Environmental Sample Classification E.S.C., Josh Katz and Kurt Zimmer

Goal: The task we were given for the bioinformatics capstone class was to construct an interface for the Pipas lab that integrated the pipeline of tools used for sequence analysis into a website. Our motivation for accomplishing this task is to make it easier for metagenomic biologists to analyze sequences using standard tools, provide a centralized resource to the relevant tools, allow researchers to visualize and understand results, be accessible to anyone with a basic biological backgrounds, and have a web server where the sequence data and outputted information can be stored and easily accessed.

Biological Background: The lab is involved in the study of metagenomics, which is the analysis of the abundant genetic material found in environmental samples. This is a way of gathering randomized genetic information, based on the location you are in, without being biased towards species already discovered, new species, or any other factors beyond the location. This allows us to possibly identify new species that we are not familiar with and possibly build complete genomes of a viral organism that we believe exists.

However there is a problem with the data set that we receive. Each sequence is processed and read so that it is only ~300 base pairs (bp) long, which is another way of saying that it is a string of 300 characters, which are the letters 'A', 'T', 'C' and 'G'. These 4 characters comprise the 4 options of nucleotides, which are the building blocks of DNA, and thus the characteristics and definition of an organism. However, while the sequences are only 300bp, a viral genome can vary in length from thousands of base pairs to over a million. For reference, the human genome is about 3 billion base pairs long, but even though viral genomes are much smaller, they aren't small enough to be able to identify them solely based on a 300bp segment.

Given this information, to discover new viruses, we must first determine if the sequence we are working with is viral, and then determine if it is part of a unique viral organism and attempt to assemble a whole genome of the possible organism. You can imagine how difficult it can be to determine what species a particular sequence belongs to, since all organisms use the same building blocks. However there are databases and tools that assist with our problem that can compare our sequence to databases.

Another issue that is faced is that there are an enormous amount of sequences involved. Each environmental sample has an unknown population of organisms and overall our lab has sequence data for over one and half million reads. This makes our task much more difficult, because there is not enough man power to go over each individual sequence and then assemble some of the sequences together into whole genomes, as it is a large task, even computationally. For example, sometimes the query processing

of a large batch of sequences on a single database using the Basic Local Alignment Search Tool (BLAST) can take weeks to complete.

Solution: For the problems that we have discussed so far, the implementation of the solution must address all of them and allow us to achieve our motivations for the project. To accomplish all of our goals we decided to write a web interface on an apache web server that is accessible from anywhere in the lab or outside with Pitt's remote access connection. We wanted anyone with a web connection and access to the lab to be able to run the set of tools that we built and also make it easy to access without any prior installation or setup procedure. This way, all the user has to do is upload a file and select their tools, inputs, and options.

To make it easy to use for anyone with a basic biological background, we constructed wrappers for the biological tools and databases that we used, so that all the user has to be aware of is what the tool is, what each option means, and how to upload a file. This and other information will be presented to the user in a yet to be created help file that is available as a standalone HTML document, or can be linked to individual topics from the appropriate link. For example, if a user wants to know what the option 'blastn' implies, there will be a link that will take them to the topic describing the available BLAST programs.

To accomplish the goals of providing useful tools and keeping them in a centralized location, we built wrappers for all of the tools that allow us to interface between the users and the command line interface. We accept the inputs from the users, including uploaded files, run the tool in the specified manner, and then return the tool output and our own output processing for easier data viewing to the user. Since all of these tools are contained on the same website, the next tool in a particular 'pipeline' sequence can then be run on the output received from the last.

And finally to allow users to upload files and view output, we created a system where user's uploaded files are allowed to be run on any tool, controlling for allowable input files. Then each tool has its own output list in an HTML file which consists of the tool's original output, the output file's location and data processing and visualization that we might do. We also have a file viewer where the user can select the type of file they want and it is presented to them in the webpage and also as a link to the original file, so that any of the output or HTML files can be viewed at any time.

Tools: The way we chose our tools to include in the website was to investigate what sort of tools were useful to members of the labs and what other modules we would need to include pulling the whole design of the website and user interface together.

Programming Languages & Technologies:: We chose our both our programming language and our technologies that we used based on what would be the easiest to work with, which allowed us to meet our goals in the project, and accomplish the goal of interacting with the user's inputs to provide relevant output data.

Perl: We started working with Perl to begin the project, since most bioinformatics tools are composed in this programming language. This allowed us to gain experience in the language so we could work on future bioinformatics projects in Perl, and it also made it easier to interface with the existing biological tools that we chose, such as Annotator. For instance, Perl contains a module called BioPerl that interfaces with the BLAST program and the National Center for Biotechnology Information (NCBI) servers. This allows us to run the Blast tool on the NCBI website and gather the report data generated. Bioperl also allows us to parse the report data into a report object, which contains fields that we can use to visualize our data, using other modules such as GD::Graph, which creates image files of pie charts out of a legend from a report file created by Annotator.

In the beginning of the project, we attempted to keep all of the programming in Perl, by using the CGI module as our scripting interface for the website. However we had problems with the high learning curve, lack of documentation compared to PHP, and the multitude of issues we had with the interface between the module and Apache. Our main issue with Apache and Perl was our inability to get Apache to recognize the Perl CGI module.

PHP: Because of all of our difficulties dealing with the CGI module of Perl, we decided to move on to a web scripting language that processed user information on the server, so we went with one of the most popular web scripting languages, PHP. We used PHP for all of our wrappers for the biological tools, which mainly involved displaying valid inputs to the users, processing those inputs, sending them off to the Perl scripts or command line interfaces, then gathering the output, processing it, and finally displaying it to the user.

HTML/CSS: The last two languages we used were HTML and CSS, which allowed us to display the default web page design, and also our PHP output onto the user's web browser. We used both of these languages minimally, especially CSS. Each page shares the same 30 lines of CSS code that defines the display of the webpage. This should be extended on before deployment, since the website is extremely under designed as far as presentation goes. The only things that we put in there were to have two separate tables for each page – the workflow and the inputs, and margins and paragraph styling.

Apache: We didn't deal much directly with the apache interface, we had to work with our sponsor, but we had many difficulties getting things installed and running, since apache is very different from the standard operating systems that we are used to. Since we couldn't access the apache account, we could never determine how to set up Perl's CGI environment using apache and dealing with several other quirks, such as PHP files being determined by apache to be text files. We had our sponsor instead install the modules on the server and used the basic web language.

Coding Environment: To write the code for the website, we had an OSX 9 Server that we worked on that we could store our files and scripts locally and would immediately appear online. Therefore we used the OSX operating system for the entire project. This includes using the terminal interface for PHP scripting. We wrote our code in vim, watching the server log by using a tail function on the apache error output, and various other common terminal functions. Since our project isn't published yet, every change we tested immediately in production on the web server using the standard web browsers Firefox and Safari.

Description of Biological Tools: All of the tools we installed into the webpage are utilizing the same algorithm, just with few minor changes in order to utilize the output in a unique manner. Since this is the case I will explain the tool which uses all parts of each of the other tools, BLAST.

Blast: BLAST is used to find similar biological sequences in a database to a query. The underlying algorithm is Smith-Waterman or longest common subsequences. The difference between BLAST and Smith-Waterman is instead of finding each longest common subsequence using direct character identity between the query and database; we instead use a scoring matrix which defines the score to add when matching any characters. So now instead of creating the identity based longest common subsequence we are matching the best scoring sequences. Also since finding the total highest scoring sequence is an NP-Hard problem, determining the sequence similarity between a 300 character query(average length of 454 sequencing data) and a trillion character database would take far too long to be useful. So instead of aligning the entire sequence, the algorithm takes short 3-9 character long subsequences from the query and each database sequence. These are called seed sequences and when are compared directly to seeds from the database if they score better than a set cutoff they will be turned into the scoring unit of BLAST a HSP. A HSP is a high scoring segment pair which is just a local alignment extended from both ends of these seed sequences. After all the HSPs have been created they are combined into one score to determine the overall similarity of the two sequences if it beats the defined cut off then it is added to the hit list in the BLAST output.

CD-HIT: CD-HIT is just the BLAST algorithm without the HSP creation. Instead of creating any alignment CD-HIT will match as many seed sequences it can and then calculate the sequence similarity in the same way that merging HSPs would be done. This allows CD-HIT to cluster similar sequences together at a far faster rate than BLAST can, thus it is used to create the non-redundant databases from the gigantic amount of sequence data stored at NCBI or UniProt, the publicly available biological sequence databases.

Clustal: Clustal operates similarly to BLAST but differently from CD-HIT, Clustal is more worried about the complete comparison of a set of sequences in order to determine just how different or similar they are. So Clustal focuses on the HSP part of the BLAST algorithm, basically creating all the possible HSPs between a set of sequences then aligning the sequences to get the highest possible score between them. If there is more than two sequences in a Clustal alignment then it uses the NJ algorithm or neighbor joining. This means that the closest related sequences are aligned first then each subsequent sequence in the same way. This creates a graph and alignment that shows you just how related a sequence set is.

Taxonomy: The final tool is the Taxonomic tool which simply utilizes BLAST output to create a tab separated document which contains all sample sequences, their statistics, and their top representative hit from the blast database. Using the top hit Taxonomy tool also queries the NCBI database and determines the family and species where that top hit belongs. Using the family and species is how our taxonomic pie chart output is created.

Description of Tool Wrappers: In this project we simply made wrappers for already established tools in order to make the utilization of them together an easier process. Also we had to design a file system dealing with our selected tools in a way that would be easy to maintain and keep the user from needing to handle file names. So we wrote PHP wrappers for Clustal, CD-HIT, and Annotator, Annotator is a Perl wrapper for BLAST that Josh wrote for the Pipas lab earlier. Annotator allows for the iterative use of BLAST to describe as much of a dataset as possible, as well as create the Taxonomic report. All the tools added to the website were command line tools that we wrote PHP wrappers for to attempt to make their use more simplistic. Our basic PHP wrapper design was to allow for as little user input to the actual command line as possible, just selection boxes of all the options as well as selection of their uploaded fasta file. We did not let them choose the name of the output; instead we made it so that the fasta file name was used in all the output created by the tools, as well as the tool name and parameters. This has two advantages, first we know what exactly the file without parsing and second it controls the redoing of the same operations on the same file.

Description of Unique Tools:

File Uploader: Fairly self explanatory. Allows users to upload their own files and then adds them to the Uploads folder in their project folder. After uploading is completed, it's available to be run on by any valid tool.

Workflow Executor: (Explained later in the Workflow Design section)

Project Login: This allows users to login to any project available on the server or create a new project. Each project stores a cookie on the user's machine for later access and a logout system is also available. Each project upon creation creates its file directory and workflow object. Each project can be deleted at anytime, at which a recursive directory deletion function is executed.

Design of the Website: As mentioned before, this was a real weakness in our project at the at the end of the semester. We never got a chance to really look at the user interface design and the presentation as far making the website look appealing and providing a complete user interface. While we were able to design all of the tools and their wrappers, including allowing the user to perform any action, we weren't able to complete help documentation and do webpage design.

However we were able to complete a large majority of the design, including displaying the tools and options in a useful way. We wanted the website to broken down into separate web pages for each tool and wrapper that we created. Not only that, we needed a persistent workflow model to be stored on the right hand side of each page that could be viewed and interacted with by the user at any time of the project.

We designed the right panel to contain almost everything a user would need regarding their workflow. Every single tool that the user has added to their workflow is displayed, along with the options they have chosen and what order it was added in. Along with that we have 3 buttons that the user can interact with that allow them to logout, delete their workflow or delete their project, the latter two requiring confirmation from the user.

Design of the Workflow Model: Because we wanted users to be able to run multiple sequence files at different times on the same tools with the same options, we decided to design a workflow model that took these things into account. The user is expected to use the workflow by adding tools and options to the workflow model through the individual tool's input pages. This will then appear in a side bar with the order that it is assigned and all of the relevant information such as the name of the tool and the description of the options.

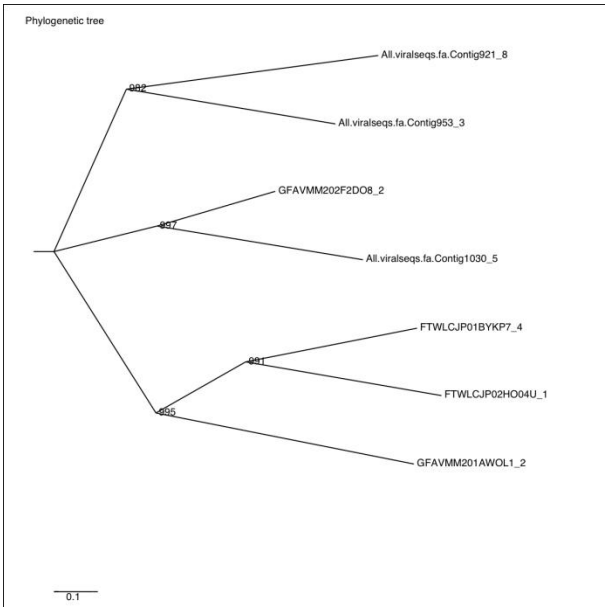
When the user is finished adding all of the necessary tools to their workflow, they can then execute it on the inputs that they have uploaded to the server, or existing files on the server. One of the real strengths of this system is that we created a way to predict outputs of each tool based on their inputs and options, so a user can select to input into a tool the output from a previous step even though the previous step has not been executed. After the user executes their workflow, all of the appropriate output is generated and presented to them upon completion. The user can then come back to their project at any time and run their workflow on a new sequence file.

Design of the File Structure: The way we set up the server's file structure was to have specific directories for each of the main parts of the website. For instance on the top level of the website, we have each of the html files for the tools, a modules folder that contains every tool and their wrappers, and a projects folder that contains every user's projects. Inside the projects folder we have each individual project's folder, which also serves as a way to check for available projects to login to on our login menu. Next in each of those, there contains a workflow data structure, then 2 folders: Outputs and Uploads. These 2 folders are self explanatory, but we created it so that each of them has folders that contain specific files. This way, and by checking file extensions, we can control the user's input into each tool, so that for example, only fasta files show up when trying to run the BLAST program. This user input validation also happens when running a tool on another tool's output while executing the workflow. For example since the taxonomy tool only takes blast report files, it can only be run on BLAST output.

Test Case: Our test case will represent how we expect a user to utilize our website with the intent to discover new viruses. If we start with a set of one million reads from an environmental sample using 454 Sanger sequencing method, then we should run a CD-HIT to remove duplicate sequences occurring in the set usually reducing the set by about a third. Now with our 650,000 remaining sequences we do a BLASTN then TBLASTX against the non-redundant NCBI database as well as a taxonomy run on the resulting blast report. Using the taxonomic charts created you can begin at any point that particularly interests you. For instance, if you are interested in Herpesviridae you would want to check if that family appears in the taxonomic pie chart. We'll say that there is a set of 100 sequences that hit a form of Herpesviridae. You would first want to verify that taxonomic classification by looking up the BLAST report for each of the 100 sequences hit. What you would look for is that the top search result was not an outlier of the dataset. For instance, there may only be one hit for your sequence just making the cutoff and if that's true you verify that the query sequence does not match by chance. After looking through your set of Herpesviridae top hits unearth a set of sequences that have high probability of being similar to some of the core Herpesviridae proteins. With this new set of information you can now move onto the final step. With a set of the Herpesviridae DNA Polymerase sequences from the NCBI database you create an

alignment profile with Clustal then we align our sample sequences to the DNA Polymerase alignment profile. There are two possible outcomes to this procedure. First we may align our sequences and find that they match perfectly to the set showing that yes there probably is Herpesviridae in the sample but nothing new. Second we align our sequences to the profile and find that the sequences are divergent enough from the profile to warrant the possibility of a novel Herpesviridae being represented in the sample, our ideal. Now the objective becomes to prove it experimentally which can be done by utilizing our sequence data to design primers for our target sequence.

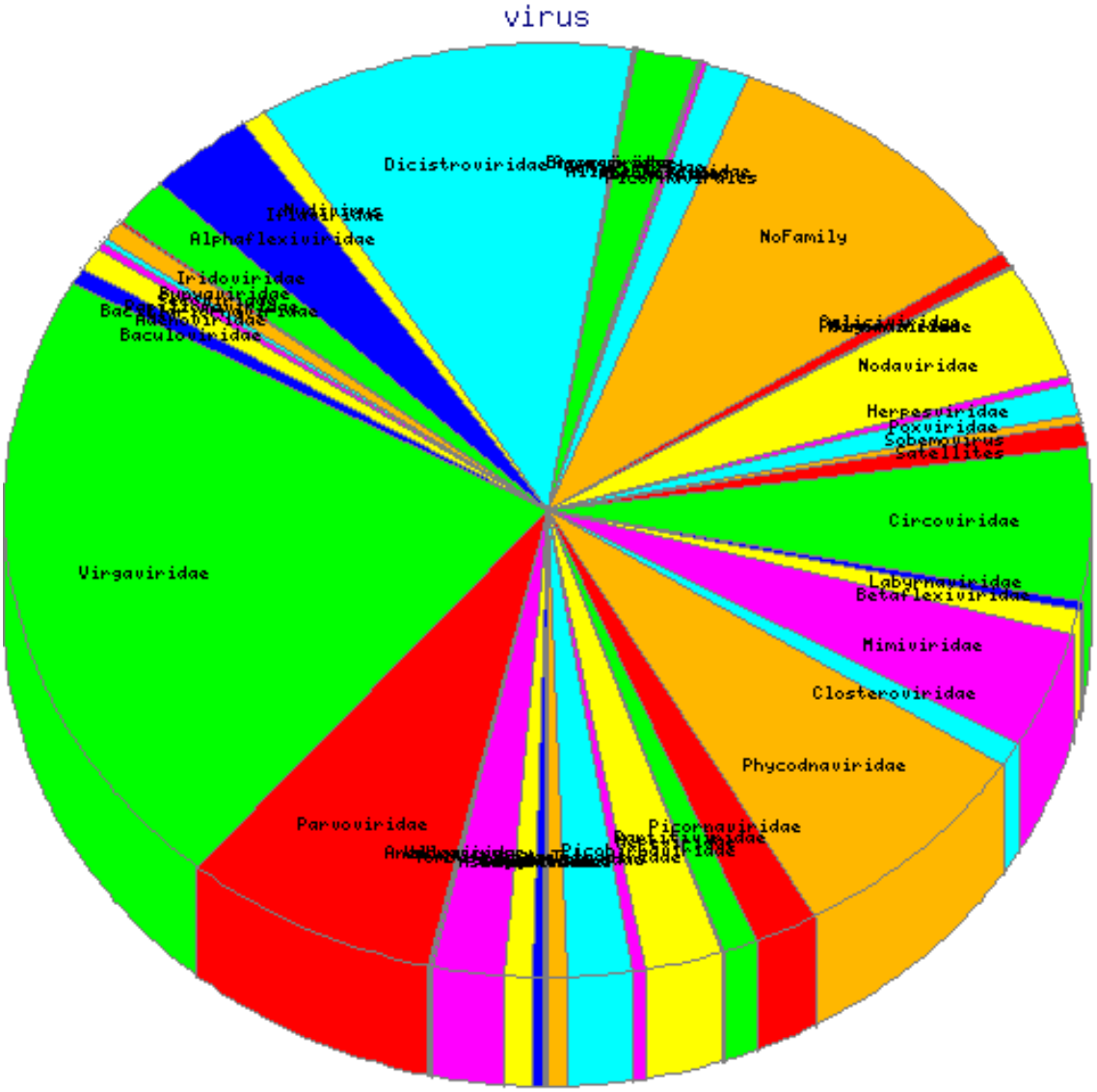
Tool Output: The output of each tool is unique but they all output text files. Currently the text files are either show within the website or are downloadable. Eventually we would like to have been able to represent all output in a picturesque manner but time was a factor and we decided to emphasis the basic running of the website. Clustal outputs both an alignment as well as a phylogenic tree both of which have already defined drawing methods.



This image was created using an online utility which took a Clustal phb file. Eventually we would want our Clustal tool to generate this image instead of just the phb file.

BLAST outputs a fasta file as well as a blast output which a large text file is showing the list of similar sequences from the database as well as some of their high scoring segment pairs. This information while not usually shown in picture form could be represented by showing the hit sequence information and the locations of the HSPs created by the query. The Taxonomy tool takes the top hit from each queried sequence and does a taxonomic query to the NCBI database. Using this information we create a tab separated document with the taxonomic and hit information included with the sequence statistics.

Then utilizing that report file we used a Perl graphics module to create a pie chart of the families and species represented in each of the defined categories, as shown below.



This is the generated image from the Taxonomy package. Unfortunately, the GD::Graphics module does not correctly create the pie chart and overwrites the labels, which is why I had to include the text legend when presenting it on the website. The Pie chart is titled the category it represents and it is organized by Virus family.

Finally we have CD-HIT which creates a cluster file which contains the sequence identifiers of a single group on a line. Also creates a fasta file which contains the longest sequences from each group. The information provided from this program does not necessarily lend itself to visual output but it could be used to enhance the proportions created by the taxonomic tool.

Output Processing (HTML, Pie Charts, etc.): Using the workflow we create an html page which shows each steps output. Also all the files are saved into folder which are viewable by the project. Thus you can load any file created by your workflow separately or you can look at the completed workflow html output. Currently, many of the output files are only text but they do have established visual output creators based on the text output which is something to be implemented.

Challenges: Our hardest challenge was the required team work, neither of us having had worked on a project in the past where our code would be expected to integrate seamlessly. I would say that while the integration of our code was not seamless our team work was effective enough to deal with the massive amount of bugs produced by it. The next challenge was the design of the website, again neither of us had any experience in trying to develop a user friendly website. However, I again think that we may have failed at implementing our design in the most efficient and user friendly method. I do believe that our design is effective and can be reutilized. Third challenge, working within our time and software constraints, with our timing we did succeed in completing a working prototype. However, as to software constraints we did not clearly define our system and requirements from the beginning and I believe that we did lose time while discovered our software constraints during development time.

Future Work: We need to develop the web interface to include JavaScript in order to create a more user friendly and less server intensive application. Some user testing would help in the design of the JavaScript user interface. Also the addition of an assembly tool would complete the one stop toolbox of metagenomics analysis. We also used my developed Perl wrapper for BLAST in the implementation of this website. With two of the tools on this website we wrote a wrapper for a wrapper. Obviously, this is horrific coding practice; a more valid method would have been to write a wrapper for the BLAST utilities then use my original libraries for verification of output.

Conclusion: Currently there is no determined best method for the analysis or metagenomic samples. However, there are widely accepted methods for the analysis of similarity of sequences and these tools are widely used for the classification of metagenomic samples. These tools are inherently very similar and thus their use in conjunction can be helpful but is not necessarily simple for non-computer oriented specialists. Thus we designed and implemented a more visual based system for the easy utilization of our tools for the description of metagenomic data. The description of the metagenomic data allows us to sift through the data in a direct method to search for reads that are significant to a specific species while also being divergent enough from the database set that it could be from a novel genome. If we discover enough of these reads within the same family then it can be attempted to experimentally prove the existence of a new genome within the sample. While our system can help with this kind of

analysis it is currently far too cumbersome for the continuous development needed for metagenomic data analysis, although the tools we selected are standards of today they may not be tomorrow and our system is completely based on their implementation. So while our code can be used today it can become outdated at any time.

Who Worked on What and What Order: Josh did the Perl BLAST and Taxonomy Wrapper, Workflow executer. Kurt did PHP wrapper for Clustal, File Uploader, HTML/CSS, and BLAST Wrapper. Everything else was debugged and programmed by both of us. We both worked around five hours a week over the semester. We both needed web programming experience for development of the webpage. We interacted with our sponsor nearly once a week and we did a lab presentation a week before the capstone. The entire project totaled 5,000 lines of code. Kurt wrote the sections: Goal, Biological Background, Solution, Tools, Programming Language and Technologies, Description of Unique Tools, Design of the Website, Design of the Workflow Model. Josh wrote Who Worked on What and What Order, Conclusion, Future Work, Challenges, Output Processing, Test Case, Description of Biological Tools, and Description of Tool Wrappers.