Spring 2012

Metagenome – Processing and Analysis

Sheetal Gosrani
San Jose State University

Metagenome – Processing and Analysis

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sheetal Gosrani

Spring 2012

SAN JOSÉ STATE UNIVERSITY

The Undersigned Project Committee Approves the Thesis Titled

Metagenome – Processing and Analysis

by

Sheetal Gosrani

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Sami Khuri, Department of Computer Science Date

Dr. Devaki Bhaya, Carnegie Institution of Science, Stanford Date

---

Dr. Mark Stamp, Department of Computer Science Date

APPROVED FOR THE UNIVERSITY

Associate Dean, Office of Graduate Studies and Research Date

Abstract

Metagenome – Processing and Analysis

by

Sheetal Gosrani

Metagenome means “multiple genomes” and the study of culture independent genomic content in environment is called metagenomics. Because of the advent of powerful and economic next generation sequencing technology, sequencing has become cheaper and faster and thus the study of genes and phenotypes is transitioning from single organism to that of a community present in the natural environmental sample. Once sequence data are obtained from an environmental sample, the challenge is to process, assemble and bin the metagenome data in order to get as accurate and complete a representation of the populations present in the community or to get high confident draft assembly.

In this paper we describe the existing bioinformatics workflow to process the metagenomic data. Next, we examine one way of parallelizing the sequence similarity program on a High Performance Computing (HPC) cluster since sequence similarity is the most common and frequently used technique throughout the metagenome data processing and analyzing steps. In order to address the challenges involved in analyzing the result file obtained from sequence similarity program, we developed a web application tool called Contig Analysis Tool (CAT). Later, we applied the tools and techniques to the real world virome metagenomic data i.e., to the genomes of all the viruses present in the environmental sample obtained from microbial mats derived from hot springs in Yellowstone National Park. There are several challenges associated with the assembly and binning of virome data particularly because of the following reasons: 1. Not many viral sequence data in the existing databases for sequence similarity. 2. No reference genome 3. No phylogenetic marker genes like the ones present in the bacteria and archaea. We will see how we overcame these problems by performing sequence similarity using CRISPR data and sequence composition using tetranucleotide analysis.

Acknowledgements

I wish to express my sincere gratitude to my advisors Dr. Sami Khuri, Professor, Department of Computer Science, San Jose State University and Dr. Devaki Bhaya, Principal Investigator, Department of Plant Biology, Carnegie Institute for Science, Stanford for providing me an opportunity to do my project on Metagenome data, for their encouragement, support and guidance throughout this project. A special thanks to National Science Foundation (NSF); research in Bhay Lab is supported, in part, by grants from NSF. I would also like to thank the team who worked on this project, Michelle Davison, Graduate Student, Bhaya Lab, Carnegie Institution of Science, Stanford and Todd Treangen, Postdoc at Mihai Pop's Lab, University of Maryland.

My special thanks to my husband, family and friends for their continuous encouragement and support throughout my Master's program.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

In this section, we describe metagenome and metagenomics followed by the reason behind the popularity and existence of this relatively new area of study. Sequencing these metagenomes produces giga-bases of DNA nucleotides which then needs to be carefully processed and analyzed for building whole-genome or to perform gene centric data analysis.

1.1 Metagenomics

Metagenomics is the study of multiple genomes i.e., metagenomes taken directly from the environment. While the traditional methods, in which organisms were cultured in predetermined media under the laboratory conditions, were able to produce a diversity profile; they missed the vast majority of biodiversity present in the environment. Recently, Kevin Chen and Lior Pachter (researchers at the University of California, Berkeley) defined metagenomics as "the application of modern genomics techniques to the study of communities of microbial organisms directly in their natural environments, bypassing the need for isolation and lab cultivation of individual species." [1]

Metagenomics is currently the only way to study genetic diversity present in the viral communities as they do not contain any universal phylogenetic marker (like 16S RNA for bacteria) which are typically used to culture bacterial organisms. Culturing the host and then infecting them with specific virsuses or viral DNA obtained from the environment in the laboratory condition is not yet streamlined. However, the metagenome sample obtained from the environment directly represent communities of population as opposed to isolated populations and thus, metagenomics may help reveal information about how the populations co-evolve.

1.2 Advancement in Sequencing Technologies

With the advent of powerful and economic next generation sequencing technologies such as Sanger sequencing or massively parallel pyrosequencing, metagenomics has become more popular. Sanger sequencing is based on chain termination with dideoxynucleotides whereas pyrosequencing is based on sequencing by synthesis method, i.e., the idea is to detect pyrophosphate release when nucleotide is incorporated. Sanger sequences are longer ~750 base pairs (bp) than pyrosequencing techniques, specifically 454 produces reads of length ~100 to ~200 bp. 454 titanium series produces reads of length ~400 to ~500 bp. Advantages of pyrosequencing over Sanger sequencing include a

much lower per-base cost and no requirement for cloning. [10] These generate sequence trace files from which base calling is done.

1.3 Bioinformatics Pipeline for metagenome processing

Once the raw reads are obtained, the data need to be processed and analyzed to see what story is hidden in it.

1.3.1 Sequence Processing:

Processing of both, the genomic and metagenomic sequence data, follow common steps like preprocessing the sequence reads, assembly, Gene Prediction and Annotation. However, the main difference between genomes and metagenomes is that the former has a fixed end-point like one or more completed chromosomes. However, in the case of metagenomes, we just get draft assemblies and may be sometimes almost complete genome of dominant populations. [1][7]

1.3.1.1 Preprocessing the sequence reads

This is a very important step in metagenome processing. It involves base calling of raw data, removal of low complexity reads, removal of contaminant sequences, and removal of outliers, i.e, reads with very short length. Base calling involves identifying DNA bases from the DNA sequencing trace files. The most commonly used base calling tool is phred. [2] phred assigns a quality value, q, to each called base based on the per-base error probability, p by using the following formula: $q = -10 \times \log_{10}(p)$.

The other tool which is used in many other researches is Prinseq [15]. Prinseq is a web as well as a standalone tool that allows to filter, trim and reformat the metagenome data. It removes low quality reads based on quality scores obtained from phred to avoid complications in assemblies and downstream analysis. It trims poly-A/T tails, repeats of A's and T's at the end of the sequence because it can result in false positives during similarity searches, since they have a good alignment with low complexity regions or sequences with tails. It removes sequences with a lot of ambiguous bases, i.e., sequences with high number of Ns. A position in the sequence where a base cannot be identified is replaced by the letter N which means it is an ambiguous base. For removing low complexity reads, it calculates the sequence complexity using both DUST and Entropy approached. DUST is the heuristic used to mask low complexity regions during BLAST search. [11] DUST computes scores based on

how often different triplets occur in the sequences and are scaled from 0 to 100 and higher scores imply lower complexity. In case of Entropy approach, entropy values of trinucleotides in the sequence is computed and scores are scaled from 0 to 100 where lower entropy would mean low complexity.

1.3.1.2 Assembly

Assembly is the process of combining reads based on similarity to obtain contiguous DNA segments called contigs. There are challenges in assembling metagenomes as there could be problems like coassembly of reads coming from different species because of non uniform species distribution. This can happen if there is high sequence similarity between reads coming from closely related species. There are many publicly available assembly programs like Phrap, Celera Assembler, Newbler but these were all designed for assembling genomes from isolates and not for metagenomes which comprise of multiple species with read coverage that is non uniform. Therefore, their performances vary significantly. To mitigate these problems for de novo assembly, we need to pass our data through more than one assembler so that it helps solving misassembly of the largest contigs. To further strengthen our assembly, we can perform multiple assemblies by tweaking parameters for a particular assembler. To be absolutely sure of our assembly so that problems do not percolate to further downstream analysis, we can perform manual inspection using scaffolding programs like ScaffViz or visualization programs like Consed. [3]

Comparative assemblies are easier to work with; where a reference genome or fully sequenced genome is passed to assembler along with the metagenome. AMOS is an assembler that performs comparative assembly.

1.3.1.3 Gene Prediction and Annotation

The process of identifying protein coding genes and RNA sequences is known as gene prediction. There are two ways of performing gene calling: one is evidenced-based and the other is ab initio gene prediction. The evidenced-based method is based on BLAST similarity search to find homologs against a database of previously found genes. The ab initio gene prediction method allows gene identification based on intrinsic features of the DNA sequence to differentiate between coding regions of a sequence from non-coding regions. This method is useful to identify those genes that do not have homologs to existing database sequences, and to find novel genes. For the ab initio method, there are many gene-prediction tools, some of which requires training data set (fgenes) while some are

self-trained on the target sequence (MetaGene, Glimmer, Genemark). MetaGene is the prokaryotic gene prediction tool developed specifically for metagenomes. The program does not require training data set and it estimates di-codon frequency from the GC content of a given sequence. [12]

In case of complete genomes, both the ways of gene prediction are employed and the hits to genes in the database act as training sets. In case of unassembled pyrosequencing reads and high complexity metagenomes, evidence-based gene prediction is the only method used because of the fragmented nature and short read lengths of these data sets; as pointed out by Mavromatis [7]. Even in case of less complex communities, it is better to perform gene prediction on both reads and contigs because reads from less abundant organism remains unassembled and these reads may contain important functionality.

The most commonly used tool to predict RNA genes like tRNA and rRNA is tRNAscan. [9]

Finally, to assign protein function to metagenome data, protein sequences are compared to the database of protein families sequences like TIGRFAM, Pfam, and COGs. [7]

1.3.2 Data Analysis

Depending on the metagenome, there are different data analysis methods. The most common analysis methods are composition analysis on contigs, reclassification of reads after preprocessing, and binning. Next, we cover the topic of binning.

1.3.2.1 Binning

The process of associating sequence data to the contributing species is known as binning. The highly reliable binning is assembly, as reads coming from same species are assembled together. This is not the case in metagenome data sets as there are chances of coassembly. The two most common ways to bin are based on sequence similarity and sequence composition. In case of sequence similarity, we compare our metagenome data using tools such as BLAST and MEGAN (Huson et al 2007), a metagenome analyzer to separate metagenome fragments based on phylogenetic groups. If the suitable marker genes are present, then assignment of fragments based on taxonomic group is feasible. However, in case of absence of marker genes for your metagenome, the other approach is to use (G+C) content along with phylogenetic information to separate fragments. The other binning method, based on sequence composition is entirely different as it makes use of oligonucleotide frequencies which

supposedly are distinct and help separate different genomes. The word length can range from 1 to 8, with longer words giving better resolution but are expensive computationally. Therefore, typical word length range from 3 to 6 bases long. This method is so far the best method. As pointed out by Teeling [19], in their experiment on 9054 genomic fragments generated from 118 complete bacterial genomes the scores and results obtained using tetranucleotide analysis were far superior compared to GC content binning method. The standalone tool available online for tetranucleotide analysis is called TETRA (Teeling et al 2004). TETRA computes z-scores from the divergence between observed versus expected tetranucleotide frequencies. To compute observed values, it counts frequencies of all $4^4 = 256$ possible tetranucleotides for DNA sequences (both forward and reverse strand). To compute expected values, it counts expected frequencies for each tetranucleotide "by means of a maximal-order Markov model from the sequences' di- and trinucleotide composition." [20]

1.4 Types of BLAST programs and existing methods to perform batch BLAST, need for parallelizing

The Basic Local Alignment Search Tool (BLAST®, Altschul et al 1990) is the most popular and widely used web based and standalone tool. It performs local pairwise alignment to find similarity between an input sequence, called a query sequence and every sequence in the database. There are 4 basic types of sequence similarity versions in BLAST. They are BLASTN, BLASTP, BLASTX, and TBLASTX programs. They differ in terms of what type the query sequence is i.e., nucleotide or protein sequence and what type of database sequences it is compared to. The BLASTN program compares nucleotide query against nucleotide database, BLASTP compares protein query against protein database. Both BLASTX and TBLASTX has query sequence as nucleotide sequence but the comparison happens to protein and nucleotide database, respectively. BLASTX translates the query sequence in all six frames and compares each such translation to sequences in the protein database. However, TBLASTX is computationally more expensive because it translates both the query sequence and nucleotide sequences in the database in all six frames and compares each translation of query sequence against each translation of every sequence in the database.

There are many different output file formats for BLAST program. For web based program HTML output format is the default one, however one can download plain text, XML, and tab delimited output formats. For batch BLAST, tab delimited is the most commonly used format.

BLAST comes as a web service, network service or as standalone distribution. The web service

is useful if one wants to compare a couple of query sequences. On the other hand, the network service is helpful for performing batch BLAST remotely on BLAST server. This is very helpful if you don't have resources to perform batch BLAST locally. However, since many jobs are queued up, the results may get delayed. They also track if the same user if sending multiple remote BLAST requests and if such requests take more CPU time, then the priority for that user is reduced which can delay your results further more. Thus, to perform similarity of many query sequences against large database the standalone BLAST distribution allows you to run batch programs locally. The advantage here is that you can run the program as many times as you want and compare query sequences to any database of your choice, even custom databases created by you. The main problem with batch BLAST locally is it takes days together for computationally expensive program like TBLASTX. Hence, there is a need to parallelize and run the task on several machines connected together. In section 2.0 we will see how we can parallelize the batch program on a cluster containing 8 nodes.

1.5 Tools to analyze the batch report

Performing batch BLAST is the first step and then extracting meaningful information out of it is another challenge. Traditionally, batch BLAST reports are analyzed using Spreadsheet program, where different columns are either filtered for some criteria like show all query sequences having hit to a specific subject sequence and so on, or sorted on some fields like sort on evalue showing hits from low evalue to high evalue. An alternative is to write Perl scripts to extract keywords and filter on fields. [16] However, this is not feasible for batch programs with 1000s of query sequences compared against massive database like Genbank's 'nt' database as it produces a very large output file. Thus, to solve this problem and simplify analysis of the batch BLAST reports, we developed a web application as explained in section 3.0.

2.0 Batch BLAST on Cluster

As mentioned earlier, to perform batch BLAST, especially TBLASTX which is computationally very expensive, there was a need for parallelization. We wrote a program which can run the BLAST program in parallel on a High Performance Computing (HPC) cluster.

2.1 Ways to perform batch BLAST

There are three ways in which parallelism can be achieved, namely by Job, Query and Database parallelism. [14] We will see below what each of them means and what are advantages and/or disadvantages of each. Later we will see which approach did we follow and how we achieved parallelism on the cluster.

2.1.1 Job parallelism

This is the simplest way to achieve parallelism in a machine which is shared by multiple users. In this case we run each user's job on a different processor. However, scheduling jobs is complex in this approach as we need to make sure that each user receives a fair share of the available resources and that no single processor is overloaded i.e., distribute jobs equally on all the processors. To solve this problem, one can add another layer of software which can manage job or batch scheduling like for e.g., Sun Grid Engine. The main disadvantage with Job parallelism approach is that no single job can take the benefit of multiple available processors in the cluster. Say for example, if there is a large job that takes days together to finish its processing, it will overload a single processor while other processors remain unused.

2.1.2 Query parallelism

In Query parallelism approach for batch BLAST, we distribute multiple query sequences on different processors / nodes. The main advantage with this approach is that there is no dependency in terms of search of different query sequences. With a lot of query sequences to be searched against an average sized database this approach gives high performance and is scalable for the fact that all the processors of each node are given work, no single processor is throttled. However, the problem with this approach is that if a scheduler is not used then it can result in overloading of a single node. We will see below how we extended this approach for BLAST parallelism by leveraging NCBI BLAST

software and writing a wrapper script.

2.1.3 Database parallelism

In Database parallelism, different processors perform BLAST search on separate pieces of the database simultaneously, like for example, for a large query sequence search against 'nt' database which is divided into 11 smaller parts, we can allocate each processor the query sequence and a part of the 'nt' database. This approach is the only way for single query searches. The main advantage is that this approach gives good performance benefits in case when the databases are large. However, the disadvantage is that it is more difficult to implement because the results of sub-searches are inter-dependent.

2.1.4 Technique applied on HPC cluster

In order to parallelize on HPC cluster at Stanford, we followed query parallelism approach and wrote a wrapper bash script to leverage the power of GNU Parallel (Tange et al 2011) tool and NCBI BLAST stand-alone program. Parallel is a shell tool to run jobs in parallel on one or more nodes in the cluster. A job can be a command or a script to be run for each line of the input. [18] We will see in detail the concrete example and parts of the wrapper script written to parallelize BLAST below in section 2.2.

2.2 Cluster Information

The High Performance Computing cluster at Carnegie Institute, Stanford is a Linux cluster of 8 nodes with each node having 12 CPU's, so a total of 96 CPU's. It has a shared file system, so that would mean, all the nodes read and write data from and to the same shared file system. However, to talk to all other nodes from the node where your program is running, we need to set up multiplex master ssh session.

2.3 Setup of BLAST

For the cluster, we downloaded Blast+ applications tar ball executable. After which we need to untar the executable and set paths in the .bashrc file as shown in Figure 1 and 2 respectively.

```
>$ untar -xvzf ncbi-blast-2.2.25+-x64-linux.tar.gz
```

Figure 1: Installing Blast+ on Cluster

The paths shown in Figure 2, need to be set for the binaries to be visible and to run the BLAST command from any folder without always giving the absolute path of BLAST programs residing in bin directory.

```
>$ cat .bashrc
export BLAST_HOME=path_to_folder/ncbi-blast-2.2.25+/bin
export PATH=$PATH:$BLAST_HOME
BLASTDB=path_to_folder/ncbi-blast-2.2.25+/db
```

Figure 2: Set up for Blast+ on cluster

Get required databases from Genbank and untar all of them. To untar all parts of 'nt' database in parallel, execute the command as shown in Figure 3.

```
>$ parallel “tar -xvzf {}” ::: nt.*.ttar.gz
```

Figure 3: Unzip all 'nt' database parts in parallel

To update databases, run the update_database Perl script present in the bin directory. It needs the database name as the parameter.

2.4 Setup to run 'parallel' command on all nodes of cluster

For the setup we need to create master ssh session, create a parallel.sshloginfile and create config file in ./ssh directory in your home folder.

In order to create multiplex master ssh connection, run the command as shown in Figure 4:

```
>$ ssh -Nf remotehostname1
>$ ssh -Nf remotehostname2
```

Figure 4: Create master ssh connection

The ssh command of Figure 4 is one of the many ways to create a master. The above commands, run a null command and tell ssh to fork into the background. The -N option is useful for port forwarding and the -f option backgrounds ssh. The idea is to set up a master ssh connection before you can use the slave connections without providing password. Setting up the master connection will require a password, and as long as the master ssh process is running and can connect to the socket "~/.ssh/CP.

%r@%h.%p", and the slaves can connect to the socket also, the slaves shouldn't need the password supplied again. The master connection is the master between node A and node B, not node A and many nodes.

Running the following 'ps' command gives the output as shown in Figure 5.

```
>$ ps -flu <your id>
F S UID PID PPID C PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
1 S <yourid> 30885  1 0 75  0 - 41564 -  2012 ? 00:01:38 ssh -Nf remotehostename1
```

Figure 5: Confirm the creation of master ssh connection

As one can see from Figure 5, since the parent process (PPID) of the ssh is 1, the process is properly backgrounded and will continue to run even after you log out of the remote system, so other processes (e.g., cron jobs) can use the master connection to invoke commands on other nodes.

To kill these connections simply run command as shown in Figure 6.

```
>$ ssh -O exit remotehostname
```

Figure 6: Kill the master ssh connection

2.4.1 Create parallel.sshloginfile:

In the file parallel.sshloginfile we just provide ssh command to all the remote hosts that you would want to connect to, as shown in Figure 7. Thus, the file contains ssh remotehostname1, ssh remotehostname2 and so on.

```
>$ cat parallel.sshloginfile
ssh remotehostname1
ssh remotehostname2
```

Figure 7: Contents of parallel.sshloginfile

We will see later in a parallel command example that to run the script passed to parallel command in parallel on all the remote hosts we need to provide a sshloginfile.

2.4.2 Create config file in ./ssh directory

```
>$ cat ~/.ssh/config
host *
controlmaster auto
forwardx11 no
controlpath ~/.ssh/CP.%r@%h:%p
```

Figure 8: ssh configuration

The contents in Figure 8 configures ssh for any host, with controlmaster ('auto' value) allowing for opportunistic multiplexing i.e., it tries using existing master connection but in case if it does not find one it will create a new connection. The controlpath parameter specifies the location to the control socket required for sharing the connection.

2.5 Example of parallel command

Now to test whether all the setup is correct and our parallel command is able to run on all the nodes, we write a simple parallel command which takes parallel.sshloginfile as parameter and executes the command on all hosts as shown in Figure 9.

```
>$ parallel --tag --nonall --sshloginfile parallel.sshloginfile hostname -s
```

Figure 9: Example parallel command to test the setup

The above parallel command runs the command "hostname -s" on all the hosts in your sshloginfile. The "--nonall" option is new, and specifies to run the command on all hosts, but not read stdin for data. The --tag allows you to see which host produced what output. Also this command should run successfully without prompting for password for all hosts. So far everything that's been done above is only possible because of the contents of our .ssh/config file. Parallel's only involvement is that it uses the master ssh connection without "knowing" about it, and can run things to remote systems without having to supply a password. [17]

2.6 Command to run batch BLAST in parallel

Now that we have a working example using Parallel command, we will see how to perform query parallelism. Lets understand the steps by the help of snippets of the bash wrapper script.

Step 1: Chop your query sequence fasta file to multiple files each having atleast one query sequence.

```
cat input.fasta | parallel --progress --tmpdir TMPDIR --pipe --recstart '>' --blocksize 1k --files 'cat'
```

Figure 10: Command to split the fasta file to multiple files in parallel

In the command shown in Figure 10, we write the fasta on standard output and pipe it as input to the 'parallel' command. The options –recstart, --blocksize and –pipe tells the 'parallel' comand to read blocksize amount of bytes from standard input and split the input based on recstart and if there is a partial record then remove it from here and prepend it to next block. The –files option tells that print the output to a file instead of standard output (stdout) and –tmpdir tells where the files should be stored. Here parallel does not execute any meaningful command/job, all we do is print nothing on stdout. The idea is to split the file in parallel to multiple files. Note that here we are executing the command on a single node only.

Step 2: We now want to pass each file created by the above step to a script, blastScript which basically has a few variables set and TBLASTX command with all the parameters.

```
ls $TMPDIR | parallel -j +0 – sshloginfile parallel.sshloginfile "blastScript {} $DB" 2>$ERRFILE
```

Figure 11: Command to run BLAST program in parallel on all nodes

In the command shown in Figure 11, we are listing all the files in the temporary directory on stdout and piping that to parallel which reads each file from stdin and passes them to the script, so now we will have the script i.e., TBLASTX command running in parallel on all the nodes and on all the processors of each node. The options –j +0 tells 'parallel' to run as many jobs are there are cpus and –sshloginfile tells that it needs to ssh to all the remote hosts mentioned in the file. As we saw earlier, our parallel.sshloginfile just contains the command 'ssh hostname' and no other information like user name, port number or password. This ssh works because the ssh client before creating a new session, sees if there is a master session which it can make use of and thats the reason we need to create all the master sessions before running these commands, otherwise we will be prompted for password for each job that 'parallel' is running in parallel. At this point we can log which files were run on which node and how

much time it took for the script to run. We can confirm by running the 'ps' command all the jobs running on that node.

Step 3: Include the output file directory, error directory and TBLASTX command in the blastScript. Do not forget to set the right permissions on the bash script. (chmod 755 blastScript). The TBLASTX command is as shown in Figure 12.

```
tblastx -db $DB -query $INFILE -outfmt "7 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore slen" -evalue 0.01 -out $OUTFILE 2> $ERRFILE
```

Figure 12: TBLASTX command

3.0 Contig Analysis Tool (CAT)

In this section we cover the web application that we developed for visualizing and analyzing batch BLAST files. As mentioned in the previous sections, BLAST is the most extensively used tool for any kind of analysis and hence there was a need to parallelize it as well as develop a rather simple visualization tool that can help see the information for each contig. The idea is to see the file level statistics as well as each contig level statistics. In order to annotate the contig, it helps to see a bird's eye view of all the sequences that matched the contig and the position on the contig sequence where there was similarity. Since, the user is having full control of adding and deleting tags for each contig, the tool helps generate highly confident annotated data. We will see the design of database and the web application in detail below.

3.1 Application Platform

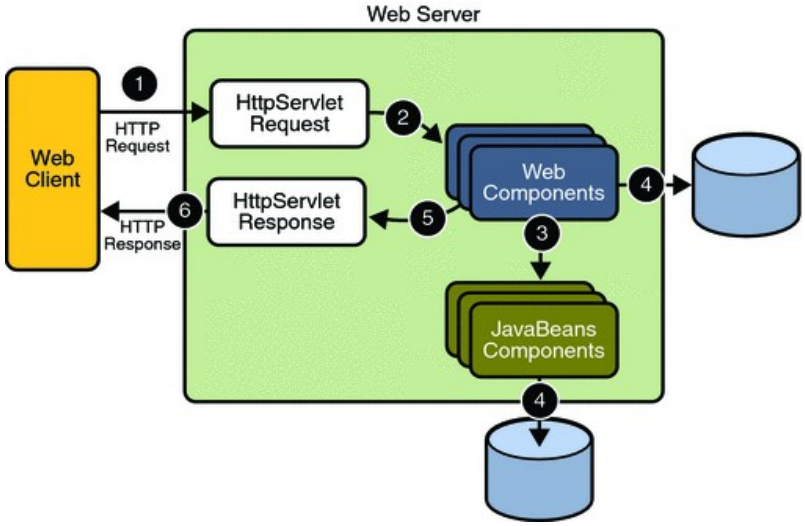


Figure 13: Java Web Application Request Handling (Source: Oracle JavaEE tutorial [4])

In order to develop this application, we selected the Java platform. The JavaEE API has many components, and for our application which is presentation oriented web application i.e. application that generates dynamic web pages in response to the requests we utilize Java Servlets and Java Server Pages (JSP) technology. The basic flow of client and server interaction is shown in Figure 13. The goal behind a N tier web application is separation of presentation, logic and data. For this application, we have build a 3 tier web application. The client starts by sending an HTTP request to the web server. The server which can understand both HTTP and Java i.e., the tomcat server converts these requests into

HTTP Servlet Request objects and passes it on to the web components. The web components interacts with either backend Java code or the database to generate the dynamic content. At this point, the web component can either send back the HTTP Servlet Response object or forward the request to another component. Eventually when the HTTP Response object is send, the web server again converts this back to HTTP response and sends it back to the client.

The web components in our application are Java Servlets and JSP pages. Java Servlets are Java classes that operate on requests and send back responses. JSP pages are muck like HTML pages, but in addition to static content it also includes a set of JSP directives or tags or scripting elements. Scripting elements are blocks of Java code embedded directly in JSP page to deliver dynamic content.

3.2 Design of our Application - MVC architecture

We have designed our application as per the Model View Controller (MVC) design architecture as shown in Figure 14. MVC design pattern for a web application separates presentation, logic and data from each other. A Model represents the state of a particular aspect of the application. The model notifies views of any changes to its state. A view accepts necessary information from the controller and renders a user interface to display. The view contains specification on how the data that it accessed from model should be represented. A controller sits between model and view and defines the behavior of an application. It handles user interactions and translates them into steps that the model needs to perform in order to reflect a change in state of the application. The controller then passes information to the view.

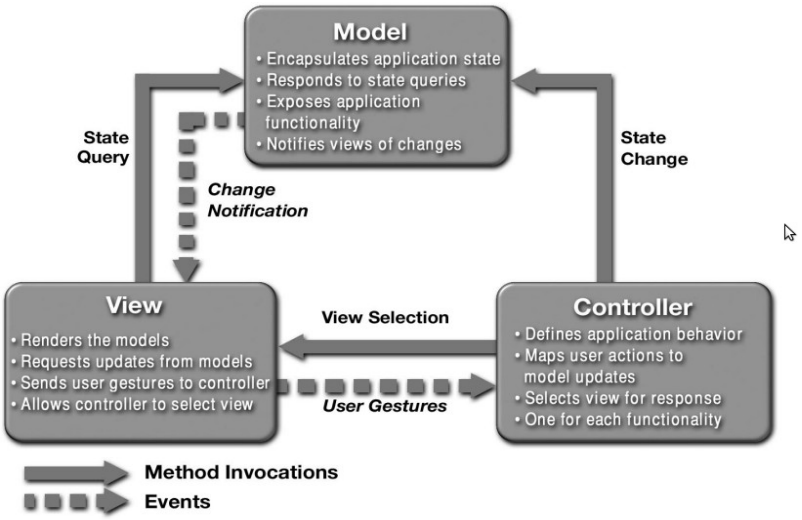
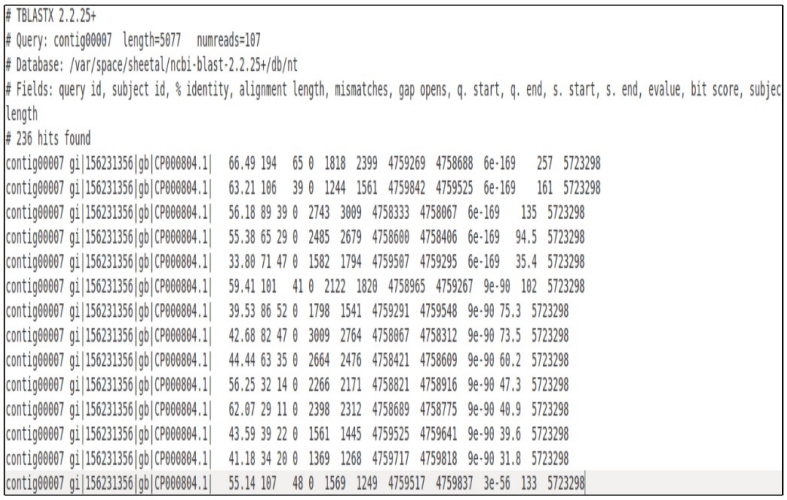


Figure 14: MVC Architecture (Source: Designing Enterprise Applications [6])

In our application, the model represents the data stored in RDBMS and the Java classes which talk to the database to access and modify the data. The HTML/JSP pages acts as views and the Java Servlets acts as controller.

3.3 Sample Input BLAST File



```
# TBLASTX 2.2.25+
# Query: contig00007  length=5077   numreads=107
# Database: /var/space/sheetal/ncbi-blast-2.2.25+/db/nt
# Fields: query id, subject id, % identity, alignment length, mismatches, gap opens, q. start, q. end, s. start, s. end, evalue, bit score, subjec
length
# 236 hits found
contig00007 gi|156231356|gb|CP000804.1|   66.49 194   65 0  1818  2399  4759269  4758688  6e-169    257  5723298
contig00007 gi|156231356|gb|CP000804.1|   63.21 106   39 0  1244  1561  4759842  4759525  6e-169    161  5723298
contig00007 gi|156231356|gb|CP000804.1|   56.18 89 39 0  2743  3009  4758333  4758067  6e-169    135  5723298
contig00007 gi|156231356|gb|CP000804.1|   55.38 65 29 0  2485  2679  4758600  4758406  6e-169   94.5  5723298
contig00007 gi|156231356|gb|CP000804.1|   33.80 71 47 0  1582  1794  4759507  4759295  6e-169   35.4  5723298
contig00007 gi|156231356|gb|CP000804.1|   59.41 101   41 0  2122  1820  4758965  4759267  9e-90  102  5723298
contig00007 gi|156231356|gb|CP000804.1|   39.53 86 52 0  1798  1541  4759291  4759548  9e-90 75.3  5723298
contig00007 gi|156231356|gb|CP000804.1|   42.68 82 47 0  3009  2764  4758867  4758312  9e-90 73.5  5723298
contig00007 gi|156231356|gb|CP000804.1|   44.44 63 35 0  2664  2476  4758421  4758609  9e-90 68.2  5723298
contig00007 gi|156231356|gb|CP000804.1|   56.25 32 14 0  2266  2171  4758821  4758916  9e-90 47.3  5723298
contig00007 gi|156231356|gb|CP000804.1|   62.07 29 11 0  2398  2312  4758689  4758775  9e-90 40.9  5723298
contig00007 gi|156231356|gb|CP000804.1|   43.59 39 22 0  1561  1445  4759525  4759641  9e-90 39.6  5723298
contig00007 gi|156231356|gb|CP000804.1|   41.18 34 20 0  1369  1268  4759717  4759818  9e-90 31.8  5723298
contig00007 gi|156231356|gb|CP000804.1|   55.14 107   48 0  1569  1249  4759517  4759837  3e-56  133  5723298
```

Figure 15: Sample BLAST file, tabular format

Figure 15 shows a screen shot of a sample BLAST file, the tabular format file. The lines starting with '#' are comment lines and all the fields description is given in the '# Fields' comment line. The lines without comments gives details of all the similar sequences or hits to our input sequence or contig. The evalue field is one important field, lower the evalue better is the similarity between our contig and the sequence in the database. Our application is modular as it can be extended to understand additional fields than the standard ones present in the file.

3.4 Database Design

Our application is storing data in relational database namely MySql database. Figure 16 shows our application's database schema diagram. The table name is at the top followed by field names, primary keys for each table are denoted by bold field names and foreign keys are denoted by the connections to the reference table field. Fields with same data types are color coded, like for example, yellow color is for Integer data type, pink for varchar data type. The user table stores information about

the user like id which is email address, name and password. The file table stores the BLAST files information that users' upload to our application. The file id is auto generated key which allows a user to store files with same names as well. The query table stores query id which in our case is the contig name. The query accession and gi number are for future use.

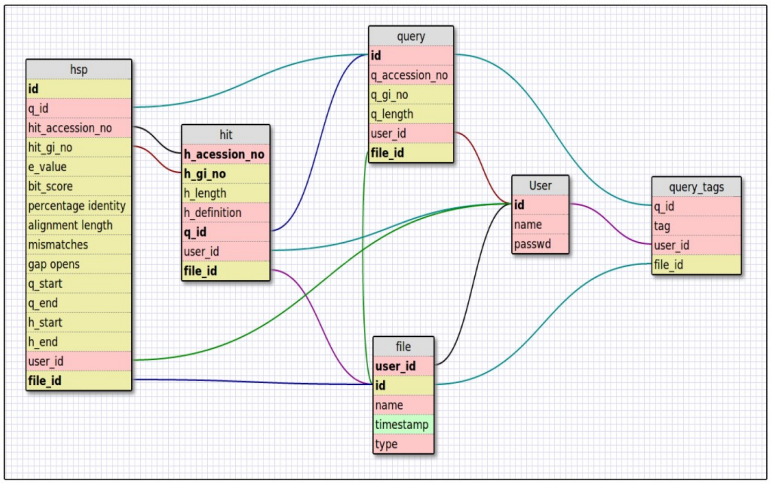


Figure 16: Schema Diagram (Generated using sqlDesigner tool)

The query id is unique for a particular file, so both query id and file id act as primary key. The query table allows to quickly compute file level statistics like total number of contigs, maximum and minimum length of a contig for a file. The hit table consists of fields like accession no and gi no (unique id given to each sequence present in the Genbank's database), hit sequence length, and hit definition. This table only consists of all unique hits for each contig. The hsp table contains all the fields that are present in the tabular BLAST file, the standard format which contains basic 12 fields.

3.5 Workflow of the CAT application

When the application starts, it shows the Login page, where the user is asked to enter their email address, shown in Figure 17.

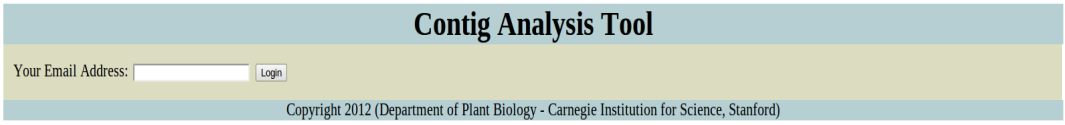


Figure 17: Login Page of CAT

For now, the application does not have user registration and profile creation functionality. The database design and application design is modular enough for inclusion of any extra features. Upon entering the email address and clicking on Login, the user is taken to the next page which is the main page of the application. On click of the Login button, the HTTP Request is send to the server. As mentioned earlier, the server converts this request to HTTP Request Servlet and passes it on to the Login.java Servlet. The application server's web container decides based on the URL in the HTTP request and the web application's deployment descriptor file (web.xml), to which web component should it pass the request. Login.java servlet stores the user information i.e., the email address in the session object, checks if the user exists in the database and if not then makes an entry of that user to the database. After this, it sends a response redirect to the Main.java servlet. Since the response is redirected by the servlet, the web application instructs the browser that it needs to fetch the new URL and hence it is a two step process. The Main.java servlet forwards the request to main.jsp which calls a model to fetch all the files uploaded by that user and for the default selected file to fetch all the contigs in that file and then puts that information in HTML drop down list and passes on the response back to the user as shown in the Figure 18. At any point if the session expires, then the user is redirected back to the login page.

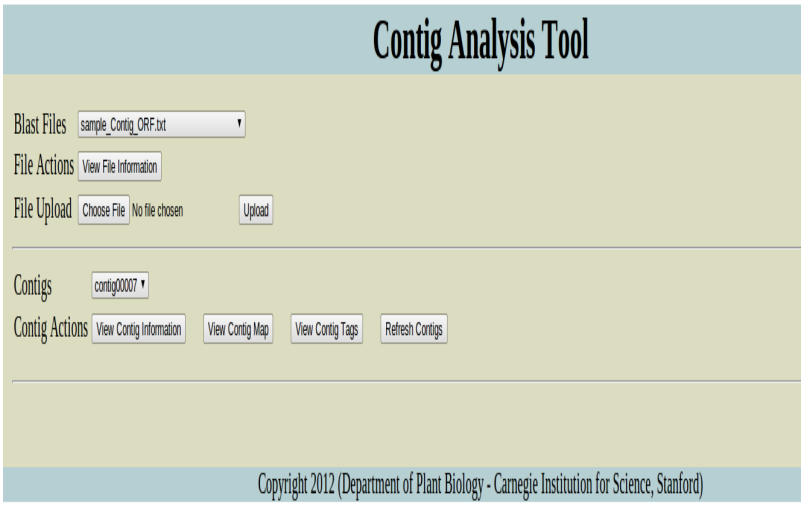


Figure 18: Main page of CAT

3.5.1 File Actions

Once on the main page, the user can perform different actions to do file level analysis, to upload a new file, or to do contig level analysis for a particular file. We will see each action in detail below.

3.5.1.1 File upload action

The file upload box when clicked will open up the browse menu of the filesystem from where the user can enter the BLAST file. As of now the application, only supports tabular BLAST output file. Onclick of upload button, the client browser send an HTTP Post request to the server, server determines based on the URL and sends the request to the servlet Main.java. The servlet creates an object that is responsible to read the file, convert it to sql statements and to call model to execute those statement to store the file data in the database. The servlet fires off a new thread and calls run method of the object it created on the new thread.

3.5.1.2 File Select Action

The user can select different uploaded files for analysis from the dropdown list. On change of the file, an asynchronous call is made to get the contigs belonging to that file and the contigs view is refreshed without loading the complete page.

3.5.1.3 View File Information Action

When the user clicks on “View File Information” button, an asynchronous (AJAX) call is made to the server and based on the URL in the request, the corresponding view is called for. In this case, the files.jsp page is called to get all the file related statistics as shown in Figure 19 below.

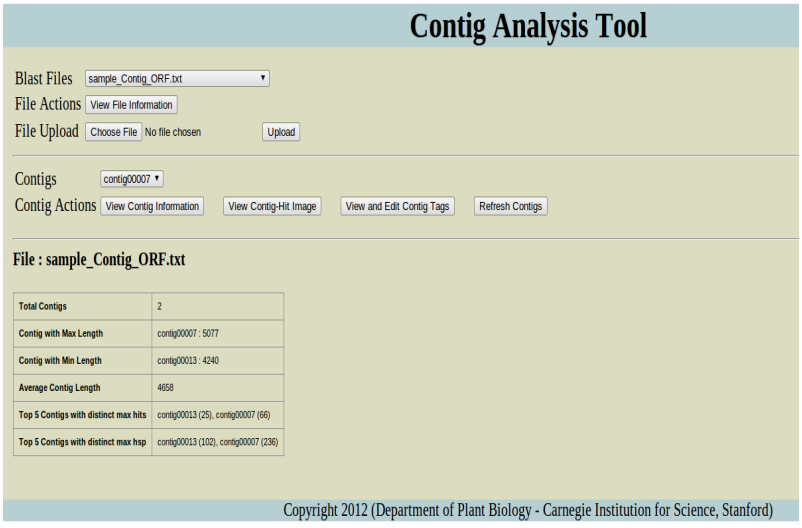


Figure 19: Result of View File Information Action

3.5.2 Contig Actions

The user can take following actions to analyze a contig for the selected file:

3.5.2.1 View Contig Information

Similar, to the file statistics, we would want to see contig level information like the selected contigs length, the maximum no of hits and hsps to that contig, the evalue range of all the hits, and so on. Similar to the file action, this is also an ajax action, so the contigs.jsp page sends back the content which gets written in the inner HTML of the div block.

3.5.2.2 View Contig-Hit Image

In order to get a bird's eye view of the BLAST hits to the selected contig, we generate an image map. The image map is generated by calling a Perl script from the program. The Perl script extends the Bioperl's Graphics library [8] to generate the image map. As shown in Figure 20 below, the horizontal graded line represents the contig length and all the yellow bars at the bottom represents hits to the contig aligned according to the start and end positions on the contig. All the High Scoring Segment Pairs (HSP) for a hit are denoted by a dotted line connector. In case if the HSP's start and end overlap then it is denoted by a dark black vertical line in the yellow bar. All the hits name are clickable and they hyperlink to the NCBI's page for that hit accession number which allows for detailed analysis on the hit.

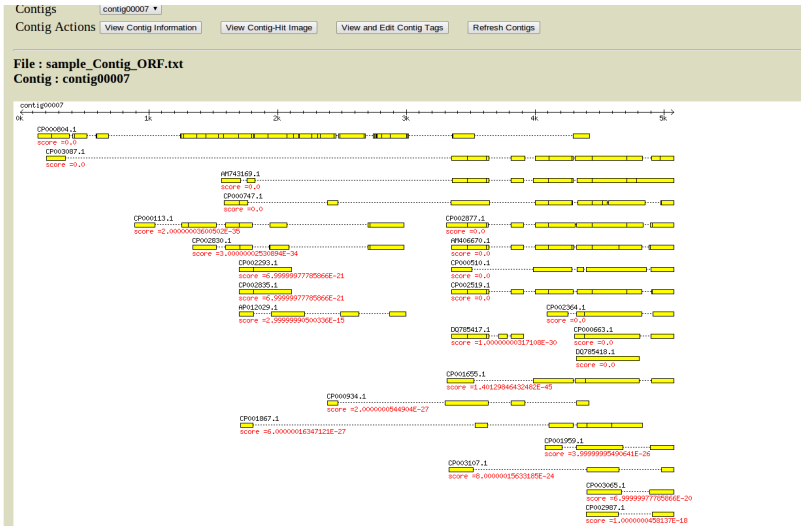


Figure 20: Result of Contig-Hit Image Action

3.5.2.3 View and Edit Contig Tags

This feature is the most useful feature to tag what you analyzed from the previously mentioned functionalities. As shown in Figure 21 below, all the tags for that contig are displayed as a list, the user can enter a tag in the text box and then click on add button to add a new tag or enter the existing tag in the text box and click on delete button to delete a tag. The add and delete button's action invokes client side JavaScript function which makes an asynchronous call to the server to add or delete the tag and refresh the list.

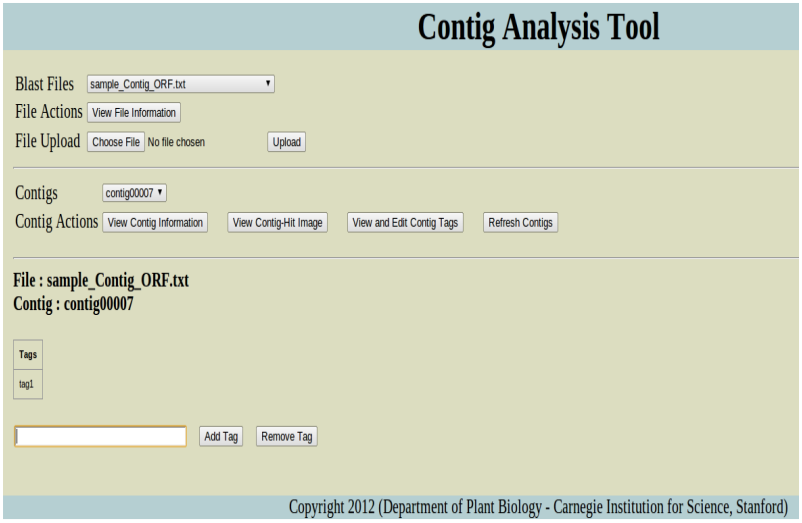


Figure 21: Result of View and Edit Contig Tags

3.5.2.4 Refresh Contigs

As previously mentioned under the File Upload section, when the user uploads a BLAST file, a new thread is spawned to store the file in the database. The thread starts loading data as soon as it is done reading information for each contig in the file. So when the response is send back for the upload action, the new file uploaded gets listed in the file list and all the contigs which are loaded in the database also gets listed for run-time analysis in the contigs list. The refresh button gives the user a functionality to list other contigs which got loaded after the response was send back.

4.0 Real World Virome Data Analysis

The virome data is obtained from the sample taken from thermophilic mircrobial mats from Octopus in Yellowstone National Park. The mat sample was centrifuged and filtered to extract viral cells and then the viral DNA was amplified using Multiple displacement amplification (MDA) reaction. MDA is a DNA amplification technique which multiplies DNA samples rapidly from very little amount of sample. The amplification reaction starts with random hexamer primers, high fidelity DNA polymerase and the template DNA. This amplified material was then sequenced using 454 titanium pyrosequencing technique. A virome database of 180,141,543bp was generated and it consisted of a total of 501,370 reads.

4.1 Virome Reads

In order to understand what the data set contains, we extracted reads length statistics, reads species distribution and read GC content distribution.

4.1.1 Read Length Distribution

In order to generate reads length distribution, we first extracted length information for each reads and then passed on the length data to R software for plotting the distribution. The read fasta file is a file of all read sequences with sequence always followed by a definition line starting with '>' (i.e., fasta format).

```
>$ ls reads.fasta | xargs grep ">" >> allreads.txt
>$ head -1 allreads.txt
>F2HZBTK02IV99S rank=0000076 x=3531.0 y=366.0 length=59
>$ ls allreads.txt | xargs cut -d' ' -f5>> readsLen.txt (extract reads name from all the deflines)
>$ head -1 readsLen.txt
Length=59
>$ ls readsLen.txt | xargs cut -d'=' -f2>> readsLength.txt
>$ head -1 readsLength.txt
59
```

Figure 22: Bash commands to extract data for read length distribution

Example defline of the read fasta file looks as below:

>F2HZBTK02IV99S rank=0000076 x=3531.0 y=366.0 length=59 (>ReadName rank x y length)

Now to extract the read name and length, we ran the following commands as shown in Figure 22.

From the deflines in the reads fasta file, we first extract all the length=x column for all the reads and put them in a file. Later, we split on '=' such that we only have the numeric length information in the file. This is an alternative approach to writing a Perl script to extract length from the fasta file. Now to plot the reads length distribution, we load the dataset in R and plot the histogram using the following commands as shown in Figure 23.

```
>$ R (start R, assuming we already have R installed)
> dat2 ← read.table("path_to_file/readsLength.txt", header=TRUE)
> attach(dat)
> hist(ReadLen, main="Read Length Distribution", xlab="Reads Length");
```

Figure 23: R commands to generate Read Length Distribution plot

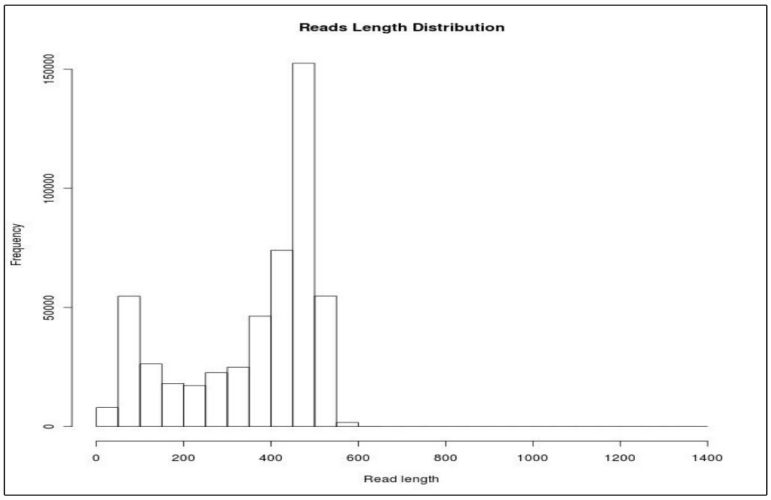


Figure 24: Read Length Distribution (Generated using R)

Figure 24 shows the distribution of reads length, we see that we have maximum reads of length 475 which conforms with the fact that 454 titanium generates reads of length ~400 to ~500 base pairs. There is a second peak at the tail because sometimes 454 preferentially amplifies smaller fragments within the input data, before amplifying larger fragments.

Table 1 shows the read length statistics like min, max, mean, median and mode of and the commands we ran in R to get them.

Reads Length Statistics	Values	R command
Minimum Read Length	40	min(ReadLength)
Maximum Read Length	1385	max(ReadLength)
Mean Read Length	359.39	mean(ReadLength)
Median Read Length	425	median(ReadLength)
Mode Read Length	475	names(sort(-table (ReadLength)))[1]

Table 1: Read Length Statistics

In order to compute mode of read length, we create another table in R and copy only the read lengths into it. Then we multiply each value by 1 and sort the values so that the topmost element gives us the most frequently occurring value ie., mode of read length.

4.1.2 Read Species Distribution

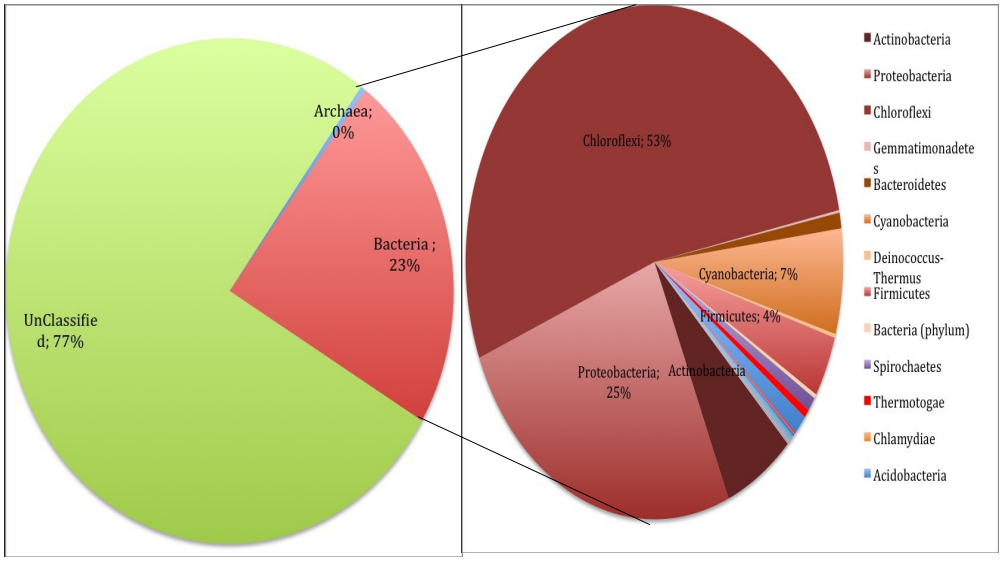


Figure 25: Read Species Classification (Using FCP [13] by Michelle and Todd)

FCP is a software package that provides classification to metagenomic fragments using composition and homology. As one can see from Figure 25, our dataset has very little bacterial and archaeal sequences whereas there were a lot of unclassified sequences, possibly representing viral sequences.

4.1.3 Reads GC- content Distribution

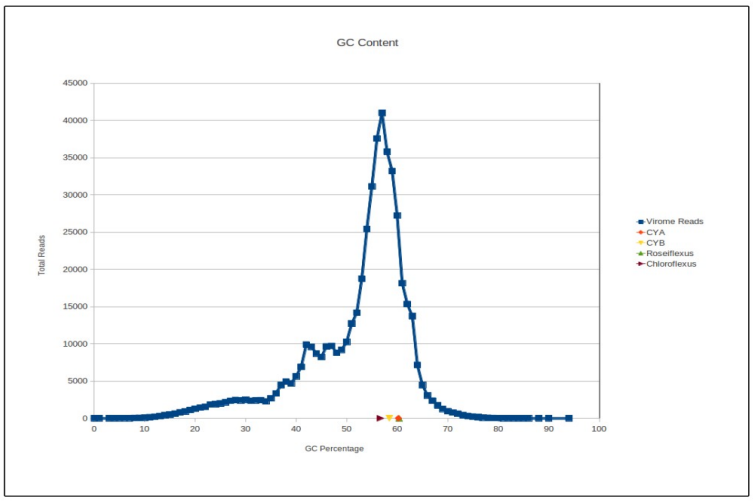
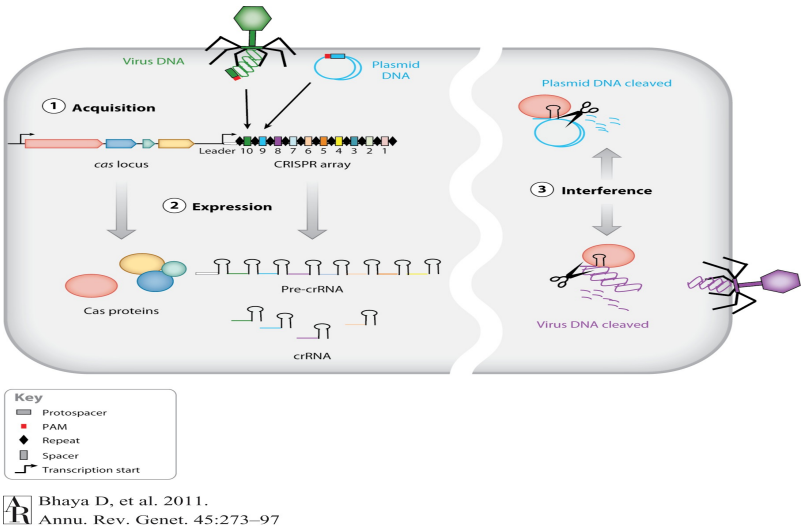


Figure 26: GC distribution of Reads

Figure 26 above shows the GC content distribution of the virome reads. The GC content of Synechococcus OS' A and Synechococcus OS' B, Roseiflexus sp. and Choloroflexus sp. are plotted on the graph as well. The GC content of our virome data is similar to that present in the above mentioned hosts.

4.2 Clustered Regularly Interspaced Short Palindromic Repeat (CRISPR) Analysis

Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR) are multiple short direct repeats found in the DNA of many bacteria and archaea.[5] The repeats are of size from about 23 to 47 base pairs. The spacers which are generally unique in a genome separate the repeats. Some spacers match to regions in phage genomes, which would mean that the host was once infected with this phage and to resist itself from further attacks it keeps a “memory” of them by inserting a spacer sequence between the repeats as shown in Figure 27.

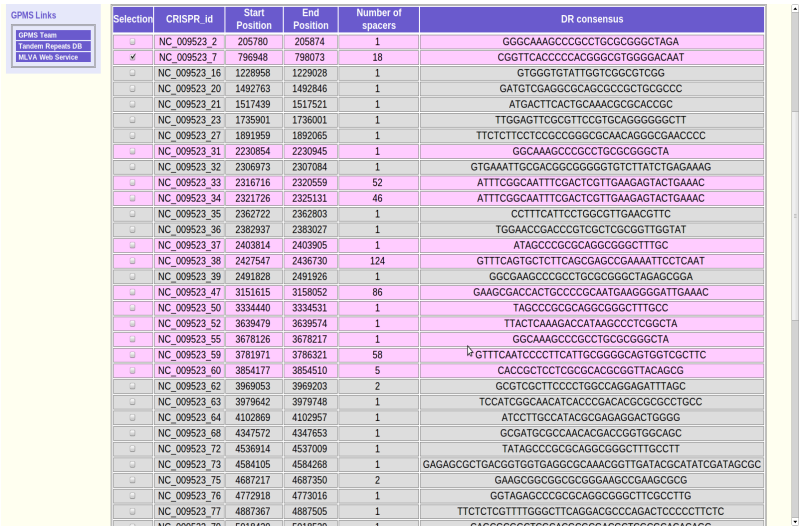


Bhaya D, et al. 2011.
Annu. Rev. Genet. 45:273–97

Figure 27: CRISPR-Cas adaptive mechanism (Source: CRISPR-Case Systems, Bhaya D. et al. 2011)

Steps performed by Michelle for CRISPR analysis

1. Extract CRISPR Repeat sequences from fully sequenced OSA, OSB', RS-1 and Chloroflexus She went to CRISPR finder website, inserted the above mentioned bacterial genomes and the program outputs potential repeat sequences as shown in Figure 28. We also did this for our 2008 metagenome, anchor genome. Potential repeats were manually inspected and only spacers were only extracted from reads containing more than three repeats. For all those filtered repeats, we pulled up its corresponding spacer sequences as shown in Figure 29.



Selection	CRISPR_id	Start Position	End Position	Number of spacers	DR consensus
☐	NC_009523_2	205780	205874	1	GGGCAAAGCCCGCCTGCGCGGGCTAGA
☑	NC_009523_7	796948	798073	18	CGGTTCACCCCCACGGGCGTGGGGACAAT
☐	NC_009523_16	1228958	1229028	1	GTGGGTGTATTGGTCGGCGTCGG
☐	NC_009523_20	1492763	1492846	1	GATGTCGAGGCGCAGCGCCGCTGCGCCC
☐	NC_009523_21	1517439	1517521	1	ATGACTTCACTGCAAACGCGCACCGC
☐	NC_009523_23	1735901	1736001	1	TTGGAGTTCGCGTTCCGTGCAGGGGGGCTT
☐	NC_009523_27	1891959	1892065	1	TTCTCTTCCTCCGCCGGGCGCAACAGGGCGAACCCC
☐	NC_009523_31	2230854	2230945	1	GGCAAAGCCCGCCTGCGCGGGCTA
☐	NC_009523_32	2306973	2307084	1	GTGAAATTGCGACGGCGGGGGTGTCTTATCTGAGAAAG
☐	NC_009523_33	2316716	2320559	52	ATTTCGGCAATTTCGACTCGTTGAAGAGTACTGAAAC
☐	NC_009523_34	2321726	2325131	46	ATTTCGGCAATTTCGACTCGTTGAAGAGTACTGAAAC
☐	NC_009523_35	2362722	2362803	1	CCTTTCATTCCTGGCGTTGAACGTTC
☐	NC_009523_36	2382937	2383027	1	TGGAACCGACCCGTCGCTCGCGGTTGGTAT
☐	NC_009523_37	2403814	2403905	1	ATAGCCCGCGCAGGCGGGCTTTGC
☐	NC_009523_38	2427547	2436730	124	GTTTCAGTGCTCTTCAGCGAGCCGAAAATTCCTCAAT
☐	NC_009523_39	2491828	2491926	1	GGCGAAGCCCCGCCTGCGCGGGCTAGAGCGGA
☐	NC_009523_47	3151615	3158052	86	GAAGCGACCACTGCCCCGCAATGAAGGGGATTGAAAC
☐	NC_009523_50	3334440	3334531	1	TAGCCCGCGCAGGCGGGCTTTGCC
☐	NC_009523_52	3639479	3639574	1	TTACTCAAAGACCATAAGCCCTCGGCTA
☐	NC_009523_55	3678126	3678217	1	GGCAAAGCCCGCCTGCGCGGGCTA
☐	NC_009523_59	3781971	3786321	58	GTTTCAATCCCCTTCATTGCGGGGCAGTGGTCGCTTC
☐	NC_009523_60	3854177	3854510	5	CACCGCTCCTCGCGCACGCGGTTACAGCG
☐	NC_009523_62	3969053	3969203	2	GCGTCGCTTCCCCTGGCCAGGAGATTTAGC
☐	NC_009523_63	3979642	3979748	1	TCCATCGGCAACATCACCCGACACGCGCGCCTGCC
☐	NC_009523_64	4102869	4102957	1	ATCCTTGCCATACGCGAGAGGACTGGGG
☐	NC_009523_68	4347572	4347653	1	GCGATGCGCCAACACGACCGGTGGCAGC
☐	NC_009523_72	4536914	4537009	1	TATAGCCCGCGCAGGCGGGCTTTGCCTT
☐	NC_009523_73	4584105	4584268	1	GAGAGCGCTGACGGTGGTGAGGCGCAAACGGTTGATACGCATATCGATAGCGC
☐	NC_009523_75	4687217	4687350	2	GAAGCGGCGGCGCGGGAAGCCGAAGCGCG
☐	NC_009523_76	4772918	4773016	1	GGTAGAGCCCGCGCAGGCGGGCTTCGCCTTG
☐	NC_009523_77	4887367	4887505	1	TTCTCTCGTTTTGGGCTTCAGGACGCCCAGACTCCCCCTTCTC

Figure 28: Sample picture for Roseiflexus Repeat sequences (Obtained from CRISPRfinder)

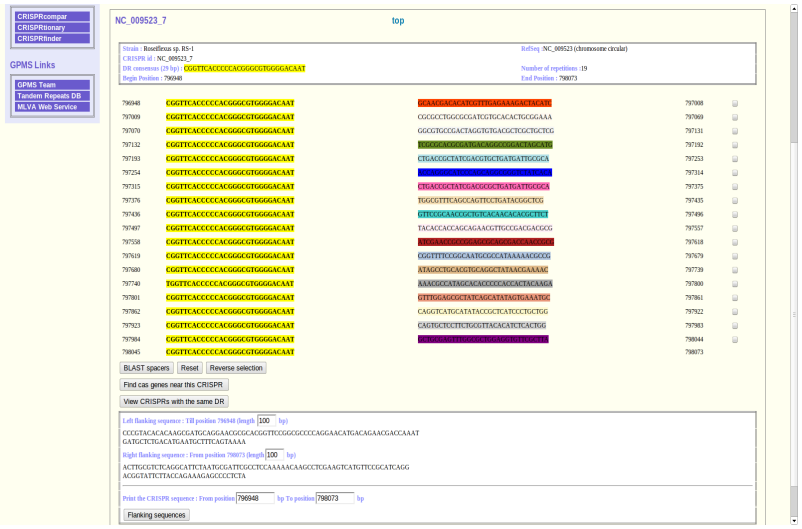


Figure 29: Sample picture for Roseiflexus Spacer sequences for a CRIRPR Id (Obtained from CRISPRfinder)

2. Then we compared the spacer sequences to our virome dataset using BLASTN. Only those reads were tagged who had 85% identity spanning 70% of the spacer length.

3. This exercise gave us a total of

   1534 Cyano spacers and 2828 Roseiflexus spacers

4.3 Read Preprocessing

Todd passed our dataset through Prinqseq (Schmieder et al 2011) for preprocessing before assembly. We removed a total of 6.5% of reads leaving 32,568 total reads. We filtered all reads which were smaller than 50 bp and reads with low complexity regions to account for MDA amplification artifacts.

4.4 Assembly

Todd performed 3 different assemblies using different filtration strategies to compare contigs generated from each of them. All 3 assemblies were performed using Newbler, an assembly software developed by 454 Life Sciences, a Roche Diagnostics Company, for de novo DNA sequence assembly of data generated by 454 technologies. The first two assembly strategies filtered reads classified as bacterial based on conservative, aggressive (strict threshold), and the last one was assembly of binned reads. In conservative approach, we removed all the reads that were classified by NUCmer (threshold,

maxmatch =30) as bacterial (37K), leaving a total of 430K reads for assembly. Later, we removed additional reads that were reclassified by BLASTN as bacterial (removed 214 contigs, 1244 reads). In the aggressive approach, we removed any read classified by FCP as bacterial (threshold, evalue=10^{-5}) which left 150k putative viral reads. The assembly resulted in 1652 total contigs (230 > 1000bp, 1446bp avg contig size). As a final strategy, we binned groups of related reads to reference phage genomes using NUCmer, and then assembled the set of recruited reads individually. This resulted in much greater specificity of the assembled target, however, the main disadvantage is significantly reduced sensitivity. We also validated our assembly by recruiting mate-pair reads from similar environments.

Finally, we now have 4371 contigs with a contig of maximum length 8459. Figure 30, shows the distribution of contig length.

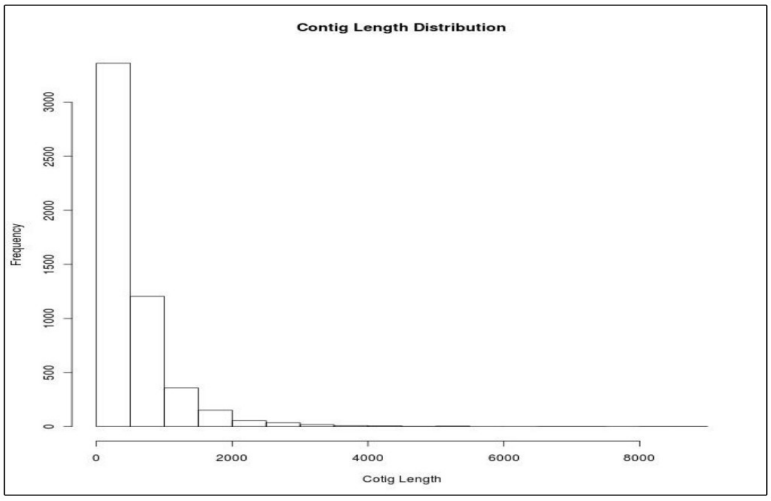


Figure 30: Contig Length Distribution (Generated using R)

4.5 Binning

We have applied both the binning methods described above in the Introduction section. Since there are no marker genes for our virome dataset, we had to rely on sequence similarity to CRISPR spacers which we discussed in the earlier section. Here we will see the results of the second approach, i.e., tetranucleotide analysis.

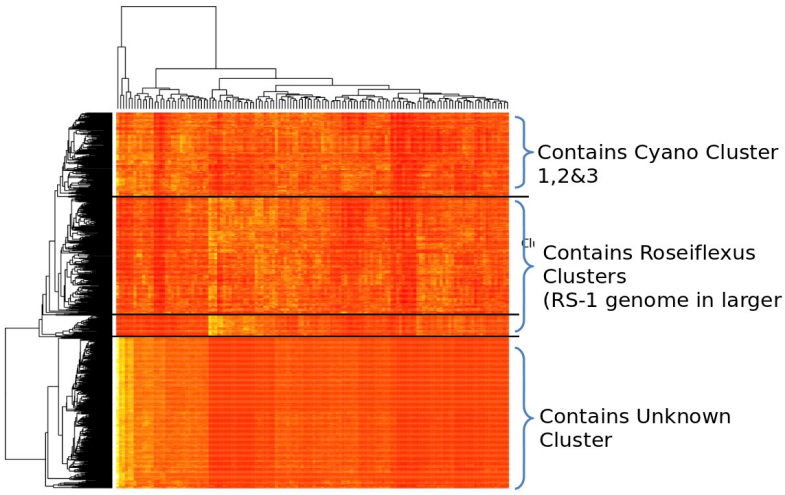


Figure 31: Tetranucleotide analysis plot (Generated by Todd)

Todd performed tetranucleotide analysis on contigs and we now have a total of 5 clusters as shown in Figure 31. The tetramer clustering was performed on the large contigs (>1000bp) shown on the left. The top part shows the clustering of 256 possible tetramers. The dark red/orange color indicates very low freq of a given tetramer while yellow/white indicates high freq. The plotted rate is simply the coverage of each tetramer in each contig (total sum/contig length). Reads containing CRISPR hits were also used to further classify tetranucleotide clustering. Cluster 1, 2 and 3 contains Cyanobacteria (both OS' A and OS' B types) CRISPR spacers and Roseiflexus cluster contains solely Roseiflexus CRISPR spacers whose counts are given in Table 2.

Cluster	Unique Spacer Hits	Total number of reads with spacer hits
Cluster 1	2	50
Cluster 2	No significant spacer hits	No significant spacer hits
Cluster 3	7	408
Roseiflexus Cluster	33	1170

Table 2: Tetranucleotide cluster classification using CRISPR hits

4.6 Gene Prediction

We have performed both gene prediction methods, evidence-based by performing TBLASTX similarity search against genbanks 'nt' database. We are yet to analyze the results. We have passed our

contigs through de novo gene prediction program, MetaGeneMark (Zhu et al 2010).

4.8 Contig Similarity

To get an insight into how similar the contigs are to each other, we performed TBLASTX and BLASTN of contigs against itself. Both the BLAST's were performed with default evalue, 10. To visualize what the BLAST results had to say, we created a 3D scatterplot of each contig against other with the 3rd axis showing total number of hits. The contigs are ranked in the sorted order of contig length and are placed on both x and y -axis as shown in Figure 32 and 33.

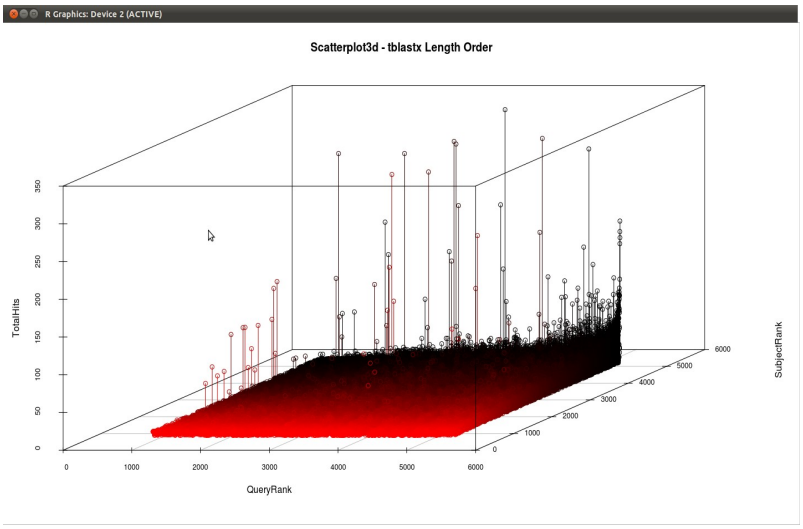


Figure 32: 3D Scatterplot of Contig vs. Contig TBLASTX (Generated using R)

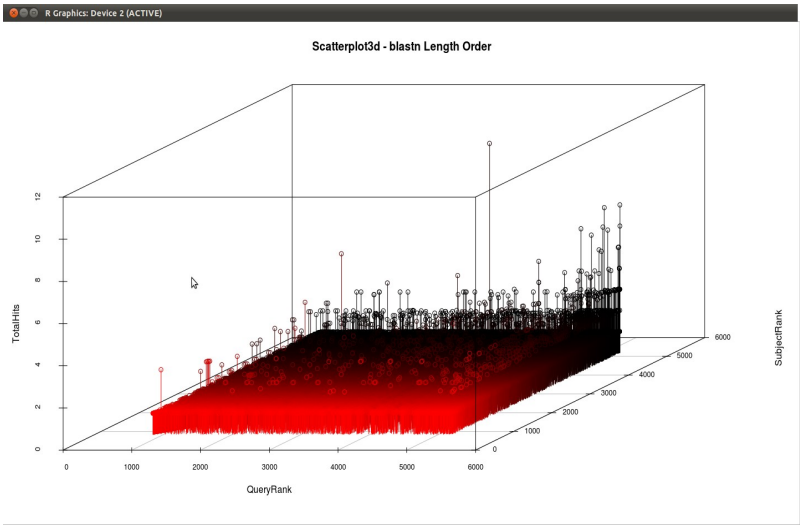


Figure 33: 3D Scatterplot of Contig vs. Contig BLASTN (Generated using R)

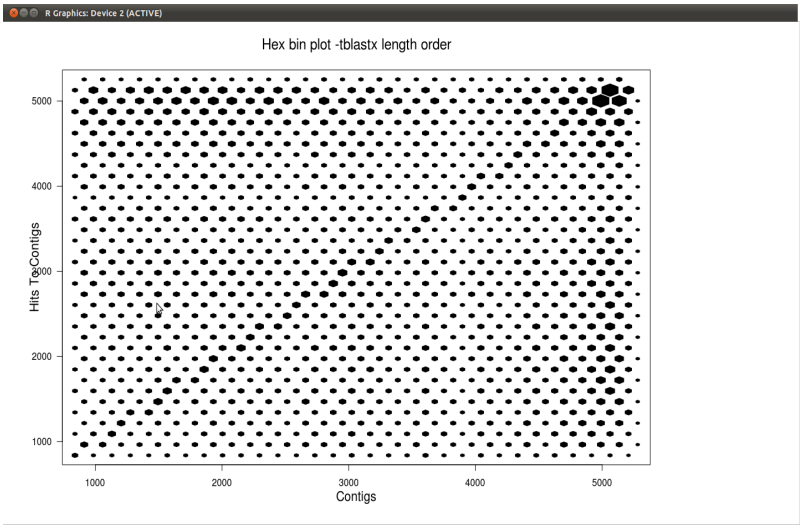


Figure 34: Hexagon plot of Contig vs. Contig TBLASTX results (Generated using R)

In order to view the plots more clearly i.e., to view bivariate plots with datasets of large n, we created another plot applying hexagon binning. In hexagon binning, the xy plane is tessellated by a regular grid of hexagons and then the count of number of points falling in each hexagon is stored in the dataset. The hexagons with count > 0 are plotted with varying the size of hexagon with respect to the count.

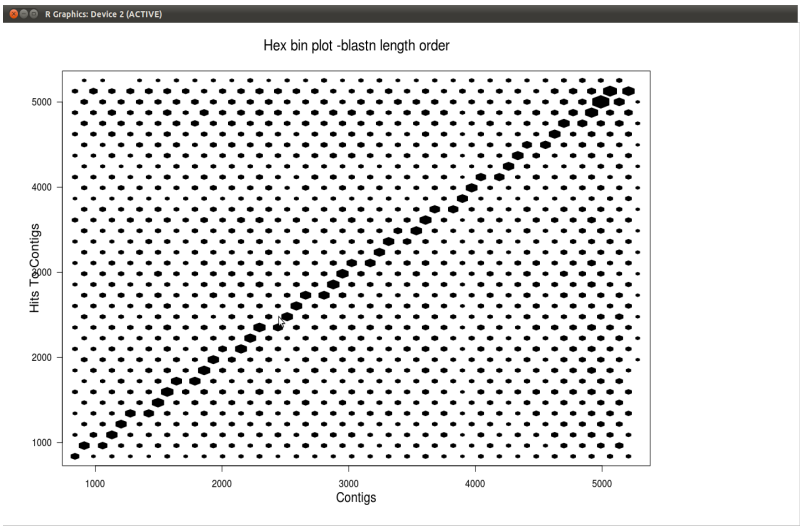


Figure 35: Hexagon plot of Contig vs. Contig BLASTN results

From the plots in Figure 33 and 35 for BLASTN results we can see that only contigs of larger size tend to recruit other large contigs. The diagonal line shows that each contig has complete similarity to itself. However, for TBLASTX from Figure 32 and 34, we see a more dense region for the larger

contigs. That would suggest some conserved domains or proteins that are present in large contigs are also seen in other large contigs. There is another less dense region at the top, kind of a dense line going from top left to right. That would mean that all our little contigs have amino acid level similarity to the larger contigs, again indicative of some similar domain or proteins present in them.

5.0 Conclusion and Future Work

In this project, we applied bioinformatics workflow to process viral DNA sequence data obtained from Yellowstone National Park. We performed sequence similarity binning on virome reads by CRISPR analysis and sequence composition binning on virome contigs by tetranucleotide analysis. In order to speed up sequence similarity using BLAST, we developed a wrapper script to run the BLAST program in parallel on a High Performance Computing Cluster. We also developed a web application called Contig Analysis Tool for the analysis of BLAST generated output file. The tool can be extended further to include the following functionalities:

1. Support for different output file formats.
2. Include an export facility - to export and save the results in a file.
3. Include export facility for each contig – to store all the raw data stored in the database in a file
4. Support for extracting functions for all the high scoring segment pairs from NCBI and then upload the information in our application to be able to view the function of each hsp in the Contig-Hit Image.
5. Login authentication functionality

References

[1] Chen K, Pachter L (2005) Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities. PLoS Comput Biol 1(2): e24. doi:10.1371/journal.pcbi.0010024

[2] Ewing B, Green P: Basecalling of automated sequencer traces using phred. II. Error probabilities. Genome Research 8:186-194 (1998).

[3] Gordon, D., C. Abajian, and P. Green.1998. Consed: a graphical tool for sequence finishing.Genome Res.8:195-202.

[4] Jendrock E., Ball J., Carson D., Evans I., Fordin S., Haase K. (September 2010). The Java EE 5 tutorial. Oracle. Retrieved April 12th, 2012, from http://docs.oracle.com/javaee/5/tutorial/doc/geysj.html

[5] Heidelberg J.F., W.C. Nelson, T. Schoenfeld and D. Bhaya Germ warfare in a microbial mat community: CRISPRs provide insights into the co-evolution of host and viral genomes. PloS ONE 2009; 4(1):e4169.

[6] Kassem N. (2000). Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition. Addison-Wesley Professional.

[7] Kunin, V., Copeland, A., Lapidus, A., Mavromatis, K., & Hugenholtz, P. (2008). A bioinformatician's guide to metagenomics. Microbiology and Molecular Biology Reviews : MMBR, 72(4), 557-78, Table of Contents. doi:10.1128/MMBR.00009-08

[8] Lincoln Stein. (2001). Bio::Graphics module on CPAN. Cold Spring Harbor Laboratory. Retrieved April 10th, 2012, from http://search.cpan.org/~lds/Bio-Graphics-2.28/lib/Bio/Graphics.pm

[9] Lowe, T. M., and S. R. Eddy. 1997. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. Nucleic Acids Res. 25:955-964.

[10] Metagenomics, Wikipedia. Retrieved on 12th April , 2012 from http://en.wikipedia.org/wiki/Metagenomics

[11] Morgulis A, Gertz EM, Schaffer AA, Agarwala R: A fast and symmetric DUST implementation to mask low-complexity DNA sequences. J. Comput. Biol 2006,13:1028-1040.

[12] Noguchi, H., J. Park, and T. Takagi. 2006. MetaGene: prokaryotic gene finding from

environmental genome shotgun sequences. Nucleic Acids Res. 34:5623-5630.

[13] Parks, D.H., MacDonald, N.J., and Beiko, R.G. (2011). Classifying short genomic fragments from novel lineages using composition and homology. BMC Bioinformatics, 12:328

[14] Rieffel M, Gill T, White W. Bioinformatics Clusters in Action. Paracel, Inc.

[15] Schmieder R and Edwards R: Quality control and preprocessing of metagenomic datasets. Bioinformatics 2011, 27:863-864. [PMID: 21278185]

[16] Schoenfeld T, Patterson M, Richardson PM, Wommack KE, Young M, et al. Assembly of viral metagenomes from yellowstone hot springs. Applied and environmental microbiology. 2008;74:4164

[17] Tange O. (n.d.). Documentation for GNU parallel. Retrieved April 15th , 2012, from http://www.gnu.org/software/parallel/man.html

[18] Tange O. GNU Parallel - The Command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.

[19] Teeling, H., A. Meyerdierks, M. Bauer, R. Amann, and F. O. Glockner. 2004. Application of tetranucleotide frequencies for the assignment of genomic fragments. Environ. Microbiol. 6:938-947.

[20] Teeling H, Waldmann J, Lombardot T, Bauer M, Glöckner FO. TETRA: a web-service and a stand-alone program for the analysis and comparison of tetranucleotide usage patterns in DNA sequences. BMC Bioinformatics 2004, 5:163