Bgh* sequences were queried against the *Erysiphe necator* reference sequences and the y-axis of each figure shows the BLAST scores when the *E. necator* reference sequences were queried against *Bgh* sequences. The solid lines drawn in each scatter plot demark the BLAST score 288. The figure on the left is a scatter plot of the reciprocal tBLASTx runs using *Bgh* CDS and the figure on the right is a scatter plot of the reciprocal BLASTx runs using *Bgh* proteins.

Using BLASTx with the *Bgh* protein sequences to order the *E. necator* reference contigs provided more hits than tBLASTx (Fig. 9). In the figure on the left there are 2165 BLAST hits with bit scores greater than 287 (medium to long length hits) when *Bgh* coding sequences (CDS) were the query and 2,425 when *E. necator* contigs were the query. In the figure on the right there are 2,898 BLAST hits with bit scores greater than 287 when *Bgh* proteins were the query and 3,289 when *E. necator* contigs were the query.

Ordering the contigs with some semblance of how they might be found within the *E. necator* genome was the next task undertaken. The best way to accomplish this task was to search for orthologs between our transcriptomic contigs and the *Bgh* proteins via reciprocal best-

hit BLAST analysis. The proteins considered problematic by the researchers who annotated the *Bgh* genome, an additional 404 protein sequences, were included in this first stage of the contig ordering process in an attempt to order the most *E. necator* contigs from the available information (Table 4).

Table 4 Results of Reciprocal Best Hit Protein BLASTs Against *Bgh* Proteins + Problematic *Bgh* Proteins

Results of Reciprocal Best Protein BLAST Hits Experiment (with problematic BG)		
Query	B. graminis	E. necator
Subject	E. necator	B. graminis
Query Contigs	6121	39686
Unique Query Sequence Hits	5162	12148
Reciprocal Hits	4522	
Query Sequences without Hits	959	27538
Non-Reciprocal Hits	640	280

* tBLASTn and BLASTx with an e-value cutoff of $1xe^{-10}$ and the parameter -F "m S" [30] were used to generate the data in this table.

The reciprocal best-hit BLAST analysis results shown in table 4 were derived using tBLASTn (*Bgh* queried against *E. necator*) and BLASTx (*E. necator* query and *Bgh*). Of the 6,121 *Bgh* sequences and 39,686 E. necator reference contigs used to perform the reciprocal blast tests, 4522 hit each other with the highest BLAST bit score. Table 4 also shows that only about a third of the *E. necator* contigs, 12,148 out of 39,686 had similarity to *Bgh* proteins an increase of approximately 1,000 contigs had hits with the addition of the problematic *Bgh* proteins. To maximize the contig ordering capability of this test, the 4,522 *E. necator* contigs determined to be orthologous to *Bgh* proteins were each followed by the *E. necator* contigs that hit the same protein to a lesser degree in descending order. By doing this we were able to order the first 12,148 contigs within the reference transcriptome based on the order of predicted proteins within the *Bgh* genome. The order of the remaining 27,538 reference contigs was based on the BLAST hits shown in figure 10.

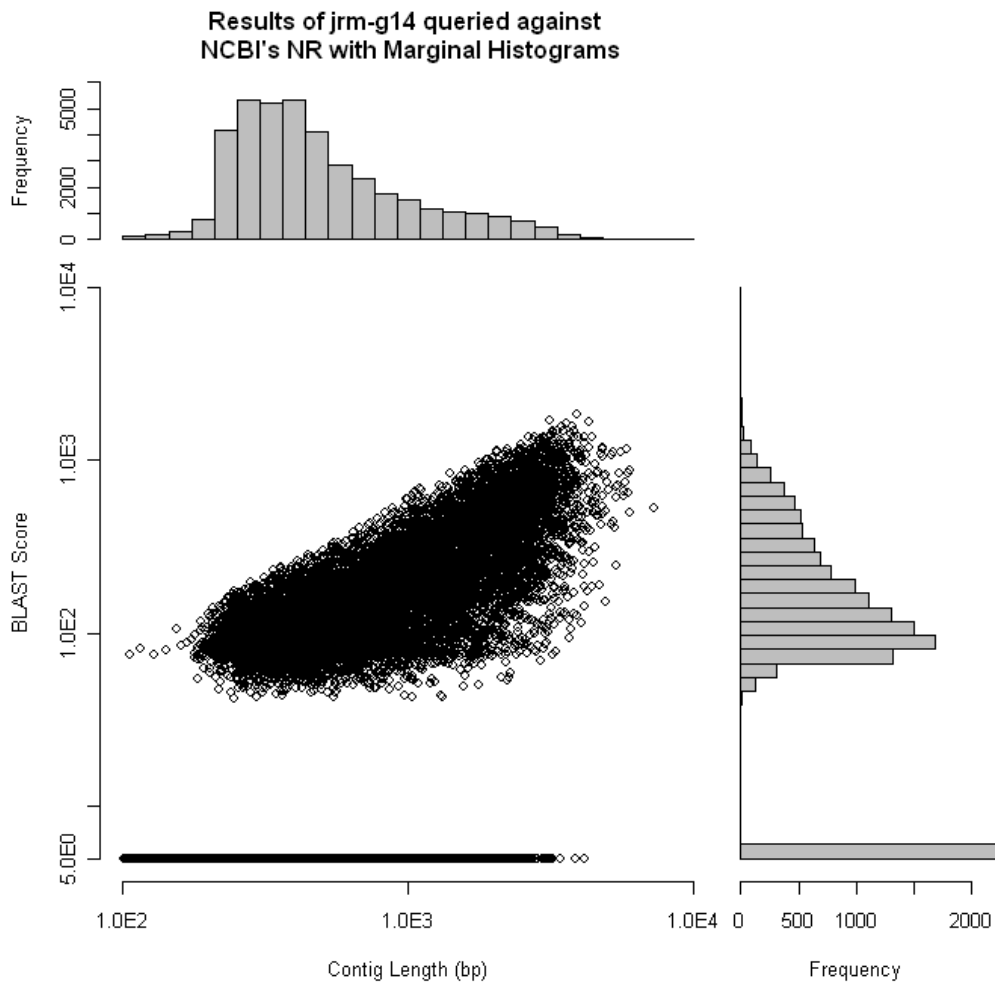

Figure 10 Scatter plot of Contig Length vs. BLAST score of the "Accurate" *de novo* Mira Assembly Queried against the NCBI's NR Protein Database with Marginal Histograms. The data shown are the BLASTX results after an e-value cutoff of $1xe^{-10}$. The majority of *Erysiphe necator* contigs (26,778) did not hit anything within the NR protein database and are shown here with scores of 5 to visualize them in the right marginal histogram while showing the entire reference data set.

There were 13,161 *E. necator* contigs matching 8,452 NR proteins at an e-value less than or equal to $1xe^{-10}$. Longer contigs have higher scoring BLAST hits, similar to figure 6. The actual BLASTx run performed to create figure 9 did not have an e-value cutoff, which resulted in 27,801 of the 39,686 *E. necator* contigs with hits to NR proteins. With the previously ordered contigs removed from this pool, we were able to order an additional 15,728 reference contigs

from highest to lowest BLAST score based on the NR protein database BLAST. The Blast2GO program was used to determine which species within NR had proteins with similarity to the assembled *E. necator* contigs (Fig. 11) [32]. The majority of the top-hit species from the Blast2GO BLAST annotation were fungal ascomycetes, most notably *Botryotinia fuckeliana* and *Sclerotinia sclerotiorum*, which are the most closely related to powdery mildews. The host plant of *E. necator* was also one of the top hit species but only 102 *E. necator* reference contigs were similar to *Vitis vinifera* further indicating that host contamination was minimal.

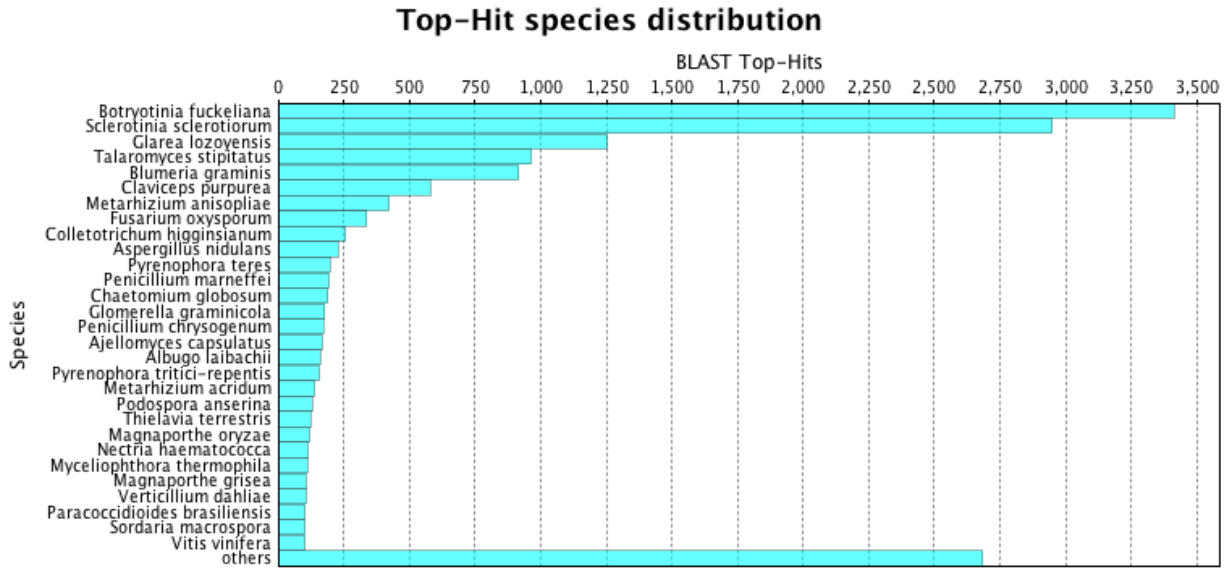

Figure 11 Distribution of the Top-Hit Species when *Erysiphe necator* reference contigs were queried against the NCBI's NR Protein database using the BLAST annotation feature of Blast2GO. An e-value cutoff of $1xe^{-3}$ was used during the BLAST analysis. The majority of top-hit species are fungal *ascomycetes* and there are only 102 *E. necator* contigs with hits to the host species *Vitis vinifera*.

This left 11,810 reference contigs to be ordered be length because they showed no similarity to *Bgh* proteins above an e-value of $1xe^{-10}$ or to any NR proteins at any level. The entire contig-ordering scheme for the 454 Mira assembled reference transcriptome can be seen in figure 12.

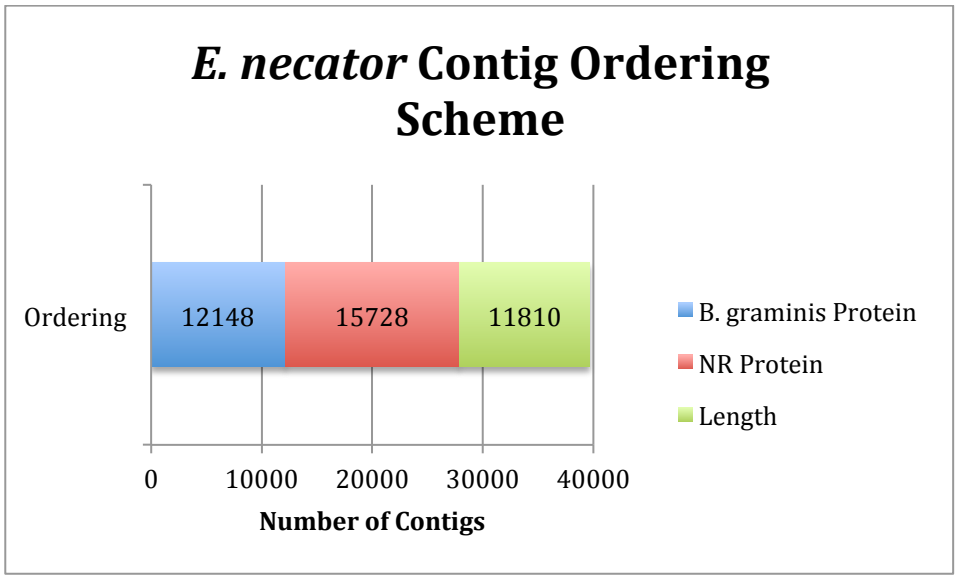

Figure 12 Contig Order of the 454 Mira *de novo* Assembly. All 39,686 reference contigs were ordered based on either BLAST hits to *Blumeria graminis* proteins, or BLAST hits to the NCBI's NR database, or by contig length.

In total, three processes were used to put the *E. necator* reference contigs into descending order of information assigned as shown in figure 12. The first 12,148 reference contigs were ordered based on orthologs to *Bgh* proteins followed by 15,728 contigs ordered based on similarity to NR proteins and the last 11,810 contigs were put into descending order of contig length.

To improve our ability to assign a potential function to the *E. necator* reference contigs they were also queried against the SwissProt database because the annotations in SwissProt are held to a higher standard than those found in the NCBI's NR protein database (Fig. 13).

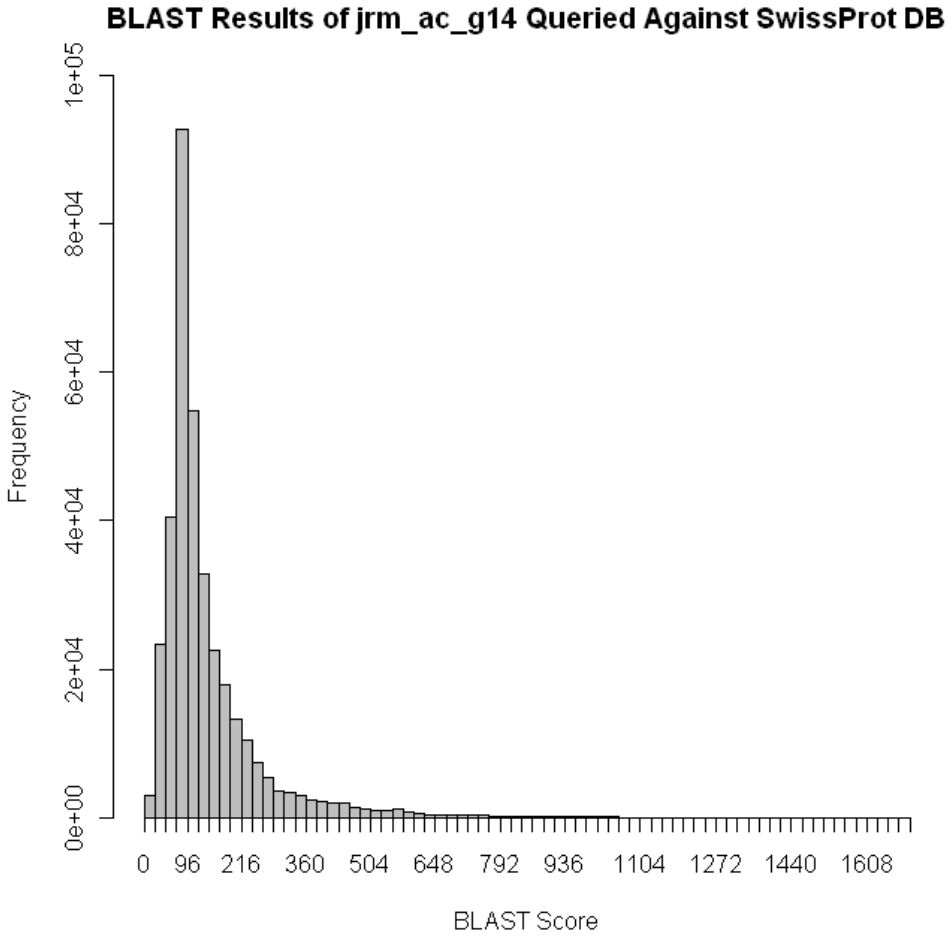

Figure 13 Histogram of BLASTx Results of the *Erysiphe necator* Reference Transcriptome Queried Against SwissProt database. This BLAST run was done with an e-value cutoff of $1xe^{-10}$.

As seen in figure 13, there were only 1,340 reference contigs with similarity to 23,078 SwissProt protein sequences with a BLAST score greater than or equal to 288, meaning that there is significant overlap between the sequences in SwissProt. However, there were significantly more contigs with hits with e-values below $1xe^{-10}$ and the large number of SwissProt proteins hit in our BLAST analysis suggests that there is a high degree of conserved features captured by the *E. necator* reference transcriptome. The highest scoring BLAST hit had a bit score of 1703.

3.3 SNP/INDEL Calling: The first step performed in the SNP/INDEL calling was to extract the sequences belonging to each isolate based on the unique barcode sequence added to each individual isolate (for number of reads per isolate see Table 1). The number of Illumina reads for each isolate used in the SNP/INDEL calling was relatively uniform (approx. 3 million reads) as shown in Table 1. The Illumina sequencing data were put through the same rigorous pre-processing as the 454-reference strain using the SeqClean program with additional screening for UniVec contamination. The cleaned reads for each isolate were then aligned to the *E. necator* reference transcriptome and SNPs and INDELs were called using the pipeline set-up by Cornell's CBSU. The results of the SNP and INDEL calling were then filtered based on quality metrics assigned to each call during the process. The final SNP and INDEL count for each isolate can be seen in figure 14.

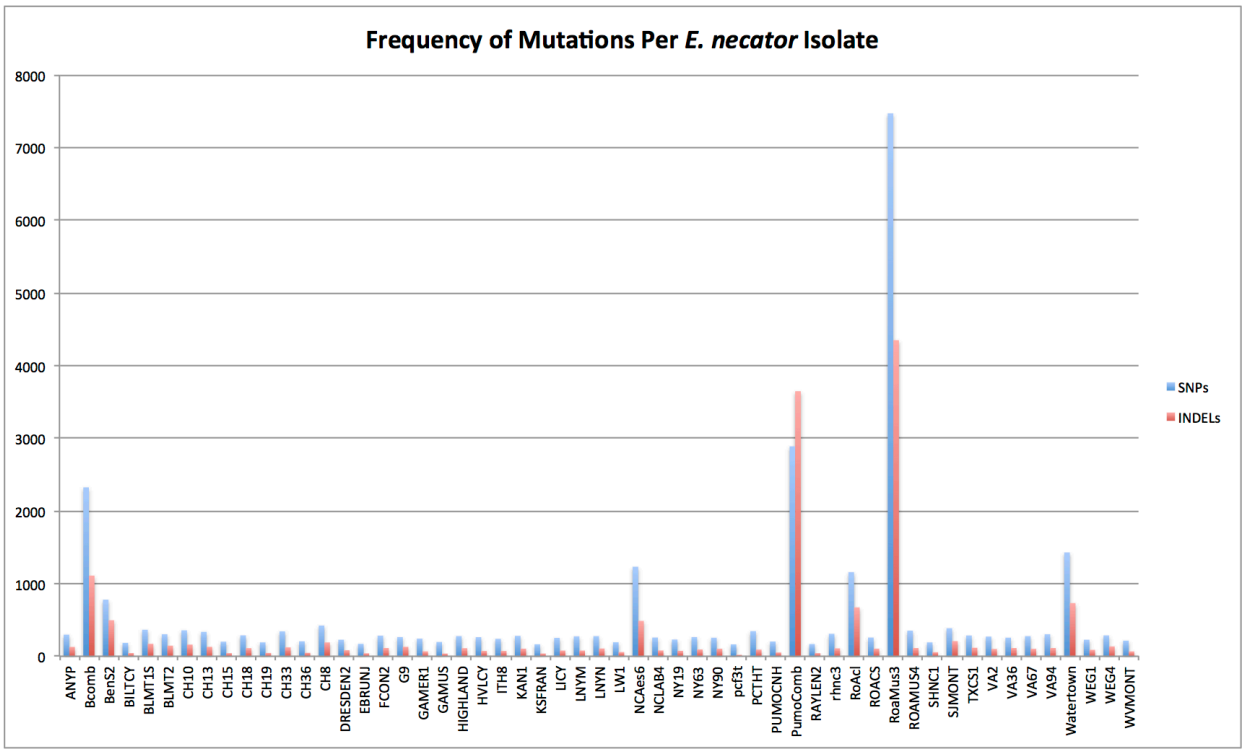

Figure 14 Histogram of SNPs and INDELs per *Erysiphe necator* Isolate. All 55 isolates used to call SNPs/INDELs against the *E. necator* reference transcriptome are shown with the number of SNPs and INDELs associated with each.

The majority of *E. necator* isolates contain comparable numbers of SNPs and INDELs as shown in figure 14. In total, there were 11,605 SNPs called and 5,248 INDELs called against the *E. necator* reference transcriptome. It is important to note that these SNPs and INDELs were spread throughout the *E. necator* reference transcriptome; even the contigs without similarity to *Bgh* or NCBI NR proteins contained SNPs/INDELs. The *E. necator* isolate RoaMus3 contains the most SNPs (7,470) and INDELs (4,349) as compared to the reference transcriptome (potentially an artifact of the barcode splitting procedure). The *E. necator* isolate pcf32 has the lowest number of SNPs (164) and INDELs (21) as compared to the reference transcriptome. The average number of SNPs called per isolate was 544 and the average number of INDELs called per isolate was 292.

Determining what the SNPs might mean in terms of their effect on the function of the mRNA in which they occur is a difficult task. To maintain the utmost confidence in defining a SNPs consequence, only those that were located within any of the *Bgh* protein, NCBI NR protein, or SwissProt protein annotations were assigned a synonymous or non-synonymous classification. The results of this classification process can be seen in figure 15.

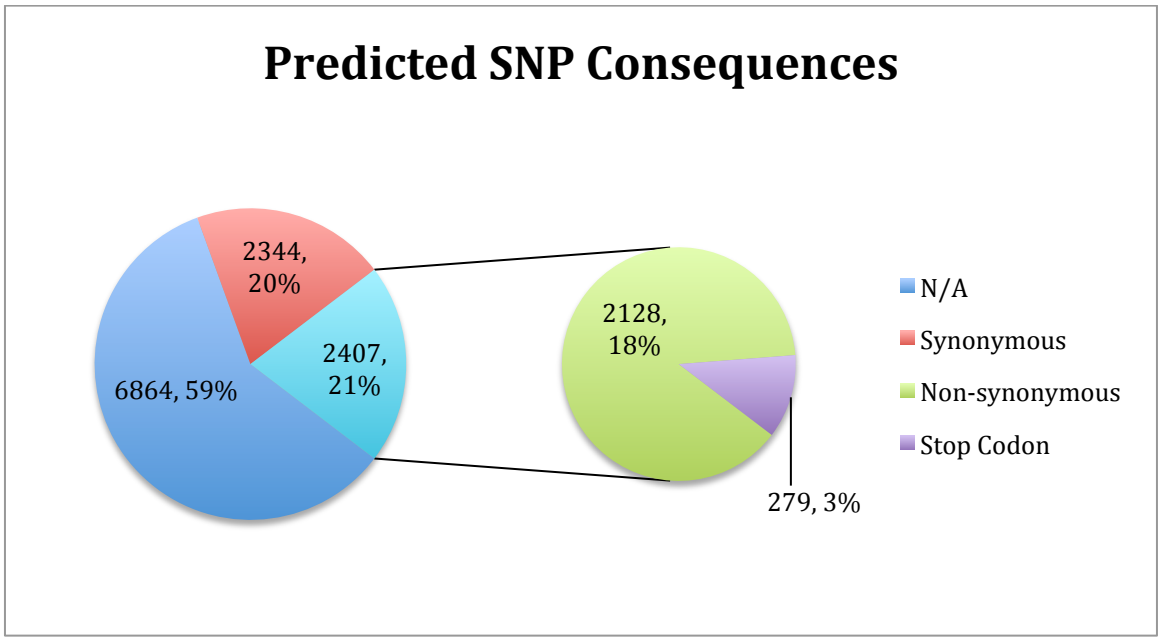

Figure 15 Predicted SNP Consequences Based on Location within BLAST Hit. The number of SNPs predicted to be synonymous and non-synonymous are shown. The secondary piechart shows that 3% of SNPs involved stop codons.

The highest scoring BLAST annotation any given SNP was located in was used to set the frame for translation of the codon to which the SNP belonged. Of the 11,605 SNPs called against the *E. necator* reference transcriptome 6,864 were not within a BLAST annotation, 2,344 were classified as synonymous (encoding the same amino acid as the reference), and 2,407 were classified as non-synonymous (encoding a different amino acid than the reference). Of the 2,407 non-synonymous classified SNPs, 279 of them either dealt with a stop codon changing to a non-stop amino acid or an amino acid changing to a stop codon.

3.4 Annotation: All of the BLAST annotations and SNP/INDEL information gained throughout this project were put into the gff3 format (standard for annotations) for viewing in the Artemis Genome Browser. The information was broken up into components and color coded for easier descrimination between all of the annotations (see Table 5).

Table 6 Annotation File Summaries.

Annotation File	Key	"/subjectDB=" qualifier	E-value cutoff	Results kept for each contig	Color Scheme	Bit Score < 280	Bit Score >= 280 and < 480	Bit Score >= 480
En_transcriptome.gff3	**contig**	**NA**	**NA**	**NA**	**SkyBlue**	135 206 250		
topSwiss.gff3	protein_match	SwissProtTopHit	1x10^-10	Highest Scoring	Purple	153 102 153	153 0 153	102 0 102
topBG.gff3	protein_match	BgTopHit	1x10^-10	Highest Scoring	Red	255 160 122	255 140 0	255 0 0
allBG.gff3	protein_match	BgAllHits	1x10^-10	All	Red	255 160 122	255 140 0	255 0 0
topViralNR.gff3	protein_match	ViralNRtopHit	1x10^-10	Highest Scoring	Yellow	255 255 153	255 255 102	255 255 0
viralNR.gff3	protein_match	ViralNRallHits	1x10^-10	All	Yellow	255 255 153	255 255 102	255 255 0
topFungalNR.gff3	protein_match	FungalNRtopHit	1x10^-30	Highest Scoring	Pink	255 182 193	255 105 180	255 20 147
fungalNR.gff3	protein_match	FungalNRallHits	1x10^-30	All	Pink	255 182 193	255 105 180	255 20 147
top5_30NR.gff3	protein_match	NCBI-NRtop5Hits	1x10^-30	5 Highest Scoring	Green	0 255 102	0 153 0	0 102 0
topNR.gff3	protein_match	NCBI-NRtopHit	NA	Highest Scoring	Green	0 255 102	0 153 0	0 102 0
						Quality Score >= 20 and < 50	Quality Score >= 50 and < 94	Quality Score > 94
En_vars.gff3	SNP	NA	NA	Filtered SNPs	Blue	0 102 153	0 51 102	0 0 102
En_vars.gff3	Indel	NA	NA	Filtered INDELs	Brown	222 184 135	205 133 63	139 69 19

* This table shows the summary information for each annotation file including the coloring scheme that allows the user to easily delineate between annotations within Artemis. The codes within the colored blocks indicate each specific color.

The *E. necator* reference transcriptome contigs were concatenated with 10 N's separating each contig to facilitate browsing the entire transcriptome in one window of the Artemis Genome Browser and annotated such that it showed up with a sky blue coloring as shown in table 6. Nine different BLAST annotation files were created with different coloring to delineate between the bit score of varying degrees of similarity. This allows the user to add in or remove information depending on their need. Additionally, a single variant annotation file containing both SNP and INDEL calls (colored differently) was also created.

An example of the landscape created in Artemis visualizing this information can be seen in figure 16. The Artemis genome browser allows one to scroll through the entire *E. necator* transcriptome and all BLAST and SNP/INDEL annotations associated with each contig. The user can also display the start and stop codons for all six frames of translation and perform searches for key words within the annotations.

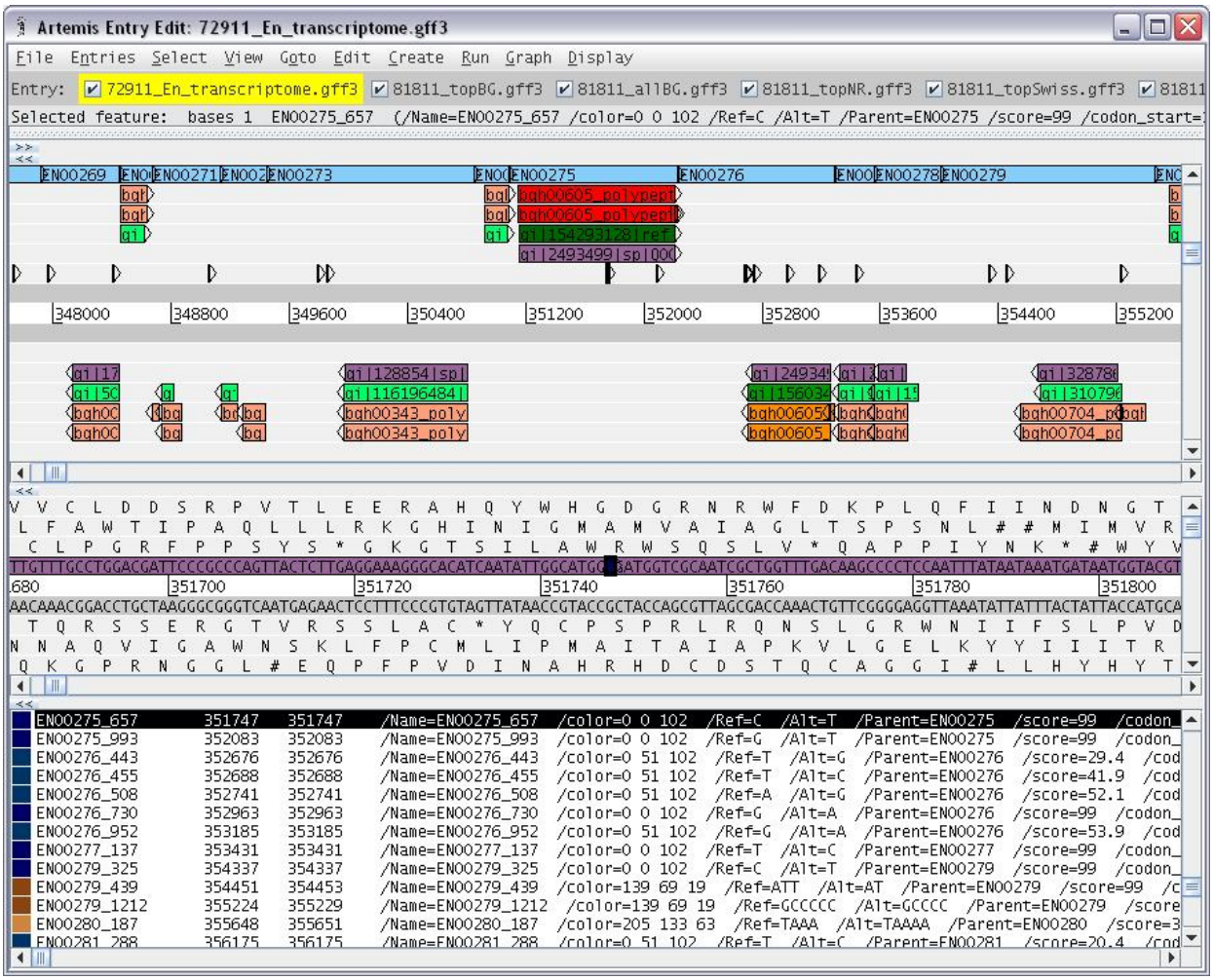

Figure 16 Example of Compiled Information about Reference *Erysiphe necator* Isolate (G14) and SNPs/INDELs Using the Artemis Genome Browser. The annotations loaded into Artemis at in this figure include: En_transcriptome.gff3, topBG.gff3, allBG.gff3, topNR.gff3, topSwiss.gff3, and En_vars.gff3. This allows the user to scroll through the compiled BLAST annotation information and SNP/INDEL information in one window.

3.5 Trinity Assembly: The number of Illumina reads (approx. 330 million) for the reference strain G14 vastly outnumbers the amount of 454 reads (approx. 0.6 million) as shown in table 1. Hybrid assembly of the 454 and Illumina data in Mira failed due to the size of the datasets and computational expense (in terms of time and resources). Therefore, we decided to perform *de novo* assembly of the Illumina data alone using the Trinity software package [16]. The full

Illumina data set (~80Gb) alone was still too large for assembly via Trinity even on a 48-core Linux machine with 512Gb of RAM and five days of run-time. Therefore, two fractions (the first 5.75 million and the first 11.5 million paired reads) of SeqClean + UniVec processed data were assembled, with and without the Jaccard_clip option (recommended for gene-dense fungal genomes), to ascertain the validity of an Illumina only assembly versus a 454 only assembly (Table 6).

Table 6 Summary Data Associated with the Four Different Trinity Assemblies.

A	
Mean	588
Standard Error	2
Median	454
Mode	201
Standard Deviation	413
Range	4,939
Minimum	201
Maximum	5,140
Sum in bp	35,251,906
Contig Count	59,932

B	
Mean	1,103
Standard Error	5
Median	627
Mode	201
Standard Deviation	1,093
Range	11,530
Minimum	201
Maximum	11,731
Sum in bp	55,963,925
Contig Count	50,718

C	
Mean	655
Standard Error	2
Median	517
Mode	276
Standard Deviation	458
Range	5,645
Minimum	201
Maximum	5,846
Sum in bp	42,921,421
Contig Count	65,536

D	
Mean	1,407
Standard Error	6
Median	842
Mode	208
Standard Deviation	1,423
Range	15,730
Minimum	201
Maximum	15,931
Sum in bp	92,218,701
Contig Count	65,536

A - Trinity Assembly of 5.75M paired-end Illumina reads with Jaccard_clip
B - Trinity Assembly of 5.75M paired-end Illumina reads without Jaccard_clip
C - Trinity Assembly of 11.5M paired-end Illumina reads with Jaccard_clip
D - Trinity Assembly of 11.5M paired-end Illumina reads without Jaccard_clip

Table 6 shows that the greatest mean contig length (1,407) and combined sum of the contig lengths in base-pairs (~92M) resulted from the assembly of 11.5 million paired-end reads of Illumina data from the reference strain G14 without the Jaccard_clip option enabled (D). In contrast, the mean contig length (655) and total sum in base-pairs (~43M) of the 11.5 million paired-end reads of Illumina data from the reference strain G14 with the Jaccard_clip option enabled (C) are significantly less. The same pattern is evident when less Illumina reads were assembled. The mean contig length (1,103) and sum in base-pairs (~56M) of the first 5.75 million paired-end reads without the Jaccard_clip option (B) are both greater than with the mean contig length (588) and sum in base-pairs (~35M) with the Jaccard_clip option enabled (A).

As with the 454 Assembly comparisons, additional information is necessary to properly determine which Trinity assembly provides the best resulting reference transcriptome. In order to better determine which Illumina only assembly is optimal each assembly was queried (BLASTx) against the *Bgh* proteins data set. The results of this BLAST analysis can be seen in table 7.

Table 7 Summarized Results of BLASTx Runs of the Four Different Trinity Assemblies Queried Against *Bgh* Proteins.

	BLAST Score	*A*	*B*	*C*	*D*
E. necator Contigs	>479	1199	4572	1338	6762
	>287	4107	7963	4617	11564
Total	>0	19984	18830	23219	25891
B. graminis Proteins	>479	836	2004	842	2102
	>287	2304	3251	2327	3305
Total	>0	4870	4882	4868	4886

A - Trinity Assembly of 5.75M paired-end Illumina reads with Jaccard_clip
B - Trinity Assembly of 5.75M paired-end Illumina reads without Jaccard_clip
C - Trinity Assembly of 11.5M paired-end Illumina reads with Jaccard_clip
D - Trinity Assembly of 11.5M paired-end Illumina reads without Jaccard_clip

* "Contigs" refers to the *Erysiphe necator* reference contigs created during the Trinity assembly and "Proteins" refers to the *Blumeria graminis* proteins that were hit from this one BLAST run. The BLAST information summarized in this

table is split into categories of blast score (greater than 479, greater than 287), and a combined total of all hits with an e-value greater than $1xe^{-10}$ in an effort to look at relatively long and medium length hits.

Table 7 shows that greatest number of *E. necator* contigs with hits to *Bgh* proteins scoring greater than 479 (6,762), greater than 287 (11,564), and overall (25,891) was the Trinity assembly of the first 11.5 million paired-end reads of the reference isolate (D). The same assembly (D) also hit the greatest number of *Bgh* proteins with scores greater than 479 (2,102), greater than 287 (3,305), and overall (4,886). The assembly of 11.5M paired-end Illumina reads with the Jaccard_clip option enabled (C) matched fewer *E. necator* contigs (23,219) to fewer *Bgh* proteins (4,868) than when the Jaccard_clip option was disabled (D). Therefore, assembly D from tables 6 and 7 was chosen as the best Trinity *de novo* assembly.

4. Discussion:

Financial and environmental issues concerning the over-use of fungicides to combat grape powdery mildew fungus, caused by *Erysiphe necator*, has spurred applied genomic research focused on this fungus [5]. The purpose of the current research project was to develop and apply transcriptomic resources in the study of *E. necator*, for which there are currently only 19 EST sequences available in GenBank. Development of a transcriptomic reference for an organism is essential for characterization of the functional elements of a genome, and is a useful tool for studying genetic diversity within a species. Most of the publicly available data for Erysiphales are from research done on *Blumeria graminis* f. sp. *hordei* (*Bgh*), barley powdery mildew, which occupies a clade within the Erysiphales but distinct from *E. necator* [2, 3]. The research described here resulted in the identification and annotation of up to 39,686 ESTs, a more than 2000-fold increase over publicly available data. Further, 11,605 SNPs were identified in transcribed sequences, providing a foundation for population genomics and genome-wide

association studies in *E. necator*.

RNA-Seq, a next-generation sequencing technique, offers an efficient means to attain genomic level data for non-model organisms, such as *E. necator* [20]. The sequencing and computational work described here required an investment of near $30,000, about 2% the cost of the *Bgh* genome project (Cadle-Davidson, personal communication). Thus, we demonstrated that a large quantity of useful information could be gathered about an organism's functional genetic content and the diversity within a species without costly whole genome sequencing and assembly.

The first step generally taken after a next-generation sequencing run is to pre-process the data. Many assembly programs like MIRA, integrate this step into the assembly process alleviating the need for the user to undertake this process prior to loading the data into the assembly software [14, 15]. However, when one relinquishes this task to the assembly software, the user also surrenders this step as a check point to evaluate the sequencing run and remove any sequences that may hinder the assembly process. As such, we opted to perform pre-processing using the SeqClean software with the additional screening for known vectors and viral contaminants from the NCBI's UniVec database [26, 27]. Figure 4 shows the results of the pre-processing performed on the Roche 454 data set of the *E. necator* reference isolate (G14) as an example of the type of information used to evaluate the sequencing done during this project. The majority of sequences in each of the three *E. necator* data sets were considered valid and none of the data sets had more than 5% vector contamination. All of the sequencing data after pre-processing with SeqClean was therefore deteremined to be of sufficient quality to move forward.

A number of parameters can be modified in the Mira assembly software, including overall quality, the percent identity necessary for read inclusion into contigs, and the number of

iterations of the assembly process to perform. The summary data for each of the three Mira assemblies (Table 2) showed that the "accurate" and "normal" *de novo* assemblies had the same mean contig length, and the mapping assembly to *Bgh* coding sequences (CDS) had a slightly greater mean contig length. All three assemblies consisted of approximately 25.5M base-pairs (Table 2). Therefore, more information was needed to delineate between the three assemblies in order to choose which assembly provided the most potentially relevant sequences.

To ascertain which assembly provided the most potentially relevant sequences, we chose to analyze reciprocal sequence similarity between each assembly and the *Bgh* protein sequences. *Bgh* is the nearest relative to *E. necator* with complete genome sequence. The use of protein sequences ensured that synonymous nucleotide differences did not bias the results. The BLAST results shown in table 3 were used in our determination that the "accurate" *de novo* assembly was the best Mira assembly of the Roche 454 data because it had more hits with high BLAST scores to the *Bgh* proteins. The proportion of high scoring hits to *Bgh* is an indication of the content of an assembly because long stretches of similarity allow one to make functional annotations with higher confidence. A more accurate assesment of the assemblies would be to look for similarity to all proteins in the NCBI's NR database but the *Bgh* proteins data set is significantly smaller and contains sequences with higher probability of sequence similarity because they are from a closely related organism. Overall, the reference transcriptome ("accurate" *de novo*) contained similarity to 84.6% of the 5,717 *Bgh* proteins, which indicates that we have captured the majority of *E. necator* genes with this reference assembly.

High scoring BLAST hits are useful for making associations between two species with as much confidence as possible and are indicative of sequences that are likely to be found within other public databases for the purpose of functional annotation. A BLAST score is derived from

the percent identity between two sequences such that the score is increased by each matching base and decreased by mismatches and gap-openings and gap-extensions [31]. Therefore, misassembly of reads into contigs resulting in chimeric sequences would not result in a single high scoring BLAST hit because each score is determined by the overlap between one query (*E. necator*) sequence and one subject (*Bgh*) sequence. A chimeric contig would display BLAST scores to multiple subject (*Bgh*) sequences. Repetitive sequences are also not an issue in this instance because without significant coverage of a repetitive region, the length of the match will be shortened (not long enough to give a high BLAST score) in the assembly process. The pre-processing done insures that low complexity reads are removed from the data set. Long contigs (>1000bp) are the result of many assembled reads and are more likely to contain long stretches of similarity to functionally annotated proteins, and therefore present higher BLAST bit scores and lower e-values. We checked that the "accurate" *de novo* Mira assembly contained these long contigs derived from numerous sequencing reads by looking at the reads per contig and subsequent length of each contig (Fig. 5). A strong positive correlation between reads per contig and contig length was found. We also checked the assertion that longer contigs (>1000bp) in general have higher scoring BLAST hit scores to *Bgh* proteins. We found that longer contigs do in fact have longer stretches of similarity than short contigs (<1000bp). Therefore, the basis for determining the "accurate" *de novo* assembly to be the best of the Mira assemblies holds.

Ensuring that a dataset is free of contamination is an important aspect of having confidence in the quality of the data. To validate that no contamination was incorporated into the "accurate" *de novo* assembly, it was checked for similarity to chloroplast and mitochondrial proteins from the NCBI's NR database with BLAST(Fig. 7). The few contigs that showed similarity to either the chloroplast or mitochondrial proteins had higher scoring BLAST hits to

Bgh proteins, which indicates that these contigs are not likely to be the result of contamination. It was determined that the similarity seen to these two potential sources of contamination did not indicate a flawed transcriptome but that the matches were due to sequence similarity between chloroplast/mitochondrial protein sequences and other, unrelated proteins. The reference transcriptome was also checked for containing the core eukaryotic genes from the CEGMA database. Again using BLAST, we found that the reference transcriptome contained similarity to 99% of the CEGMA proteins, which suggests that the reference transcriptome likely contains the vast majority of expressed genes of *E. necator*, at least in the tissues sampled. However, the technique used to obtain samples of *E. necator* for RNA-Seq may not have sampled the haustorium, the only fungal cell that interacts directly with the host plant cell, thereby missing the production site of the RNAs that are translated and secreted. One of the goals of transcriptomic analysis is to identify candidate secreted proteins, as they are likely to be under significant evolutionary pressure to overcome the host response (apoptosis) to their detection in host epidermal cells [3]. Thus, even though 99% of the core genes were detected, some of the most important genes in host-pathogen interactions may have been missed. Future experiments could target transcripts from the haustorium and be used to confirm whether they were captured by this reference transcriptome or not.

Ordering the reference transcriptome to reflect contig positions within the genome was not possible due to the lack of a whole genome sequence for *E. necator*. However, we co-opted the order of genes of closely related *Bgh* that has been fully sequenced and annotated to order contigs by determining the orthologous genes between them [20]. We accomplished this through reciprocal best-hit BLAST analysis [30]. In order to determine which *Bgh* data set (CDS or protein) to use for the reciprocal BLAST searches we ran both and compared them (Fig. 9). The

reciprocal BLAST test using the *Bgh* proteins resulted in more higher scoring matches than when the *Bgh* CDS were used. At this point we decided to add in the set of *Bgh* proteins considered problematic (404) by the researchers who sequenced and manually annotated its genome in order to gain as much ordering power as possible for the reference transcriptome. The reciprocal best-hit BLAST analysis was re-run including the problematic proteins and 4,522 orthologs were found between *E. necator* and *Bgh* (Table 4). To maximize the contig ordering capability of this test, the 4,522 *E. necator* contigs determined to be orthologous to *Bgh* proteins were each followed by the *E. necator* contigs that hit the same protein to a lesser degree in descending order. Gene fragmentation and reduced representation are well-known downfalls of next-generation sequencing assemblies and explain why multiple *E. necator* contigs have regions of similarity (not covering the entire contig sequence) to the *B. graminis* proteins [35]. This allowed us to place a total of 12,148 *E. necator* contigs in a biologically relevant order, an increase of approximately 1,000 contigs with the addition of the problematic *Bgh* proteins. However, this left the remaining 27,538 contigs unordered.

The ordering of the remaining contigs was done based on similarity to the entire NCBI NR database (Fig. 10). There were a total of 27,801 contigs with BLAST hits to the NCBI's NR proteins from a BLAST run without an e-value cutoff. No cutoff was used during this BLAST so that we could maximize the number of contigs ordered even if the matches were in fact random. Removing the 12,148 contigs previously ordered, we were able to order the remaining contigs (15,728) with similarity to the NR proteins in descending order of BLAST score. We also used the NCBI's NR protein BLAST against the *E. necator* reference contigs to determine the top-hit species distribution using Blast2GO (Fig. 11) [32]. The top-hit species distribution shows that the majority of NCBI' NR annotations are from fungal *ascomycetes* (many of which are also

plant pathogens). A very small number of *E. necator* reference contigs (102) had similarity to the host species proteins (*V. vinifera*), which further indicated that host contamination was minimal. This result also gives one confidence that the annotations based on similarity to NCBI NR protein sequences are in fact derived from fungal species and more likely to be biologically relevant. The remaining 11,810 reference contigs without similarity to *Bgh* or NR proteins were put into descending order of length. In essence, the *E. necator* reference transcriptome was put into the order of highest degree of information gathered to lowest (Fig. 12).

The functional annotation of a transcriptome is necessary to understand what protein each transcript (contig) is likely to encode. Functional annotations are generally based on similarity to known protein sequences that have already been functionally annotated with a high-throughput technique. As such, the *E. necator* reference transcriptome was queried (BLASTx) against the NCBI's NR protein database and the SwissProt data base in addition to the annotations already provided from the BLAST search against the *Bgh* protein data set. The *Bgh* protein functional annotations are not publicly available but their similarity to the reference contigs allows one to target those contigs which have a high probability of being vital to the survival of *E. necator*. A total of 27,801 reference contigs were annotated based on similarity to NCBI NR proteins (no e-value cutoff) and 6,659 reference contigs were annotated based on similarity to SwissProt proteins (e-value cutoff $1xe^{-10}$). These annotations were put into the gff3 format, standard for annotaion browsing, for use with the Artemis Genome Browser (Table 6). There are some disadvantages to developing a reference transcriptome data set and using BLAST to annotate it. For example, we have no way of evaluating whether the assembly process produced full-length ESTs or determining the proportion of contigs in the data set that are chimeric (containing sequence from two or more transcripts) without targeted re-sequencing of problematic contigs.

Additionally, the annotations based on BLAST results against the largest public repositories of protein annotations and the *Bgh* protein dataset do not generally span the entire sequence of a contig. Therefore, high scoring BLAST annotations that cover a relatively small portion of a contigs sequence may not provide an accurate determination of a contig's potential function (after translation). The discrepancy in annotation coverage of the contig sequence may be due to the presence of sequences for translation machinery binding. Problems with the reference transcriptome may be resolved through the use of protein prediction software like GeneMark-E [36] and could lead to higher confidence functional annotations from programs like InterProScan [37]. However, this route would likely overlook the contigs that did not show any similarity to known proteins and potentially lead to loss of valuable information because these contigs were shown to be relevant during the SNP/INDEL calling process by having high quality SNPs and INDELs found within them.

Even though the reference transcriptome created during this project may contain flaws such as chimeric or incomplete sequences, we were able to make valid functional annotations to approximately two thirds of the contigs. For example, contig EN00002 (1,018bp) is likely to be a 2-hydroxyacid dehydrogenase, with hits to both the NCBI's NR and SwissProt databases at e-vaules of $4xe^{-86}$ and $1xe^{-69}$ respectively. This contig also had a significant hit to the *Bgh* protein bgh01634_polypeptide at an e-value of $2xe^{-103}$ and no SNPs or INDELs. The contig EN00263 (1,384bp) had significant hits to the *Bgh* protein bgh00694_polypeptide (e-value: $6xe^{-111}$), hypothetical protein BC1G_07261 [*Botryotinia fuckeliana* B05.10] (e-value: $5xe^{-84}$) from the NCBI NR database, and Low-affinity iron/zinc ion transport protein fet4 (e-value: $1xe^{-55}$) from the SwissProt database. Contig EN00263 is therefore most likely an iron/zinc ion transport protein and is an instance where use of the SwissProt database allowed us to make a meaningful

functional annotation when the highest scoring hit to the NCBI NR database was a hypothetical protein. This contig may also be of note because it contains two SNPs (one synonymous and one not within a BLAST hit) when compared to the 55 non-reference *E. necator* isolates. Additionally, contig EN01270 (2,456bp) had similarity to bghP000684000001001 (e-value: 0), alpha-L-rhamnosidase B [*Aspergillus fumigatus* Af293] (e-value: 0) from the NCBI NR database, and no hit to the SwissProt database. Therefore, contig EN01270 is most likely codes for an alpha-L-rhamnosidase B protein and is an instance where the SwissProt database may not have contained a similar protein. This contig may be of particular significance because it contains 6 SNPs when compared to the 55 non-reference *E. necator* isolates. Many of the reference contigs were found to have the highest similarity to hypothetical proteins in the NCBI's NR database. This is unavoidable because of the amount of hypothetical annotations that are allowed into the NR database and a side effect of the rapid growth and availability of large-scale sequencing. There is a significant need for experimental validation and functional determination of proteins to expand the set of well-annotated public available resources like SwissProt.

SNP/INDEL analysis between isolates of a species is an effective way of characterizing the diversity within a population. There were a total of 11,605 SNPs called and 5,248 INDELs called against the *E. necator* reference transcriptome using the Illumina derived sequence data sets from the 55 additional isolates (Fig. 14). The average number of SNPs called per isolate was 544 and the average number of INDELs called per isolate was 292. One isolate, RoaMus3, had significantly greater numbers of SNPs and INDELs than the averages. This is likely due to the barcoding process and the fact that this isolate was not barcoded and all of the sequences that were considered unmatched during barcode splitting process were associated with RoaMus3. As

such, the SNPs and INDELs associated with RoaMus3 from this analysis will not be used to make determinations about the phenotype exhibited by the RoaMus3 isolate in further studies. Any other isolates deviation from these averages is likely due to more reads passing the cleaning process than the majority of isolates allowing for higher quality SNP/INDEL calls. This could be checked by normalizing in regards to the number of reads used to make the SNP/INDEL calls per isolate and the number of SNPs and INDELs called. One should also note that the samples for the 55 additional isolates used for SNP/INDEL calling were not normalized, which could have resulted in fewer polymorphisms called because of uneven coverage within the reference transcriptome.

The determination of whether a SNP is synonymous or non-synonymous in terms of the amino acid codon in which it occurs is important to understanding its potential consequence. We used the *Bgh* protein, NCBI NR protein, and SwissProt protein annotations to predict the frame of the sequence in which a SNP occurred in order to make a determination of the amino acid encoded by reference codon and the codon with the SNP included. Of the 11,605 SNPs called against the reference contigs we found that roughly the same number of SNPs (20-21%) were synonymous and non-synonymous (Fig. 15). SNPs that change the stop codon of a transcript are especially interesting because these polymorphisms have a high probability of disrupting protein function. Of the 11,605 SNPs there were 279 (3%) in this category (Fig. 15). Identification of these polymorphisms will direct researchers to target the contigs containing these types of SNPs and the transcripts they represent for further study. The SNPs and INDELs called during this project will be used in the future for a transcriptome wide association study (TWAS) in combination with phenotypic data to determine which polymorphisms correspond to the phenotypic differences seen amongst the isolates.

With the results of the annotation and SNP/INDEL identification using the 454 reference transcriptome it was clear that there was room for improvement of the reference. Not all of the *Bgh* genes were found within the 454 *E. necator* reference and there are clear signs that some contigs were the result of chimeric assembly and incomplete transcripts were present. In an attempt to improve upon the 454 reference transcriptome the G14 isolate was again sequenced, using the Illumina platform. Ideally, a hybrid assembly of the Illumina and Roche 454 data using Mira would increase the sequencing coverage and provide a higher quality reference. Unfortunately multiple attempts at a hybrid assembly using Mira were unsuccesful due to lack of computational resources and time. The hybrid assembly process involves a time-consuming sequential step and an enormous amount of RAM is necessary (256Gb) to load the entire Illumina data set into memory. Attempts to use fractions of the Illumina reference data were also unsucessful due to time constraints. Because of this we assembled the Illumina data *de novo* using the Trinity sequences assembler.

Trinity has a smaller computational resource footprint than Mira, but the entire Illumina data set (~330M reads) was still too large to attain an assembly in a reasonable amount of time (less than 7 days). To overcome the problems associated with such a large data set and still gain an assembly of the Illumina data in a *de novo* fashion we used the first 5.75M and the first 11.5M paired-end Illumina reads in four assemblies. Four assemblies were necessary to ascertain at which point the addition of more sequencing data was worthwhile and whether the Jaccard_clip option (recommended for fungal data sets) was advantageous (Tables 6 and 7). The Trinity assembly of the 11.5M paired-end Illumina reads without the Jaccard_clip option had the greatest mean contig length (1,407) and had the greatest total number of base-pairs (~92M bp) of

the four assemblies performed. The same assembly also had the greatest number of high scoring BLAST hits to *Bgh* proteins.

From tables 6 and 7 it is clear that the Jaccard_clip option resulted in shorter length contigs that matched fewer *Bgh* proteins than when it was disabled. This may be due to the Jaccard_clip option having a too lax criteria for splitting contigs. Therefore, the Jaccard_clip option does not seem to provided a better assembly with the *E. necator* Illumina data set. The addition of more Illumina reads to the assembly process increased the average contig length, number of contigs, and their sum in base pairs. As such, it appears that 11.5 million reads is probably not the upper limit at which an increased number of reads can add significant value to a given assembly. However, it is advisable to balance the need for additional reads and potentially higher quality assemblies against the time and resources available. The increased contig length could lead to higher quality annotation compared with the Roche 454 only assemblies. However, the longer contig lengths could be a failure to delineate between two separate transcripts (not using Jaccard_clip), which could be catastrophic in terms of assigning potential function to any given contig. Without more information, it is not clear whether the Trinity *de novo* assembly provided a better assembly than the Mira *de novo* assembly. A BLAST search of the 11.5M paired-end read Trinity assembly (without Jaccard_clip) against the NCBI's NR database will likely provide a better means of differentiating between the two assembly programs and sequencing platforms. The greater number of contigs and the total sum of base-pairs of the Illumina reference transcriptome would however make all of the downstream annotation and SNP/INDEL calling analysis take a significantly longer time and use more computational resources. If time and resources are not an issue this type of assembly may be worthwhile to researchers because it is more likely to provide a robust and all encompassing transcriptomic

reference.

5. Conclusion:

The Roche 454 derived reference transcriptome created for *Erysiphe necator* and the functional annotation and polymorphic characterization performed during this project will assist researchers in understanding which genes are responsible for the differences between different *E. necator* isolates and may allow for increased efficacy of attempts to control this fungus and its infection of grape vines. In total, we made over 50,000 annotations and called 11,605 SNPs and 5,248 INDELs against the 39,686 contigs that make up this transcriptome. These data sets will be made publicly available and be used to further study *E. necator* and other Erysiphales.

We have successfully shown that RNA-Seq is a viable option for researchers of non-model organisms to obtain large quantities of useful information at a relatively low cost. A lack of whole-genome sequence presents unique problems, however. The unavoidable presence of un-anchored and unordered contigs, potentially chimeric contigs, and incomplete contigs are some of the problems that need to be addressed in order to attain the highest level of confidence in a reference transcriptome. In the future, we would like to determine which sequencing technology (Roche 454 or Illumina) provides data allowing for the best de novo assembly and develop criteria for the removal of contigs that are unlikely to biologically relevant. The reference transcriptome can be used to create markers that will allow researchers to target specific *E. necator* genes for further study. Additionally, the SNP and INDEL information will be used in the future to make phenotypic associations, through a TWAS, amongst the isolates used in this study. The reference transcriptome will also most likely be run through gene prediction software and annotated using protein domain prediction software.

6. Works Cited

1. Spanu PD, et al. (2010) Genome Expansion and Gene Loss in Powdery Mildew Fungi Reveal Tradeoffs in Extreme Parasitism. *Science,* 330:1543-1546.

2. Brewer MT and Milgroom M. (2010) Phylogeography and population structure of the grape powdery mildew fungus, *Erysiphe necator,* from diverse *Vitis* species. *BMC Evolutionary Biology*, 10:268.

3. Glawe DA. (2008) The Powdery Mildews: A Review of the World's Most Familiar (Yet Poorly Known) Plant Pathogens. *Annual Review of Phytopathology*, 46:27-51.

4. Saenz GS and Taylor JW. (1999) Phylogeny of the Erysiphales (powdery mildews) inferred from internal transcribed spacer ribo- somal DNA sequences. *Canadian Journal of Botany,* 77:150–168.

5. Cadle-Davidson L, Wakefield L, Seem R, and Gadoury D. (2009) Specific Isolation of RNA from the Grape Powdery Mildew Pathogen *Erysiphe necator,* an Epiphytic, Obligate Parasite. *Journal of Pytopathology*, 158:69-71.

6. Gadoury DM, Cadle-Davidson L, Wilcox W, Dry I, Seem R, and Milgroom M. (2012) Grapevine powdery mildew (Erysiphe necator): a fascinating system for the study of the biology, ecology and epidemiology of an obligate biotroph. *Molecular Plant Pathology*, 13:1–16.

7. Calonnec A, Cartolaro P, Poupot C, Dubourdieu D, and Darriet P. (2004) Effects of *Uncinula necator* on the yield and quality of grapes (*Vitis vinifera)* and wine. *Plant Pathology*, 53:434-445.

8. National Center for Biotechnology Information. (2010). *Erysiphe necator* taxonomy browser. http://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?mode=Info&id=52586, verified on October 23, 2011.

9. Schena M, Shalon D, Davis RW, and Brown PO. (1995) Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science,* 270:467-470.

10. Mardis ER. (2008) Next-generation DNA sequencing methods. *Annual Review Genomics and Human Genetics*, 9:387-402.

11. Marioni JC, Mason CE, Mane SM, Stephens M, Gilad Y. (2008) RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research*, 18:1509-1517.

12. Wang Z, Gerstein M, and Snyder M. (2009) RNA-Seq a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10:57–63.

13. Wang Z, Gerstein M, and Snyder M. (2009) RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics,* 10:57–63.

14. Chevreux B, Pfisterer T, Drescher B, Driesel AJ, Muller WEG, Wetter T, and Suhai S. (2004) Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Research*, 14:1147–1159.

15. MIRA [http://chevreux.org/projects_mira.html].

16. Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, Amit I, Adiconis X, Fan L, Raychowdhury R, Zeng Q, Chen Z, Mauceli E, Hacohen N, Gnirke A, Rhind N, di Palma F, Birren BW, Nusbaum C, Lindblad-Toh K, Friedman N, and Regev A. (2011) Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotechnology*, 29:644-652.

17. Li H and Durbin R. (2000) Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*, 25:1754-1760.

18. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, and 1000 Genome Project Data Processing Subgroup. (2009) The Sequence alignment/map (SAM) format and SAMtools. *Bioinformatics*, 25:2078-2079.

19. Rutherford K, Parkhill J, Crook J, Horsnell T, Rice P, Rajandream MA, and Barrell B. "Artemis: sequence visualization and annotation." *Bioinformatics*, 10:944-945.

20. Parchman T, Geist K, Grahnen J, Benkman C, and Buerkle CA. (2010) Transcriptome sequencing in an ecologically important tree species: assembly, annotation, and marker discovery. *BMC Genomics*, 11:180-196.

21. Garber M, Grabherr M, Guttman M, and Trapnell C. (2011) Computational methods for transcriptome annotation and quantification using RNA-seq. *Nature Methods,* 8:469–477.

22. Zerbino DR and Birney E. (2008) Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Research,* 18:821-829.

23. Robertson G, Schein J, Chiu R, Corbett R, Field M, Jackman SD, Mungall K, Lee S, Okada HM, Qian JQ, Griffith M, Raymond A, Thiessen N, Cezard T, Butterfield YS, Newsome R, Chan SK, She R, Varhol R, Kamoh B, Prabhu AL, Tam A, Zhao Y, Moore RA, Hirst M, Marra MA, Jones SJ, Hoodless PA, and Birol I. (2010) *De novo* assembly and analysis of RNA-seq data. *Nature Methods*, 7:909-912.

24. Wilming L and Harrow J. (2009) "Gene Annotation Methods" in *Bioinformatics: Tools and Applications*; D. Edwards et al. eds.; Springer Scienc+Business Media. Chapter 6, 121-136.

25. FastX-toolkit [http://hannonlab.cshl.edu/fastx_toolkit/index.html].

26. SeqClean [http://compbio.dfci.harvard.edu/tgi/software/].

27. UniVec [http://www.ncbi.nlm.nih.gov/VecScreen/UniVec.html]

28. SFF extract [http://bioinf.comav.upv.es/sff_extract/].

29. Parra G, Branam K, and Korf I. (2007) CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. *Bioinformatics*, 23:1061-1067.

30. Moreno-Hagelsieb G and Latimer K. (2008) Choosing BLAST options for better detection of orthologs as reciprocal best hits. *Bioinformatics*, 24:319-324.

31. Altschul SF, Gish W, Miller W, Myers EW, and Lipman DJ. (1990) Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410.

32. Conesa A, Götz S, Garcia-Gomez JM, Terol J, Talon M, and Robles M. (2005) Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research. *Bioinformatics*, 21: 3674-3676.

33. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, Fuellen G, Gilbert JG, Korf I, Lapp H, Lehväslaiho H, Matsalla C, Mungall CJ, Osborne BI, Pocock MR, Schattner P, Senger M, Stein LD, Stupka E, Wilkinson MD, Birney E. (2002) The Bioperl toolkit: Perl modules for the life sciences. *Genome Research*, 12:1611–1618.

34. Lischer HEL and Excoffier L. (2012) PGDSpider: An automated data conversion tool for connecting population genetics and genomics programs. *Bioinformatics,* 28:298-299.

35. Ye L, Hillier LW, Minx P, Thane N, Locke DP, Martin JC, Chen L, Mitreva M, Miller JR, Haub KV, Dooling DJ, Mardis ER, Wilson RK, Weinstock GM, and Warren WC. (2011) A vertebrate case study of the quality of assemblies derived from next-generation sequences. *Genome Biology*, 12:R31.

36. Borodovsky M and McIninch J. (1993) GeneMark: parallel gene recognition for both DNA strands. *Computers and Chemistry*, 17:123-133.

37. Quevillon E, Silventoinen V, Pillai S, Harte N, Mulder N, Apweiler R and Lopez R. (2005) InterProScan: protein domains identifier. *Nucleic Acids Research*, 33:W116–W120.

7. Appendix 1

7.1. cleanBLAST.pl

```
# Program to remove all secondary blast hits Leaving the single
# highest scoring blast hit for a given query sequence. The input
# must be sorted by the query name followed by the blast score.
#
#       example: sort -k1,1 -k12,12nr blastResults
#
#
#       @author: Jason Myers
#       @date: 06/21/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 1) {

      print "Usage: perl cleanBLAST.pl srt_blastResults\n";
         exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $lastQuery = "";
open INFILE, "$infile", or die $!;
my $outfile = "cl_" . $infile;
open OUTFILE, ">>$outfile", or die $!;
#loop over the input
while(<INFILE>){
         my $string = $_;
         chomp($string);
         my ($query, $subject, $identity, $alLength, $mismatch, $gap,
                  $qStart, $qEnd, $sStart, $send, $eVal, $score)  =
                  split(/\t/, $string, 12);
         if($query eq $lastQuery){
         } else {
                  print OUTFILE $string, "\n";
                  $lastQuery = $query;
         }

}
close(INFILE);
close(OUTFILE);
```

```
# END
exit;
```

7.2. recipHitsEval.pl

```
#
# Determine if reciprocal blasts correspond. The blast
# input files must be cleaned using cleanBLAST.pl (m8).
# The idList file is from a .gff3 file sorted by contig
# in the first column then the proteinID in column 2.
#
#
#       @author: Jason Myers
#       @date: 07/06/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 4) {
                        #       cleaned - sort -k1,1 -k12,12nr file > out
                        #               - cleanBlast.pl out

    print "Usage: perl recipHitsEval.pl cleanedG14 cleanedBG bgIDList outfile\n";
        exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
my $infile2 = $ARGV[2];
my $outfile = $ARGV[3];
# open the first file
open INFILE2, "$infile2", or die $!;
my $iter = 0;
my @bgId = ();
my %contigs;
my %g14Query;
my %g14Score;
my %g14eVal;
my %g14Subject;
my %g141Score;
my %g141eVal;
my %bgSubject;
my %bgScore;
my %bgeVal;
```

```
my $agree = 2; #set to the error code
#loop over the input
while(<INFILE2>){
	my $string = $_;
	chomp($string);
	my ($contigName, $bg)  = split(/\t/, $string, 2);
	if(exists $contigs{$bg}){
		print "Warning: $bg has a duplicated contig.\n";
	} else {
		$contigs{$bg} = $contigName;
		#preserve the order of the id's
		$bgId[$iter] = $bg;
		$iter++;
	}
}
close(INFILE2);
open INFILE, "$infile", or die $!;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my ($query, $subject, $identity, $alLength, $mismatch, $gap,
		$qStart, $qEnd, $sStart, $send, $eVal, $score)  =
		split(/\t/, $string, 12);
	if($score > $g14Score{$subject}){
		$g14Query{$subject} = $query;
		$g14Score{$subject} = $score;
		$g14eVal{$subject} = $eVal;
	}
	if($score > $g141Score{$query}){
		$g14Subject{$query} = $subject;
		$g141Score{$query} = $score;
		$g141eVal{$query} = $eVal;
	}
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
#loop over the input
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	my ($query, $subject, $identity, $alLength, $mismatch, $gap,
		$qStart, $qEnd, $sStart, $send, $eVal, $score)  =
		split(/\t/, $string, 12);
	if($score > $bgScore{$query}) {
		$bgSubject{$query}= $subject;
```

```
                $bgScore{$query} = $score;
                $bgeVal{$query} = $eVal;
        }
}

close(INFILE1);
open OUTFILE, ">>$outfile", or die $!;
print OUTFILE "Contig\ttransId\tG14hit\tG14score\ttransHit\ttransScore\tagree\n";
my $max = $#bgId;
my $bgNoHit;
my $g14NoHit;
for(my $it = 0; $it <= $max; $it++){
        $agree = 2; #error code
        $bgNoHit = 0;
        $g14NoHit = 0;
        my $current = $bgId[$it];
        print OUTFILE $contigs{$current}, "\t", $current, "\t";
        if(!exists $bgSubject{$current}){
                $bgSubject{$current} = "no_hit";
                $bgScore{$current}= 1;
                $bgeVal{$current}= 10;
                $bgNoHit++;
        }
        if(!exists $g14Query{$current}){
                $g14Query{$current} = "no_hit";
                $g14Score{$current}= 1;
                $g14eVal{$current}= 10;
                $g14NoHit++;
        }
        if($g14Subject{$bgSubject{$current}} eq $current ){
                print OUTFILE $bgSubject{$current}, "\t"
                , $g141eVal{$bgSubject{$current}}, "\t"
                , $g141Score{$bgSubject{$current}}, "\t";
                $agree = 0;
        } else {
                print OUTFILE $g14Query{$current}, "\t"
                , $g14eVal{$current}, "\t"
                , $g14Score{$current}, "\t";
                $agree = 10;
        }
        print OUTFILE $bgSubject{$current}, "\t"
                , $bgeVal{$current}, "\t"
                , $bgScore{$current}, "\t";
        if($bgNoHit == 1 && $g14NoHit == 1){
                $agree = 5;
        }
```

```
        print OUTFILE "$agree\n";
}
close(OUTFILE);
# END
exit;
```

7.3. getContigOrder.pl

```
#
#       Determine the order of contigs based on reciprocal
#       blast information from recipHitsEval.pl
#
#       @author: Jason Myers
#       @date: 06/24/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 2) {
                                # resorted means sort the cleaned blast result
                                # using sort -k2,2 -k12,12nr file.txt > outfile.txt
      print "Usage: perl getContigOrder.pl fixed_recirocalInfo cleaned_resorted_G14_blast\n";
        exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
# open the first file
open INFILE, "$infile", or die $!;
# reciprocal info arrays
my @g14 = ();
my @bg= ();
my @codes= ();
# blast hit info arrays
my @blastQu = ();
my @blastSub = ();
my @blastSc = ();
my @blastE = ();
my @blastQuS = ();
my @blastQuE = ();
my @blastSubS = ();
my @blastSubE = ();
```

```
# finished contig list arrays
my @list = ();
my @codeList = ();
# misc
my $iter = 0;
my $flag = 0;
my $it = 0;
my $num = 0;
#loop over the input get the reciprocal info
while(<INFILE>){
	my $string = $_;
	chomp($string);
	#for agree: 0=reciprocalMatch, 5=reciprocalMisMatch, 10=MisMatch
	my ($contigName, $bgID, $g14query, $g14Score, $BGquery, $BGscore, $agree)  =
split(/\t/, $string, 7);
	if($flag == 0){
		$flag++;
	} elsif($agree == 0 || $agree == 10) {
		if($g14query ne "no_hit"){
			$g14[$iter] = $g14query;
			$bg[$iter] = $bgID;
			$codes[$iter] = $agree;
			$iter++;
		}
	}
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
#loop over the input get the blastInfo
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	my ($query, $subject, $identity, $alLength, $mismatch, $gap,
		$qStart, $qEnd, $sStart, $sEnd, $eVal, $score)  =
		split(/\t/, $string, 12);
	$blastQu[$it] = $query;
	$blastSub[$it]  = $subject;
	$blastSc[$it]  = $score;
	$blastE[$it] = $eVal;
	$blastQuS[$it]  = $qStart;
	$blastQuE[$it]  = $qEnd;
	$blastSubS[$it]  = $sStart;
	$blastSubE[$it]  = $sEnd;
	$it++;
}
close(INFILE1);
```

```
#sort
for(my $t = 0; $t <= $#bg ; $t++){
	$list[$num] = $g14[$t];
	$codeList[$num] = $codes[$t];
	$num++;
	for(my $i = 0; $i <= $#blastQu; $i++){
		if($bg[$t] eq $blastSub[$i]){
			if($g14[$t] eq $blastQu[$i]){
			}else{
				$list[$num] = $blastQu[$i];
				#15 signifies that these are not reciprocal best hits
				$codeList[$num] = 15;
				$num++;
			}
		}
	}
}
my $outfile = "contigOrder.txt";
open OUTFILE, ">>$outfile", or die $!;
#header
print OUTFILE "g14\tg14Start\tg14End\tSub\tSubStart\tSubEnd\tEval\tScore\tOrigin\n";
$num = 0;
#print
for(my $t = 0; $t <= $#list ; $t++){
	for(my $i = 0; $i <= $#blastQu; $i++){
		if($list[$t] eq $blastQu[$i]){
			print OUTFILE $blastQu[$i], "\t", $blastQuS[$i], "\t", $blastQuE[$i], "\t",
			$blastSub[$i], "\t", $blastSubS[$i], "\t", $blastSubE[$i], "\t",
			$blastE[$i], "\t", $blastSc[$i], "\t", $codeList[$num], "\n";
			$num++;
		}
	}
}
close(OUTFILE);
# END
exit;
```

8. Appendix 2

8.1. getContigOrderNRBLAST.pl

```
#
#	Determine the order of contigs based on an NR
#	blast result, disregarding already ordered contigs.
#
#	@author: Jason Myers
#	@date: 06/28/2011
```

```
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 2) {
                                        # sorted blastOutput accomplished
                                        # using sort -k12,12nr file.txt > outfile.txt
                                        # may need to be done after cleaning
      print "Usage: perl getContigOrderNRBLAST.pl contigOrder.txt SortedBlastOutput.txt\n";
         exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
# open the first file
open INFILE, "$infile", or die $!;
my $flag = 0;
my %firstHash;
#loop over the input get the reciprocal info
while(<INFILE>){
         my $string = $_;
         chomp($string);
         #for origin: 0=reciprocalMatch, 5=reciprocalMisMatch, 10=MisMatch,
15=secondaryMatchToBG
         my ($g14, $g14Start, $g14End, $BG, $BGstart, $BGend, $eVal, $score,
                  $origin)  = split(/\t/, $string, 9);
         if($flag == 0){
                  $flag++;
         } else{
                  $firstHash{"$g14"} = $origin;
         }
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
my $outfile = "NRcontigOrder.txt";
open OUTFILE, ">>$outfile", or die $!;
#header
print OUTFILE "g14\tg14Start\tg14End\tNR\tNRstart\tNRend\tEval\tScore\tOrigin\n";
#loop over the input get the blastInfo
while(<INFILE1>){
         my $string = $_;
         chomp($string);
         my ($query, $subject, $identity, $alLength, $mismatch, $gap,
```

```
                $qStart, $qEnd, $sStart, $sEnd, $eVal, $score)  =
                split(/\t/, $string, 12);
        if(exists $firstHash{$query}){
        } else {
                print OUTFILE $query, "\t", $qStart, "\t", $qEnd, "\t",
                $subject, "\t", $sStart, "\t", $sEnd, "\t", $eVal, "\t",
                $score, "\t20\n";
        }
}
close(INFILE1);
close(OUTFILE);
# END
exit;
```

9. Appendix 3

9.1. getContigOrderLength.pl

```
#
#       Determine the order of contigs based contig Length
#       disregarding already ordered contigs.
#
#       @author: Jason Myers
#       @date: 06/28/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 2) {
                                # sortedContigLength is accomplished
                                # using sort -k2,2nr file.txt > outfile.txt
                                # COMBINEDcontigOrder means to combine the output
                                # of the getContigOrder.pl and getContigOrderNRBLAST.pl
                                # programs (remove intervening header)
      print "Usage: perl getContigOrderLength.pl COMBINEDcontigOrder
SortedContigLengthFile\n";
        exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
# open the first file
```

```
open INFILE, "$infile", or die $!;
my $flag = 0;
my %firstHash;
#loop over the input get the reciprocal info
while(<INFILE>){
	my $string = $_;
	chomp($string);
	# For origin: 0=reciprocalMatch, 5=reciprocalMisMatch,
	#	10=MisMatch, 15=secondaryMatchToBG, 20=matchToNR
	my ($g14, $g14Start, $g14End, $BG, $BGstart,
		$BGend, $eVal, $score, $origin)  = split(/\t/, $string, 9);
	if($flag == 0){
		$flag++;
	} else{
		$firstHash{"$g14"} = $origin;
	}
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
my $outfile = "LENGTHcontigOrder.txt";
open OUTFILE, ">>$outfile", or die $!;
#header
print OUTFILE "g14\tg14Start\tg14End\tSub\tSubStart\tSubEnd\tEval\tScore\tOrigin\n";
#loop over the input get the blastInfo
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	my ($query, $length)  = split(/\t/, $string, 2);
	if(exists $firstHash{$query}){
	} else {
		print OUTFILE $query, "\t1\t", $length, "\tN/A\tN/A\tN/A\tN/A\tN/A\t25\n";
	}
}
close(INFILE1);
close(OUTFILE);
# END
exit;
```

10. Appendix 4

10.1. finishOrdering.pl

```
#
#	Take in a file with the contig list in order,
#	a list of contig lengths by ID, and a fasta file
#	containing all of the contigs then output a newly
#	ordered fasta file with correct ID's, a fasta file
#	with all the sequences under one ID with oligo-N's
#	seperating the different contigs, and a gff3 file
#	that specifies which contig is which.
#
#
#	@author: Jason Myers
#	@date: 06/28/2011
#
#!/usr/bin/perl
use strict;
use Bio::SeqIO;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 3) {
		#COMBINEDcontigOrderFile: combination of the outputs of the following:
		#		getContigOrder.pl
		#		getContigOrderNRBLAST.pl
		#		getContigOrderLength.pl
		#
		#ContigLengthFile: should be in the form: ContigID[TAB]ContigLength
		#
		#FastaFile: Containing all of the sequences that you wish to order
	print "Usage: perl finishOrdering.pl COMBINEDcontigOrderFile contigLengthFile
FastaFile\n";
	exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
my $infile2 = $ARGV[2];
# open the first file
open INFILE, "$infile", or die $!;
my @ids = ();
my @newNames = ();
my $iter = 0;
my $flag = 0;
```

```
my %hash;
my %lengths;
my $iden = '';
#start at 1 because there is no contig 0
my $i = 1;
# start at 1 because there is no base 0
my $curLength = 1;
# pad with 11 even though there are only 10
# N's seperating the contigs to get the correct
# coordiantes.
my $Npad = 11;
# The number of bases to be printed per line
my $cutoff = 60;
my $carry = '';
my $tempStr = '';
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my ($g14, $g14Start, $g14End, $SUB
	, $SUBStart, $SUBEnd, $eVal, $score
	, $origin) = split(/\t/,$string, 9);
	if($flag == 0 ){
		#skip header
		$flag++;
	} else {
		#store contig ID's
		$ids[$iter] = $g14;
		$iter++;
	}
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
#loop over the input
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	my ($contigName, $contigLength) = split(/\t/,$string, 2);
	$lengths{"$contigName"} = $contigLength;
}
close(INFILE1);
my $inSeq = Bio::SeqIO->new(-file => "$infile2",
				-format => 'Fasta');
my $outSeq = Bio::SeqIO->new(-file => ">g14Ordered.fasta",
				-format => 'Fasta');
#loop over all of the sequences and store them in the hash
```

```
while(my $seq = $inSeq->next_seq() ){
	my $newKey = $seq->primary_id;
	$hash{"$newKey"} = $seq;
}
# loop over the contig ID's array and print out the
# sequences in the correct order
for(my $t = 0; $t <= $#ids; $t++){
		#change the contig name before printing
		$iden = "EN" . sprintf("%05d",$i);
		$newNames[$t] = $iden;
		$hash{$ids[$t]}->display_name($iden);
		$outSeq->write_seq($hash{$ids[$t]});
		$i++;
}
my $outfile = "En_transcriptome.fasta";
open OUTFILE, ">>$outfile" or die $!;
print OUTFILE ">En_transciptome\n";
for(my $t = 0; $t <= $#ids; $t++){
		$tempStr = $carry;
		$tempStr .= $hash{$ids[$t]}->seq();
		$tempStr .= "NNNNNNNNNN";
		my $strLength = length($tempStr);
		my $num = int($strLength / $cutoff);
		if(($strLength % $cutoff) != 0){
			$num++;
		}
		for(my $k = 0; $k < $num; $k++){
			if($k == ($num - 1)){
				$carry = substr($tempStr, $k * $cutoff, $cutoff);
			} else {
				print OUTFILE substr($tempStr, $k * $cutoff, $cutoff), "\n";
			}
		}
}
print OUTFILE $carry, "\n";
close(OUTFILE);
my $outfile2 = "En_transcriptome.gff3";
open OUTFILE2, ">>$outfile2" or die $!;
print OUTFILE2 "##gff-version 3\n";
#loop for the number of contigs and output the gff3 file
for(my $r = 0; $r <= $#newNames; $r++){
	print OUTFILE2 $newNames[$r], "\t";
	print OUTFILE2 "Mira_assembly\t";
	print OUTFILE2 "contig\t";
	print OUTFILE2 $curLength, "\t";
	$curLength += ($lengths{$ids[$r]} - 1);
```

```
        print OUTFILE2 $curLength, "\t";
        $curLength += $Npad;
        print OUTFILE2 "0\t";
        print OUTFILE2 ".\t";
        print OUTFILE2 "0\t";
        print OUTFILE2 "ID=", $newNames[$r], ";Name=", $newNames[$r], ";Alias=",
$ids[$r], ";color=135 206 250\n";
}
# END finishOrdering.pl
exit;
```

11. Appendix 5

11.1. fixGff3fromBlast.pl

```
#
#       This program takes in a file containing information about the
#       start and end of contigs withiin the transcriptome as a whole
#       as well as naming conventions of the contigs, a gff3 file from
#       the bp_search2gff.pl program, a color scheme to be associated
#       with the annotations that varies by shade according to BLAST
#       score. (<279 = light, >279<480 = medium, >479 dark), the blast
#       ouput file to incorporate the e-value information, the name of
#       the DB quereid against for reference purposes, and allows the
#       user to specify the output file name.
#
#       @author: Jason Myers
#       @date: 06/28/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
                        # ContigInfo:
ActualContigName<TAB>start<TAB>end<TAB>OriginalContigName
unless( $numArgs == 6) {
      print "Usage: perl fixGff3fromBlast.pl contigInfo gff3FromBlast\n",
            "\tcolor(green/purple/yellow/red/gray/blue/pink/brown)\n",
            "\tblastOutput subjectDBname outfile\n";
        exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
my $infile2 = $ARGV[3];
```

```
my $dbName = $ARGV[4];
my $outfile = $ARGV[5];
my $colorFlag = 0;
if($ARGV[2] eq "green"){
	$colorFlag++;
} elsif($ARGV[2] eq "purple"){
	$colorFlag = 2;
} elsif($ARGV[2] eq "yellow"){
	$colorFlag = 3;
} elsif($ARGV[2] eq "red"){
	$colorFlag = 4;
} elsif($ARGV[2] eq "gray"){
	$colorFlag = 5;
} elsif($ARGV[2] eq "blue"){
	$colorFlag = 6;
} elsif($ARGV[2] eq "pink"){
	$colorFlag = 7;
} elsif($ARGV[2] eq "brown"){
	$colorFlag = 8;
} else {
	print "Warning: The color you have chosen is not an option.\nExiting...\n";
	exit;
}
# open the first file
open INFILE, "$infile", or die $!;
my $flag = 0;
my %hash;
my %eVals;
my $field1 = "contig";
my $field2 = "start";
my $field3 = "end";
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my ($newName, $start, $end, $oldName) = split(/\t/,$string, 9);
	$hash{"$oldName"}{"$field1"} = $newName;
	$hash{"$oldName"}{"$field2"} = $start;
	$hash{"$oldName"}{"$field3"} = $end;
}
close(INFILE);
open INFILE2, "$infile2", or die $!;
#loop over the input
while(<INFILE2>){
	my $string = $_;
	chomp($string);
```

```
		my ($query, $subject, $iden, $align, $mis,
			$gap, $qs, $qe, $ss, $se, $eval, $score)
			= split(/\t/,$string, 12);
		$eVals{"$query"}{"$subject"}= $eval;
}
open INFILE1, "$infile1", or die $!;
open OUTFILE, ">>$outfile" or die $!;
#loop over the input
while(<INFILE1>){
		my $string = $_;
		chomp($string);
		my ($featName, $source, $type, $s, $e, $score,
			$strand, $phase, $att) = split(/\t/,$string, 9);
		my ($tag, $ref) = split(/=/,$att,2);
		my $color = "";
		if($flag ==0){
			$flag++;
			print OUTFILE $string, "\n";
		} elsif($type eq "match") {
		} else {
			if( $tag eq "iD"){
				$tag = "ID";
			}
			print OUTFILE $featName, "\t", $source, "\t", $type, "\t",
				($s + $hash{$featName}{$field2} - 1), "\t",
				($e + $hash{$featName}{$field2} - 1), "\t",
				$score, "\t", $strand, "\t", $phase, "\t",
				"Name=", $ref, ";",
				$tag, "=", $hash{$featName}{$field1}, ";";
				if($colorFlag == 1){
					if($score > 479){
						$color = "0 102 0";
					} elsif($score > 279){
						$color = "0 153 0";
					} else{
						$color = "0 255 102";
					}
				}elsif($colorFlag == 2){
					if($score > 479){
						$color = "102 0 102";
					} elsif($score > 279){
						$color = "153 0 153";
					} else{
						$color = "153 102 153";
					}
				}elsif($colorFlag == 3){
```

```
        if($score > 479){
                $color = "255 255 0";
        } elsif($score > 279){
                $color = "255 255 102";
        } else{
                $color = "255 255 153";
        }
}elsif($colorFlag == 4){
        if($score > 479){
                $color = "255 0 0";
        } elsif($score > 279){
                $color = "255 140 0";
        } else{
                $color = "255 160 122";
        }
}elsif($colorFlag == 5){
        if($score > 479){
                $color = "238 233 233";
        } elsif($score > 279){
                $color = "205 201 201";
        } else{
                $color = "139 137 137";
        }
}elsif($colorFlag == 6){
        if($score > 479){
                $color = "0 0 102";
        } elsif($score > 279){
                $color = "0 51 102";
        } else{
                $color = "0 102 153";
        }
}elsif($colorFlag == 7){
        if($score > 479){
                $color = "255 20 147";
        } elsif($score > 279){
                $color = "255 105 180";
        } else{
                $color = "255 182 193";
        }
}elsif($colorFlag == 8){
        if($score > 479){
                $color = "139 69 19";
        } elsif($score > 279){
                $color = "205 133 63";
        } else{
                $color = "222 184 135";
```

```
                                }
                        }
                        print OUTFILE "color=", $color, ";subjectDB=", $dbName,
                                ";e-value=", $eVals{$featName}{$ref}, "\n";
        }
}
close(INFILE1);
close(OUTFILE);
# END finishOrdering.pl
exit;
```

12. Appendix 6

12.1. vcfFilter.pl

```
#
#       Filter out variants that do not meet specific criteria
#       from a vcf file.
#
#       @author: Jason Myers
#       @date: 08/04/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 1) {
      print "Usage: perl vcfFilter.pl vcfFile\n";
         exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
# open the first file
open INFILE, "$infile", or die $!;
#set quality limits
my $lowQual = 20;
my $lowDP = 5;
my $highDP = 10000;
my $lowMQ = 20;
my $nonBases = "[BDEFHIJKLMNOPQRSUVXYZ]";
my $outfile = "flt_" . $infile;
open OUTFILE, ">>$outfile" or die $!;
my $outfile1 = "disc_" . $infile;
```

```
open OUTFILE1, ">>$outfile1" or die $!;
my $headerFlag = 0;
my $badFlag = 0;
# initialize record keeping
my $q = 0;
my $ld = 0;
my $hd = 0;
my $lm = 0;
my $br = 0;
my $ba = 0;
my $qC = 0;
my $ldC = 0;
my $hdC = 0;
my $lmC = 0;
my $brC = 0;
my $baC = 0;
my $badCount = 0;
my $goodCount = 0;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	if($headerFlag < 2){
		$headerFlag++;
		# print out the header line
		print OUTFILE $string, "\n";
		if($headerFlag == 1){
			# print out the header line plus header info for the new fields
			print OUTFILE1 $string, "\t",
			"LowQual\tLowMQ\tLowDP\tHighDP\tLowAF\tHighAF\tAmbRef\tAmb
			Alt\n";
		}
	} else {
		# split the line into its various parts
		my ($contigName, $location, $thing, $ref, $alt, $qual,
		$thing1, $stats, $order, $isoInfo) = split(/\t/,$string, 10);
		# split to ge the first letter of the stats field to see if
		# the current variant is an INDEL or a SNP
		my ($first, $right) = split(//,$stats,2);
		# initialize some variables
		my($iden, $readDepth, $alleleFreq, $mapQual);
		# if this variant is an INDEL
		if($first eq 'I'){
			# get the read depth
			my ($junk, $readDep, $aFreq, $con, $perGT, $mapQ, $junk1) =
			split(/;/,$stats,7);
```

```
		($iden, $readDepth) = split(/=/, $readDep, 2);
		if($iden ne "DP") {
			print "INFO field formatted incorrectly: Indel-DP\n";
			exit;
		}
		# get the mapping quality
		($iden, $mapQual) = split(/=/, $mapQ, 2);
		if($iden ne "MQ") {
			print "INFO field formatted incorrectly: Indel-MQ\n";
			exit;
		}
	} else {
	# otherwise this should be a SNP
		# get the read depth
		my ($readDep, $aFreq, $con, $perGT, $mapQ, $junk1) = split(/;/,$stats,6);
		($iden, $readDepth) = split(/=/, $readDep, 2);
		if($iden ne "DP") {
			print "INFO field formatted incorrectly: SNP-DP\n";
			exit;
		}
		# get the mapping quality
		($iden, $mapQual) = split(/=/, $mapQ, 2);
		if($iden ne "MQ") {
			print "INFO field formatted incorrectly: SNP-MQ\n";
			exit;
		}
	}
	#do the record keeping for this line
	if($qual < $lowQual){
		$q++;
		$qC++;
		$badFlag++;
	}
	if($mapQual < $lowMQ){
		$lm++;
		$lmC++;
		$badFlag++;
	}
	if($readDepth < $lowDP){
		$ld++;
		$ldC++;
		$badFlag++;
	}
	if($readDepth > $highDP){
		$hd++;
		$hdC++;
```

```
                $badFlag++;
        }
        if($ref =~ m/$nonBases/){
                $br++;
                $brC++;
                $badFlag++;
        }
        if($alt =~ m/$nonBases/){
                $ba++;
                $baC++;
                $badFlag++;
        }

        # redirect the line depending on whether the current variant passed all of the filter
        # criteria or not
        if($badFlag > 0){
                # print to the discard file
                print OUTFILE1 $string,
"\t",$q,"\t",$lm,"\t",$ld,"\t",$hd,"\t",$br,"\t",$ba,"\n";
                $badCount++;
        } else {
                # print to the flt file
                print OUTFILE $string, "\n";
                $goodCount++;
        }
        # reset the record keeping variables
        $badFlag = 0;
        $q = 0;
        $lm = 0;
        $ld = 0;
        $hd = 0;
        $br = 0;
        $ba = 0;
    }
}
close(INFILE);
close(OUTFILE);
close(OUTFILE1);
        # print out the record keeping information
        print "Number of \"good\" variants: ", $goodCount, "\n";
        print "Number of \"bad\" variants: ", $badCount, "\n";
        print "Number discarded for low QUAL: ", $qC, "\n";
        print "Number discarded for low MQ: ", $lmC, "\n";
        print "Number discarded for low DP: ", $ldC, "\n";
        print "Number discarded for high DP: ", $hdC, "\n";
        print "Number discarded for nonBase REF: ", $brC, "\n";
```

```
		print "Number discarded for nonBase ALT: ", $baC, "\n";
# END
exit;
```

12.2. removeMultiIndels.pl

```
#
#	Filter out INDEL variants with multiple alleles from a vcf Files
#
#	@author: Jason Myers
#	@date: 08/04/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 1) {
	print "Usage: perl RemoveMultiIndels.pl INDELvcfFile\n";
	exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
# open the first file
open INFILE, "$infile", or die $!;
my $lowQual = 20;
my $lowDP = 5;
my $highDP = 10000;
my $lowMQ = 20;
my $nonBases = "[BDEFHIJKLMNOPQRSUVXYZ]";
my $outfile = "flt_" . $infile;
open OUTFILE, ">>$outfile" or die $!;
my $outfile1 = "disc_" . $infile;
open OUTFILE1, ">>$outfile1" or die $!;
my $headerFlag = 0;
my $badFlag = 0;
my $multiBADflag = 0;
my $q = 0;
my $ld = 0;
my $hd = 0;
my $lm = 0;
my $br = 0;
my $ba = 0;
my $qC = 0;
```

```
my $ldC = 0;
my $hdC = 0;
my $lmC = 0;
my $brC = 0;
my $baC = 0;
my $badCount = 0;
my $goodCount = 0;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	if($headerFlag < 12){
		print OUTFILE $string, "\n";
		if($headerFlag == 11){
			print OUTFILE1 $string, "\t",
			"LowQual\tLowMQ\tLowDP\tHighDP\tLowAF\tHighAF\tAmbRef\tAmb
			Alt\n";
		}
		$headerFlag++;
	} else {
		my ($contigName, $location, $thing, $ref, $alt, $qual,
		$thing1, $stats, $order, $isoInfo) = split(/\t/,$string, 10);
		my ($first, $right) = split(//,$stats,2);
		my($iden, $readDepth, $alleleFreq, $mapQual);
		if($first eq 'I'){
			my ($junk, $readDep, $aFreq, $con, $perGT, $mapQ, $junk1) =
			split(/;/,$stats,7);
			($iden, $readDepth) = split(/=/, $readDep, 2);
			if($iden ne "DP") {
				print "INFO field formatted incorrectly: Indel-DP\n";
				exit;
			}
			($iden, $mapQual) = split(/=/, $mapQ, 2);
			if($iden ne "MQ") {
				print "INFO field formatted incorrectly: Indel-MQ\n";
				exit;
			}
			if($alt =~ m/,/){
				$multiBADflag++;
			}
		} else {
			my ($readDep, $aFreq, $con, $perGT, $mapQ, $junk1) = split(/;/,$stats,6);
			($iden, $readDepth) = split(/=/, $readDep, 2);
			if($iden ne "DP") {
				print "INFO field formatted incorrectly: SNP-DP\n";
				exit;
```

```
                }
                ($iden, $mapQual) = split(/=/, $mapQ, 2);
                if($iden ne "MQ") {
                        print "INFO field formatted incorrectly: SNP-MQ\n";
                        exit;
                }
        }
        if($qual < $lowQual){
                $q++;
                $qC++;
                $badFlag++;
        }
        if($mapQual < $lowMQ){
                $lm++;
                $lmC++;
                $badFlag++;
        }
        if($readDepth < $lowDP){
                $ld++;
                $ldC++;
                $badFlag++;
        }
        if($readDepth > $highDP){
                $hd++;
                $hdC++;
                $badFlag++;
        }
        if($ref =~ m/$nonBases/){
                $br++;
                $brC++;
                $badFlag++;
        }
        if($alt =~ m/$nonBases/){
                $ba++;
                $baC++;
                $badFlag++;
        }
        if($badFlag > 0){
                print OUTFILE1 $string,
"\t",$q,"\t",$lm,"\t",$ld,"\t",$hd,"\t",$br,"\t",$ba,"\n";
                $badCount++;
        }elsif($multiBADflag > 0){
                print OUTFILE1 $string,
"\t",$q,"\t",$lm,"\t",$ld,"\t",$hd,"\t",$br,"\t",$ba,"\*\n";
                $badCount++;
        } else {
```

```
                        print OUTFILE $string, "\n";
                        $goodCount++;
                }
                $badFlag = 0;
                $multiBADflag = 0;
                $q = 0;
                $lm = 0;
                $ld = 0;
                $hd = 0;
                $br = 0;
                $ba = 0;
        }
}
close(INFILE);
close(OUTFILE);
close(OUTFILE1);
                print "Number of \"good\" variants: ", $goodCount, "\n";
                print "Number of \"bad\" variants: ", $badCount, "\n";
                print "Number discarded for low QUAL: ", $qC, "\n";
                print "Number discarded for low MQ: ", $lmC, "\n";
                print "Number discarded for low DP: ", $ldC, "\n";
                print "Number discarded for high DP: ", $hdC, "\n";
                print "Number discarded for nonBase REF: ", $brC, "\n";
                print "Number discarded for nonBase ALT: ", $baC, "\n";
# END
exit;
```

12.3. vcf2Fasta.pl

```
#
#       Program to convert a vcf file with bi-allelic polymorphisms
#       to csv and fasta with per isolate and per variant deliniation.
#
#               THE INDEL FIXER
#
#       Uses tCoffee to insert '-' into the reference and alternative
#       indel calls so that they match-up correctly.
#
#
#       @author: Jason Myers
#       @date: 08/11/2011
#
#!/usr/bin/perl
use strict;
use Bio::Tools::Run::Alignment::TCoffee;
```

```
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
                        # flt_flt: means that the file has been filtered by
                        #         qual, mapQual, and non-bases from the ref and alt
                        #-----   then filtered to remove the multiallelic variants
unless( $numArgs == 1) {
      print "Usage: perl vcf2fasta.pl flt_flt_vcfFile >> log.txt 2>&1\n";
         exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
# open the first file
open INFILE, "$infile", or die $!;
my ($outfile, $oldExt) = split(/\./, $infile, 2);
$outfile .= ".csv";
open OUTFILE, ">>$outfile" or die $!;
my $headerFlag = 0;
my @isoNames = ();
my $headingNum = 9; # the number of columns in the header before isolate names
my $isolate = 0;
my $poly = 0;
my @toFactory = ();
my @order = ();
my @data = ();
my $numToAln = 0;
my $missingData = Bio::Seq->new(-seq => "N"); # place holder to be changed to '#' later
                                              # Bio::seq doesn't allow the sequence to be '#'
# header for the isolate name column
print OUTFILE "Isolate";
# Build a tcoffee alignment factory
my @params = ('ktuple' => 2, 'matrix' => 'BLOSUM');
my $factory = Bio::Tools::Run::Alignment::TCoffee->new(@params);
#loop over the input file
while(<INFILE>){
        # get line and house-clean
        my $string = $_;
        chomp($string);
        $numToAln = 0;
        @toFactory = ();
        @order = ();
        $isolate = 0;
        # If your at the head of the file
        if($headerFlag <= 1){
                # if your at the second line of the header
```

```
        if($headerFlag == 1){
                # get the isolate names
                @isoNames = split(/\t/, $string);
                # remove the preceeding column headings
                for(my $i = 0; $i < $headingNum; $i++){
                        shift(@isoNames);
                }
                # push the name of the reference to the front of the
                # array of isolate names
                unshift(@isoNames, "REFERENCE_G14");
        }
        $headerFlag++;
} else {
        # print out a comma to coninue the csv header line
        print OUTFILE ",";
        my ($contigName, $location, $thing, $ref, $alt, $qual,
        $thing1, $stats, $order, @isoInfo) = split(/\t/,$string);
        # print out the Variant name for the header
        print OUTFILE ($contigName . "_" . $location);
        # grab the reference sequence and place it in the array
        # for the multiple alignment
        my $tempSeq = Bio::Seq->new(-seq => "$ref");
        $toFactory[$numToAln] = $tempSeq;
        # mark the fact that there is a sequence for this isolate
        $order[$isolate] = 1;
        $numToAln++;
        $isolate++;
        # loop over the isolate genotype information
        foreach my $iso (@isoInfo){
                # get at the PL info (likelihoods)
                my ($pl, $gt, $gq) = split(/:/, $iso, 3);
                my($pl1, $pl2, $pl3) = split(/,/, $pl, 3);
                # if there were no reads for the current isolate
                # at the current position
                if($pl1 == 0 && $pl2 == 0 && $pl3 == 0){
                        # mark that there is no sequence info
                        $order[$isolate] = 0;
                } else {
                        # if the called genotype is homozygous reference
                        if($gt eq "0/0"){
                                # mark that this is the same as the reference
                                $order[$isolate] = 3;
                        } elsif($gt eq "1/1"){
                                # add the alternative to the alignment
                                $order[$isolate] = 1;
                                $tempSeq = Bio::Seq->new(-seq => "$alt");
```

```
                                        $toFactory[$numToAln] = $tempSeq;
                                        $numToAln++;
                                } else {
                                        # add the reference to the alignment
                                        $order[$isolate] = 2;
                                        $tempSeq = Bio::Seq->new(-seq => "$ref");
                                        $toFactory[$numToAln] = $tempSeq;
                                        $numToAln++;
                                }
                        }
                        $isolate++;
                }
                # get a reference to the array with seq objects to be aligned
                my $toFactory_ref = \@toFactory;
                # perform the alignment using the factory
                my $aln = $factory->align($toFactory_ref);
                # start the alignment iterator at 1 because underlying bioperl calls for it
                my $alnPOS = 1;
                # loop over the order array
                for(my $j = 0; $j < $isolate; $j++){
                        # if no data
                        if($order[$j] == 0){
                                $data[$j][$poly] = $missingData;
                        # if heterozygous
                        } elsif($order[$j] == 2){
                                $data[$j][$poly] = $missingData;
                                $alnPOS++;
                        # if homozygous reference
                        } elsif($order[$j] == 3){
                                $data[$j][$poly] = $aln->get_seq_by_pos(1);
                        # if homozygous alternative
                        } else {
                                $data[$j][$poly] = $aln->get_seq_by_pos($alnPOS);
                                $alnPOS++;
                        }
                }
                $poly++;
        }
}
close(INFILE);
print OUTFILE "\n";
#print out the data in CSV format
for(my $isoIT = 0; $isoIT < $isolate; $isoIT++){
        # printing out the isolate name
        my $tempStr = $isoNames[$isoIT];
        $tempStr =~ s/cl_od3_//;
```

```
		$tempStr =~ s/cl_//;
		$tempStr =~ s/\.bam//;
		print OUTFILE $tempStr;
		# printing out the allele data
		for(my $polIT = 0; $polIT < $poly; $polIT++){
			print OUTFILE ",";
			# replace N's with #'s
			if($data[$isoIT][$polIT]->seq() eq "N"){
				print OUTFILE "#";
			} else {
				print OUTFILE $data[$isoIT][$polIT]->seq();
			}
		}
		print OUTFILE "\n";
}
close(OUTFILE);
my ($outfile1, $oldExt1) = split(/\./, $infile, 2);
$outfile1 .= ".fasta";
open OUTFILE1, ">>$outfile1" or die $!;
my $newLineFlag = 0;
# print out the fasta file
for(my $isoIT = 0; $isoIT < $isolate; $isoIT++){
		#print out the isolate name
		my $tempStr = $isoNames[$isoIT];
		$tempStr =~ s/cl_od3_//;
		$tempStr =~ s/cl_//;
		$tempStr =~ s/\.bam//;
		print OUTFILE1 ">", $tempStr, "\n";
		# print out the data
		for(my $polIT = 0; $polIT < $poly; $polIT++){
			$newLineFlag = 0;
			# replace N's with #'s
			if($data[$isoIT][$polIT]->seq() eq "N"){
				print OUTFILE1 "# ";
			} else {
				print OUTFILE1 $data[$isoIT][$polIT]->seq(), " ";
			}
			#
			if($polIT > 0 && ($polIT % 15) == 0){
				print OUTFILE1 "\n";
				$newLineFlag++;
			}
		}
		if($newLineFlag == 0){
			print OUTFILE1 "\n";
		}
```

```
}
close(OUTFILE1);
# END
exit;
```

12.4. getSNPVars.pl

```
#
#       Create a csv and file from a vcf and fasta
#       of SNPs. The fasta should be generated using
#       PGDSpider2.jar.
#
#       @author: Jason Myers
#       @date: 08/05/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
        # flt_   means that the vcfFile was already run through vcfFilter.pl
unless( $numArgs == 2) {
      print "Usage: perl GetSNPvars.pl flt_vcfFile flt_fasta\n";
        exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
# open the first file
open INFILE, "$infile", or die $!;
open INFILE1, "$infile1", or die $!;
my $outfile = "en_SNP_info.csv";
open OUTFILE, ">>$outfile" or die $!;
my $flag = 0;
my $headerFlag = 0;
my @isoNames = ();
my @varNames = ();
my $varNum = 0;
my $isoNum = 9;
print OUTFILE "Isolate,";
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
```

```
	if($headerFlag == 0 || $headerFlag == 1){
		$headerFlag++;
	} else {
		my ($contigName, $location, $thing, $ref, $alt, $qual,
		$thing1, $stats, $order, $isoInfo) = split(/\t/,$string, 10);
			print OUTFILE ($contigName . "_" . $location), ",";
	}
}
close(INFILE);
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	my (@vars) = split(//,$string);
	if($vars[0] eq ">"){
		my (@head) = split(/\s+/,$string);

		my $tempStr = $head[0];
		$tempStr =~ s/>cl_od3_//;
		$tempStr =~ s/>cl_//;
		$tempStr =~ s/\.bam//;
		print OUTFILE "\n", $tempStr, ",";
	} else {
		$string =~ s/\s+/,/g;
		print OUTFILE $string;
	}
}
close(INFILE1);
print OUTFILE "\n";
close(OUTFILE);
# END
exit;
```

12.5. varFilter.pl

```
#
#
#	Program to convert a vcf file with atleast bi-allelic polymorphisms
#	to csv and fasta with per isolate and per variant deliniation and
#	any variants with either only missing data or reference as the
#	alternative call.
#
#
#	@author: Jason Myers
#	@date: 08/12/2011
#
#!/usr/bin/perl
use strict;
```

```
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 2) {
	print "Usage: perl varFilter.pl VariantCsvFile 1(SNP)|2(INDEL)\n";
	exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $code = $ARGV[1];
my $lineBreak;
my $missingData;
if($code == 1){
	$lineBreak = 35;
	$missingData = "N";
} else {
	$lineBreak  = 15;
	$missingData = "#";
}
# open the first file
open INFILE, "$infile", or die $!;
my $outfile = "flt_" . $infile;
open OUTFILE, ">>$outfile" or die $!;
my $headerFlag = 0;
my @header = ();
my $isolate = 0;
my $poly = 0;
my @order = ();
my @data = ();
#loop over the input file
while(<INFILE>){
	# get line and house-clean
	my $string = $_;
	chomp($string);
	# If your at the head of the file
	if($headerFlag == 0){
		@header = split(/,/, $string);
		$headerFlag++;
	} else {
		my (@curLine) = split(/,/, $string);

		for(my $i = 0; $i <= $#curLine; $i++){
			$data[$isolate][$i] = $curLine[$i];
		}
		$isolate++;
```

```
		}
	}
	close(INFILE);
	my $realFlag;
	$order[0] = 1;
	for(my $polyIT = 1; $polyIT <= $#header; $polyIT++){
		my $curRef = $data[0][$polyIT];
		$realFlag = 0;
		for(my $isoIT = 1; $isoIT < $isolate; $isoIT++){
			if($curRef eq $data[$isoIT][$polyIT] || $data[$isoIT][$polyIT] eq $missingData){
				if($missingData eq "N" && $data[$isoIT][$polyIT] eq $missingData){
					$data[$isoIT][$polyIT] = "#";
				}
			} else {
				$realFlag++;
			}
		}

		if($realFlag > 0){
			$order[$polyIT] = 1;
		} else {
			$order[$polyIT] = 0;
		}
	}
	#print out the csv header
	for(my $polIT = 0; $polIT <= $#order; $polIT++){
		if($order[$polIT] == 1){
			print OUTFILE $header[$polIT], ",";
		}
	}
	print OUTFILE "\n";
	#print out the csv data
	for(my $isoIT = 0; $isoIT < $isolate; $isoIT++){
		for(my $polIT = 0; $polIT <= $#order; $polIT++){
			if($order[$polIT] == 1){
				print OUTFILE $data[$isoIT][$polIT], ",";
			}
		}
		print OUTFILE "\n";
	}
	close(OUTFILE);
	my ($outfile1, $oldExt1) = split(/\./, $outfile, 2);
	$outfile1 .= ".fasta";
	open OUTFILE1, ">>$outfile1" or die $!;
	my $newLineFlag = 0;
	my $numPrinted;
```

```
for(my $isoIT = 0; $isoIT < $isolate; $isoIT++){
	print OUTFILE1 ">", $data[$isoIT][0], "\n";
	$numPrinted = 1;
	for(my $polIT = 1; $polIT <= $#order; $polIT++){
		$newLineFlag = 0;
		if($order[$polIT] == 1){
			print OUTFILE1 $data[$isoIT][$polIT], " ";
			$numPrinted++;
		}
		if(($numPrinted % $lineBreak) == 0){
			print OUTFILE1 "\n";
			$newLineFlag++;
			$numPrinted = 1;
		}
	}
	if($newLineFlag == 0){
		print OUTFILE1 "\n";
	}
}
close(OUTFILE1);
# END
exit;
```

13. Appendix 7

13.1. vcf2gff3.pl

```
#
#	FOR VARIANT CALL FORMAT FILES
#
#	Take a csv header file and a vcf file
#	and create a gff3 file.
#
#
#	@author: Jason Myers
#	@date: 08/12/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
			# combinedCSVheader:	The header lines from the SNP
			#				and INDEL csv files combined
```

```
                        #                       into a single line.
unless( $numArgs == 3) {
      print "Usage: perl vcf2gff3.pl contigInfo combinedCSVheader combinedVCF \n";
         exit;
}
# get a handle on the input file
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
my $infile2 = $ARGV[2];
my $colorFlag = 0;
# open the first file
open INFILE, "$infile", or die $!;
my $flag = 0;
my %hash;
my $field1 = "contig";
my $field2 = "start";
my $field3 = "end";
#loop over the input
while(<INFILE>){
         my $string = $_;
         chomp($string);
         my ($newName, $start, $end, $oldName) = split(/\t/,$string, 4);
         $hash{"$newName"} = $start;
}
close(INFILE);
open INFILE1, "$infile1", or die $!;
my @header = ();
my %varNames;
#loop over the input
while(<INFILE1>){
         my $string = $_;
         chomp($string);
         @header = split(/,/, $string);
         shift(@header);
         foreach my $varName (@header){
                  $varNames{"$varName"} = 0;
         }
}
close(INFILE1);
open INFILE2, "$infile2", or die $!;
my ($fileName, $type) = split(/\./,$infile2, 2);
my $outfile = $fileName . ".gff3";
open OUTFILE, ">>$outfile" or die $!;
print OUTFILE "##gff-version 3\n";
my $polyType = "";
my $color = "";
```

```
my $iCount = 0;
my $sCount = 0;
my $source = "CBSU+JRM";
my $polyName = "";
my $headerFlag = 0;
#loop over the input
while(<INFILE2>){
	my $string = $_;
	chomp($string);
	if($headerFlag <= 1){
		$headerFlag++;
	} else {
		my ($contigName, $location, $thing, $ref, $poly, $qual,
		$thing1, $stats, $order, $isoInfo) = split(/\t/,$string, 10);
		my $tempkey = $contigName . "_" . $location;
		if(exists $varNames{$tempkey}){
			my ($first, $right) = split(//,$stats,2);
			if($first eq 'I'){
				$polyType = "Indel";
				$iCount++;
				$polyName = $polyType . "$iCount";
				if($qual > 94){
					$color = "139 69 19";
				} elsif($qual > 12){
					$color = "205 133 63";
				} else{
					$color = "222 184 135";
				}
			} else  {
				$polyType = "SNP";
				$sCount++;
				$polyName = $polyType . "$sCount";
				if($qual > 94){
					$color = "0 0 102";
				} elsif($qual > 12){
					$color = "0 51 102";
				} else{
					$color = "0 102 153";
				}
			}
			print OUTFILE $polyName, "\t", $source, "\t", $polyType, "\t",
				($location + $hash{$contigName} - 1), "\t",
				($location + $hash{$contigName} - 2 + length($ref)), "\t",
				$qual, "\t+\t0\t",
				"Name=", $tempkey, ";Parent=", $contigName, ";Ref=",
				$ref, ";Alt=", $poly;
```

```
                        print OUTFILE ";color=", $color, "\n";
                }
        }
}
close(INFILE2);
close(OUTFILE);
# END
exit;
```

14. Appendix 8

14.1. correlateSNPLoci.pl

```
#
#       Take in the top hit from BLAST annotations, the reference fasta,
#       and the SNP/INDEL gff3 file and determine if a SNP is within a BLAST
#       annotation. If so, use the highest scoring BLAST annotation to set the
#       frame, call the amino acid for the reference and with the SNP incorporated.
#       Then, determine if the two amino acids are synonymous or non-synonymous.
#
#       @author: Jason Myers
#       @date: 05/23/2012
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
unless( $numArgs == 6) {
      print "Usage: perl correlateSNPLoci.pl topSwissGff3 topNRgff3 topBGgff3 refFasta
snpIndelGff3 outFile\n";
        exit;
}
# get a handle on the input files
my $infile = $ARGV[0];
my $infile1 = $ARGV[1];
my $infile2 = $ARGV[2];
my $infile3 = $ARGV[3];
my $infile4 = $ARGV[4];
my $outfile = $ARGV[5];
# open the first file
open INFILE, "$infile", or die $!;
my %blast1 = ();
my $flag = 0;
```

```
#loop over the input
print "Reading in file 1... \n";
while(<INFILE>){
	my $string = $_;
	chomp($string);
	if($flag ==0){
		$flag++;
	} else {
		my ($featName, $source, $type, $s, $e, $score,$strand, $phase, $att) =
		split(/\t/,$string, 9);
		my ($name, $desc, $parent, $color, $subject, $eVal) = split(/;/,$att, 6);
		$parent =~ s/Parent=//;
		my $curStr = $s . ',' . $e . ',' . $score . ',' . $strand;
		$blast1{ $parent } = $curStr;
	}
}
close(INFILE);
# open the first file
open INFILE1, "$infile1", or die $!;
my %blast2 = ();
my $flag = 0;
#loop over the input
print "Reading in file 2... \n";
while(<INFILE1>){
	my $string = $_;
	chomp($string);
	if($flag ==0){
		$flag++;
	} else {
		my ($featName, $source, $type, $s, $e, $score,$strand, $phase, $att) =
		split(/\t/,$string, 9);
		my ($name, $desc, $parent, $color, $subject, $eVal) = split(/;/,$att, 6);
		$parent =~ s/Parent=//;
		my $curStr = $s . ',' . $e . ',' . $score . ',' . $strand;
		$blast2{ $parent } = $curStr;
	}
}
close(INFILE1);
# open the first file
open INFILE2, "$infile2", or die $!;
my %blast3 = ();
my $flag = 0;
#loop over the input
print "Reading in file 3... \n";
while(<INFILE2>){
	my $string = $_;
```

```
		chomp($string);
		if($flag ==0){
			$flag++;
		} else {
			my ($featName, $source, $type, $s, $e, $score,$strand, $phase, $att) =
			split(/\t/,$string, 9);
			my ($name, $parent, $color, $subject, $eVal) = split(/;/,$att, 5);
			$parent =~ s/Parent=//;
			my $curStr = $s . ',' . $e . ',' . $score . ',' . $strand;
			$blast3{ $parent } = $curStr;
		}
}
close(INFILE2);
# open the first file
open INFILE3, "$infile3", or die $!;
my %reference = ();
my $flag = 0;
#loop over the input
print "Reading in file 4... \n";
while(<INFILE3>){
	my $string = $_;
	chomp($string);
	if($flag ==0){
		$flag++;
	} else {
		my (@curLine) = split(//,$string);
			foreach my $nuc (@curLine){
				$reference{ $flag } = $nuc;
				$flag++;
			}
	}
}
close(INFILE3);
open OUTFILE, ">>$outfile", or die $!;
# Amino-acid translations
my $tr1 = "TTT";
my $tr2 = "TTC";
my $tr3 = "TTA";
my $tr4 = "TTG";
my $tr5 = "CTT";
my $tr6 = "CTC";
my $tr7 = "CTA";
my $tr8 = "CTG";
my $tr9 = "ATT";
my $tr10 = "ATC";
my $tr11 = "ATA";
```

```
my $tr12 = "ATG";
my $tr13 = "GTT";
my $tr14 = "GTC";
my $tr15 = "GTA";
my $tr16 = "GTG";
my $tr17 = "TCT";
my $tr18 = "TCC";
my $tr19 = "TCA";
my $tr20 = "TCG";
my $tr21 = "CCT";
my $tr22 = "CCC";
my $tr23 = "CCA";
my $tr24 = "CCG";
my $tr25 = "ACT";
my $tr26 = "ACC";
my $tr27 = "ACA";
my $tr28 = "ACG";
my $tr29 = "GCT";
my $tr30 = "GCC";
my $tr31 = "GCA";
my $tr32 = "GCG";
my $tr33 = "TAT";
my $tr34 = "TAC";
my $tr35 = "TAA";
my $tr36 = "TAG";
my $tr37 = "CAT";
my $tr38 = "CAC";
my $tr39 = "CAA";
my $tr40 = "CAG";
my $tr41 = "AAT";
my $tr42 = "AAC";
my $tr43 = "AAA";
my $tr44 = "AAG";
my $tr45 = "GAT";
my $tr46 = "GAC";
my $tr47 = "GAA";
my $tr48 = "GAG";
my $tr49 = "TGT";
my $tr50 = "TGC";
my $tr51 = "TGA";
my $tr52 = "TGG";
my $tr53 = "CGT";
my $tr54 = "CGC";
my $tr55 = "CGA";
my $tr56 = "CGG";
my $tr57 = "AGT";
```

```
my $tr58 = "AGC";
my $tr59 = "AGA";
my $tr60 = "AGG";
my $tr61 = "GGT";
my $tr62 = "GGC";
my $tr63 = "GGA";
my $tr64 = "GGG";
my %trans;
$trans{ $tr1 } = 'Phe';
$trans{ $tr2 } = 'Phe';
$trans{ $tr3 } = 'Leu';
$trans{ $tr4 } = 'Leu';
$trans{ $tr5 } = 'Leu';
$trans{ $tr6 } = 'Leu';
$trans{ $tr7 } = 'Leu';
$trans{ $tr8 } = 'Leu';
$trans{ $tr9 } = 'Ile';
$trans{ $tr10 } = 'Ile';
$trans{ $tr11 } = 'Ile';
$trans{ $tr12 } = 'Met';
$trans{ $tr13 } = 'Val';
$trans{ $tr14 } = 'Val';
$trans{ $tr15 } = 'Val';
$trans{ $tr16 } = 'Val';
$trans{ $tr17 } = 'Ser';
$trans{ $tr18 } = 'Ser';
$trans{ $tr19 } = 'Ser';
$trans{ $tr20 } = 'Ser';
$trans{ $tr21 } = 'Pro';
$trans{ $tr22 } = 'Pro';
$trans{ $tr23 } = 'Pro';
$trans{ $tr24 } = 'Pro';
$trans{ $tr25 } = 'Thr';
$trans{ $tr26 } = 'Thr';
$trans{ $tr27 } = 'Thr';
$trans{ $tr28 } = 'Thr';
$trans{ $tr29 } = 'Ala';
$trans{ $tr30 } = 'Ala';
$trans{ $tr31 } = 'Ala';
$trans{ $tr32 } = 'Ala';
$trans{ $tr33 } = 'Tyr';
$trans{ $tr34 } = 'Tyr';
$trans{ $tr35 } = 'STOP';
$trans{ $tr36 } = 'STOP';
$trans{ $tr37 } = 'His';
$trans{ $tr38 } = 'His';
```

```
$trans{ $tr39 } = 'Gln';
$trans{ $tr40 } = 'Gln';
$trans{ $tr41 } = 'Asn';
$trans{ $tr42 } = 'Asn';
$trans{ $tr43 } = 'Lys';
$trans{ $tr44 } = 'Lys';
$trans{ $tr45 } = 'Asp';
$trans{ $tr46 } = 'Asp';
$trans{ $tr47 } = 'Glu';
$trans{ $tr48 } = 'Glu';
$trans{ $tr49 } = 'Cys';
$trans{ $tr50 } = 'Cys';
$trans{ $tr51 } = 'STOP';
$trans{ $tr52 } = 'Trp';
$trans{ $tr53 } = 'Arg';
$trans{ $tr54 } = 'Arg';
$trans{ $tr55 } = 'Arg';
$trans{ $tr56 } = 'Arg';
$trans{ $tr57 } = 'Ser';
$trans{ $tr58 } = 'Ser';
$trans{ $tr59 } = 'Arg';
$trans{ $tr60 } = 'Arg';
$trans{ $tr61 } = 'Gly';
$trans{ $tr62 } = 'Gly';
$trans{ $tr63 } = 'Gly';
$trans{ $tr64 } = 'Gly';
print OUTFILE "Contig\tLocation\tReference\tSNP(s)\tRef->Alt1\tRef->Alt2\tRef-
>Alt3\tBlastDBforFrame\n";
# open the first file
open INFILE4, "$infile4", or die $!;
my $flag = 0;
#loop over the input
print "Reading in file 5... \n";
print "Writing to output file...\n\n";
while(<INFILE4>){
	my $string = $_;
	chomp($string);
	if($flag ==0){
		$flag++;
	} else {
	   my ($featName, $source, $type, $s, $e, $score,$strand, $phase, $att) = split(/\t/,$string,
9);
			if($type eq "SNP"){
			my ($name, $parent, $ref, $alt, $color) = split(/;/,$att, 5);
				$name =~ s/Name=//;
				$ref =~ s/Ref=//;
```

```
$alt =~ s/Alt=//;
my ($contigID, $snpLoc) = split(/_/,$name, 2);

print OUTFILE $contigID, "\t", $snpLoc, "\t", $ref, "\t", $alt;
my $bStart = 0;
my $bEnd = 0;
my $bStrand = '+';

my $bFlag1 = 0;
my $bStart1 = 0;
my $bEnd1 = 0;
my $bStrand1 = '+';
my $bScore1 = 0;
my $bFlag2 = 0;
my $bStart2 = 0;
my $bEnd2 = 0;
my $bStrand2 = '+';
my $bScore2 = 0;
my $bFlag3 = 0;
my $bStart3 = 0;
my $bEnd3 = 0;
my $bStrand3 = '+';
my $bScore3 = 0;
if(exists $blast1{$contigID}){
        ($bStart1, $bEnd1, $bScore1, $bStrand1) =
        split(/,/,$blast1{$contigID}, 4);
        $bFlag1++;
}
if(exists $blast2{$contigID}){
        ($bStart2, $bEnd2, $bScore2, $bStrand2) =
        split(/,/,$blast2{$contigID}, 4);
        $bFlag2++;
}
if(exists $blast3{$contigID}){
        ($bStart3, $bEnd3, $bScore3, $bStrand3) =
        split(/,/,$blast3{$contigID}, 4);
        $bFlag3++;
}
if(($bScore1 >= $bScore2) and ($bScore1 >= $bScore3)){
        $bStart = $bStart1;
        $bEnd = $bEnd1;
        $bStrand = $bStrand1;
        $bFlag1++;
} elsif($bScore2 >= $bScore3){
        $bStart = $bStart2;
        $bEnd = $bEnd2;
```

```
		$bStrand = $bStrand2;
		$bFlag2++;
	} else {
		$bStart = $bStart3;
		$bEnd = $bEnd3;
		$bStrand = $bStrand3;
		$bFlag3++;
	}
	my $refProt = "";
	my $refStr = "";
	my $aProt = "";
	my $aStr = "";
	my @altSeq = split(/,/, $alt);
	my $multFlag = 0;

	if($s >= $bStart and $s <= $bEnd){
		$bFlag1++;
		$bFlag2++;
		$bFlag3++;
		if($bStrand eq '+'){
			my $diff = $s - $bStart;
			my $fullCodons = 0;
			my $change = 0;
			my $diff1 = 0;

			if($diff < 3){
				$diff1 = $diff;
			} else {
				$fullCodons = int($diff / 3);
				$change = $bStart + ($fullCodons * 3);
				$diff1 = $s - $change;
			}

			if($diff1 == 0){
				$refStr = $reference{$s} . $reference{($s + 1)} .
				$reference{($s + 2)};
				$refProt = $trans{ $refStr };

				foreach my $curAlt (@altSeq){
					$aStr = $curAlt . $reference{($s + 1)} .
					$reference{($s + 2)};
					$aProt = $trans{ $aStr };
					if($refProt eq $aProt){
						print OUTFILE "\tSYN:", $refProt, "-
						>", $aProt;
					}else{
```

```
                                print OUTFILE "\tNON:". $refProt,
                                "->", $aProt;
                        }
                        $multFlag++;
                }
        } elsif($diff1 == 1){
                $refStr = $reference{($s - 1)} . $reference{$s} .
                $reference{($s + 1)};
                $refProt = $trans{ $refStr };

                foreach my $curAlt (@altSeq){
                        $aStr = $reference{($s - 1)} . $curAlt .
                        $reference{($s + 1)};
                        $aProt = $trans{ $aStr };
                        if($refProt eq $aProt){
                                print OUTFILE "\tSYN:", $refProt, "-
                                >", $aProt;
                        }else{
                                print OUTFILE "\tNON:". $refProt,
                                "->", $aProt;
                        }
                        $multFlag++;
                }
        } elsif($diff1 == 2){
                $refStr = $reference{($s - 2)} . $reference{($s - 1)} .
                $reference{$s};
                $refProt = $trans{ $refStr };

                foreach my $curAlt (@altSeq){
                        $aStr = $reference{($s - 2)} . $reference{($s
                        - 1)} . $curAlt;
                        $aProt = $trans{ $aStr };
                        if($refProt eq $aProt){
                                print OUTFILE "\tSYN:", $refProt, "-
                                >", $aProt;
                        }else{
                                print OUTFILE "\tNON:". $refProt,
                                "->", $aProt;
                        }
                        $multFlag++;
                }
        }
} else {
        my $diff = $bEnd - $s;
        my $fullCodons = 0;
        my $change = 0;
```

```
my $diff1 = 0;

if($diff < 3){
        $diff1 = $diff;
} else {
        $fullCodons = int($diff / 3);
        $change = $bEnd - ($fullCodons * 3);
        $diff1 = $change - $s;
}

if($diff1 == 0){
        $refStr = $reference{$s} . $reference{($s - 1)} .
        $reference{($s - 2)};
        $refProt = $trans{ $refStr };

        foreach my $curAlt (@altSeq){
                $aStr = $curAlt . $reference{($s - 1)} .
                $reference{($s - 2)};
                $aProt = $trans{ $aStr };
                if($refProt eq $aProt){
                        print OUTFILE "\tSYN:", $refProt, "-
                        >", $aProt;
                }else{
                        print OUTFILE "\tNON:". $refProt,
                        "->", $aProt;
                }
                $multFlag++;
        }
} elsif($diff1 == 1){
        $refStr = $reference{($s + 1)} . $reference{$s} .
        $reference{($s - 1)};
        $refProt = $trans{ $refStr };

        foreach my $curAlt (@altSeq){
                $aStr = $reference{($s + 1)} . $curAlt .
                $reference{($s - 1)};
                $aProt = $trans{ $aStr };
                if($refProt eq $aProt){
                        print OUTFILE "\tSYN:", $refProt, "-
                        >", $aProt;
                }else{
                        print OUTFILE "\tNON:". $refProt,
                        "->", $aProt;
                }
                $multFlag++;
        }
```

```
                                        } elsif($diff1 == 2){
                                                $refStr = $reference{($s + 2)} . $reference{($s + 1)}
                                                . $reference{$s};
                                                $refProt = $trans{ $refStr };

                                                foreach my $curAlt (@altSeq){
                                                        $aStr = $reference{($s + 2)} . $reference{($s
                                                        + 1)} . $curAlt;
                                                        $aProt = $trans{ $aStr };
                                                        if($refProt eq $aProt){
                                                                print OUTFILE "\tSYN:", $refProt, "-
                                                                >", $aProt;
                                                        }else{
                                                                print OUTFILE "\tNON:". $refProt,
                                                                "->", $aProt;
                                                        }
                                                        $multFlag++;
                                                }
                                        }
                                }
                        } else {
                                print OUTFILE "\tNotInBlastHit\t\t";
                        }
                        if($multFlag == 1){
                                print OUTFILE "\t\t";
                        } elsif($multFlag == 2){
                                print OUTFILE "\t";
                        }
                        if($bFlag1 == 3){
                                print OUTFILE "\ttop-SwissProt\n";
                        }elsif($bFlag2 == 3){
                                print OUTFILE "\ttop-NCBI-NR\n";
                        }elsif($bFlag3 == 3){
                                print OUTFILE "\ttop-Bgraminis-CDS\n";
                        }else{
                                print OUTFILE "\tN/A\n";
                        }
                }
        }
}
close(INFILE4);
close(OUTFILE);
# END
exit;
```

15. Appendix 9

15.1. compileContigData.pl

```
#	Create spreadsheet # 1 (deliverable)
#	resulting file should contain SNP, INDEL, and Mutation counts.
#	The output also has per kb stats for each and the top hit
#	to b. graminis, NR, and swissprot with e-values.
#
#
#	@author: Jason Myers
#	@date: 08/02/2011
#
#!/usr/bin/perl
use strict;
#use warnings;
# Making sure that the correct number of arguments are given by the user and
# that the program displays an accurate message in case the user does not.
my $numArgs = $#ARGV + 1;
                        # ContigInfor:
ActualContigName<TAB>start<TAB>end<TAB>OriginalContigName
                        # GFF3files: 1 hit per contig (highest scoring)
                        # vcfFile: output of SNP/INDEL calling pipeline
unless( $numArgs == 9) {
      print "Usage: perl compileContigData.pl vcfFile topBlumeriaGFF3 topNRGFF3",
                        " topSwissProtGFF3 csvFile topBGhit topNRblast topSwissBlast
contigInfo\n";
         exit;
}
# get a handle on the input files
my $vcfFile = $ARGV[0];
my $blum = $ARGV[1];
my $nr = $ARGV[2];
my $swiss = $ARGV[3];
my $csvFile = $ARGV[4];
my $bgBlastFile = $ARGV[5];
my $nrBlastFile = $ARGV[6];
my $swissBlastFile = $ARGV[7];
my $contigInfo = $ARGV[8];
#vairable definitions
my $flag = 0;
my $index = 0;
my $indFlag = 0;
my %snpCount;
my %indelCount;
my %blumHit;
my %bgEvals;
```

```
my %nrHit;
my %swissHit;
my %nrGI;
my %nrEvals;
my %swissGI;
my %swissEvals;
my %csvList;
my %contigLength;
my @contigNames =();
my @OLDcontigNames =();
my $nrLink = "";
my $swissLink = "";
my $beginLink = "=HYPERLINK(\"http://www.ncbi.nlm.nih.gov/protein/";
# open the contigInfo file
open INFILE, "$contigInfo", or die $!;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my ($newName, $start, $end, $oldName) = split(/\t/,$string, 4);
	$contigLength{"$newName"} = $end - $start;
	$contigNames[$index] = $newName;
	$OLDcontigNames[$index] = $oldName;
	$index++;
}
close(INFILE);
# open the vcf file
open INFILE, "$csvFile", or die $!;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my (@csvHeader) = split(/,/,$string);
	shift(@csvHeader);
	foreach my $csvHead (@csvHeader){
		$csvList{"$csvHead"} = 0;
	}
}
close(INFILE);
# open the blast file
open INFILE, "$bgBlastFile", or die $!;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	my ($query, $subject, $iden, $align, $mis,
```

```
            $gap, $qs, $qe, $ss, $se, $eval, $score)
            = split(/\t/,$string, 12);
        $bgEvals{"$query"} = $eval;
}
close(INFILE);
# open the blast file
open INFILE, "$nrBlastFile", or die $!;
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
        my ($query, $subject, $iden, $align, $mis,
            $gap, $qs, $qe, $ss, $se, $eval, $score)
            = split(/\t/,$string, 12);
        $nrEvals{"$query"} = $eval;
}
close(INFILE);
# open the blast file
open INFILE, "$swissBlastFile", or die $!;
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
        my ($query, $subject, $iden, $align, $mis,
            $gap, $qs, $qe, $ss, $se, $eval, $score)
            = split(/\t/,$string, 12);
        $swissEvals{"$query"} = $eval;
}
close(INFILE);
# open the vcf file
open INFILE, "$vcfFile", or die $!;
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
        if($flag == 0 || $flag ==1){
                $flag++;
        } else {
                my ($chrom, $pos, $var, $ref, $alt, $qual, $fil,
                        $info, $format, $isolates) = split(/\t/,$string, 10);
                my $tempString = $chrom . "_" . $pos;
                if(exists $csvList{$tempString}){
                        if($info =~ m/INDEL/){
                                $indelCount{"$chrom"}++;
                        } else {
                                $snpCount{"$chrom"}++;
```

```
                        }
                }
        }
}
close(INFILE);
# open the topBLUMERIAgff3 file
open INFILE, "$blum", or die $!;
$flag = 0;
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
        if($flag ==0){
                $flag++;
        } else {
                my ($oldContig, $source, $type, $s, $e, $score,
                        $strand, $phase, $att)  = split(/\t/, $string, 9);
                my($name, $parent, $color, $subDB,
                        $eVal) = split(/;/, $att, 6);
                my($tag, $hitName) = split(/=/,$name, 3);
                $blumHit{"$oldContig"} = $hitName;
        }
}
close(INFILE);
# open the topNRgff3 file
open INFILE, "$nr", or die $!;
$flag = 0;
#loop over the input
while(<INFILE>){
        my $string = $_;
        chomp($string);
        if($flag ==0){
                $flag++;
        } else {
                my ($oldContig, $source, $type, $s, $e, $score,
                        $strand, $phase, $att)  = split(/\t/, $string, 9);
                my($name, $desc, $parent, $color, $subDB,
                        $eVal) = split(/;/, $att, 6);
                my($head, $hitProt, $nothing) = split(/\"/,$desc, 3);
                $nrHit{"$oldContig"} = $hitProt;
                my($tag, $hitName) = split(/=/,$name, 3);
                my($junk1, $keep, $junk2, $junk3) = split(/\|/,$hitName, 4);
                $nrGI{"$oldContig"} = $keep;
        }

}
```

```
close(INFILE);
# open the topSWISSgff3 file
open INFILE, "$swiss", or die $!;
$flag = 0;
#loop over the input
while(<INFILE>){
	my $string = $_;
	chomp($string);
	if($flag ==0){
		$flag++;
	} else {
		my ($oldContig, $source, $type, $s, $e, $score,
			$strand, $phase, $att)  = split(/\t/, $string, 9);
		my($name, $desc, $parent, $color, $subDB,
			$eVal) = split(/;/, $att, 6);
		my($head, $hitProt, $nothing) = split(/\"/,$desc, 3);
		$swissHit{"$oldContig"} = $hitProt;
		my($tag, $hitName) = split(/=/,$name, 3);
		my($junk1, $keep, $junk2, $junk3) = split(/\|/,$hitName, 4);
		$swissGI{"$oldContig"} = $keep;
	}
}
close(INFILE);
my $outfile = "En_contigData.txt";
open OUTFILE, ">>$outfile" or die $!;
print OUTFILE "ContigID\tLength(bp)\tSNPcount\tSNPs/kb\tINDELcount\tIndels/kb\t",
	"MutationCount\tMutations/kb\tB.graminis_Hit\tB.graminis_eValue\tNR_GI\tNR_eValu
	e\t", "NR_Desc\tSwissProt_GI\tSwissProt_eValue\tSwissProt_Desc\n";
#loop over the contigs
for(my $i = 0; $i < $index; $i++){
	print OUTFILE $contigNames[$i], "\t", $contigLength{$contigNames[$i]}, "\t";
	if(!exists $snpCount{$contigNames[$i]}){
		$snpCount{$contigNames[$i]} = 0;
	}
	print OUTFILE $snpCount{$contigNames[$i]}, "\t",
		(($snpCount{$contigNames[$i]} * 1000) / $contigLength{$contigNames[$i]}),
"\t";
	if(!exists $indelCount{$contigNames[$i]}){
		$indelCount{$contigNames[$i]} = 0;
	}
	print OUTFILE $indelCount{$contigNames[$i]}, "\t",
		(($indelCount{$contigNames[$i]} * 1000) / $contigLength{$contigNames[$i]}),
"\t";
	print OUTFILE ($snpCount{$contigNames[$i]} + $indelCount{$contigNames[$i]}), "\t",
		((($snpCount{$contigNames[$i]} + $indelCount{$contigNames[$i]}) * 1000) /
		$contigLength{$contigNames[$i]}), "\t";
```

```
	if(!exists $blumHit{$OLDcontigNames[$i]}){
		print OUTFILE "N/A\t";
	} else {
		print OUTFILE $blumHit{$OLDcontigNames[$i]}, "\t";
	}
	if(!exists $bgEvals{$OLDcontigNames[$i]}){
		print OUTFILE "N/A\t";
	} else {
		print OUTFILE $bgEvals{$OLDcontigNames[$i]}, "\t";
	}
	if(!exists $nrGI{$OLDcontigNames[$i]}){
		$nrHit{$OLDcontigNames[$i]} = "N/A";
		$nrLink = "N/A";
	} else {
		$nrLink = $beginLink . $nrGI{$OLDcontigNames[$i]} . "\", " .
		$nrGI{$OLDcontigNames[$i]} . ")";
	}
	if(!exists $nrEvals{$OLDcontigNames[$i]}){
		$nrEvals{$OLDcontigNames[$i]} = "N/A";
	}

	print OUTFILE $nrLink, "\t", $nrEvals{$OLDcontigNames[$i]}, "\t",
	$nrHit{$OLDcontigNames[$i]}, "\t";
	if(!exists $swissGI{$OLDcontigNames[$i]}){
		$swissHit{$OLDcontigNames[$i]} = "N/A";
		$swissLink = "N/A";
	} else {
		$swissLink = $beginLink . $swissGI{$OLDcontigNames[$i]} . "\", " .
		$swissGI{$OLDcontigNames[$i]} . ")";
	}
	if(!exists $swissEvals{$OLDcontigNames[$i]}){
		$swissEvals{$OLDcontigNames[$i]} = "N/A";
	}
	print OUTFILE $swissLink, "\t", $swissEvals{$OLDcontigNames[$i]}, "\t",
$swissHit{$OLDcontigNames[$i]}, "\t";
	print OUTFILE "\n";
}
close(OUTFILE);
# END
exit;
```