5-20-2011

PARSES: A Pipeline for Analysis of RNA-Sequencing Exogenous Sequences

Joseph Coco
University of New Orleans

PARSES: A Pipeline for Analysis of RNA-Sequencing Exogenous Sequences

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Bioinformatics

By

Joseph Coco

B.S., University of New Orleans, New Orleans, Louisiana, 2009

May 2011

ACKNOWLEDGEMENTS

The work contained herein was made possible with the support and guidance provided by my advisor, Dr. Christopher Taylor. The area I chose to work in is one that he has been researching alongside Dr. Erik Flemington of Tulane Cancer Center. In addition, I have utilized some of his code for basic data manipulation.

Funding to support this research has been provided by NIH ARRA supplement NOT-OD-09-060 and the Research Institute for Children.

I would like to thank Dr. Erik Flemington's lab at Tulane Cancer Center for helpful suggestions throughout the course of this work and their patience with providing feedback to me. I'd also like to thank my collaborators Dr. Dongxiao Zhu and his research group and Ms. Qi Zhang at the University of New Orleans for their helpful suggestions.

Finally, I would like to thank Shaun Jackman and Daniel Huson for their assistance in using their tools and quick fixes for bugs.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

RNA-Sequencing (RNA-Seq) has become one of the most widely used techniques to interrogate the transcriptome of an organism since the advent of next generation sequencing technologies [1]. A plethora of tools have been developed to analyze and visualize the transcriptome data from RNA-Seq experiments, solving the problem of mapping reads back to the host organism's genome [2] [3]. This allows for analysis of most reads produced by the experiments, but these tools typically discard reads that do not match well with the reference genome. This additional information could reveal important insight into the experiment and possible contributing factors to the condition under consideration. We introduce PARSES, a pipeline constructed from existing sequence analysis tools, which allows the user to interrogate RNA-Sequencing experiments for possible biological contamination or the presence of exogenous sequences that may shed light on other factors influencing an organism's condition.

Keywords: *exogenous agents, RNA-Seq, contamination, sequence alignment, cancer etiology, sequence assembly, taxonomical classification, cancer treatment.*

CHAPTER 1 – INTRODUCTION

Sequence Alignment

In order to ascertain meaningful information from sequenced deoxyribonucleic acid (DNA) it must be compared to an annotated sequence. This process is called sequence alignment and has many forms.

Needleman-Wunsch

The Needleman-Wunsch algorithm (Equation **1**) is a dynamic programming, optimal, global alignment algorithm for determining similarity between coding sequences.

$$F(i,j) = max\begin{cases} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{cases}$$

Equation 1 where $s(x_i, y_j)$ represents the amount of shared information between the sequences and d represents the penalty incurred for a gap in alignment.

Smith-Waterman

The Smith-Waterman algorithm (Equation **2**) is similar to Needleman-Wunsch algorithm except that it is a local alignment algorithm. It is a dynamic programming, optimal, local alignment algorithm for determining similarity between coding sequences. It identifies homologous regions between sequences and incorporates a system of penalties incurred by inserting gaps. Local alignment is chosen and gaps are allowed because DNA accumulates noise through mutation over time.

$$F(i,j) = max\begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{cases}$$

Equation 2 where $s(x_i, y_j)$ represents the amount of shared information between the sequences and d represents the penalty incurred for a gap in alignment.

BLAST+

The Smith-Waterman algorithm is the most accurate for database searches but is both time and CPU intensive. Therefore, for larger searches, the Basic Local Alignment Search Tool (BLAST) was created. BLAST is a widely used heuristic local alignment tool which indexes very high scoring matches over short stretches of the sequence. These sequences are called seeds and are grown in later processes to find alignments. Thresholds are set on how far out each extension from the seed can reach and how much an alignment score must grow for the extension to be added [4].

BLAST initially uses a modification of the Smith-Waterman or Sellers algorithms to classify high-scoring segment pairs (HSPs) whose scores are maximized with respect to extensions and trimming. Random sequences are generated to determine the likelihood of a high score occurring by chance. The model forces random alignments to be negative to ensure longer alignments are not selected by default [5]. The expectation value (evalue) represents the number of hits expected assuming completely random sequences when searching a database of a particular size. However, in order for a scoring system to be meaningful it must be relative to each of the alignments as seen in Equation **3**, which is known as a bit score. Given sufficiently large sequences of length m and n, the evalue can be characterized by Equation **4**. The evalue decreases exponentially with respect to score S [6].

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

Equation 3 where λ is a scale for database size, S represents the score, and K is a scale for scoring system.

$$E = mn2^{-S'}$$

Equation 4 where m and n are lengths of sequences being aligned, λ is a scale for database size, and S represents the score.

TopHat

An organism's transcriptome represents the coding material at a specific moment within the organism. Alternate splicing of genes requires alignment which is more complicated than aligning to a reference genome as there can be many different permutations of gene transcription.

TopHat maps reads in two phases, the first of which uses the Burrows-Wheeler algorithm potentially with mismatches against the entire reference genome, the second enumerates the island sequences extracted for all canonical donor and acceptor sites and computes all pairings of the selected sites that form canonical introns formed between neighboring islands. Following, any instances of two segments of the same read being mapped far from each other or of internal segments failing to map are recorded as possible splice junctions. Finally, the reads from the second procedure are checked against the reads from the first procedure and reported as splicing junction reads [3].

RNA-Seq

The 'next generation' messenger ribonucleic acid (RNA) sequencing technology, RNA-Seq, is similar in function to microarray sequencing. It uses unique mapping and quantifying transcriptome methods which allows for isolation of isoforms for all reads, an improvement over previous deep-sequencing methods [1].

In order for RNA-Seq to begin there must be a reference genome to which the sample can align. RNA-Seq uses statistical analysis of the length of an alignment, or expression profile, to the reference genome to distinguish the difference between technology-based artifacts and biological discoveries. This step is known as building a DNA library [7]. After this initial step high-quality RNA is extracted from a sample. It is then purified from contaminating DNA and protein by filtered for Poly-A tails using Magnetic Oligo(dT)

Beads. The RNA is then fragmented via divalent cations, usually into strands 200 base pairs in length and subsequently randomly probed in overlapping segments of six base-pairs (bp), both of which produces randomly reverse-transcribed RNA—generally with reverse transcriptase and DNA polymerase—into a complementary DNA (cDNA). The ends of the DNA are repaired via a T4 DNA polymerase and Klenow DNA polymerase; the 3' end is further processed, and the ends are ligated. Once this is done a DNA template is created in order to PCR the DNA strands [8].

Next a process called bridge amplification is performed. First the DNA is randomly fragmented and both ends are ligated with adapters known as bridges. Following, single strands of the DNA are randomly, densely bound to a flow cell channel—a glass slide. Then free nucleotides and enzymes are added to bind the bridges to the substrate and to create a second strand for each fragment, forming a U-like structure. The solution is then denatured causing the strands to break and leave only a single bridge attached to the substrate. This denaturation is repeated, creating a cluster station.

These DNA attached to a flow cell must then be sequenced. For efficiency reasons, sequencing uses a reversible dye-terminator technology and is done in parallel. Initially, four labeled reversible terminators are added to the flow cell along with primers and DNA polymerase and the base pairs are excited via a laser. An image is produced for all fluorescently labeled base pairs, photographing a single layer of the flow cell, after which the base pair's terminal 3' blocker and fluorophore are removed to allow the next set of nucleotides in each fragment to be imaged [9]. The intensity of the image represents the statistical certainty of each base pair known as its quality score and the color represents Adenine, Cytosine, Thymine, or Guanine [10]. The throughput is hundreds of millions of reads ranging from 30-100 base pairs [11].

RNA-Seq vs. Microarray

RNA-Seq is largely replacing Microarrays, or the more modern tiling arrays, because they allow identification of novel splicing isoforms which are alternate ways to transcribe a DNA sequence. In addition, RNA-Seq is cheaper, does not require sequencing be tailored to target for specific types of experiments or organisms, has a broader and more specific quality score range, can detect otherwise unannotated transcript regions, is sensitive to even low concentrations of DNA, measures absolute concentration instead of relative, has little background noise, and can be scaled to any sequencing depth desired [12].

RNA-Seq in Cancer Research

A traditional example of using RNA-Seq in cancer research is Maher et. al's sequencing of chronic myelogenous leukemia cell line K562 in search of regulatory elements of a gene tainting an oncogene. They detected 111 chimaera gene fusions as potential causal genetic aberrations in humans, many of which were subsequently experimentally confirmed and shown to represent acquired somatic mutations isolated to tumor cells. TMPRSS2-ERG was of particular interest but 56 other distinct gene fusions were also discovered using VCaP cells. Identifying gene fusions offers therapeutic targets for treating prostate and lung cancers.

Contamination

Contamination in an issue for adaptor-ligation based methods such as RNA-Seq. Thus, in order to deter results or research being altered due to erroneous data preventative procedures must be in place, particularly, one at the end of the process to accredit the results beyond the intuition and knowledge of professionals. Particularly in exploratory research, in which the researcher is not necessarily expecting a specific outcome, such measures would be useful. Contaminants will be referred to as exogenous agent coding reads from analysis of the host organism which were introduced during the procedures involved in reading the RNA sequences from the host organism. Therefore, to distinguish this definition, any biological contamination of foreign organisms occupying the host will be referred to as exogenous agents.

Extent of Contamination

In an experiment judging the amount of contamination within popular, public sequencing databases greater than 20% of non-primate genomes have been found to be contaminated with human DNA. The primate specific AluY repeat was used as query with BLASTN or BLAT to search the databases. It was considered to be contamination if the alignment reported greater than 98% identity and mapped to a single locus in the human genome [13].

Practical Limitations Caused by Contamination

Removal of contaminating DNA is a common process in analyzing ancient DNA. Contamination is seen as the largest obstacle to ancient DNA analysis [14]. Strict mechanisms are put in place such as removing the outer-layer of samples, ultra-violet light irradiation, washing with HCl then water, minimal human contact, disposable lab equipment, designated labs, etc. Even with such strict precautions it is common place to find a copious amount of contaminants in such samples due to various forms of degradation. Some researchers create individual assays for each type of contaminant which is believed to permeate the samples, often relying on haplotype frequency differences, in conjunction with real-time PCR to identify contaminants. Researchers often will not publish their work if contaminants are discovered but presumably if a standard system for removing contaminants were to be agreed upon it may become more common, opening up an abundance of data to anthropologists and geneticists [15].

De Novo Transcriptome Assembly

RNA-Seq reads are often short to the point of being unable to claim with any statistical certainty that a read aligns to a particular organism. Assembly has become an important procedure in many studies.

A de Bruijn graph is used to reduce the computational complexity of de novo assembly by creating contigs, or longer reads composed from stitching together smaller reads, see Figure 1 for an example. De Bruijn graphs are directed graphs which record adjacency by breaking reads into k-mers and using overlaps between the first and last characters of a specified length k-1, which represents homogenous overlaps between sequences, to determine adjacency. Unfortunately, in order to determine the ideal k-mer length the algorithm must be run for all possible values, then an N50 score is computed which is the value for which the weighted median such that 50% of the assembly is within the contigs is equal to or

larger than the specified value. The assembly with the maximum N50 score represents the ideal value for k. The best k value is generally a function of coverage, read length, and error rate [16]. The maximum size of the graph is 4^k, with memory usage increasing as a function of the biological variation and sequencing errors [17].

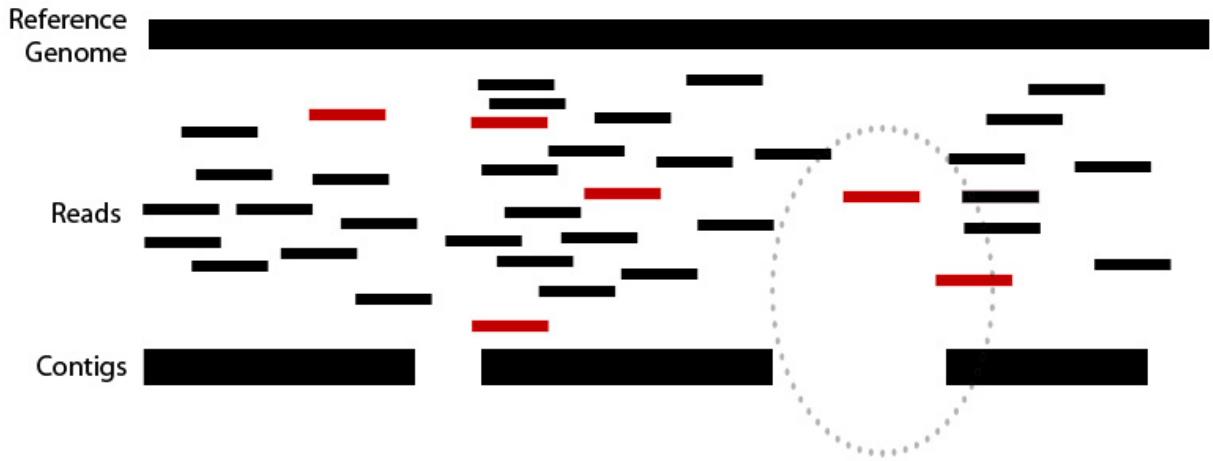

Figure 1 Depiction of de novo assembly process. Red reads represent those not used in the construction of contigs. The circled area shows a read not used in a contig though there is coverage of the genome in that area.

As genome coverage grows so does assembly contig stability and length. De novo assembly has been used to construct 30 million bases of sequence with 66,921 contigs of length 10,951 bp or less. When assembling reads from hg18 only 0.004% did not align which was believed to have been caused by bacterial contamination.

Further the amount of support for individual alleles in the de Bruijn graph correlates with expression levels [16].

Taxonomical Analysis

Through Similarity-Based Methods

MEGAN, the metagenome analyzer, uses BLAST to perform heuristic alignments and bins the alignments via a lowest common ancestry algorithm. This allows reads which can be assigned at a species level to be assigned to taxa near the leaves of the NCBI tree, whereas widely conserved sequences are assigned to higher-order taxa further up the tree. Reads that are not assigned by the lowest common ancestor algorithm are grouped separately in a not assigned category. This approach tends to be computational expensive because all sequences must be compared to a database of all recorded sequences, such as NT, ENV-NT, or WGS [18].

Through Composition-Based Methods

Phymm takes advantage of conserved regions of genomes by identifying short oligonucleotide usage patterns [19]. The frequencies of these short sequences contain information regarding phylogeny. This process can suffer from over fitting if fragments are too short however.

Phymm uses an ab initio classification method utilizing a Bayesian decision machine which uncovers the taxonomic root of a read with its maximum a posteriori probability calculated with interpolated Markov Models. It has the ability to use multiple mers to classify a sequence [20].

Diffuse large B-cell lymphoma

In western countries, diffuse large B-cell lymphoma (DLBCL) is the most common adult non-Hodgkin lymphoma. It is a high grade, aggressive lymphoma which generally presents in a single, intranodal location in a wide age range of people. The median age is 70, though younger people often get the disease especially if they are immunosuppressed for which 40% of tumors are extranodal. The average survival time for untreated DLBCL is 17 months but it is potentially curable with combination chemotherapy and adjuvant radiotherapy.

There are 5 variants and 3 subtypes, but they all present and respond to treatment similarly. A patient's lymph node structures are replaced by sheets of large lymphoma cells, causing 30% of patients to get fever, night sweats, or weight loss. Bone marrow involvement is less frequent and peripheral blood involvement, such as Hypergammaglobulinemia, only occurs in the fatal phase. Anemia, fever, weight loss, and skin rashes are common in advanced stages in the elderly [21].

CHAPTER 2 – METHODS AND TOOLS

Data Source and Processing

DLBCL and Follicular Lymphoma

The DLBCL and Follicular Lymphoma (FL) sequence data were obtained from the NCBI SRA database as part of the NCI Cancer Genome Characterization Initiative (CGCI) (SRP001599). The data were prepared from fresh frozen biopsies. The RNA was then extracted and polyA selecting using oligo(dT) selection procedure. Finally, paired-end sequences were obtained for each sample and ran through an Illumina Genome Analyzer II [22]. Only one lane per sample was used in the processing of the data through PARSES and the data sets were occasionally truncated to 770MB if the files were too large to fit into memory when BLASTed. The samples IPI scores were varied.

Prostate Adenocarcinoma

The Prostate Adenocarcinoma sequencing data were also obtained from the NCBI SRA database as part of a transcription-induced chimeras and gene fusions analysis from three Homo sapiens. Single-end RNA-Seq was performed on the primary prostate tumors and the normal adjacent tissue samples, which were reviewed by board-certified pathologists. An Illumina Genome Analyzer was used for sequencing. The tumor samples were found to be at least 70% tumor content [23].

Normal Lymph Node

The normal lymph node sequencing data was added to EBI database as part of the bodymap2 transcriptome project. The subjects varied in race, sex, and ages from 19 to 86 years. The sequences were subjected to two rounds of oligo(dT) beads binding and underwent conventional, paired-end RNA-Seq preparation using 15 PCR cycles. The final product was normalized by DSN digestion and used for sequencing. An Illumina HiSeq 2000 was used for sequencing 1µg of each paired-end of RNA [24].

Neanderthal

The Neanderthal sequencing data was harvested from the Neanderthal fossil Vi-80 metagenome data generated from whole genome shotgun sequencing. The sample was extracted from the Homo sapiens neanderthalensis fossil long bone from a cave in Vindija, Croatia. Because the complete Neandertal genome was not readily available the Homo sapiens neanderthalensis mitochondrion genome was used as the host organism for PARSES, but no reads within the small sample were aligned to it. Further, it is believed the Neanderthal genome was constructed from these sequences, amongst others, and so Novoalign and TopHat would simply remove all the reads [31].

Mutu

The cells are Mutu I cl. 14, selected for its responsiveness to Transformation Growth Factor β (TGF-β) mediated reactivation to EBV. The cells were extracted from EBV-positive Burkitt's lymphoma cell line from humans in which EBV was primarily latent. The cells were infected with the retrovirus based vector pMSCV-puro-GFP which expresses two genes: Puro resistance gene which was driven by a dedicated promoter and GFP which was driven by a separate promoter. The virus was produced by co-transfecting retroviral vectors with packing vectors into 293 cells and filtered through 0.45 micron filters. Two separate infections of cells were carried out of miRNA 146a and miRNA-155 with each retrovirus. Cells were grown for roughly 10-14 days post-infection. The microRNA (miRNA) contains the pre-miRNA as well as an additional 100-200 bps upstream and downstream to ensure the presence of all associated processing signals. RNA was generated from the cells using Qiagen kits and preps were performed by an expert RNA technician. The majority of the RNA revealed high expression of mature miRNAs [26].

Databases

The nucleotide sequence (NT) database and whole genome shotgun sequencing (WGS) database are NCBI compiled databases which are similar in utility. These databases must be in a BLAST format for BLAST but formatters are available and preformatted databases are regularly distributed. Each sequence of the millions among all BLAST databases has a unique GI identifier. BLAST databases are updated as often as possible, ideally daily. The minimally redundant NT database is a compilation based on informational content and is populated with all reads from traditional GenBank, EMBL, and DDBJ databases excluding other compilation-based databases. WGS is populated with genome assemblies for different organisms [25]. Its reads are generally annotated.

The Taxonomy Identification (TaxID) database links NCBI sequences to taxonomies by associating each GI to a taxonomy identification number. It is updated weekly [26].

Rake

A pipeline can best be described as a group of related tasks, each with preferences and dependencies. Rake provides both file task and non-file task dependency resolution, is available on virtually all platforms, and contains all the functionality of Ruby including the ability to execute shell commands.

Novoalign

Novoalign is a highly accurate sequence alignment tool which is optimized for short sequences. It allows for a maximum mismatch to be set, has a Message Protocol Interface (MPI) version, and is multi-threaded. An index of the reference genome must be created unique to Novoalign before it can be used. It is available on MacOS, Linux, and Windows systems [2].

TopHat

TopHat uses Bowtie to align splice-junction spanning reads. It allows for a maximum mismatch to be set and is multi-threaded. An index of the reference genome must be created unique to Bowtie before it

can be used. It is available on MacOS, Linux, Windows (through Cygwin) systems and also as a module for the central Galaxy server online [3].

BLASTN

BLASTN is a nucleotide alignment tool from the BLAST+ suite of tools. It can search any properly formatted database and there are several regularly updated versions of common databases available online. It is available on MacOS, Linux, and Windows systems and also as a web application with limited functionality. It can be used to infer sequence function, taxonomy, and phylogeny. It can provide output in several formats with varying degrees of information:

- Pairwise, for human readability.
- BLASTTAB, for ease of parsing by scripts.
- BLASTXML, for universal ability to be parsed.

The BLASTTAB format does not contain taxonomy information or base pair specific alignment information but pairwise and BLASTXML do. BLASTN also provides for various filters such as soft masking, DUST, and expectation value (evalue) thresholds. Soft masking is either turned on, which reduces low information segments, or off. However, the evalue threshold is represented by a floating point. It is a gauge of background noise. As an evalue approaches zero, matches are deemed more significant [27].

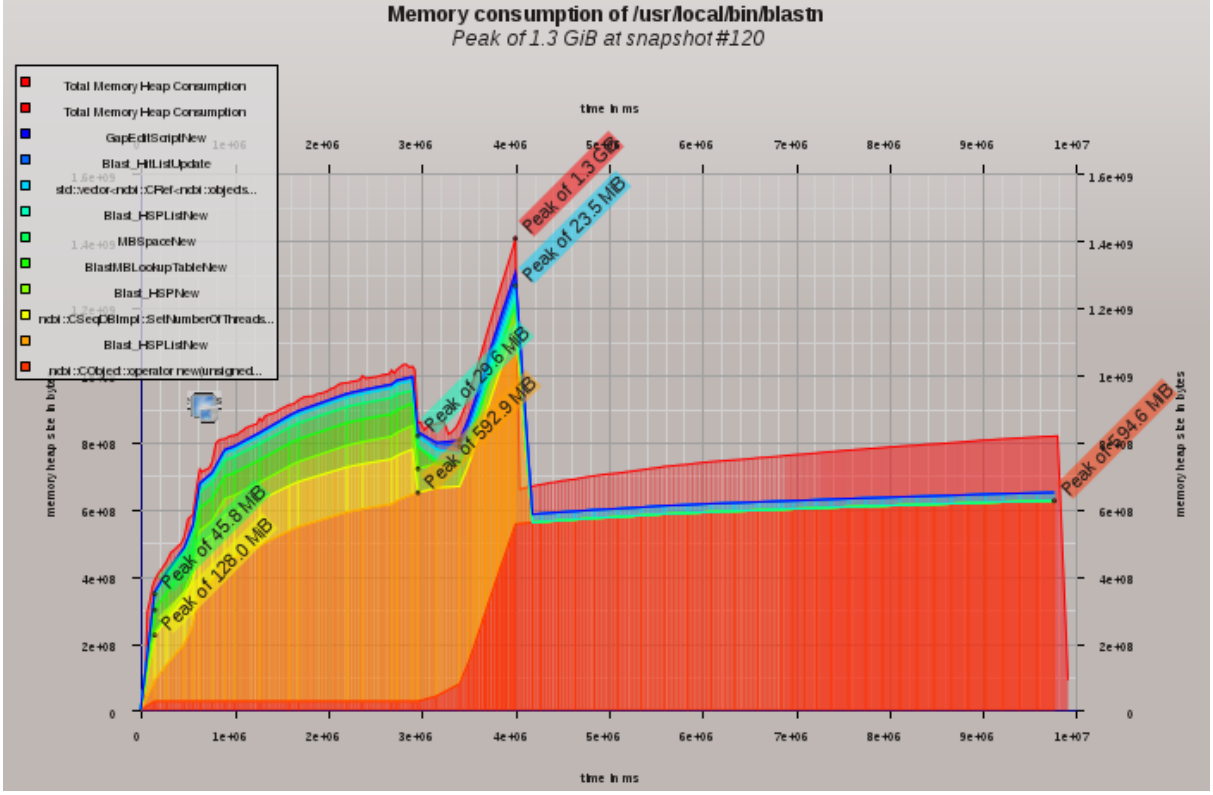

Figure 2 BLASTN execution for 10000 reads using BLASTTAB output format. It represents the main memory and time bottleneck of PARSES.

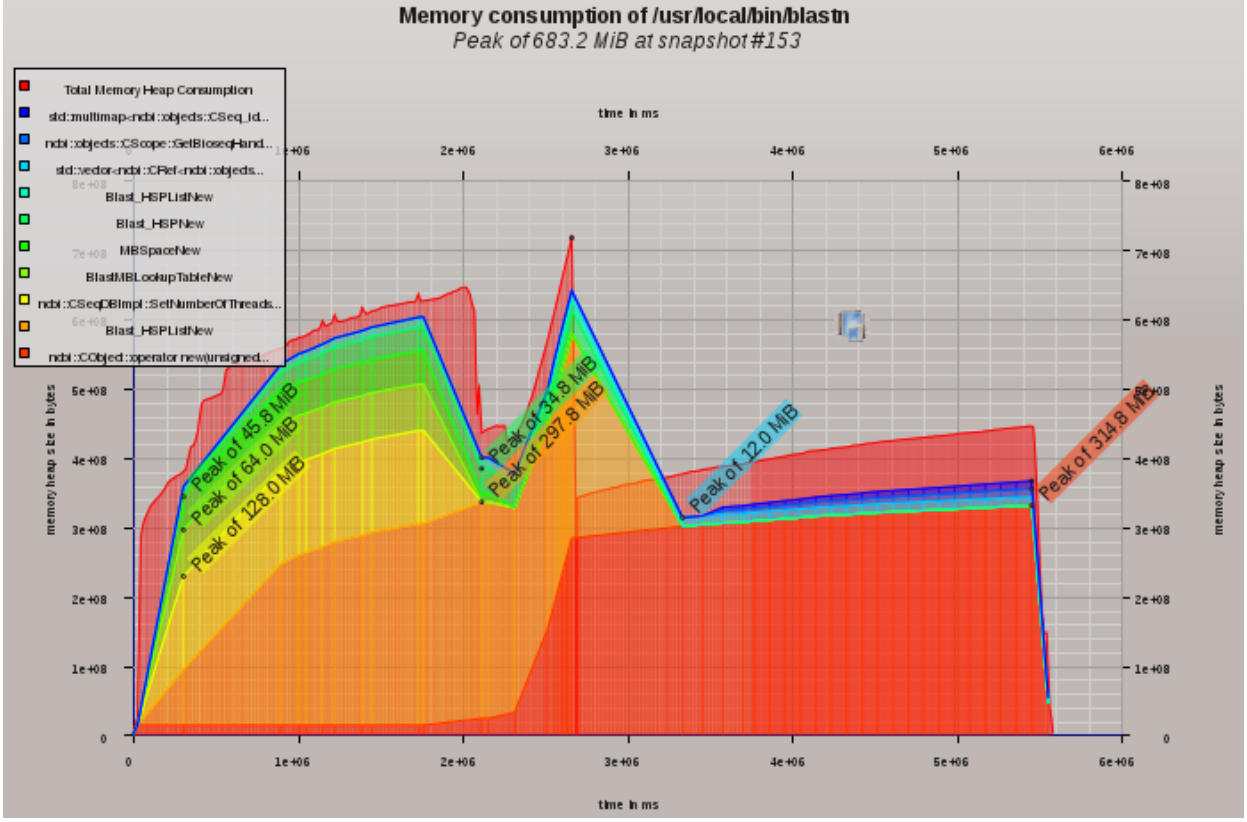

Figure 3 5000 BLASTTAB BLASTN execution has roughly half the memory footprint of a comparable 10000 BLASTN execution.

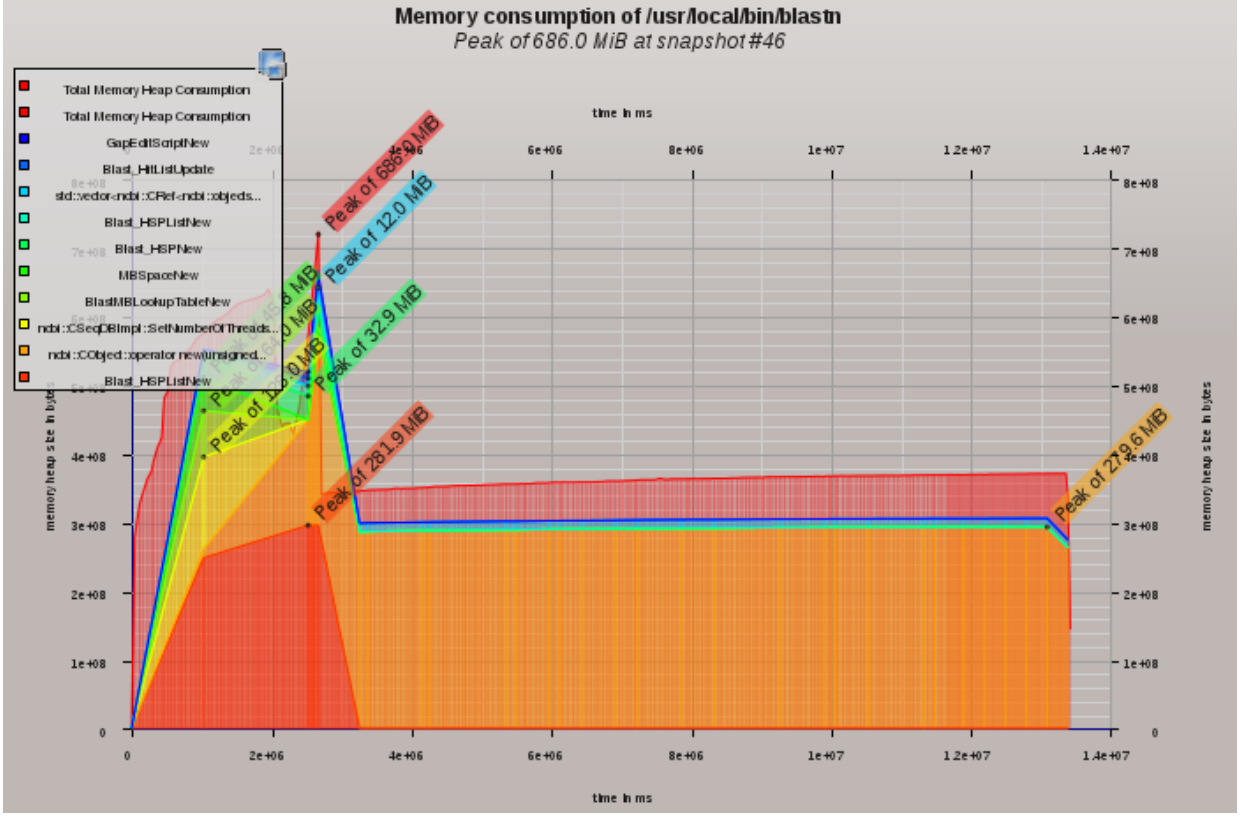

Figure 4 It takes more than twice as long to run 5000 reads through pairwise BLASTN than with BLASTTAB BLASTN.

MEGAN

MEGAN is a metagenomic analysis tool which means it can also perform taxonomical analysis. It is available on MacOS, Linux, and Windows systems. It requires preprocessing with BLASTN or BLASTX but then allows an interactive view of the data. MEGAN also offers classification of SEED and KEGG for reads. It provides comparative features between datasets and can produce various graphs representing taxonomies, rarefaction, comparative data set distance charts, microbial attributes, SEED genomic comparative annotation, and KEGG higher-order systemic behavior analysis. MEGAN also allows for the modification of the least common ancestor (LCA) algorithm interactively by altering the values of min support, min score, top percentage, and win score. Where min support specifies the number of hits in a single taxonomy in order to be classified in that taxonomy, min score specifies the minimum alignment bit score required for a hit to be classified, top percentage specifies the lowest score allowed for a classified hit with respect to the highest scoring hit in the same taxonomy, and win score specifies the minimum score for a hit to be classified as long as any of the hits within the taxonomy meet the win score. See Figure **5** for details on changing the LCA parameters [28].

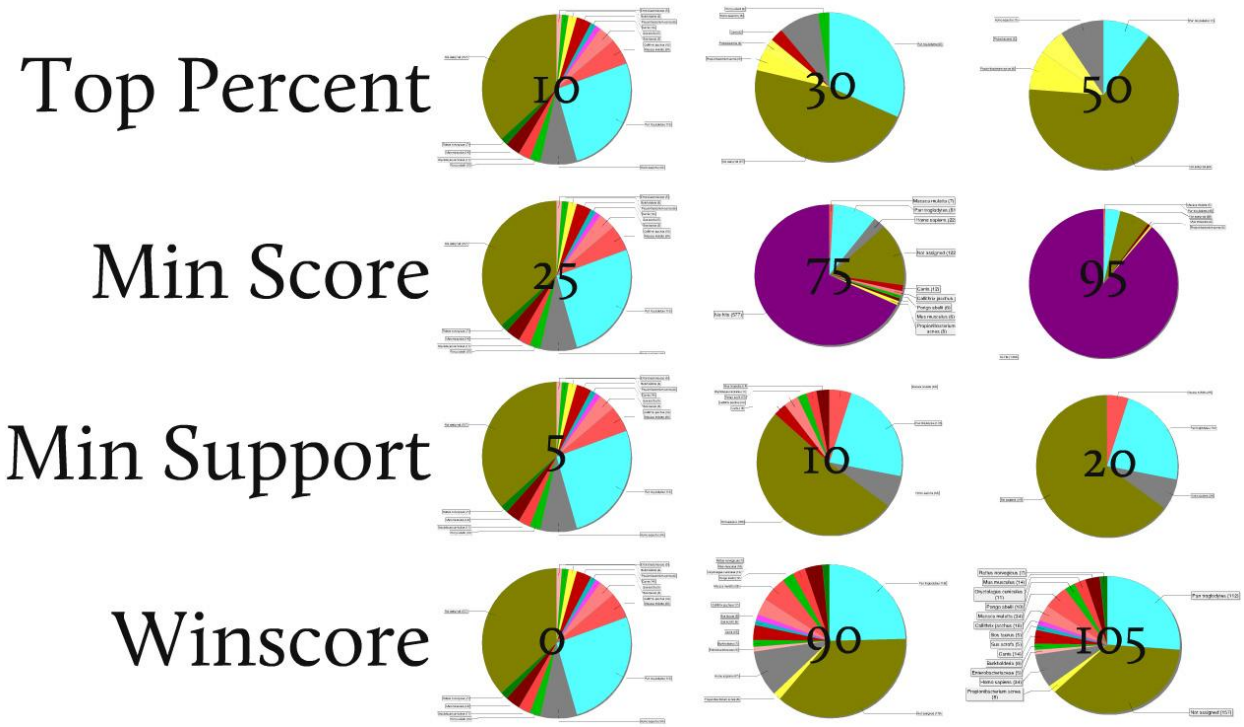

Figure 5 Relative taxonomies changing with respect to LCA parameters.

MEGAN has been shown to produce no false positives when assigning taxonomies to reads in simulations with reads as short as 35 bps and similarly for reads randomly collected from B. bacteriovorus HD100 of 100 bps. Additionally, in simulation when ignoring BLAST hits with bit scores less than 35, MEGAN has been shown to produce small numbers of false positives if the organism has not been sequenced into the provided database [18]. However, given the error rates of sequencing data, synthetic metagenome data created from US National Center for Biotechnology Information (NCBI) RefSeq database showed an accuracy of less than 40% with 400 bp reads at the varying clade levels for MEGAN [20].

ABYSS

ABySS is a de novo assembler. It requires a k-mer be set between 1 and 96 though it is recommended a k-mer less than 7 not be used and by default it is compiled with a maximum k-mer of 64 to conserve memory [29]. It has the ability to filter unchaste reads, masked bases, low quality bases, read length, k-mer coverage, and short bubbles. It assembles in two stages, the first building initial contigs by dividing the reads into k-mer length substrings and filtering read errors, the second contigs are extended by resolving ambiguities through paired-end information if available. It can output k-mer coverage histograms, overlap graphs, and popped bubbles list. It is available on MacOS, Linux, Windows (through Cygwin) systems [16].

CHAPTER 3 - DESIGN AND IMPLEMENTATION

Pipeline Overview

Our pipeline leverages existing tools for sequence similarity search, assembly, and metagenomics in order to explore the presence of exogenous sequences in RNA-Seq data. The goal is to search the sequence reads against a very large, broad database in order to find the most likely origin of the reads which do not match well to the host genome. BLAST+ provides a fast search but does not always yield optimal results and it is difficult and time consuming to interpret the output of BLAST+ searches that yield many results. We employ a tool for metagenomics that provides convenient visualization and taxonomic analysis of BLAST+ results called MEGAN. The biological researcher can interactively refine the views provided by MEGAN and extract interesting reads which were assigned to specific taxonomical categories for further investigation. Since these reads are relatively short, by transcriptional standards, de novo sequence assembly can be performed in order to reconstruct the transcripts in an extracted taxonomical group in order to interrogate the longer transcripts further. This process allows the researcher to proceed from RNA-Seq to de novo assembled transcripts of exogenous DNA with minimal interaction and supervision of the process and without having to learn all of the tools involved in the mapping and processing of sequences. See Figure **6** for details on the data flow of PARSES.

PARSES was designed to discover exogenous, potentially cancer-promoting DNA from tumor cells in humans and because RNA-Seq is highly reproducible PARSES can easily be used to discover contamination in experiments when given biological or technical replicates. With little extra effort, PARSES can also be used to discover exogenous agents in other organisms.

PARSES is designed for researchers to analyze their own data sets but it's possible for it to be executed by biocurators and/or database maintainers to assign a degree of confidence of purity to individual data sets or even executed by a third party which could maintain confidence of data purity across databases. Further it could be used in an automated form on similar data sets to report statistical deviances which may be of interest.

PARSES is capable of removing particular reads from data sets or extracting certain reads from data sets, potentially cleansing a contaminated data set. This is particularly beneficial in fields in which contamination is a large hurdle such as ancient DNA analysis.

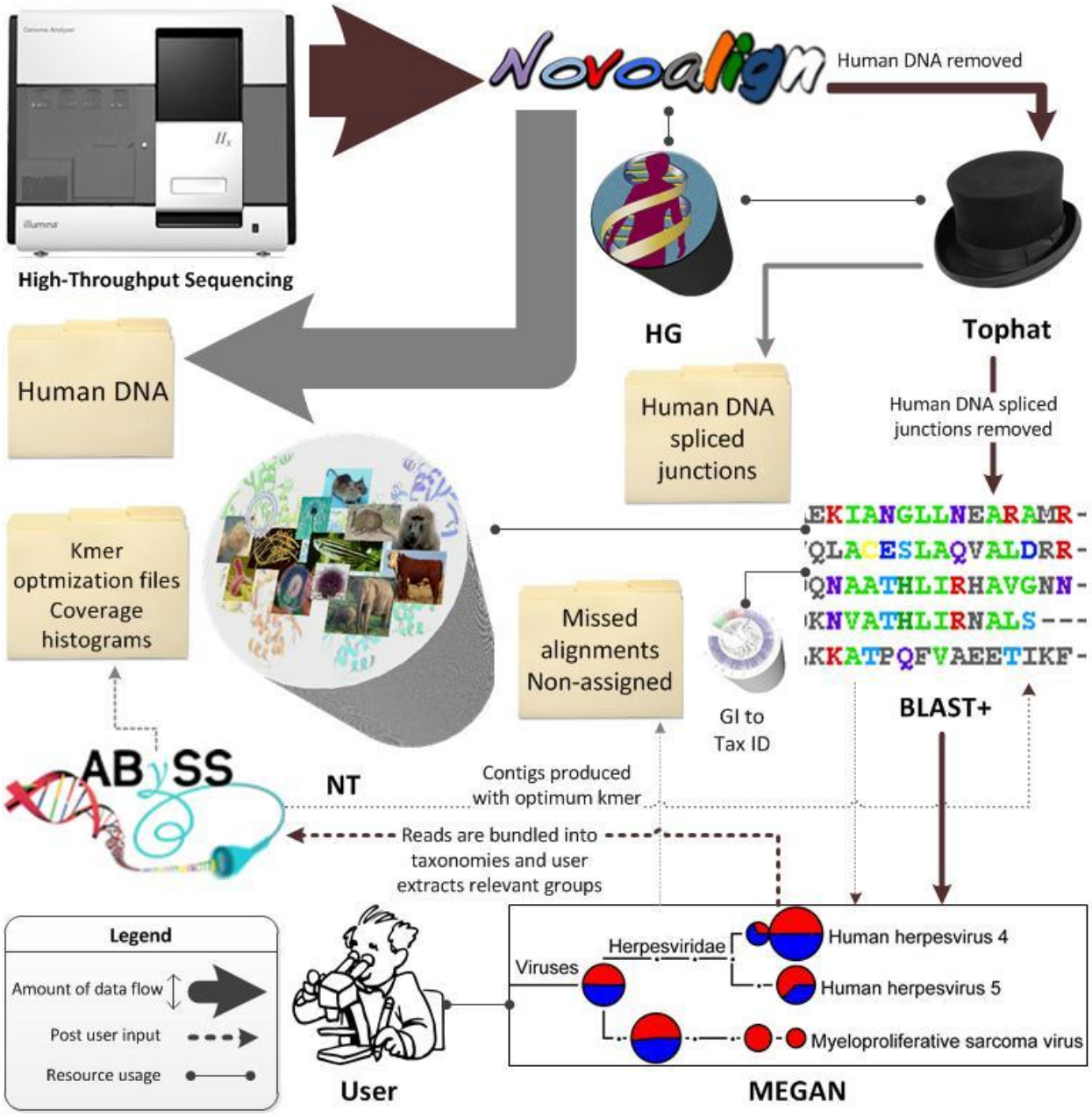

Figure 6 Data flow through PARSES.

Task Dependency Resolution

Rake is an ideal candidate to glue together the tools of PARSES because its constructs abstract the logic of the pipeline to the task level. In many ways it can be considered a domain specific language for pipelines.

Rake's file task dependency resolution allows PARSES to automatically determine the degree to which a data set has been processed and begins execution with the appropriate task. File timestamp

comparisons are utilized to achieve this smart execution. Further, the user can specify a particular step in the pipeline to stop the analysis, but by default the pipeline will fully execute.

Rake's non-file task dependency resolution allows the user to execute PARSES unencumbered by the installations of the underlying tools until they are required, at which point the latest version of the tool will be automatically downloaded, configured, and installed via heavy use of regular expressions, Rake's native Net/HTTP and Net/FTP libraries, and a series of shell commands. In addition, it will allow users to generate reports when such functionality is implemented and will automatically execute PARSES to the necessary point to generate said report [31].

Implementation

PARSES begins by parsing all the configurations for the system or assigned in the past but it then determines where to begin execution of the pipeline. It sequentially executes the following assuming all resources are available to it:

1. novoalign aligns the reads to the host organism, human by default, splitting the reads into uniquely mapped, repeat mapped, quality-controlled, and unmapped reads. All but the unmapped reads are discarded for PARSES's analysis, though the frequencies of all the reads are logged.
2. Xnovotonm script removes all the reads aligned from the unmapped Novoalign output from the original data set.
3. TopHat aligns the splice-junction reads to the host organism.
4. Samtools converts the BAM binary file produced by TopHat to a SAM tab-delimited ASCII file.
5. Xextractspans script isolates the splice-junction reads.
6. Xfilterspans script removes all the reads aligned from TopHat from the original data set.
7. parallelBlast script launches BLASTN in parallel if the computer has enough processing and memory resources to be capable of multiple BLASTN instances. BLASTN aligns the exogenous reads to a compilation-based database. The NT database is used by default but many other databases such as WGS, HTGS, or ENV_NT could also serve the same purpose. DUST filtering is removed if reads are shorter than 50 bps. Soft masking is turned on for all reads and the maximum number of threads is utilized. A configurable evalue of 0.001 is set and the format of BLASTTAB is selected for maximum efficiency.
8. addTaxon script fetches the TaxID for every hit reported from BLASTN and appends it in the last column of the BLASTTAB file.
9. MEGAN produces the RMA file which correlates reads with hits and further with taxonomies and limits hits to being binned into a single taxonomy. Lenient LCA values are used so researchers can make a more informed decision about their data. Min score is set to 35, min support to 1% of the total reads or 5 if 1% is less than five, top percent to 10, and win score to 0.
10. MEGAN launches its GUI allowing the researcher an interactive view of their data. At this point the researcher may change the way the data is visualized, inspect specific

taxonomies which provides details of the hits and reads, export reports of the data, compare data with other data sets, change the LCA parameters, exclude taxonomies which forces reads to be reassigned, remove potential hits which forces reads to be reassigned, store reads and hits in a PostGRES database, set the number of reads used, highlight differences in reads, and extract reads associated with taxonomies which is necessary for the latter half of PARSES [28] [18].

11. abyssKmerOptimizer script launches ABySS for every possibly k-mer in parallel, iterating for each set of reads extracted by taxonomy from MEGAN. It uses a glob to iterate through each file and Parallel::Iterator to invoke ABySS with each k-mer simultaneously. The fac.pl script is invoked to calculate the N50 score of each file and the k-mer optimized files are combined into a single file.
12. BLASTN aligns the hand-picked exogenous contigs to a compilation-based database such as NT. The arguments used are the same except the evalue which is increased to 100 since it is more computational feasible to do so with the small data set.
13. MEGAN produces the RMA file which correlates the hand-picked exogenous contigs with hits and further with taxonomies and limits hits to being binned into a single taxonomy. At this point the researcher can be more confident of the presence of an organism when reported in MEGAN. The arguments used are the same as the first execution of MEGAN.
14. MEGAN launches its GUI allowing the researcher an interactive view of their data and the ability to generate reports and perform comparative analysis.

Record

Logging is implemented using the native logger library built into Ruby. A log is generated for each data set PARSES analyzes and is updated appropriately with a timestamp every time the data set is processed with all invoked shell commands and any relevant comparative data such as the percent of reads the newly processed data set represents.

A specific function is used to execute shell commands which ensures proper logging including harvesting execution time information from each command. All errors encountered during execution of the pipeline will be displayed to the user in addition to being logged.

Usability

PARSES is command line-based and requires at least the sequence name as a parameter to run, though if the sequence has not been run before the file and read type must also be supplied. The exception to this is if the user is entering an install or clean mode in which no arguments other than the task must be specified. PARSES can invoke entire pipeline with single command including installation of required tools and databases. The researcher can select read type including FASTA, Solexa, Illumina 1.3 and Illumina 1.5.

The latest versions of the all the resources utilized by PARSES are automatically downloaded, configured, and executed if they are not already on the system. In order to locate databases and indices PARSES

utilizes the locate command unless it is not accessible or does not find the file, then the find command is used.

PARSES uses native Ruby libraries to poll the websites of all its component tools and databases to determine the latest version of each tool and download them. It does not update software already on the system though the user could simply remove the software if they wished PARSES to update the tool.

PARSES is an independent tool, requiring only to be marked as executable and for gcc, Ruby, and Rake to be installed.

Usage and mechanism documentation exists in addition to providing easy access to a list of all tasks the pipeline can perform.

PARSES functions on both Mac OS X and Linux. It may execute on other UNIX-based systems, but they are not officially supported for the time being. Most of the underlying tools support Windows, however the pipeline is currently tied into a *NIX environment.

Information about the environment in which the pipeline is executing is harvested via various shell commands. Information such as the operating system, number of CPU cores, total system memory, locate database location, and the default shell of the user is extracted for optimizing execution of the underlying tools.

PARSES estimates the amount of memory needed to blast a FASTA file, splits it into equally sized chunks, and recombines them after blasting. BLAST+ chunking allows larger data sets to be processed on a desktop computer because less reads will be held in memory as overhead.

Should PARSES fail to install any tool required for its execution or if the researcher desires to use a specific version of software but does not wish to manually install it, PARSES can be set to download software from a repository maintained online or on the user's system. This also means PARSES will not need to be updated in the event one of software repositories changes; it can merely be accessed through the repository.

PARSES also allows a user to bastardize the pipeline, either selecting a file on which to perform a single task or selecting a file to continue the pipeline from a specified point without previous steps ever being executed.

Rake supports a clean task which removes all temporary files created by PARSES and a clobber task which removes all files associated with PARSES. Numerous files are generated by PARSES and it is unreasonable to expect the user to be capable of identifying all of them, thus, safe and easy clean-up for restarting the analysis or removing temporary files is provided.

PARSES appears to not noticeably take longer to process a single read if there are a greater number of reads. That is to say, the Pearson correlation coefficient is 0.253 when tested on distinct data sets ranging in size from 4062476 reads to 18294116 reads implies a small to medium positive correlation

(See Figure **7**). However, when the largest run is removed from the processing, which did thrash on the BLASTN execution, the Pearson correlation coefficient goes down to -0.0024.

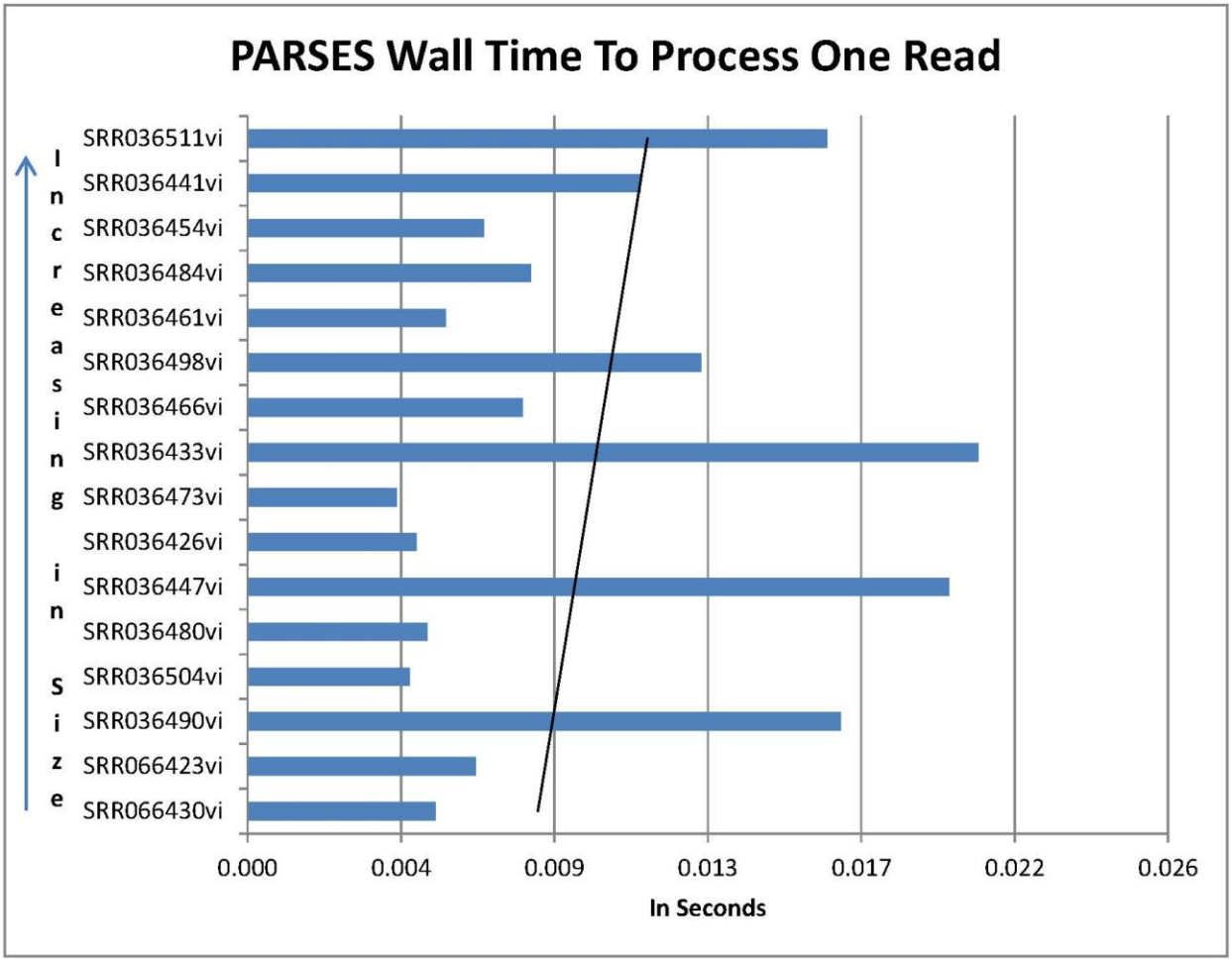

Figure 7 Pearson correlation coefficient is 0.253 which can be explained by disk thrashing for larger data sets. It is believed that if the benchmark machine had more memory there would be no correlation.

PARSES execution time can vary wildly based on the number of host organism reads in the data set. If there are many, there are less reads to search against the larger database NT which considerably decreases execution time (See Figure **8**), even among data sets with the same number of reads (See Figure **9**).

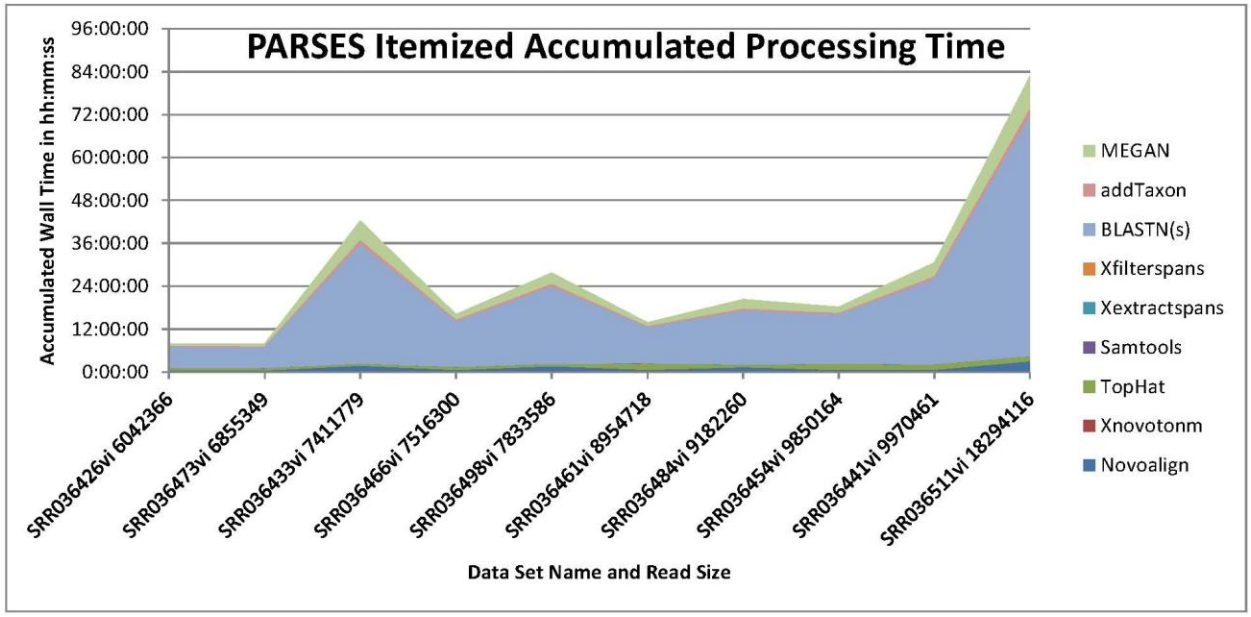

Figure 8 Demonstrating average run time.

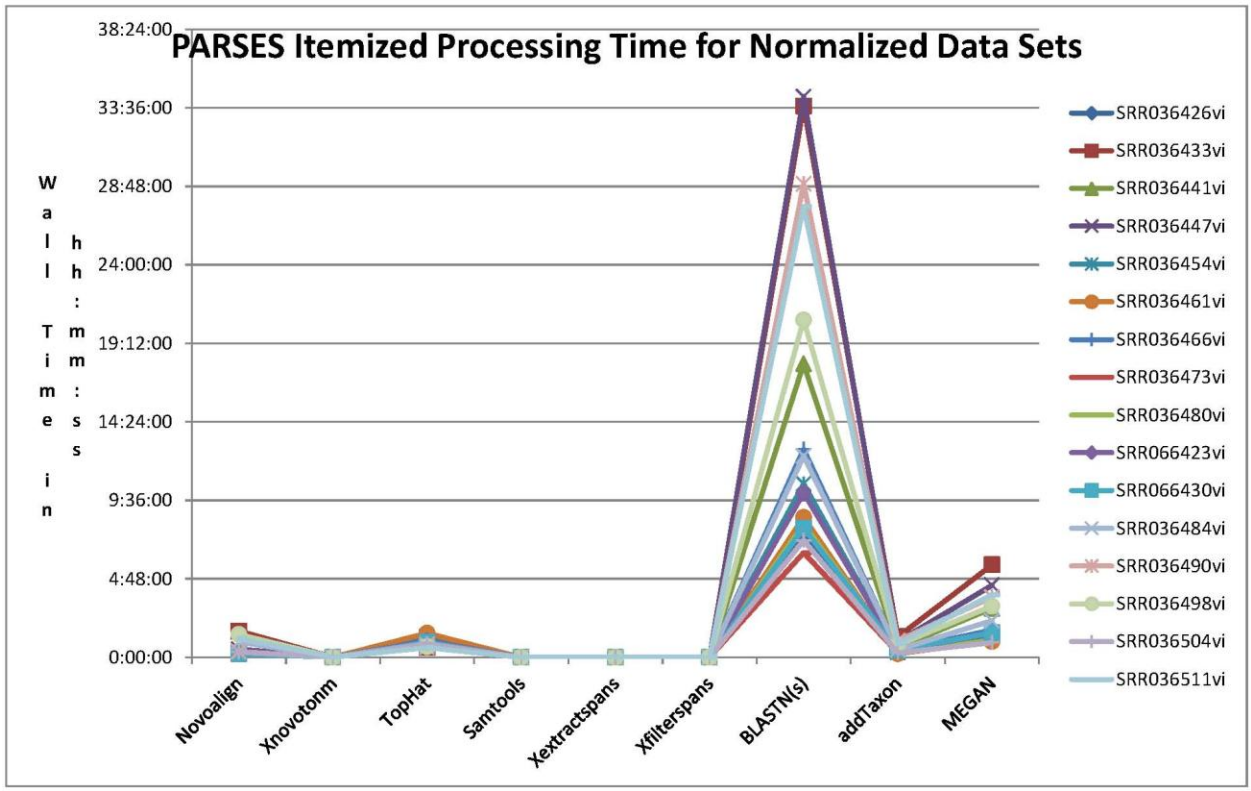

Figure 9 Benchmark normalized to 750,000 reads. The varying ratios of endogenous to exogenous DNA explains the deviating TopHat, BLASTN, and MEGAN running times.

Customization

In the event the host organism for the data set is non-human PARSES allows for a FASTA file to be identified from which it will create indices and execute as the host organism.

PARSES allows for individual steps to be performed outside of the scope of PARSES, allowing the user to manually use other tools on their data as long as the output is in a similar format.

Serialization is implemented using the native YAML library built into Ruby which offers configurations for both the program and the execution of each specific data set to be stored for later usage. In addition, the files are written to disk in a human-alterable form, so if the researcher has unique needs he or she can change most options of the underlying tools.

CHAPTER 4 - RESULTS

DLBCL

In recent, ongoing research by Dr. Erik Flemington and his lab there was roughly 0.5 to 2 Acinetobacter, a Gram-negative genus of Bacteria, reads for every 100 human reads in DLBCL tumors, which is contrary to the controls, each containing no Acinetobacter DNA as seen in Figure **11**. Due to such a strong trend, Dr. Flemington postulates it has a causative effect in tumorgenesis or tumor growth for DLBCL but has yet to experimentally prove this. We believe this postulate to be reasonable as it has been shown 70% of AIDS-associated immunoblastic DLBCL cases present with Epstein-Barr virus (EBV) as well [21].

Acinetobacter strains are ubiquitous, non-fermentative, non-fastidious organisms. They often present as nosocomial infections, particularly in immunocompromised patients. Though they can adapt quickly to new antibodics they are easily treated which means DLBCL may be treatable with antibodics as opposed to the current method of chemotherapy, thereby reducing the danger and invasiveness of treatment. [30]

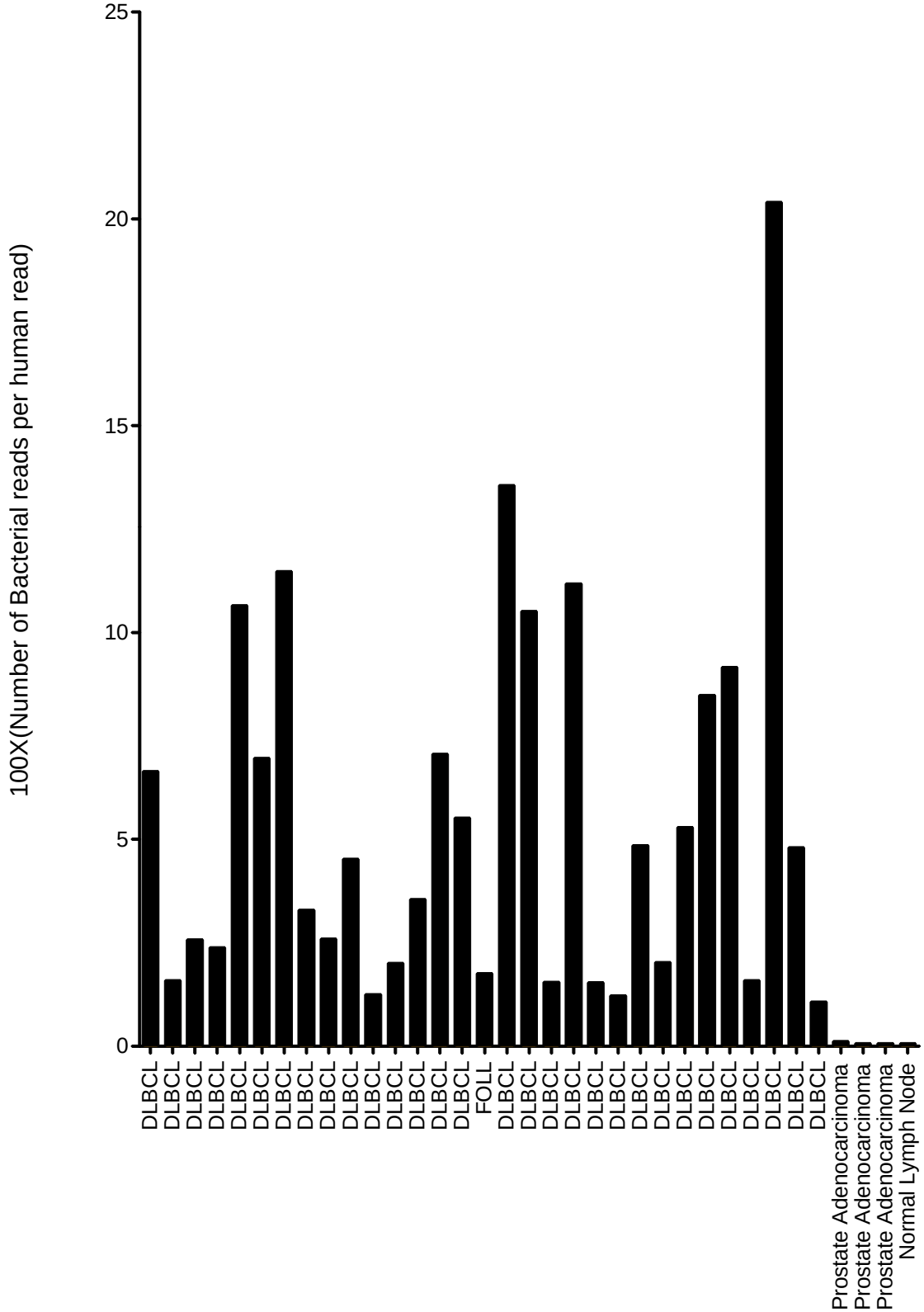

Figure 10 Negligible number of bacteria reads from prostate cancer and normal lymph nodes.

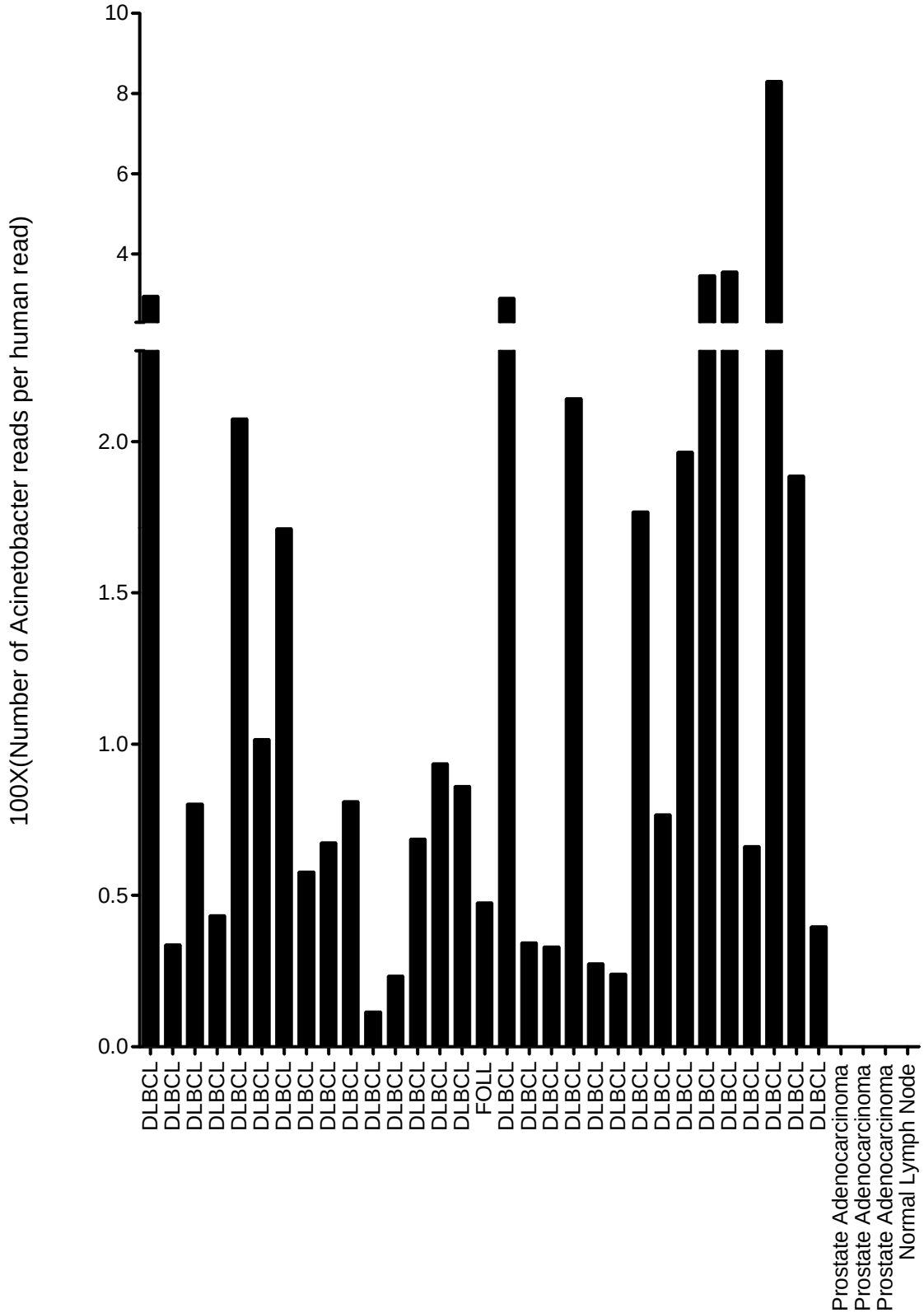

Figure 11 No Acinetobacter reads found in prostate cancer or normal lymph node samples.

Neanderthal Human Contamination

PARSES was executed on 10,000 reads from the Neanderthal fossil Vi-80 metagenome data.

Assuming the researcher was investigating reads differing from particularly primates, such as Homo sapiens, they could potentially remove those reads and continue their analysis as seen in Figure **14**.

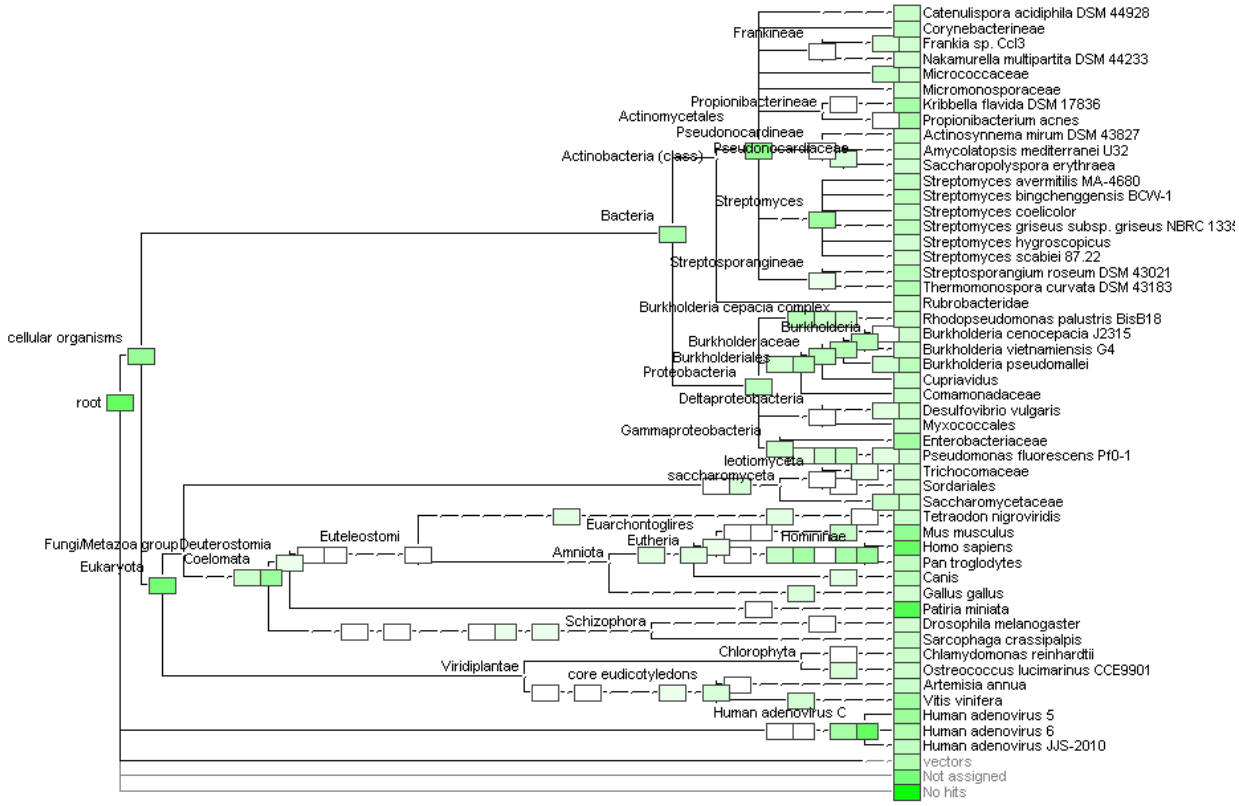

Figure 12 Phylogeny tree heat map of first 10,000 reads from Neanderthal fossil Vi-80.

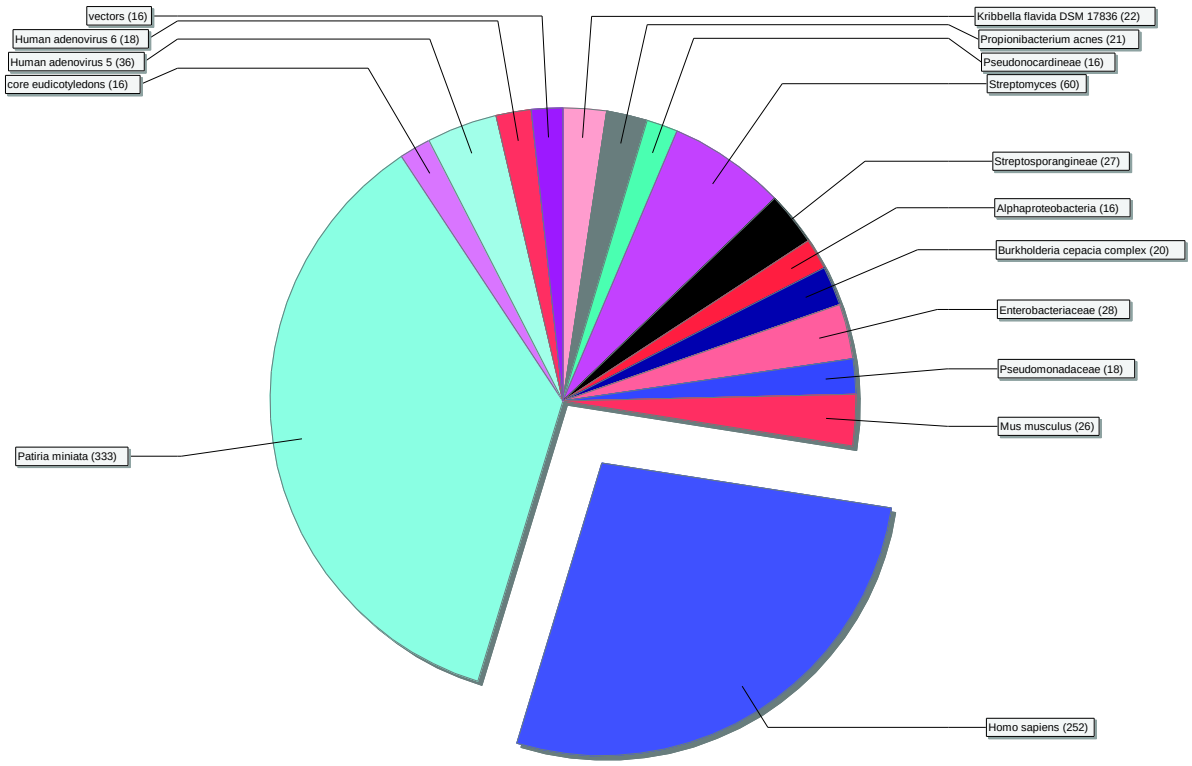

Figure 13 Depiction of number of Homo sapiens reads from Neanderthal data set at species-level.

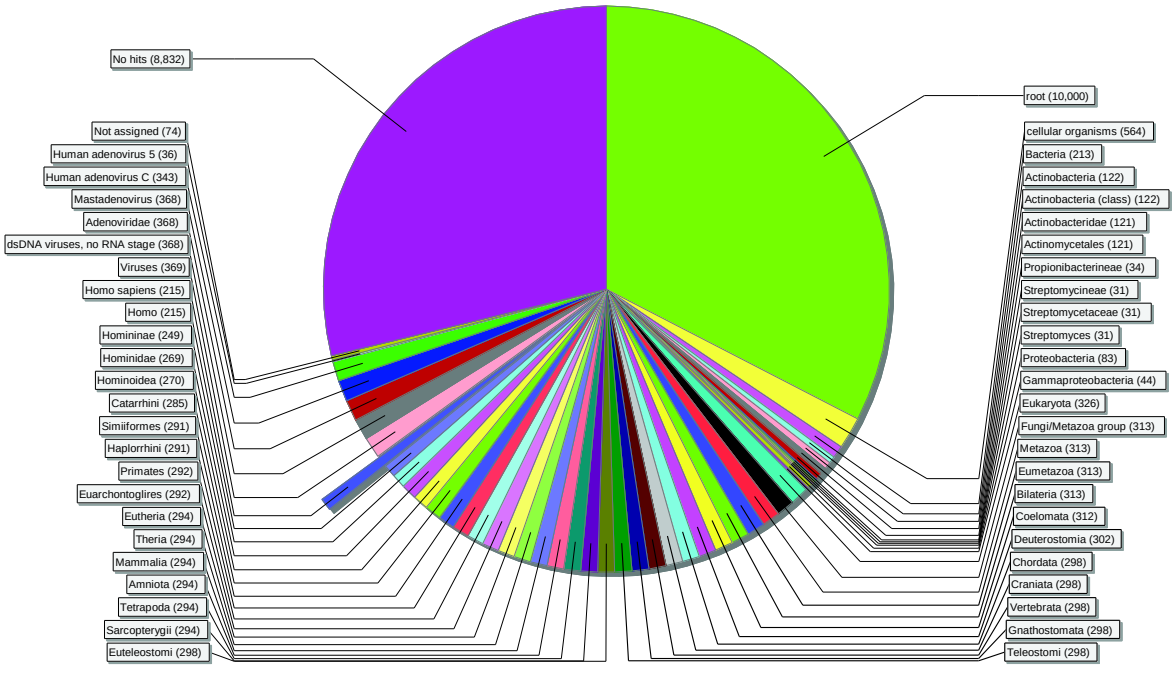

Figure 14 Removing Homo sapiens reads from Neanderthal data set.

Mutu Contamination

The Mutu data set was found to have little to no contamination between biological replicates as seen in Figure **15**. Even when LCA parameters which inherently weed out potential contaminants are set low, see Figure **16**, there is no visually identifiable contamination. When differences in the data sets are brought to the surface, see Figure **17**, it appears there could be contamination but this data only represents 0.02% of all the reads which is not statistically significant given the average quality score of the data sets' reads, see Figure **19**. In addition, given that most of taxonomies in these highly disjoint taxonomies cluster around 100, the assigned min support for this execution of MEGAN, it is highly likely these reads are being binned into different taxonomies simply due to minor, between 1-30, variations in the number of reads for each taxonomy between data sets.

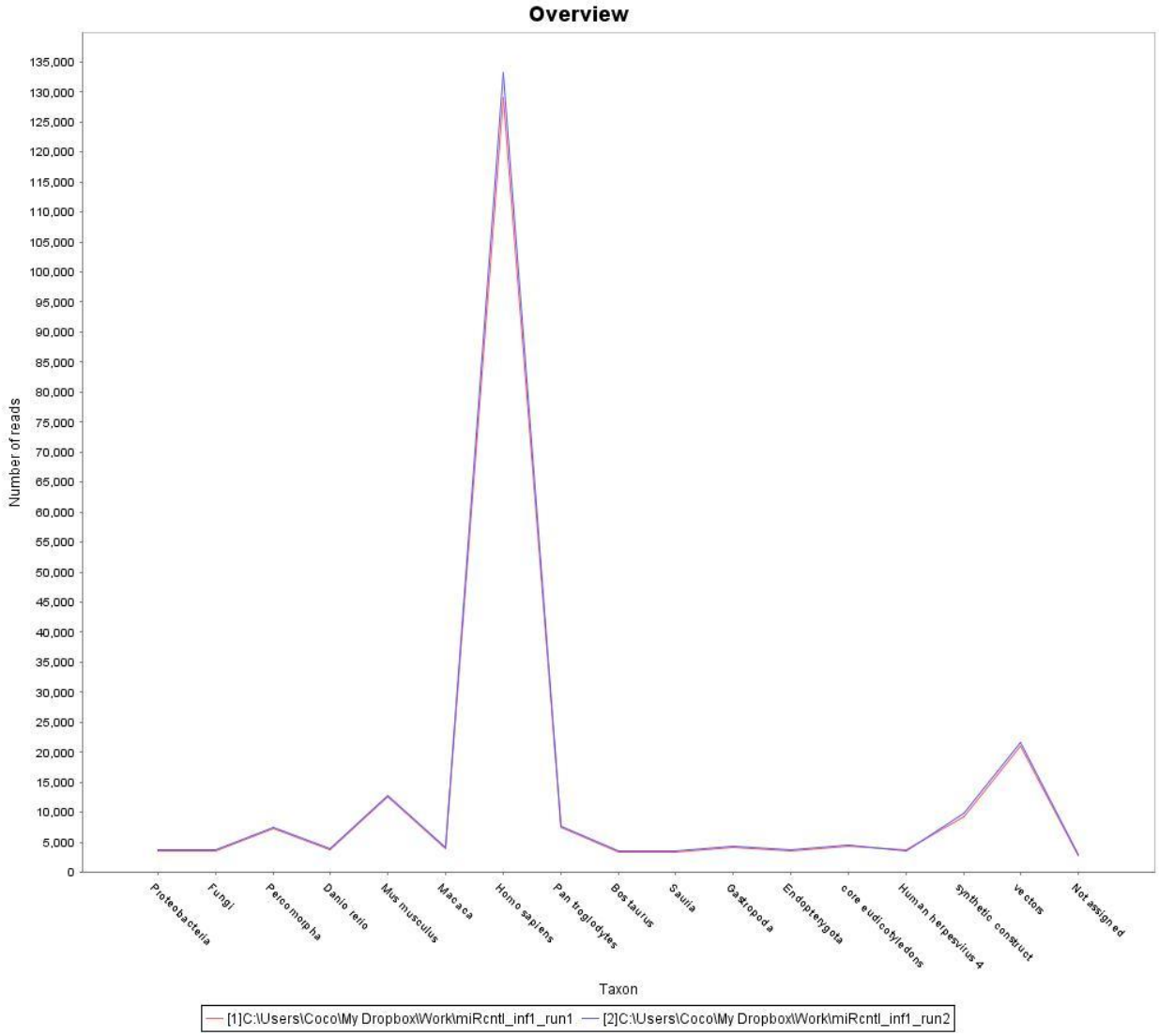

Figure 15 Little variation at species level with similar, strict LCA parameters.

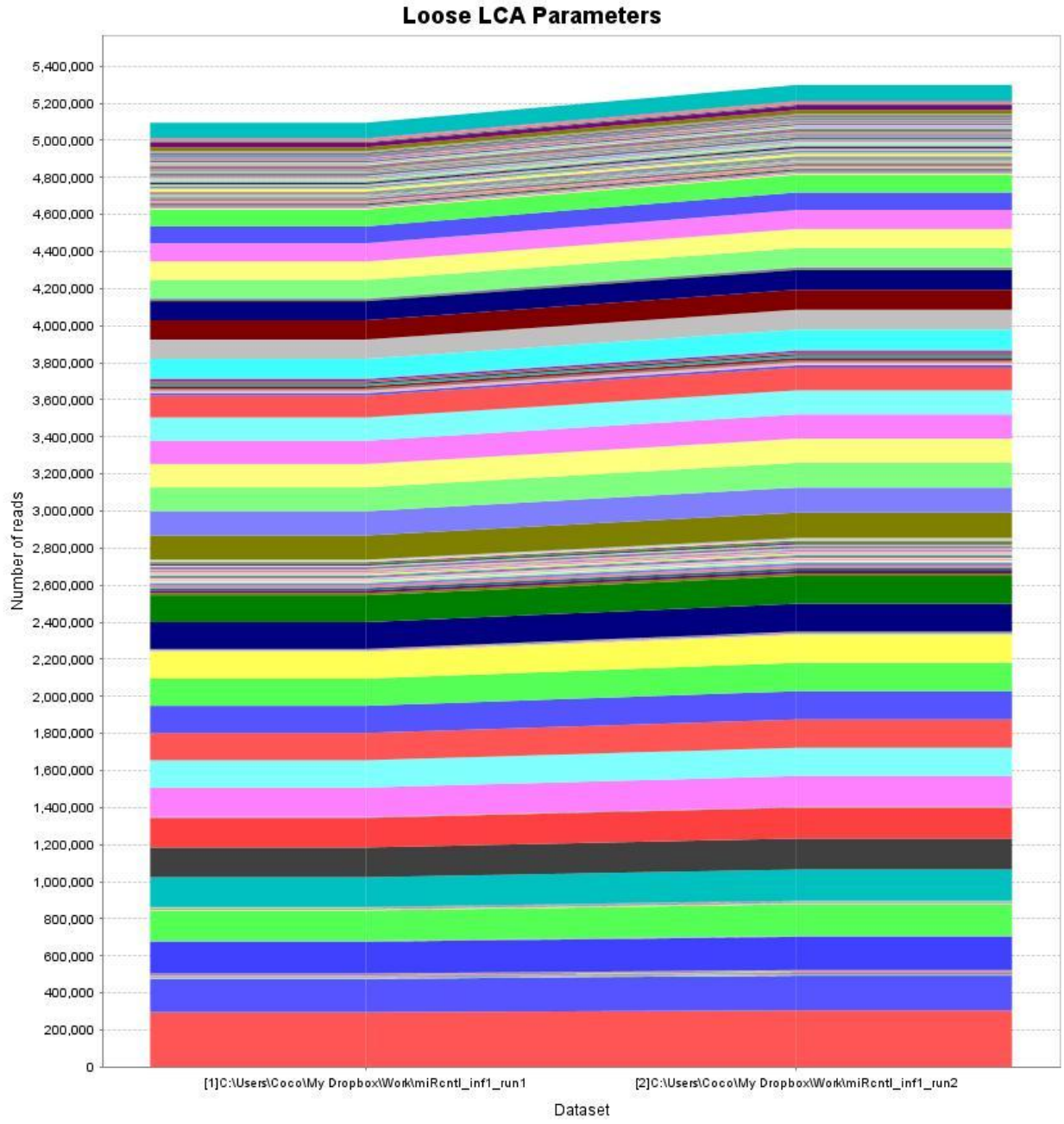

Figure 16 Few differences across entire data set even with loose LCA parameters.

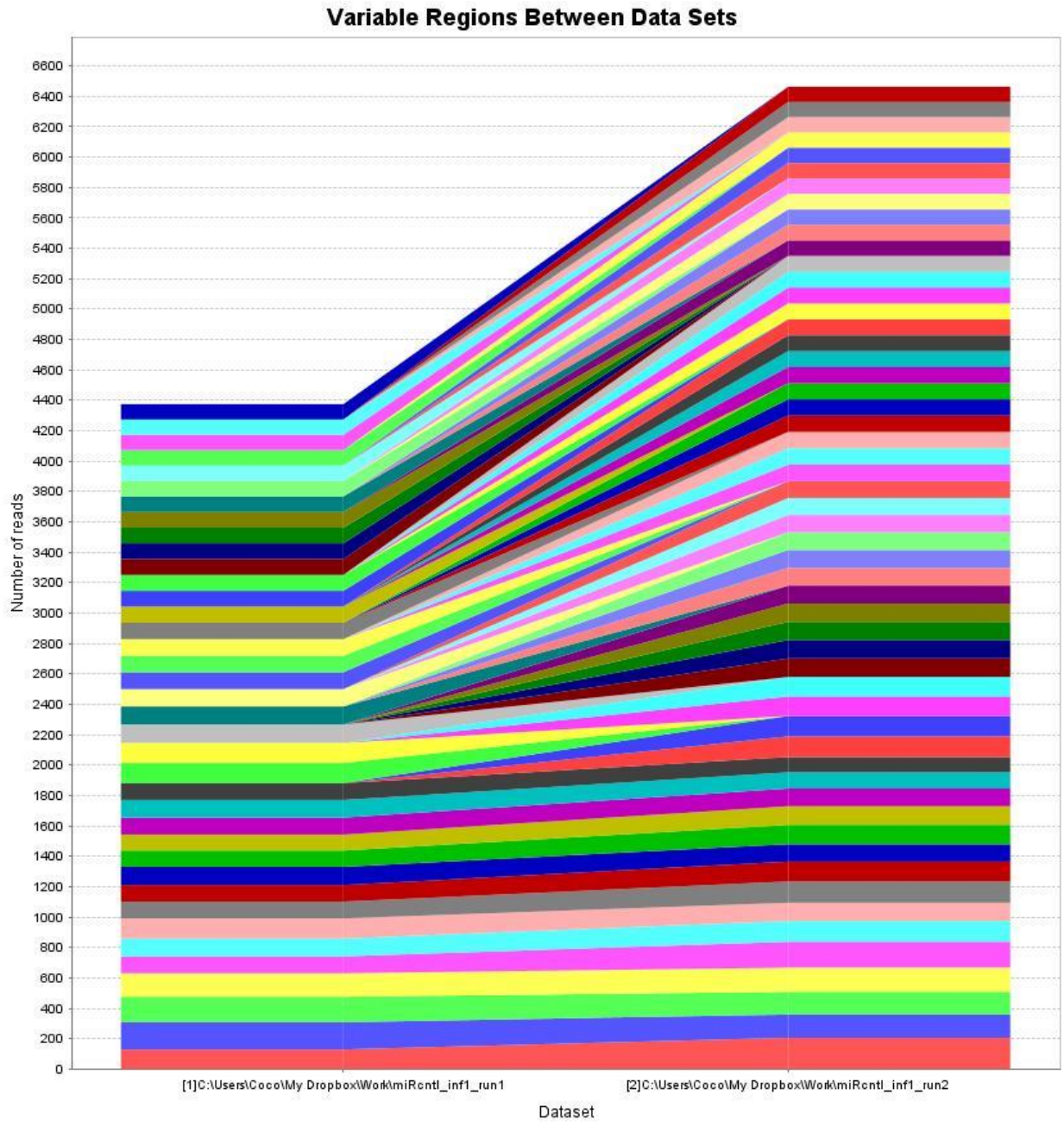

Figure 17 Selected highly variable taxonomies between data sets. These regions only represent 10,834 reads between both data sets of the 51,427,652 possible reads which is 0.02% of the total reads.

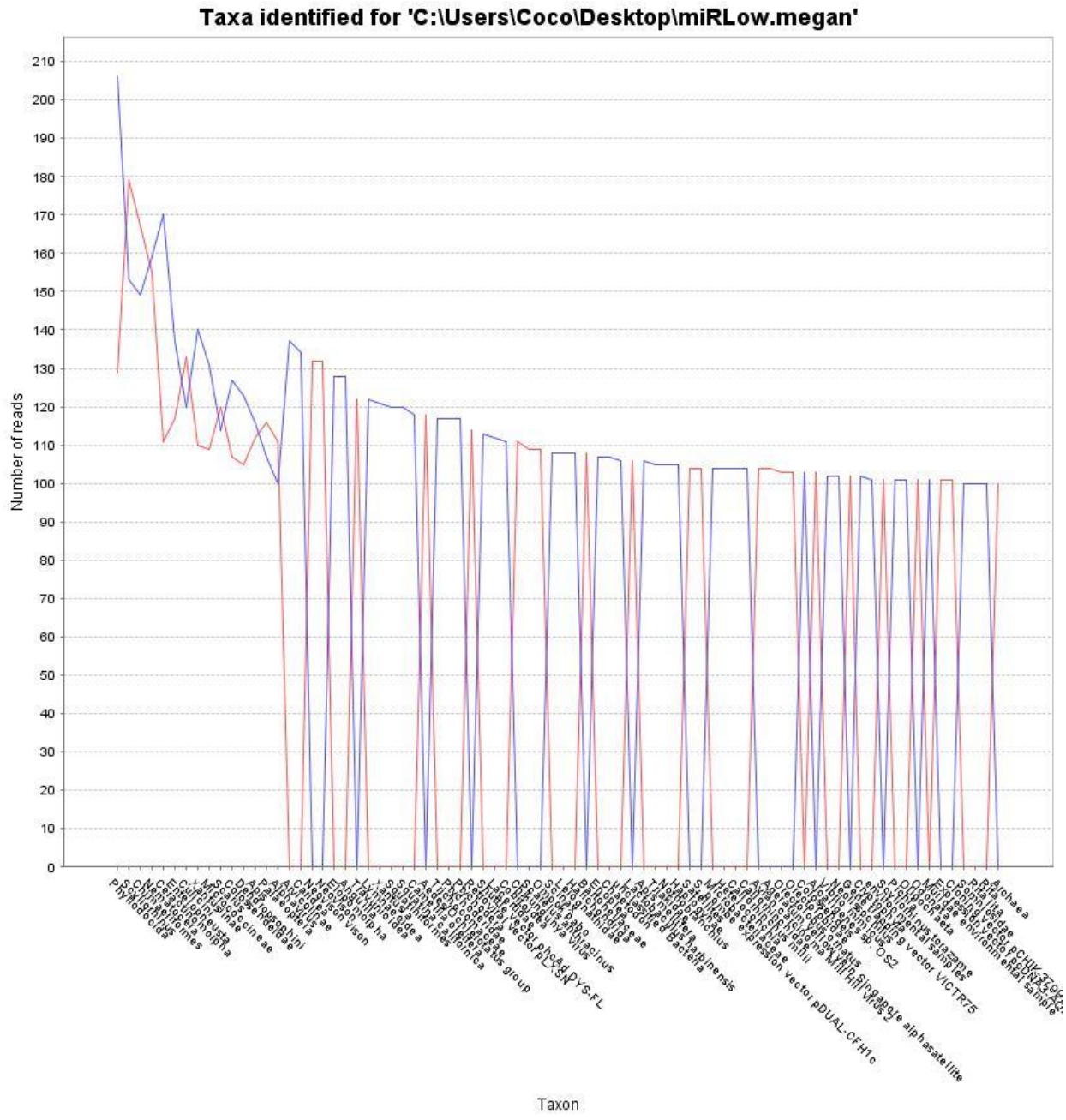

Figure 18 Selected highly variable taxonomies between data sets. These regions only represent 10,834 reads between both data sets of the 51,427,652 possible reads which is 0.02% of the total reads.

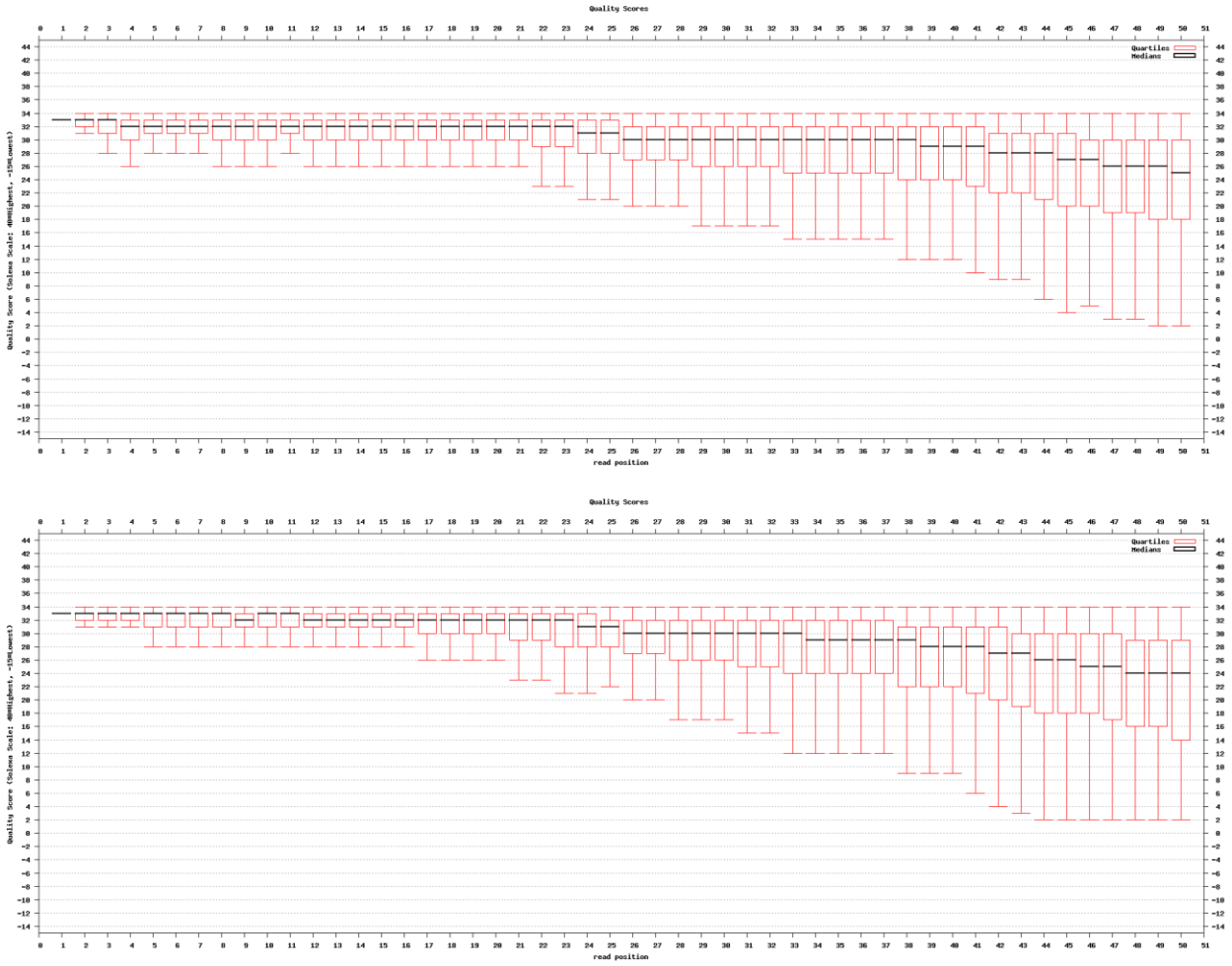

Figure 19 Quality scores of both data sets similar and of reasonable quality. Top represents first data set and bottom the second.

Noise vs. Contamination or Exogenous Agents

De novo assembly can efficiently separate noise due to low quality reads in databases, low quality reads in the researcher's data set, or organism homology by assembling longer reads such that low quality base pairs are replaced with higher quality base pairs, provided enough coverage exists. For example, 5,258 50-bps reads which were classified as Pan troglodytes from the Mutu I data set were extracted and assembled which produced 21 reads (among others which were classified further up the hierarchy) with an average length of 88.85-bps. This effectively rules out Pan troglodytes as a potential exogenous agent or contaminant.

CHAPTER 5 - CONCLUSION

Cancer Research

With the exponential increase in sequencing throughput of recent years there has a large push in genome annotation, yet the exogenous agents are simply discarded or possibly receive a footnote in the publication [26]. PARSES addresses this, exemplifying the usefulness of such data, particularly in searching for etiology in cancers. In addition, it could be used to discover the origins of cancers. It is ultimately an effective, usable tool though it relies on the researcher's knowledge to determine if a taxonomy represents an exogenous agent or a contaminating agent.

Contamination

PARSES applies basic concepts of detecting and removing contamination. More thorough reproduction of contamination removal must be done before it can be accepted as a standard. PARSES can be used before data analysis to ensure sanitation, during data analysis to remove contaminants, or after data analysis to give further validity to a researcher's results. PARSES can potentially be used to offset the amount of contaminants recently discovered in DNA databases by simply appending it to any future data sets which would be used in those databases and running it on current databases to invalidate questionable sequences.

Process

PARSES is available at GitHub via https://github.com/Lythimus/PARSES. Detailed documentation including an overview of PARSES's technologies, tools, requirements, installation information, features, configuration, results, et cetera is also available.

Alternate Software

Framework

Though the framework for PARSES is extensive and easily extendable, rather than developing an entire framework, it would have been simpler in terms of maintenance, usability, and development to use or extend an existing framework such as Galaxy. Galaxy supports many of the tools used in PARSES and extending it is a simple process of crafting XML files. All of the installers could be added to Galaxy and though it is not designed for GUI-based tools such as MEGAN they can be included. The researcher would have a wealth of tools at their disposal to manipulate their data and could customize the pipeline with the tools they are most familiar with or their data is best catered to [32] [33].

```
<tool id="abyssKmerOpt" name="Abyss with Kmer Optimization" version="1.0.0">
<description>Assemble short unpaired reads with optimum kmer</description>
<command interpreter="perl">abyssKmerOptimizer.pl $infile $minKmer $maxKmer $qualityType
$parallel $outfile</command>
```

```
<inputs>
  <param name="infile" type="data" format="fasta|fastq" label="Unpaired read
sequences" />
  <param name="minKmer" type="integer" value="7" label="Min Kmer" help="Minimum
kmer length" min="7" />
    <param name="maxKmer" type="integer" value="50" label="Max Kmer"
help="Maximum kmer length, not to exceed read length" min="7" max="96" />
  <param name="qualityType" type="select" force_select="true" label="Quality Type">
    <option value="--illumina-quality">Illumina</option>
    <option value="--standard-quality">Standard</option>
   </param>
   <param name="parallel" type="boolean" label="Parallel execution on kmers"
checked="true" truevalue="true" falsevalue="''" help="Requires perl module
Parallel::Iterator" />
</inputs>
<outputs>
   <data name="outfile" format="fasta" />
</outputs>

<help>
**What it does**

ABySS is a de novo sequence assembler that is designed for short reads. It is run with all the k-mers in
ranged and the contigs with the optimum k-mer computed via N50 score is returned.

.. image:: http://www.bcgsc.ca/platform/bioinfo/software/abyss/screenshot

**Reference**

http://www.bcgsc.ca/platform/bioinfo/software/abyss
</help>
</tool>
```

Code Snippet 1 ABySS parallel executions Galaxy module.

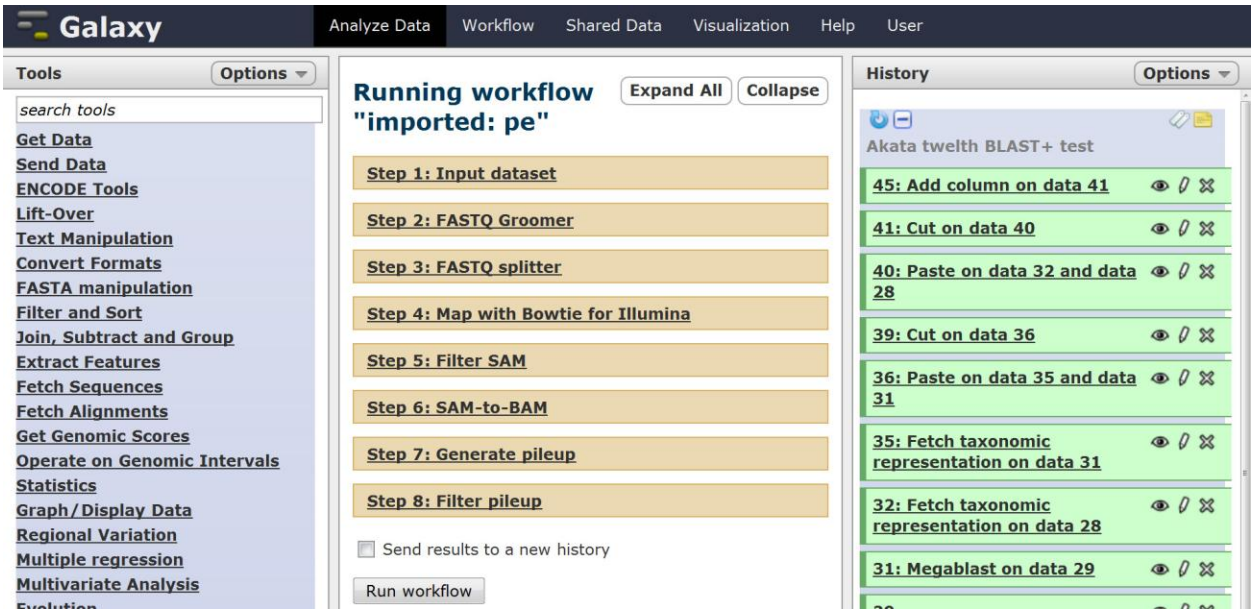

Figure 20 A workflow in the popular Galaxy web interface.

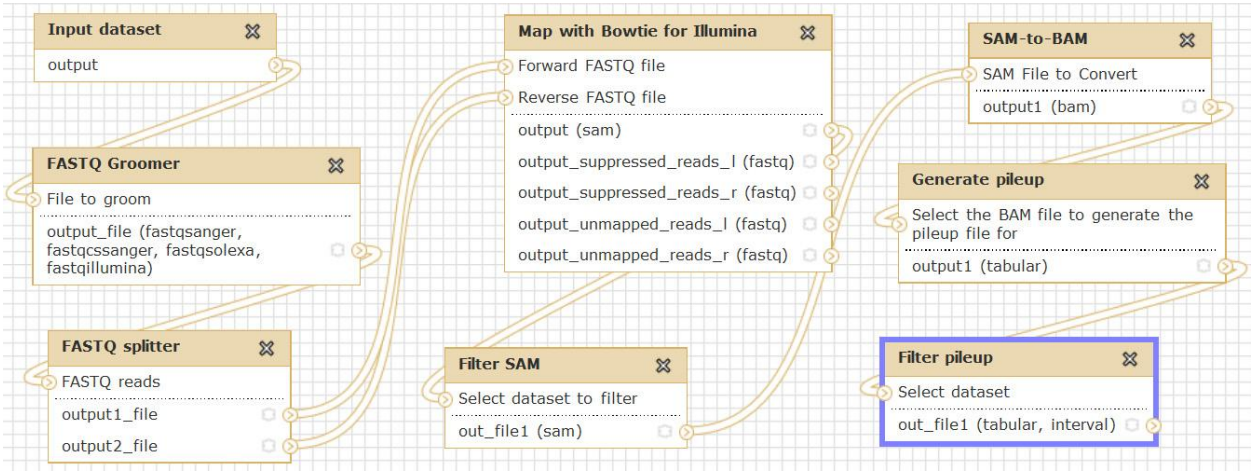

Figure 21 A graphical representation of a workflow in Galaxy web interface.

Taxonomical Analysis

Due to computational and accuracy concerns a composition-based taxonomical analysis was considered over a similarity-based method as a consequence of technical issues with using Phymm and the unavailability of other tools such as RAIphy. Research has shown that PhymmBL and RAIphy perform better than simply BLAST+ in general and extensively better on short reads [20].

De Novo Assembly

Velvet was also considered for De Novo Assembly, which is similar in function to ABySS except it is more computationally expensive and limits the possible k-mer choices [34]. It received a greater N50 score on multiple executions but not consistently greater and often negligible. Combining Velvet and ABySS and

ranking the results by N50 was considered but it was determined ABySS is sufficient and it addresses the computational and time issues affecting PARSES.

Dependency Resolution

Make, a more popular dependency resolution tool based on shell scripting, has the basic functionality of Rake, but it is by far more difficult to use. For instance, there are many cases in which whitespace usage drastically changes the interpretation of code as seen in Code Snippet **2**.

```
var1 = 23   # 'var1=23' is correct.
# On line above, Bash attempts to execute command "var1" with the arguments "=" and "23".
```

Code Snippet 2 Example of Make's whitespace parsing.

Also, numerical and string comparison operations are different, shell scripts are not guaranteed to perform comparably in different environments, shells do not necessarily have access to subshell's environment variables, et cetera. These issues plaguing make render it difficult to work with large on complex systems [35].

APPENDIX A: PARSES CODE

Rakefile.rb

```
require 'logger'
require 'yaml'
require 'net/http'
require 'net/ftp'
require 'uri'
require 'rake/clean'
require 'csv'

##### DISCLAIMER
# This product may not be used for any sort of financial gain. Licenses for both MEGAN and Novoalign are strictly for non-profit research at non-profit institutions and academic usage.

PROG_NAME = 'PARSES'
VER = '0.44'
PROG_DIR = File.dirname(__FILE__)
MEGAN_EXPANSION = 'expand direction=vertical; update;'

# Execute a bash command, time it, and log it.
def safeExec(command, log, sequence, errorMessage)
	results=`{ /usr/bin/time #{command} } 2> .time`
	if $?.exitstatus != 0
		puts "#{errorMessage} (status = #{$?.exitstatus})"
		log.info("#{errorMessage} (status = #{$?.exitstatus})")
	else
		timeFile = File.open(".time")
		File.open(".time").first =~ /(\d+).\d+ real/
		timeFile.close
		sequence.executionTime = sequence.executionTime + $1.to_i
		log.info("#{command} execution time:
#{Time.at($1.to_i).gmtime.strftime('%R:%S')}")
	end
	return $?.exitstatus, results
end

# Determine if software is installed
def command?(name)
	`which #{name}`
	$?.success?
end

# Determine if installing an application
installMode = false
cleanMode = false
for arg in ARGV do
```

```
        installMode = true if arg =~ /install|index|clobber|clean/i
        cleanMode = true if arg =~ /clobber|clean/i
end

# Columnar representation of a Novoalign file
class NOVOFILE
        READ_HEADER=0
        READ_TYPE=1
        SEQUENCE=2
        QUALITY=3
        ALIGNMENT_TYPE=4
        ALIGNMENT_SCORE=5
        ALIGNMENT_QUALITY=6
        REVERSE_COMPLEMENT_ALIGNMENT_SCORE=7
        ALIGNMENT_HEADER=8
        ALIGNMENT_OFFSET=9
        ALIGNMENT_STRANDEDNESS=10
        PAIRED_END1=11
        PAIRED_END2=12
        PAIRED_END3=13
        MISMATCHES_INSERTIONS_DELETEIONS=14
        HUMAN_TAXONOMY_NUMBER=9606
end

# Links to each program which may be used should the tool not be automatically downloaded properly
class ProgramRepository
        attr_accessor :megan, :tophat, :abyss, :novoalign, :blast, :bowtie, :samtools, :hg
end

def numberOfHostOrganismReads(sequence)
        output = File.open( "#{ENV['seq']}.csv", 'wb' )
        taxons = `MEGAN +g -f "#{sequence.abyssPath}" -x "select all; update; uncollapse
all; update; uncollapse subtrees; update; list summary=all; quit;"`
        hostOrganism = notAssigned = noHits = exogenousAgents = 0
        hostOrganism = $1.chomp.to_i + `egrep -cv '>'
"#{sequence.blast1Path}"`.chomp.to_i + `egrep -c '^[ACTGN]'
"#{sequence.removeNonMappedPath}"`.chomp.to_i if taxons=~
/#{sequence.hostOrganismName}: (\d+)/i
        notAssigned = $1.chomp.to_i if taxons=~ /Reads unassigned: (\d+)/i
        noHits = $1.chomp.to_i if taxons=~ /Reads with no hits: (\d+)/i
        exogenousAgents = $1.chomp.to_i if taxons=~ /Reads assigned: (\d+)/i
        output.write("#{sequence.hostOrganismName},#{hostOrganism}\nNot
Assigned,#{notAssigned}\nNo Hits,#{noHits}\nExogenous
Agents,#{exogenousAgents}")
        output.close
        `MEGAN +g -E -x "import format=CSV file='#{ENV['seq']}.csv'; exportchart
format=PDF file='#{ENV['seq']}.breakdown.pdf' data=taxa type=bar; quit;"`
end

# Create MEGAN parsable CSV file from a Novoalign FASTA output file.
def novoToMeganCsv(novoFile)
        output = File.open( "#{novoFile}.csv", 'wb' )
```

```
        CSV.foreach(novoFile) do |row|
                rowSplit = row.to_s.split("\t")

output.write("#{rowSplit[NOVOFILE::READ_HEADER]},#{NOVOFILE::HUMAN_TAXONOMY
_NUMBER},#{rowSplit[NOVOFILE::ALIGNMENT_SCORE]}\n") unless rowSplit[0][0].chr
== '#'
        end
        output.close
end

# Follow redirecting links. Used to obtain latest versions of files.
def fetch(uri_str, limit = 10)
        raise ArgumentError, 'HTTP redirect too deep' if limit == 0
        response = Net::HTTP.get_response(URI.parse(uri_str))
        case response
        when Net::HTTPSuccess     then response
        when Net::HTTPRedirection then fetch(response['location'], limit - 1)
        else response.error!
        end
end

# Program settings such as paths to various databases and indices
class Settings
        attr_accessor :humanGenomeDatabase, :ntDatabase, :giTaxIdNuclDatabase,
:bowtieIndex, :novoIndex
end

# Load program configuration file
if File::exists?(File.expand_path("~/.#{PROG_NAME}")) #Program has been executed before
        progSettingsFile = File.open(File.expand_path("~/.#{PROG_NAME}"))
        progSettings = YAML.load(progSettingsFile)
else
        progSettings = Settings.new
end



### GATHER SEQUENCE RUN INFORMATION FROM PREVIOUS RUNS
seqName = "#{ENV['seq']}".chomp
abort 'Must specify a sequence to analyze via the seq= option' if
seqName.empty? and !installMode
abort 'Perl must be installed.' if !command? 'perl' and !installMode
abort 'gcc must be installed' if !command? 'gcc'
seqFileName = "#{ENV['file']}".chomp
dataType = "#{ENV['type']}".chomp
useRepo = "#{ENV['repo']}".chomp
truncate = "#{ENV['truncate']}".chomp
forceFile = "#{ENV['forcefile']}".chomp
indexName = "#{ENV['indexName']}".chomp
indexGlob = "#{ENV['indexGlob']}".chomp
```

```
# Specify properties about this individual data set/run
class Sequence
	attr_accessor :readLength, :dataType, :hostOrganismName, :executionTime, :filePath,
:numOfReads, :novoalignPath, :removeNonMappedPath, :blast1Path, :megan1Path,
:abyssPath, :abyssPathGlob, :minKmerLength, :maxKmerLength, :blastPathGlob,
:blast2Path, :eValue1, :eValue2, :blastOutputFormat, :megan2Path, :expansionNumber,
:maxMatches, :minScoreByLength, :topPercent, :winScore, :minSupport, :useCogs,
:useGos, :useSeed, :useKegg, :imageFileType, :pipeEndPath
	def initialize(filePath, dataType)
		# If the file exists, get its number of reads and the read length
		if File.exists?(filePath)
			f = File.new(filePath)
			line = f.gets
			line = f.gets while line[0] =~ /\W/
			@readLength=line.length
			f.close
			@numOfReads="#{`egrep -c '^[ACTGN]' "#{filePath}"`}".chomp.to_i
		else
			filePath=''
		end
		@dataType=dataType
		@hostOrganismName="Homo sapiens"
		@executionTime = 0.0 # The total amount of time is seconds this run has receive
processing time
#PATHS
		@filePath=filePath
		@novoalignPath="#{filePath}.novo"
		@removeNonMappedPath="#{novoalignPath}.NM.fasta"
		@blast1Path="#{removeNonMappedPath}.nospans"
		@megan1Path="#{blast1Path}.blast"
		@abyssPath="#{megan1Path}.megan.rma"
		@abyssPathGlob="reads-*.fasta"
		@blastPathGlob="#{abyssPathGlob}.*.kmer.contigs.fa.kmerOptimized.fa"
		@blast2Path="#{abyssPath}.kmerOptimized.fa"
		@megan2Path="#{blast2Path}.blast"
		@pipeEndPath="#{megan2Path}.megan.rma"
#ABYSS
		@minKmerLength=7
		@maxKmerLength=readLength
#BLAST
		@eValue1=0.001
		@eValue2=100
		@blastOutputFormat=6
#MEGAN
		@expansionNumber=10
		@minScoreByLength=35
		@topPercent=10.0
		@winScore=0.0
		@minSupport=0.01
		@useCogs='false'
		@useGos='false'
```

```
            @useSeed='false'
            @useKegg='false'
            @imageFileType='PDF'
        end
end

# Novoalign supports PRB, PRBnSEQ, QSEQ Illumina from Bustard, ABI Solid color space FASTA, ABI Solid color space with qual file, and color space FASTQ.
# Tophat supports colorspace FASTA and space-delimited interger files
# Data types supported
class DataType
        attr_accessor :novoalign, :tophat, :abyss
        def initialize(dataType)
            case dataType
            when 'fasta'
                @novoalign='-F FA'
                @tophat=''
                @abyss = ''
            when 'sanger'
                @novoalign='-F STDFQ'
                @tophat=''
                @abyss='--standard-quality'
            when 'solexa'
                @novoalign='-F SLXFQ'
                @tophat='--solexa-quals'
                @abyss='--illumina-quality'
            when 'illumina1.3'
                @novoalign='-F ILMFQ'
                @tophat='--solexa1.3-quals'
                @abyss='--illumina-quality'
            when 'illumina1.5'
                @novoalign='-F ILMFQ'
                @tophat='--solexa1.3-quals'
                @abyss='--illumina-quality'
            end
        end
end
# Load information concerning this data set/run
if (!installMode)
        if File::exists?(".#{seqName}") #Sequence has been run before
            seqFile = File.open(".#{seqName}", "r+")
            sequence = YAML.load(seqFile)
            #Was a new file or file type entered from commandline?
            if seqFileName.empty?
                seqFileName = sequence.filePath
            else
                sequence.filePath = seqFileName
            end
            if dataType.empty?
                dataType = sequence.dataType
            else
```

```
            sequence.dataType = dataType
        end
    elsif seqFileName.empty? or dataType.empty? #Sequence has not been run and a file or data type is not specified
        abort "Must specify a file of data to analyze for the first execution of the process as well as it's data type via the file= and type= options" if !installMode
    else #Sequence has not been run, but file and file type are specified
        seqFile = File.open(".#{seqName}", "w+")
        sequence = Sequence.new(seqFileName, dataType)
    end
    setDataTypes = DataType.new(sequence.dataType)

    log = Logger.new("#{seqName}.log")
    log.info("Begin run for seq=#{seqName} file=#{seqFileName} type=#{dataType} task=#{ARGV[-1]}")
else
    sequence = Sequence.new('~', 'solexa') #garbage data just to get things installed
end

# Clean and clobber
if cleanMode
    seqNameEmpty = seqName.empty?
    seqName = FileList[".[a-zA-Z0-9]*"] if seqNameEmpty #CLEAN or CLOBBER all files if no sequence is specified
    seqName.each{ | sn |
        seqFile = File.open(".#{sn}")
        seqFileName = YAML.load(seqFile).filePath
        seqFileName = YAML.load(seqFile).filePath if seqNameEmpty and
        CLEAN.include("#{sn}_tophat_out")
        CLEAN.include("#{seqFileName}.novo.counts")
        CLEAN.include("#{seqFileName}.novo.NM.fasta.nospans.details")
        CLEAN.include("#{seqFileName}.novo.NM.fasta.nospans.mergedBlast")
        CLEAN.include("#{seqFileName}.novo.NM.fasta.nospans.[a-z][a-z]")
        CLEAN.include("#{seqFileName}.novo.NM.fasta.nospans.[a-z][a-z].blast")
        CLEAN.include("reads-*.fasta.*.kmer.contigs.fa")
        CLEAN.include("reads-*.fasta.*.kmer.contigs.coverage")
        CLEAN.include(".time")
        CLOBBER.include("#{seqFileName}.*")
        CLOBBER.include("reads-*")

CLOBBER.exclude("#{seqFileName}.novo.NM.fasta.nospans.blast.megan.rma.kmerOptimized.fa.blast.megan.rma")

CLOBBER.exclude("#{seqFileName}.novo.NM.fasta.nospans.blast.megan.rma.kmerOptimized.fa.blast.pdf")
        CLOBBER.include(".#{sn}")
        CLOBBER.include("#{sn}.log")
    }
end
```

```
### GATHER SYSTEM INFORMATION

# Determine which program to use to find files and find the first hit
locate = (command? 'locate') && (!ENV['LOCATE_PATH'].to_s.empty? or File.exists?
'/var/lib/mlocate/mlocate.db')
def findFile(filename, locate)
	path = ''
	path = `locate #{filename} | head -1`.chomp if locate
	path = `find / -name #{filename} | head -1`.chomp if path.to_s.empty?
	return path.to_s
end

# Find a novo index of the specificied criteria, make one if not found
def buildNovoIndex(name, pathGlob)
	novoIndex=findFile("#{name}*.ndx", locate)
	if novoIndex.to_s.empty?
		novoIndex="#{File.dirname(pathGlob)}/#{name}.ndx"
		`novoindex "#{novoIndex}" "#{pathGlob}"`
	end
	return novoIndex.to_s.chomp
end

# Find a bowtie index of the specificied criteria, make one if not found
def buildBowtieIndex(name, pathGlob)
	bowtieIndex=findFile("#{name}*.ebwt", locate)
	bowtieIndex= $1 if bowtieIndex =~ /(.*#{name}.*)\.\d+\.ebwt/i
	resourceFiles=''
	if bowtieIndex.to_s.empty?
		bowtieIndex="#{File.dirname(pathGlob)}/#{name}"
		FileList[pathGlob].each do |filename|
			resourceFiles << filename + ','
		end
		resourceFiles.chomp!(',')
		sh %{
			cd "#{File.dirname(pathGlob)}";
			bowtie-build "#{resourceFiles}" "#{name}";
		}
	end
	return bowtieIndex.to_s.chomp
end

# shell=`ps -p $$ | tail -1 | awk '{print $NF}'` #supposedly more accurate method to return shell, but is
returning sh instead of bash
shell = File.basename(ENV['SHELL']).chomp

# Represents operating system of current environment
class OS
	OSX=0
	LINUX=1
	BSD=2
```

```
        SOLARIS=3
        WINDOWS=4
end

# Determine OS
osName = `uname -s`.chomp!
case osName
when 'Darwin'
        os = OS::OSX
when 'Linux'
        os = OS::LINUX
when /BSD/
        os = OS::BSD
when 'SunOS'
        os = OS::SOLARIS
when /CYGWIN/
        os = OS::WINDOWS
end

arch = `uname -m` =~ /64/ ? 64 : 32 #Determine architecture
ncpu = -1
memInGigs = -1
progRepo = ProgramRepository.new

#Download program links repositories
case os
when OS::OSX
        ncpu = `sysctl -n hw.ncpu`.chomp.to_i # determine number of cpus for MacOS
        "#{`/usr/sbin/system_profiler SPHardwareDataType | grep Memory`}" =~
/Memory:\s+(\d+)\s*GB/; # determine amount of memory in MacOS
        memInGigs = $1.chomp.to_i
        if useRepo
                Net::HTTP.start('cloud.github.com', 80) { |http| # download repository for MacOS
                        progRepo =
YAML.load(http.get('/downloads/Lythimus/PARSES/.programRepositoryMac.txt').body)
                }
        end
when OS::LINUX
        `cat /proc/cpuinfo | grep -G processor.*:.* | tail -n 1` =~ /processor.*:.*(\d+)/
        ncpu = $1.chomp.to_i + 1 # determine number of cpus for Linux
        memInGigs = %x[echo `cat /proc/meminfo | grep MemTotal` | sed  "s/[^0-
9]//g"].to_i/2**20 # determine amount of memory in Linux
        if useRepo
                Net::HTTP.start('cloud.github.com', 80) { |http| # download repository for Linux
                        progRepo =
YAML.load(http.get('/downloads/Lythimus/PARSES/.programRepositoryLinux.txt').bod
y)
                }
        end
end

## consider adding time of processing to all logs
```

```
log.info("Executing #{PROG_NAME + ' v' + VER} in a #{osName} environment
with #{memInGigs}GB of memory and #{ncpu} processing cores with a #{arch}-
bit architecture.") if !installMode

########## PIPELINE
desc 'Novoalign - Align reads in to base genome'
task :alignSequence => [:novoalignInstall, :hgInstall, :novoIndex]
file sequence.novoalignPath => sequence.filePath do
      puts 'Sequence Alignment'
      if indexName.empty? or indexGlob.empty?
            novoIndex = progSettings.novoIndex
      else
            novoIndex = buildNovoIndex(indexName, indexGlob)
      end
      seqFileName = sequence.filePath if forceFile != 'true'
      exitStatus = safeExec("novoalign -d \"#{novoIndex}\"
#{setDataTypes.novoalign} -f \"#{seqFileName}\" >
\"#{sequence.novoalignPath}\";", log, sequence,
                  'Novoalign sequence alignment not performed')
      if exitStatus == 0
            readsLeft=`egrep -c '@' "#{sequence.novoalignPath}"`
            log.info("#{readsLeft} human reads discovered which is
#{sequence.numOfReads/readsLeft*100}% of total.")
      end
end

desc 'Xnovotonm - Harvest non-base organism reads'
task :removeHuman => :alignSequence
file sequence.removeNonMappedPath => sequence.novoalignPath do
      puts 'RemoveHuman'
      seqFileName = sequence.novoalignPath if forceFile != 'true'
      exitStatus = safeExec("\"#{PROG_DIR}/Xnovotonm.pl\" \"#{seqFileName}\";",
log, sequence,
                  'Removal of Human mapped reads from novoalign results not
performed')
      if exitStatus == 0
            readsLeft=`egrep -c '^[ACTGN]' "#{sequence.removeNonMappedPath}"`
            log.info("Reduced to #{readsLeft} non-mapped reads which is
#{sequence.numOfReads/readsLeft*100}% of total.")
      end
end

desc 'Tophat - Align spanning reads in order to remove base organism reads'
task :removeSpans => [:bowtieIndex, :tophatInstall, :removeHuman]
file sequence.blast1Path => sequence.removeNonMappedPath do
      puts 'RemoveSpans'
      ## DIDN'T IMPLEMENT forceFile BECAUSE TOO COMPLICATED
      if indexName.empty? or indexGlob.empty?
            bowtieIndex = progSettings.bowtieIndex
      else
            bowtieIndex = buildBowtieIndex(indexName, indexGlob)
```

```
    end
    safeExec("tophat -p #{ncpu} #{setDataTypes.tophat} --output-dir
\"#{ENV['seq']}_tophat_out\" \"#{bowtieIndex}\" \"#{sequence.filePath}\";", log,
sequence,
            'TopHat sequence alignment not performed')
    safeExec("samtools view -h -o
\"#{ENV['seq']}_tophat_out/accepted_hits.sam\"
\"#{ENV['seq']}_tophat_out/accepted_hits.bam\";", log, sequence,
            'Samtools conversion not performed')
    safeExec("\"#{PROG_DIR}/Xextractspans.pl\"
\"#{ENV['seq']}_tophat_out/accepted_hits.sam\";", log, sequence,
            'Spanning region extraction not performed')
    exitStatus = safeExec("\"#{PROG_DIR}/Xfilterspans.pl\"
\"#{sequence.removeNonMappedPath}\"
\"#{ENV['seq']}_tophat_out/accepted_hits.sam.spans\";", log, sequence,
            'Filter of spanning regions not performed')
    if exitStatus == 0
        readsLeft=`egrep -cv '>' "#{sequence.blast1Path}"`
        log.info("Reduced to #{readsLeft} reads which is
#{sequence.numOfReads/readsLeft*100}% of total.")
    end
end

desc 'BLAST - Associate reads with organisms.'
task :localAlignReads => [:blastInstall, :ntInstall, :removeSpans]
file sequence.megan1Path => sequence.blast1Path do
    # This is looking sloppy :/
    puts 'localAlignReads'
    dust = "-dust no" if sequence.readLength < 50 # Dust filtering should be disabled for
short reads
    seqFileName = sequence.blast1Path if forceFile != 'true'
    pieces = ([(memInGigs/30), ncpu].min)
    pieces = 1 if pieces == 0
    pieceSize = `wc -l "#{seqFileName}"`.chomp.to_i / pieces
    pieceSize = pieceSize + (pieceSize % 2) # This needs to be changed if paired-end read suppor
tis ever implemented
    `split -l #{pieceSize} "#{seqFileName}" "#{seqFileName}."`
    blastPieces = FileList["#{seqFileName}.[a-zA-Z0-9][a-zA-Z0-9]"]
    piecesLeft = blastPieces.length
    fileCount = [pieces, blastPieces.length].min
    @blastCommands = []
    blastPieces.each { | blastPiece |
        @blastCommands << "blastn -db \'#{progSettings.ntDatabase}\' -
soft_masking true #{dust} -num_threads #{ncpu} -evalue #{sequence.eValue1} -
outfmt #{sequence.blastOutputFormat} -query \'#{blastPiece}\' -out
\'#{blastPiece}.blast\' &"
        fileCount = fileCount - 1
        if fileCount == 0
            piecesLeft = piecesLeft - pieces
```

```
                safeExec("\"#{PROG_DIR}/parallelBlast.sh\"
#{@blastCommands.join(' ')}", log, sequence, #This is not a space in the join, rather looks like a
space. It is used by the BASH script as a delimiter.
                        'BLAST of reads not performed')
                @blastCommands = []
                fileCount = [pieces, piecesLeft].min
            end
        }
        `cat #{seqFileName}.[a-z][a-z].blast > #{seqFileName}.mergedBlast`
        safeExec("\"#{PROG_DIR}/addTaxon.pl\"
\"#{progSettings.giTaxIdNuclDatabase.to_s}\" \"#{seqFileName}.mergedBlast\"
\"#{seqFileName}\";", log, sequence,
                'Adding taxon to end of file not performed')
end

desc 'MEGAN - Separate reads into taxonomies.'
task :metaGenomeAnalyzeReads => [ :meganInstall, :localAlignReads]
file sequence.abyssPath => sequence.megan1Path do
        puts 'metaGenomeAnalyzeReads'
        ## DIDN'T IMPLEMENT forceFile BECAUSE TOO COMPLICATED

        `MEGAN +g -V -E -x 'version; quit;'` =~ /MEGAN.*version\s*(\d+\.?\d*)/
        if ($1.to_f < 4.0)
            safeExec("MEGAN +g -E -x \"import blastfile='#{sequence.megan1Path}'
readfile='#{sequence.blast1Path}' meganfile='#{sequence.abyssPath}'
minscore=#{sequence.minScoreByLength} toppercent=#{sequence.topPercent}
winscore=#{sequence.winScore} minsupport=#{[(sequence.minSupport*`egrep -cv '>'
"#{sequence.blast1Path}"`.chomp.to_i).to_i, 5].max} summaryonly=false
usecompression=true usecogs=#{sequence.useCogs} usegos=#{sequence.useGos}
useseed=false; #{MEGAN_EXPANSION*sequence.expansionNumber} uncollapse all;
update; exportgraphics format='#{sequence.imageFileType}'
file='#{sequence.megan1Path + '.' + sequence.imageFileType.downcase}'
REPLACE=true; quit;\";", log, sequence,
                'MEGAN processing of BLASTed reads not performed')
            exitStatus, results = safeExec("MEGAN -f \"#{sequence.abyssPath}\" -x
\"#{MEGAN_EXPANSION*sequence.expansionNumber} uncollapse all;\";", log,
sequence,
                'Opening MEGAN file not performed')
        else
            safeExec("MEGAN +g -E -x \"import blastfile='#{sequence.megan1Path}'
fastafile='#{sequence.blast1Path}' meganfile='#{sequence.abyssPath}'
minscore=#{sequence.minScoreByLength} toppercent=#{sequence.topPercent}
winscore=#{sequence.winScore} minsupport=#{[(sequence.minSupport*`egrep -cv '>'
"#{sequence.blast1Path}"`.chomp.to_i).to_i, 5].max} usecogs=#{sequence.useCogs}
useseed=#{sequence.useSeed} usekegg=#{sequence.useKegg};
#{MEGAN_EXPANSION*sequence.expansionNumber} select nodes=all; uncollapse
subtrees; update; exportimage file= '#{sequence.megan1Path + '.' +
sequence.imageFileType.downcase}' format= '#{sequence.imageFileType}'
REPLACE=true; quit;\";", log, sequence,
                'MEGAN processing of BLASTed reads not performed')
```

```
        exitStatus, results = safeExec("MEGAN -f \"#{sequence.abyssPath}\" -x
\"#{MEGAN_EXPANSION*sequence.expansionNumber} uncollapse all;\";", log,
sequence,
                'Opening MEGAN file not performed')
    end
    if exitStatus == 0
        totalReads = $_.chomp.to_i if (results =~ /Total reads:\s*(\d+)/)
        assignedReads = $_.chomp.to_i if (results =~ /Assigned reads:\s*(\d+)/)
        unassignedReads = $_.chomp.to_i if (results =~ /Unassigned reads:\s*(\d+)/)
        noHits = $_.chomp.to_i if (results =~ /Reads with no hits:\s*(\d+)/)
        log.info("Potential Exogenous Reads: #{totalReads}. Assigned Reads:
#{assignedReads} which is #{assignedReads/sequence.numOfReads}% of entire
data set. Unassigned Reads: #{unassignedReads} which means LCA hides
#{unassignedReads/totalReads}% of the data. No hits: #{noHits} which is
#{noHits/sequence.numOfReads} of the entire data set.")
    end
end

desc 'ABySS - Assemble reads associated with clusters of taxonomies.'
task :denovoAssembleCluster => [:abyssInstall, :parallelIteratorInstall,
:metaGenomeAnalyzeReads]
file sequence.blast2Path => FileList["#{sequence.abyssPathGlob}"] do
    puts 'denovoAssemblyCluster'
    seqFileName = sequence.abyssPathGlob if forceFile != 'true'
    exitStatus=-1
    FileList["#{seqFileName}"].each { | abyssFiles |
        exitStatus = safeExec("\"#{PROG_DIR}/abyssKmerOptimizer.pl\"
#{abyssFiles} #{sequence.minKmerLength} #{sequence.maxKmerLength}
#{setDataTypes.abyss};", log, sequence,
                'ABySS not performed')
    }
    `cat #{sequence.blastPathGlob} > #{sequence.blast2Path}` if forceFile != 'true'
    if exitStatus == 0
        coverageThreshold = $_ if (results =~ /(Using a coverage threshold of \d+)/)
        medianKmerCoverage = $_ if (results =~ /(The median k-mer coverage is
\d+)/)
        assembly = $_ if (results =~ /(Assembled \d+ k-mer in \d+ contigs)/)
        FileList["#{sequence.blastPathGlob}"].first =~
/#{sequence.abyssPathGlob}.(\d+)\.kmer\.contigs\.fa\.kmerOptimized\.fa/ if forceFile !=
'true'
        kmer = $_
        log.info("#{coverageThreshold}. #{medianKmerCoverage}. #{assembly}.
The optimum kmer is #{kmer}.")
    end
end

desc 'BLAST - Associate contigs with organisms.'
task :localAlignContigs => [:blastInstall, :ntInstall, :denovoAssembleCluster]
file sequence.megan2Path => sequence.blast2Path do
    puts 'localAlignContigs'
    seqFileName = sequence.blast2Path if forceFile != 'true'
```

```
        safeExec("blastn -db \"#{progSettings.ntDatabase}\" -soft_masking true -
num_threads #{ncpu} -evalue #{sequence.eValue2} -outfmt
#{sequence.blastOutputFormat} -query \"#{seqFileName}\" -out
\"#{sequence.megan2Path}.noTax\";", log, sequence,
                'BLAST of contigs not performed')
        safeExec("\"#{PROG_DIR}/addTaxon.pl\"
\"#{progSettings.giTaxIdNuclDatabase.to_s}\" \"#{sequence.megan2Path}.noTax\"
\"#{seqFileName}\";", log, sequence,
                'Adding taxon to end of BLAST contigs file not performed')
end

desc 'MEGAN - Separate contigs into taxonomies.'
task :metaGenomeAnalyzeContigs => [ :meganInstall, :localAlignContigs]
file sequence.pipeEndPath => sequence.megan2Path do
        ## DIDN'T IMPLEMENT forceFile BECAUSE TOO COMPLICATED
        puts 'metaGenomeAnalyzeContigs'
        `MEGAN +g -V -E -x 'version; quit;'` =~ /MEGAN.*version\s*(\d+\.?\d*)/
        if ($1.to_f < 4.0)
                safeExec("MEGAN +g -E -x \"import blastfile='#{sequence.megan2Path}'
readfile='#{sequence.blast2Path}' meganfile='#{sequence.pipeEndPath}'
minscore=#{sequence.minScoreByLength} toppercent=#{sequence.topPercent}
winscore=#{sequence.winScore} minsupport=#{[(sequence.minSupport*`egrep -cv '>'
"#{sequence.blast2Path}"`.chomp.to_i).to_i, 5].max} summaryonly=false
usecompression=true usecogs=#{sequence.useCogs} usegos=#{sequence.useGos}
useseed=false; #{MEGAN_EXPANSION*sequence.expansionNumber} uncollapse all;
update; exportgraphics format='#{sequence.imageFileType}'
file='#{sequence.megan2Path + '.' + sequence.imageFileType.downcase}'
REPLACE=true; quit;\";", log, sequence,
                        'MEGAN processing of BLASTed contigs not performed')
                exitStatus, results = safeExec("MEGAN -f
\"#{sequence.pipeEndPath}.rma\";", log, sequence,
                        'Opening MEGAN file not performed')
        else
                safeExec("MEGAN +g -E -x \"import blastfile='#{sequence.megan2Path}'
fastafile='#{sequence.blast2Path}' meganfile='#{sequence.pipeEndPath}'
minscore=#{sequence.minScoreByLength} toppercent=#{sequence.topPercent}
winscore=#{sequence.winScore} minsupport=#{[(sequence.minSupport*`egrep -cv '>'
"#{sequence.blast2Path}"`.chomp.to_i).to_i, 5].max} usecogs=#{sequence.useCogs}
useseed=#{sequence.useSeed} usekegg=#{sequence.useKegg}; update; set
context=seedviewer; #{MEGAN_EXPANSION*sequence.expansionNumber} select
nodes=all; uncollapse subtrees; update; exportimage file=
'#{sequence.megan2Path + '.' + sequence.imageFileType.downcase}' format=
'#{sequence.imageFileType}' REPLACE=true; quit;\";", log, sequence,
                        'MEGAN processing of BLASTed contigs not performed')
                exitStatus, results = safeExec("MEGAN -f
\"#{sequence.pipeEndPath}.rma\";", log, sequence,
                        'Opening MEGAN file not performed')
        end
        if exitStatus == 0
                totalReads = $_.chomp.to_i if (results =~ /Total reads:\s*(\d+)/)
```

```
            assignedReads = $_.chomp.to_i if (results =~ /Assigned reads:\s*(\d+)/)
            unassignedReads = $_.chomp.to_i if (results =~ /Unassigned reads:\s*(\d+)/)
            noHits = $_.chomp.to_i if (results =~ /Reads with no hits:\s*(\d+)/)
            log.info("Potential Exogenous Reads: #{totalReads}. Assigned Reads:
#{assignedReads}. Unassigned Reads: #{unassignedReads}. No hits: #{noHits}.")
      end
end



########## INSTALLATIONS

desc 'Install latest version of human genome database.'
task :hgInstall do
      if progSettings.humanGenomeDatabase.to_s.chomp.empty?
            progSettings.humanGenomeDatabase=File.dirname(findFile('chr*.fa', locate))
            if progSettings.humanGenomeDatabase.to_s.chomp == '.'
                  hg = ''
                  ftp = Net::FTP.new('hgdownload.cse.ucsc.edu')
                  ftp.login()
                  ftp.passive=true
                  ftp.chdir('apache/htdocs/goldenPath')
                  ftp.list("hg[0-9][0-9]").last =~ /.*(hg\d+).*/
                  hg = $1
                  ftp.chdir(hg + '/bigZips')
                  ftp.getbinaryfile('chromFa.tar.gz')
                  ftp.close
                  sh %{
                        mkdir /usr/share/#{hg};
                        tar -xzf chromFa.tar.gz -C /usr/share/#{hg};
                        rm chromFa.tar.gz;
                  }
                  progSettings.humanGenomeDatabase = '/usr/share/' + hg
            end
      end
end

desc 'Create an index for novoalign of the human genome database.'
task :novoIndex => [:hgInstall, :novoalignInstall] do
      if progSettings.novoIndex.to_s.empty?
            progSettings.novoIndex=buildNovoIndex('hgChrAll',
"#{progSettings.humanGenomeDatabase.to_s}/chr[0-9XY]*.fa")
      end
end

desc 'Create an index for bowtie/tophat of the human genome database.'
task :bowtieIndex => [:hgInstall, :bowtieInstall] do
      if progSettings.bowtieIndex.to_s.empty?
            progSettings.bowtieIndex=buildBowtieIndex('hgChrAll',
"#{progSettings.humanGenomeDatabase.to_s}/chr[0-9XY]*.fa")
      end
```

```
end

desc 'Install latest version of Novoalign.'
task :novoalignInstall do
	if !command? 'novoalign'
		novoalign = ''
		Net::HTTP.start('www.novocraft.com', 80) { |http|
			http.get('/main/releases.php', 'Referer' =>
'http://www.novocraft.com/').body =~ /(V\d+\.\d+\.\d+)/
			novoalign = $1
			File.open('novocraft.tar.gz', 'w'){ |file|
				if useRepo == true
					link = progRepo.novoalign
				else
					case os
					when OS::OSX then
						link = '/downloads/' + novoalign + '/novocraft' +
novoalign + '.MacOSX.tar.gz'
					when OS::LINUX then
						link = '/downloads/' + novoalign + '/novocraft' +
novoalign + '.gcc.tar.gz'
					end
				end
				file.write(http.get(link, 'Referer' =>
'http://www.novocraft.com/').body)
			}
		}
		sh %{
			tar -xzf novocraft.tar.gz -C /usr/bin;
			ln -s /usr/bin/novocraft/isnovoindex /usr/bin;
			ln -s /usr/bin/novocraft/novo2maq /usr/bin;
			ln -s /usr/bin/novocraft/novo2paf /usr/bin;
			ln -s /usr/bin/novocraft/novoalign /usr/bin;
			ln -s /usr/bin/novocraft/novobarcode /usr/bin;
			ln -s /usr/bin/novocraft/novoindex /usr/bin;
			ln -s /usr/bin/novocraft/novoutil /usr/bin;
		}
	end
end

desc 'Install latest version of Bowtie.'
task :bowtieInstall do
	if !command? 'bowtie'
		bowtie = ''
		Net::HTTP.start('bowtie-bio.sourceforge.net', 80) { |http|
			http.get('/index.shtml').body =~
/https:\/\/sourceforge\.net\/projects\/bowtie-bio\/files\/bowtie\/(\d*\.\d*\.\d*)/
		}
		Net::HTTP.start('softlayer.dl.sourceforge.net', 80) { |http|
			bowtie = $1
			File.open('bowtie-' + bowtie + '-src.zip', 'w'){ |file|
```

```
                        if useRepo == true
                                link = progRepo.bowtie
                        else
                                link = '/project/bowtie-bio/bowtie/' + bowtie + '/bowtie-
' + bowtie + '-src.zip'
                        end
                        file.write(http.get(link).body)
                }
        }
        sh %{
                unzip -d /usr/bin bowtie-#{bowtie}-src.zip;
                cd /usr/bin/bowtie-#{bowtie};
                make;
                ln -s /usr/bin/bowtie-#{bowtie}/bowtie /usr/bin;
                ln -s /usr/bin/bowtie-#{bowtie}/bowtie-build /usr/bin;
                ln -s /usr/bin/bowtie-#{bowtie}/bowtie-inspect /usr/bin;
        }
    end
end

desc 'Install latest version of Samtools.'
task :samtoolsInstall do
    if !command? 'samtools'
        samtools = ''
        File.open('samtools.tar.bz2', 'w'){ |file|
            if useRepo == true
                link = progRepo.samtools
            else
                link =
'http://sourceforge.net/projects/samtools/files/latest'
            end
            file.write(fetch(link).body)
        }
        `tar -jxf samtools.tar.bz2 -C /usr/bin`
        `rm samtools.tar.bz2`
        samtools = `ls /usr/bin | grep samtools | head -1`.chomp!
        sh %{
            cd /usr/bin/#{samtools};
            make;
            cp libbam.a /usr/lib;
            mkdir /usr/include/bam;
            cp *.h /usr/include/bam;
            ln -s /usr/bin/#{samtools}/samtools /usr/bin;
        }
    end
end

desc 'Install latest version of Tophat.'
task :tophatInstall => [:samtoolsInstall, :bowtieInstall] do
    if !command? 'tophat'
        tophat = ''
        Net::HTTP.start('tophat.cbcb.umd.edu', 80) { |http|
```

```
                http.get('/index.html').body =~ /\.\/downloads\/(tophat-
\d+\.\d+\.\d+\.tar\.gz)/
                tophat = $1
                File.open(tophat, 'w'){ |file|
                        if useRepo == true
                                link = progRepo.tophat
                        else
                                link = '/downloads/' + tophat
                        end
                        file.write(http.get(link).body)
                }
        }
        sh %{
                tar -xzf #{tophat} -C /usr/bin;
                rm #{tophat};
                cd /usr/bin/#{tophat.chomp!('.tar.gz')};
                ./configure;
                make;
                make install;
        }
    end
end

desc 'Install latest version of ABySS with Google Sparsehash.'
task :abyssInstall do
    if !command? 'ABYSS'
        Net::HTTP.start('code.google.com', 80) { |http|
                http.get('/p/google-sparsehash/').body =~ /(http:\/\/google-
sparsehash\.googlecode\.com)(\/files\/sparsehash-\d+\.\d+\.tar\.gz)/
        }
        base = File.basename($1)
        gsh = $2
        Net::HTTP.start(base, 80) { |http|
                File.open(File.basename(gsh), 'w'){ |file|
                        if useRepo == true
                                link = progRepo.abyss
                        else
                                link = gsh
                        end
                        file.write(http.get(link).body)
                }
        }
        sh %{
                tar -xzf #{File.basename(gsh)} -C /usr/bin;
                rm #{gsh};
                cd /usr/bin/#{File.basename(gsh, '.tar.gz')};
                ./configure;
                make;
                make install;
        }

        abyss = ''
```

```
            Net::HTTP.start('www.bcgsc.ca', 80) { |http|
                  http.get('/downloads/abyss/?C=N;O=D').body =~ /(abyss-
\d+\.\d+\.\d+\.tar\.gz)/
                  abyss = $1
                  File.open(abyss, 'w'){ |file|
                        file.write(http.get('/downloads/abyss/' + abyss).body)
                  }
            }
            sh %{
                  tar -xzf #{abyss} -C /usr/bin;
                  rm #{abyss};
                  cd /usr/bin/#{abyss.chomp('.tar.gz')};
                  ./configure CPPFLAGS=-I/usr/local/include --prefix=/usr/bin --enable-
maxk=96 && make && make install;
            }
      end
end

desc 'Install latest version of BLAST+.'
task :blastInstall do
      blast = ''
      if !command? 'blastn'
            ftp = Net::FTP::new('ftp.ncbi.nlm.nih.gov')
            ftp.login()
            ftp.passive=true
            ftp.chdir('blast/executables/blast+/LATEST')
            ftp.list("ncbi-blast*+-src.tar.gz").last =~ /.*(ncbi-blast.*\+\-
src\.tar\.gz).*/
            blast = $1
            ftp.getbinaryfile(blast)
            ftp.close
            sh %{
                  tar -xzf #{blast} -C /usr/bin;
                  cd /usr/bin/#{blast.chomp!('.tar.gz')}/c++;
                  ./configure --without-debug --with-mt --with-build-root=ReleaseMT;
                  cd ReleaseMT/build;
                  make all_r;

                  #sudo rm /usr/bin/blastdb_aliastool /usr/bin/dustmasker /usr/bin/rpstblastn
/usr/bin/blastdbcheck /usr/bin/gene_info_reader /usr/bin/segmasker /usr/bin/blastdbcmd
/usr/bin/gumbelparams /usr/bin/seqdb_demo /usr/bin/blast_formatter /usr/bin/srsearch /usr/bin/blastn
/usr/bin/makeblastdb /usr/bin/tblastn /usr/bin/blastp /usr/bin/makembindex /usr/bin/tblastx
/usr/bin/blastx /usr/bin/project_tree_builder /usr/bin/convert2blastmask /usr/bin/psiblast
/usr/bin/windowmasker /usr/bin/datatool /usr/bin/rpsblast
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastdb_aliastool /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/dustmasker /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/rpstblastn /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastdbcheck /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/gene_info_reader /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/segmasker /usr/bin;
                  ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastdbcmd /usr/bin;
```

```
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/gumbelparams /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/seqdb_demo /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blast_formatter /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/srsearch /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastn /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/makeblastdb /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/tblastn /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastp /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/makembindex /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/tblastx /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/blastx /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/project_tree_builder /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/convert2blastmask /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/psiblast /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/windowmasker /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/datatool /usr/bin;
                ln -s /usr/bin/#{blast}/c++/ReleaseMT/bin/rpsblast /usr/bin;
        }
    end
end

desc 'Install latest version of the NT database.'
task :ntInstall => :blastInstall do
    if ENV['BLASTDB'].to_s.empty?
        if progSettings.ntDatabase.to_s.empty?
            progSettings.ntDatabase = findFile('nt.nal', locate).to_s.chomp('.nal')
            if progSettings.ntDatabase.to_s.empty?
                progSettings.ntDatabase='/usr/share/nt/nt'
                sh %{
                    mkdir /usr/share/nt;
                    cd /usr/share/nt;
                    /usr/bin/ncbi-blast*/c++/src/app/blast/update_blastdb.pl nt;
                    for i in nt*.tar.gz; do tar -xzf $i; rm $i; done;
                }
            end
        else
            `echo "export BLASTDB=#{progSettings.ntDatabase.to_s}" >>
~/.#{shell}rc`
        end
    end
end

desc 'Install latest version of MEGAN.'
task :meganInstall do
    if !command? 'MEGAN'
        megan = ''
        Net::HTTP.start('ab.inf.uni-tuebingen.de', 80) { |http|
            http.get('/software/megan/welcome.html').body =~
/(\/data\/software\/[^"]*\/download\/welcome\.html)/
            downloadDir = File.dirname($1)
            http.get("#{$1}").body =~ /(MEGAN_unix_\d+_\d+\.sh)/
```

```
                megan = downloadDir + "/#{$1}"
                File.open("#{File.basename(megan)}", 'w'){ |file|
                if useRepo == true
                        link = progRepo.megan
                else
                        link = '/data/software/megan/download/' + megan
                end
                file.write(http.get(link).body)
                }
        }
        megan = File.basename(megan)
        sh %{
                chmod +x #{megan};
                ./#{megan};
                rm #{megan};
        }
        megan = `which MEGAN`.chomp
        `ln -s "#{findFile(MEGAN)}" /usr/bin` if megan.empty?
        megan = `which MEGAN`.chomp
        if (arch == 64) # If CPU architecture is 64-bit, allow for more than 2GB of RAM and force
64-bit Java.
                text = File.new(megan).read.gsub(/"\$prg_dir\/\$progname" "-server" "-
Xms\d+." "-Xmx\d+."/, "\"$prg_dir/$progname\" \"-server\" \"-d64\" \"-
Xms#{memInGigs}G\" \"-Xmx#{memInGigs}G\"")
                File.open(megan, 'w+'){ |file| file.write(text) }
        end
    end
end

desc 'Install GI to Taxonomy ID database.'
task :giTaxIdNuclInstall do
    progSettings.giTaxIdNuclDatabase=findFile('gi_taxid_nucl.dmp', locate)
    if progSettings.giTaxIdNuclDatabase.to_s.chomp == '.'
        ftp = Net::FTP::new('ftp://ftp.ncbi.nih.gov')
        ftp.login()
        ftp.passive=true
        ftp.chdir('pub/taxonomy')
        ftp.getbinaryfile('gi_taxid_nucl.zip')
        ftp.close
        `unzip -d /usr/share gi_taxid_nucl.zip`
        progSettings.giTaxIdNuclDatabase='/usr/share/gi_taxid_nucl.dmp'
    end
end

desc 'Install latest version of Parallel::Iterator for perl.'
task :parallelIteratorInstall do
    `perl -MParallel::Iterator -e 1`
    if $?.exitstatus != 0
        `perl -MCPAN -e 'install Parallel::Iterator'`
    end
end
```

```
desc 'Index the genome of a specified organism with Novo and Bowtie.'
task :otherIndex do
	puts 'OtherIndex'
	if !indexName.empty? and !indexGlob.empty?
		buildNovoIndex(indexName, indexGlob)
		buildBowtieIndex(indexName, indexGlob)
	end
end

task :default do
	Rake::Task[:metaGenomeAnalyzeContigs].invoke
end

desc 'Install latest version of everything.'
task :install do
	#$ replace with regular expression on for each task at some point
	Rake::Task[:novoalignInstall].invoke
	Rake::Task[:bowtieInstall].invoke
	Rake::Task[:hgInstall].invoke
	Rake::Task[:novoIndex].invoke
	Rake::Task[:bowtieIndex].invoke
	Rake::Task[:samtoolsInstall].invoke
	Rake::Task[:tophatInstall].invoke
	Rake::Task[:abyssInstall].invoke
	Rake::Task[:blastInstall].invoke
	Rake::Task[:ntInstall].invoke
	Rake::Task[:meganInstall].invoke
	Rake::Task[:giTaxIdNuclInstall].invoke
	Rake::Task[:parallelIteratorInstall].invoke
end

desc 'Automatically saving any settings changes which may have been made'
task :reserialize do
   # Reserialize object in case any changes have been made
   seqFile = File.open(".#{seqName}", 'w')
	progSettingsFile = File.open(File.expand_path("~/.#{PROG_NAME}"), 'w')
	YAML.dump(sequence, seqFile) if !installMode
	YAML.dump(progSettings, progSettingsFile)
	seqFile.close
	progSettingsFile.close
end

#Allow for pipeline override, chopping off the front of the pipeline
if truncate != 'true'
	task :alignSequence => sequence.novoalignPath if
!File.exists?(sequence.novoalignPath)
	task :removeHuman => sequence.removeNonMappedPath if
!File.exists?(sequence.removeNonMappedPath)
	task :removeSpans => sequence.blast1Path if !File.exists?(sequence.blast1Path)
	task :localAlignReads => sequence.megan1Path if
!File.exists?(sequence.megan1Path)
```

```
    task :metaGenomeAnalyzeReads => sequence.abyssPath if
!File.exists?(sequence.abyssPath)
    task :denovoAssembleCluster => sequence.blast2Path if
!File.exists?(sequence.blast2Path)
    task :localAlignContigs => sequence.megan2Path if
!File.exists?(sequence.megan2Path)
    task :metaGenomeAnalyzeContigs => sequence.pipeEndPath if
!File.exists?(sequence.pipeEndPath)
end

# invoke reserialize after all tasks have been executed
current_tasks =  Rake.application.top_level_tasks
current_tasks << :reserialize
Rake.application.instance_variable_set(:@top_level_tasks, current_tasks)
```

abyssKmerOptimization.pl

```
#!/usr/bin/perl
use strict;
use Parallel::Iterator qw( iterate_as_array );
use File::Basename;

### author: Joseph Coco
# Determine the best size for the kmer to use for ABYSS, the DeNovo
# Assembler, and execute the assembly with that size for all data sets
# provided. Outputs several files containing the input file name prepended
# with kmer.contigs.fasta. Operates in parallel.
#
# Optimizes ABYSS using the kmer which produces the best N50 score.
#
# NOTE: ABYSS must be within the executing user's PATH.
#
# ARGV0 = Regex to query files paths.
# ARGV1 = Minimum kmer to attempt.
# ARGV2 = Maximum kmer to attempt.
# ARGV3 = Data Type parameter. Either --illumina-quality or --standard-quality
#
###
my @reads = glob $ARGV[0];
my $sequence;
foreach my $seq (@reads){ # perform on each input file
    $sequence = $seq;
    # execute ABYSS with all kmer in parallel
    iterate_as_array( \&abyss, [$ARGV[1]..$ARGV[2]] );
    # pull out best N50 score and make a copy of the file with best score
    my $dir = dirname($0);
    my @facBestResult = sort qx{ "$dir/fac.pl" $sequence.*.contigs.fa };
    $_ = $facBestResult[-1];
    split ('\t');
    chomp $_[1];
    `cp $_[1] $_[1].kmerOptimized.fa`;
```

```
}

sub abyss{
	my $kmer = shift;
	$kmer = $kmer+7;
	`ABYSS -k $kmer $ARGV[3] --out=$sequence.$kmer.kmer.contigs.fa $sequence --
coverage-hist=$sequence.$kmer.kmer.contigs.coverage`;
}
```

addTaxon.pl

```
#!/usr/bin/perl -w
##########################################################################
############################################################

# First collect the GI numbers from the blast hit file.  The taxonomy file is huge, so we only want to build the
# hash for those entries that appear in the blast hit file.
%GtoT = ();
open( BHF, "<$ARGV[1]" );
print STDERR "\nCollecting GI numbers from blast hit file...";
while( $hit = <BHF> )
{
	next if ($hit =~ /^#/);
	chomp($hit);
	($readid, $subjid) = split "\t", $hit;
	$readid = $readid;
	($gi, $gid) = split '\|', $subjid;
	$gi = $gi;
	$GtoT{$gid} = -1;
}
close(BHF);
print STDERR "done.\n";

# Now process the huge gi to taxid translation file, keeping translations for gis that we saw in the blast hit
file
open( GTN, "<$ARGV[0]" );
print STDERR "\nBuilding gi to taxid translation hash...";
while( $line = <GTN> )
{
	chomp($line);
	($gin, $tin) = split "  ", $line;
	if( defined $GtoT{$gin} )
	{
		if( $GtoT{$gin} == -1 )
		{
			$GtoT{$gin} = $tin;
		}
		elsif( $GtoT{$gin} != $tin )
		{ print STDERR "\nWARNING: GIN $gin has more than one TIN associated
with it!\n"; }
	}
```

```
}
close(GTN);
print STDERR "done.\n";

# Now build the fasta hash
%fasta = ();
open( FASTA, "<$ARGV[2]" );
print STDERR "\nBuilding fasta hash...";
@mfa = <FASTA>;
close(FASTA);
for( $mi = 0; $mi <= $#mfa; $mi++ )
{
	$mine = $mfa[$mi];
	if( $mine =~ /^>/ )	# Check to see if it is a header line
	{
		chomp( $mine );
		$seq = $mfa[++$mi];
		chomp( $seq );
		while( ($mi < $#mfa) && ($mfa[$mi+1] =~ /^[-A-Za-z]/) )
		{	# Continue grabbing lines until the next non-sequence line
			$tmine = $mfa[++$mi];
			chomp($tmine);
			$seq .= $tmine; # append this line to the end of the sequence
		}
		# Put this sequence into the hash
		$mine =~ s/>//; # strip the > symbol off the sequence name
		$fasta{$mine} = $seq;
	}
}
print STDERR "done.\n";

open( TAXID, ">$ARGV[2].blast" );
open( BHF, "<$ARGV[1]" );
while( $hit = <BHF> )
{
	next if ($hit =~ /^#/);
	chomp($hit);
	($readid, $subjid) = split "\t", $hit;
	$readid = $readid;
	($gi, $gid) = split '\|', $subjid;
	$gi = $gi;
	print TAXID "$hit\t$GtoT{$gid}\n";
}
close(BHF);
foreach $key ( keys %fasta )
{
	print TAXID "$key	gi|000000|gb|AY000000.0|	0.00	0	0
0	0	0	0	0	0	0	-1\n";
}

close(TAXID);
```

fac.pl

```
#!/usr/bin/perl
use strict;
use Getopt::Std qw'getopts';

# Altered from original form. Possible written by Shaun Jackman. Unknown.

my %opt;
getopts 'hHjt:', \%opt;
my $opt_threshold = defined $opt{'t'} ? $opt{'t'} : 100;
my $opt_filename = $opt{'H'} || (@ARGV > 1 && !$opt{'h'});
my $opt_jira = $opt{'j'};

sub eng($)
{
	my $x = shift;
	return $x if $x < 10000000;
	return substr($x / 1000000, 0, 5) . 'e6';
}

my ($short, $sum, $ambiguous, $any, $other);
my @x;

sub count($$)
{
	my $id = shift;
	my $seq = uc shift;
	#print $id, length $seq;
	#print "\n";
	my $x = $seq =~ tr/ACGT//;
	my $colourspace = $seq =~ tr/0123//;
	die unless $x == 0 || $colourspace == 0;
	$x = $colourspace if $x == 0;
	my $myambiguous = $seq =~ tr/KYSBWRDMHV//;
	my $myany = $seq =~ tr/N//;
	if ($x < $opt_threshold) {
		$short++;
		return;
	}
	$sum += $x;
	$ambiguous += $myambiguous;
	$any += $myany;
	$other += (length $seq) - $x - $myambiguous - $myany;
	push @x, $x;
}

sub fac($)
{
	my $path = shift;
	$short = $sum = 0;
	$ambiguous = $any = $other = 0;
	@x = ();
```

```
        my $id;
        my $seq;
        open IN, "<$path" or die "$path: $!\n";
        while (<IN>) {
                chomp;
                if (/^>/) {
                        count $id, $seq if defined $id;
                        $id = $_;
                        $seq = '';
                } else {
                        $seq .= $_;
                        push @x, $seq;
                }
        }
        count $id, $seq if defined $id;
        close IN;

        my $n = @x;
        if ($n > 0) {
                my $mean = int $sum / $n;

                @x = sort { $a <=> $b } @x;
                my $min = $x[0];
                #my $q1 = $x[@x/4];
                my $q2 = $x[@x/2];
                #my $q3 = $x[@x*3/4];
                my $max = $x[-1];

                my ($nn50, $n50);
                my $n50sum = 0;
                while ($n50sum < $sum / 2) {
                        $nn50++;
                        $n50 = pop @x;
                        $n50sum += $n50;
                }
                #my $np = int 100 * $n50sum / $sum;

                my $ntotal = eng $short + $n;
                my $sumeng = eng $sum;
print "$n50\t$path\n";
=pod
                        print "$ntotal$," if $opt_threshold > 0;
                        print $n, $nn50, $min, $q2, $mean, $n50, $max;
                        #printf "$,%#.3g", $sum;
                        print $, . eng($sum);
                        print "$,ambig=$ambiguous" if $ambiguous > 0;
                        print "$,any=$any" if $any > 0;
                        print "$,other=$other" if $other > 0;
                        print "$,$path" if $opt_filename;
                        print "\n";
#=cut
```

```
                format Spaces =
@<<<<<<<@<<<<<<@<<<<<<@<<<<<<@<<<<<<@<<<<<<@<<<<<<@<<<<<<@<<<<<<
^<<<<<<<<<<<<<
$ntotal, $n, $nn50, $min, $q2, $mean, $n50, $max, $sumeng, $path

^<<<<<<<<<<<<<~~
$path
.
                format Pipes =
|@<<<<<<<|@<<<<<|@<<<<<|@<<<<<|@<<<<<|@<<<<<|@<<<<<|@<<<<<|@<<<<<<|@*|
$ntotal, $n, $nn50, $min, $q2, $mean, $n50, $max, $sumeng, $path
.
                $~ = $opt_jira ? 'Pipes' : 'Spaces';
                $^ = $opt_jira ? 'Pipes_TOP' : 'Spaces_TOP';
                $: = '/- ';
                write;
#=pod
                print "$,ambig=$ambiguous" if $ambiguous > 0;
                print "$,any=$any" if $any > 0;
                print "$,other=$other" if $other > 0;
                print "$,$path" if $opt_filename;
                print "\n";
=cut
    } else {
        print STDERR "warning: `$path' is empty\n";
    }
}

format Spaces_TOP =
n      n:@<<< n:N50  min    median   mean   N50    max    sum
$opt_threshold
.
format Pipes_TOP =
||n    ||n:@<<||n:N50||min  ||median  ||mean||N50  ||max  ||sum   ||
$opt_threshold
.

=pod
$, = "\t";
if ($opt_threshold > 0) {
        print 'n', "n:$opt_threshold$,";
} else {
        print "n$,";
}
print 'n:N50', 'min', 'median', 'mean', 'N50', 'max', "sum\n";
=cut

@ARGV = ('-') if @ARGV == 0;
fac $_ foreach @ARGV;
```

parallelBlast.sh

```
#!/bin/bash

$@
wait
```

Xextractspans.pl

```
#!/usr/bin/perl -w

foreach $fn (@ARGV)
{
	unless( -r $fn )
	{
		print STDERR "\nFile $fn is not readable.  Skipping this one.\n";
		next;
	}

	open( INF, "<$fn" );
	open( OUF, ">$fn.spans" );
	open( NOR, ">$fn.mapped" );
	while( $line = <INF> )
	{
		if( $line =~ /^@/ )
		{ next; }
		@cols = split "	", $line;
		if( defined $cols[5] && length($cols[5]) > 3 )
		{ print OUF $line; }
		else
		{ print NOR $line; }
	}
	close(INF);
	close(OUF);
	close(NOR);
}
```

Xfilterspans.pl

```
#!/usr/bin/perl -w
##########################################################################
############################################################
# This script takes fasta files of nonmapped reads and corresponding sam files of span reads from tophat and
#
# filters the span reads out of the nonmapped reads.  It prints a new fasta file with the span reads removed.
#
# There must be an even number of files specified on the command line and the respective paired files must
match. #
# This is not checked by the script, the user should inspect the details file to ensure the files matched.
#
```

```
# For example: Filterspans.pl a.NM.fasta b.NM.fasta c.NM.fasta a.sam.spans b.sam.spans c.sam.spans
#
##########################################################################
##########################################################

# First check to make sure the command line parameters are correct; if not print usage to stderr and exit
unless (($#ARGV % 2) == 1)
{
	print STDERR "You must specify an even number of files on the command
line.\n";
	print STDERR "usage: $0 file1.NM.fasta [file2.NM.fasta...]
file1.sam.spans [file2.sam.spans...]\n";
	exit;
}

for( $fi = 0; $fi <= ($#ARGV / 2); $fi++ )
{
	$fastan = $ARGV[$fi];
	$spann = $ARGV[$fi + ($#ARGV / 2) + 1];

	unless( -r $fastan )
	{
		print STDERR "File $fastan is not readable.  Skipped this pair.\n";
		next;
	}
	unless( -r $spann )
	{
		print STDERR "File $spann is not readable.  Skipped this pair.\n";
		next;
	}
	unless( $fastan =~ /.NM.fasta$/ && $spann =~ /.sam.spans$/ )
	{
		print STDERR "Filename extensions are not in proper format.  Skipped
this pair.\n";
		next;
	}

	print STDERR "Filtering spans in $spann from $fastan.\n";

	# Read the span reads into a hash
	%spanr = ();
	open( SPAN, "<$spann" );
	$spanreads = 0;
	while( $line = <SPAN> )
	{
		++$spanreads;
		($rname) = split " ", $line;
		$rname = ">" . $rname . "/1";
		if( defined $spanr{$rname} )
		{
			++$spanr{$rname};
			#print STDERR "FYI: Found a duplicated read name in the spans file: $rname\n";
```

```
        }
        else
        { $spanr{$rname} = 1; }
    }
    close( SPAN );

    # Read the nonmapped reads from fasta and filter out those that are in the spanreads
    open( NM, "<$fastan" );
    open( NS, ">$fastan.nospans" );
    open( DET, ">$fastan.nospans.details" );
    print DET "Filtering spans in $spann from $fastan. Make sure these files
match!\n";
    $nonmappedreads = 0;
    $filtered = 0;
    $kept = 0;
    $line = <NM>;
    chomp($line);
    $notdone = 1;
    do
    {
        $head = $line;

        # Read up the sequence
        $seq = <NM>;
        chomp($seq);
        do
        {
            if( $line = <NM> )
            {
                chomp( $line );
                unless( $line =~ /^>/ )
                { $seq .= $line; }
            }
            else
            {
                $notdone = 0;
            }
        } until ( ($notdone == 0) || ($line =~ /^>/) );

        ++$nonmappedreads;
        if( defined $spanr{$head} )
        {       # This read was one of the spans so filter it
            ++$filtered;
        }
        else
        {       # This read was not a span so keep it
            ++$kept;
            print NS "$head\n$seq\n";
        }
    } while( $notdone );
    close( NM );
    close( NS );
```

```
	print DET "There were $spanreads reads in the span file.\n";
	print DET "There were $nonmappedreads reads in the nonmapped fasta
file.\n";
	print DET "I filtered $filtered of these reads and kept $kept of these
reads.\n";
	close( DET );
}
```

Xnovotonm.pl

```
#!/usr/bin/perl -w
##################################################################
##############################################################
# This script takes files of mapped sqeuence reads in novo format as input.  For each input file, it produces a
#
# file as output with nonmapped (NM) reads and a counts file.                                    #
##################################################################
##############################################################

unless( $#ARGV >= 0 )
{
	print STDERR "usage: $0 mappedseqfile1 ...\n";
	exit;
}

for( $f = 0; $f <= $#ARGV; $f++ )
{
	# Open the input and output files
	open( INF, "<$ARGV[$f]" );
	open( NM, ">$ARGV[$f].NM.fastq" );
	open( NMA, ">$ARGV[$f].NM.fasta" );

	$tot = $uniq = $rep = $qc = $nm = $ql = 0;
	while( $line = <INF> )
	{
		# Skip any lines that aren't reads
		unless( $line =~ /@>/ )
		{ next; }

		++$tot;
		# Parse the line
		chomp($line);
		($head, $slr, $seq, $rdqual, $status, $alscore, $alqual, $chr, $pos, $strand) =
split("\t", $line);
		$slr = $slr; $alscore = $alscore; $alqual = $alqual; $chr = $chr; $pos = $pos;
$strand = $strand;
		print $status
		if( $status eq "U" )
		{ ++$uniq; }
		elsif( $status eq "R" )
		{ ++$rep; }
		elsif( $status eq "QC" )
```

```
        { ++$qc; }
        elsif( $status eq "NM" )
        {
            ++$nm;
            print NM "$head\n$seq\n+\n$rdqual\n";
            $head =~ s/^@/>/;
            print NMA "$head\n$seq\n";
        }
        elsif( $status eq "QL" )
        { ++$ql; }
        else
        { print STDERR "Unknown status $status. Skipped this read.\n" }
    }

    close(INF);
    close(NM);
    close(NMA);

    open( COU, ">$ARGV[$f].counts" );
    print COU "Total Reads: $tot\n";
    print COU "Unique Maps: $uniq\n";
    print COU "Repeat Maps: $rep\n";
    print COU "No Mappings: $nm\n";
    print COU "Quality Con: $qc\n";
    print COU "Low Scoring: $ql\n";

    close(COU);
}
```

BIBLIOGRAPHY

[1] Zhong Wang, Mark Gerstein, and Michael Snyder, "RNA-Seq: a revolutionary tool for transcriptomics.," *Nature reviews. Genetics*, vol. 10, no. 1, pp. 57-63, 2009.

[2] Novocraft Technologies. (2011, March) Novocraft. [Online]. http://www.novocraft.com/wiki/tiki-index.php?page=Novoalign+User+Guide

[3] Cole Trapnell, Lior Pachter, and Steven L Salzberg, "TopHat: discovering splice junctions with RNA-Seq.," *Bioinformatics (Oxford, England)*, vol. 25, no. 9, pp. 1105-11, 2009.

[4] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. New York: Cambridge University Press , 1999.

[5] E. J. Gumbel, "Statistics of extremes," *Columbia University Press*, 1958.

[6] S. F. Altschul and W. Gish, "Local alignment statistics," *Meth. Enzymol*, pp. 460-480, 1996.

[7] Ali Mortazavi, Brian A Williams, Kenneth Mccue, Lorian Schaeffer, and Barbara Wold, "Mapping and quantifying mammalian transcriptomes by RNA-Seq," *Nature Methods*, pp. 1-8, 2008.

[8] Illumina, Inc., mRNA Sequencing Sample Preparation Guide, September 2009.

[9] Illumina, Inc., Preparing Samples for Digital Gene Expression-Tag Profiling with DpnII, 2007.

[10] Michael L. Metzker, "Sequencing technologies — the next generation," *Nature Reviews Genetics*, pp. 31-46, 2009.

[11] Elaine R Mardis, "Next-generation DNA sequencing methods.," *Annual review of genomics and human genetics*, vol. 9, pp. 387-402, 2008.

[12] John C Marioni, Christopher E Mason, Shrikant M Mane, Matthew Stephens, and Yoav Gilad, "RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays.," *Genome research*, pp. 1509-17, 2008.

[13] Mark S. Longo, Michael J. O'Neill, and Rachel J. O'Neill, "Abundant Human DNA Contamination Identified in Non-Primate Genome Databases," *PLoS ONE*, pp. 1-4, 2011.

[14] Svante Pääbo et al., "Genetic analyses from ancient DNA," *Annual review of genetics*, pp. 645-679, 2004.

[15] Connie J. Mulligan, "Anthropological Applications of Ancient DNA: Problems and Prospects," *American Antiquity*, pp. 365-380, 2006.

[16] Inanç Birol et al., "De novo transcriptome assembly with ABySS.," *Bioinformatics (Oxford, England)*, vol. 25, no. 21, pp. 2872-7, 2009.

[17] Illumina, Inc., De Novo Assembly Using Illumina Reads, October 13, 2009.

[18] Daniel H Huson, Alexander F Auch, Ji Qi, and Stephan C Schuster, "MEGAN analysis of metagenomic data," *Genome Research*, pp. 377-386, 2007.

[19] Arthur Brady and Steven L Salzberg, "Phymm and PhymmBL: Metagenomic Phylogenetic Classification with Interpolated Markov Models," *Nature Methods*, vol. 6, no. 9, pp. 673-676, 2010.

[20] Ozkan U Nalbantoglu, Samuel F Way, Steven H Hinrichs, and Khalid Sayood, "RAIphy : Phylogenetic classification of metagenomics samples using iterative refinement of relative abundance index profiles RAIphy : Phylogenetic classification of metagenomics sam- ples using iterative refinement of relative abundance index profiles," *BMC Bioinformatics*, Janurary 2011.

[21] Harry L. Ioachim and L. Jeffrey Medeiros, *Ioachim's Lymph Node Pathology: Fourth edition*. Philadelphia, PA: Lippincott Williams & Wilkins, 2008.

[22] RD Morin et al., "Somatic mutations altering EZH2 (Tyr641) in follicular and diffuse large B-cell lymphomas of germinal-center origin," *Nat Genet.*, pp. 181-185, 2010.

[23] Serban Nacu et al., "Deep RNA sequencing analysis of readthrough," *BMC Medical Genomics*, pp. 1-22, 2011.

[24] Irina Khrebtukova. (2011, March) EMBL-EBI. [Online]. http://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-513

[25] Guenter Stoesser et al., "The EMBL Nucleotide Sequence Database," *Nucleic Acids Research*, pp. 21-26, 2001.

[26] Guorong Xu et al., "Transcriptome and targetome analysis in MIR155 expressing cells using RNA-seq," *RNA*, vol. 16, no. 8, pp. 1610-1622, June 2010.

[27] Tao Tao. (2007, December) FTP Downloadable BLAST Databases From NCBI. [Online]. http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/blastdb.html

[28] Dennis a Benson, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers, "GenBank.," *Nucleic acids research*, vol. 38, no. Database issue, pp. D46--51, 2010.

[29] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman, "Basic local alignment search tool.," *Journal of molecular biology*, vol. 215, no. 3, pp. 403-10, 1990.

[30] D.H. Huson, S. Mitra, H.-J. Ruscheweyh, N. Weber, and S. C. Schuster, "Integrative analysis of environmental sequences using MEGAN 4," 2011.

[31] Shaun Jackman. (2010, December) very large MAX_KMER size - ABySS | Google Groups. [Online]. http://groups.google.com/group/abyss-users/browse_thread/thread/1066990323413de/6294ac871b456fa7?lnk=gst&q=max+kmer#6294ac871b456fa7

[32] J. Weirich. (2009, May) Rake. [Online]. http://rubyforge.org/projects/rake/index.html

[33] E. Bergogne-Berezin, Marie-Laure Joly-Guillou, and Kevin J. Towner, *Acinetobacter: Microbiology, Epidemiology, Infections, Management*. Boca Raton, FL: CRC Press, 1995.

[34] J Goecks, A Nekrutenko, J Taylor, and The Galaxy Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol.*, p. R86, 2010.

[35] D. Blankenberg et al., "Galaxy: a web-based genome analysis tool for experimentalists," *Current Protocols in Molecular Biology*, pp. 1-21, 2010.

[36] Jared T Simpson et al., "ABySS: a parallel assembler for short read sequence data.," *Genome research*, vol. 19, no. 6, pp. 1117-23, 2009.

[37] Mendel Cooper. (2010, March) Thu Linux Documentation Project. [Online]. http://tldp.org/LDP/abs/html/

[38] Samuel Marguerat and Jürg Bähler, "RNA-seq: from technology to biology.," *Cellular and molecular life sciences : CMLS*, vol. 67, no. 4, pp. 569-79, 2010.

VITA

Joseph Robert Coco was born in Baton Rouge, Louisiana and received his B.S. degree in Computer Science from the University of New Orleans. He was admitted to the graduate school of the University of New Orleans in Fall 2009, where he worked as a teacher's assistant under the guidance of Dr. Jaime Nino until Summer 2010 where he worked as a research assistant for Dr. Christopher Taylor.

Part of his graduate studies were spent researching the prediction of transcription factor binding sites based on virus classes but the majority was spent researching exogenous agents of tumors from RNA-Seq experiments. When not working for Dr. Taylor, he worked at the University of New Orleans' Research and Technology Park for the Space and Naval Warfare (SPAWAR) department of the Navy.