VIRUSLAND DOCUMENTATION

TABLE OF CONTENTS

Documentation as of 06.01.2015

INSTALLATION

System Requirements

- *nix OS
- Minimum RAM requirement= 16GB; dependent upon Velvet needs (function of k, number of reads, and read size)
- Java runtime environment version 1.6 or higher
- Tested in BioLinux7 and Ubuntu 12.04.1 LTS, 64-bit OS

Required Compilers/Interpreters and Software

Compilers (typically distributed in *nix OS): javaC, g++ and python

Software:
Velvet: https://github.com/dzerbino/velvet/tree/master

- Reference Velvet's README.txt for instructions for compiling.
- VirusLand assumes location of Velvet is within the /bin directory.
- Tested using version 1.0.19 (current as of June 1, 2015)

Emboss: http://emboss.sourceforge.net/download/

- Reference Emboss site listed here for instructions regarding unpacking and compiling.
- VirusLand assumes location of Emboss is within the /bin directory.
- Tested using version 6.6.0 (current as of June 1, 2015)

Lambda: http://www.seqan.de/projects/lambda/

- Reference Lambda site for instructions regarding unpacking and compiling.
- VirusLand assumes location of Lambda is within the /bin directory.
- Tested using version 0.4.7 (current as of June 1, 2015)

Bowtie2: http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

- Reference Bowtie2 site for instructions regarding unpacking and compiling.
- VirusLand assumes location of Bowtie2 is within the /bin directory
- Adding Bowtie 2 directory to your PATH environment variable.
- Tested using version 2.1.0 (current as of June 1, 2015)

PAUDA: http://ab.inf.uni-tuebingen.de/software/pauda/

- Application must be unzipped; VirusLand assumes location is within the /bin directory
- PAUDA is dependent upon the Bowtie2 software. Bowtie2 must be installed as well.
- Tested using version 1.0.1 (current as of June 1, 2015)

Each tool should be added to your PATH environment variable. Note, the Bio-Linux OS includes Velvet, EMBOSS and Bowtie2 requiring only the installation of Lambda and PAUDA.

Installing VirusLand

Unzip the VirusLand installation package maintaining the file structure.

INTRODUCTION

This paper presents the documentation of the VirusLand software developed for the taxonomic and functional classification of raw viral sequences within large metagenomic datasets. VirusLand has been created as a means to perform metageonomic analysis that is expedient and efficient without sacrificing accuracy and precision. It is meant to be simple and straightforward to use for those with little programming or scripting knowledge, while still being open and extend-able to those with more computational knowledge. This document starts with the objectives of Virus Land and an overview of modules. Source code for the modules is available in the /src/uncompiled/ folder within the installation package.

OBJECTIVES

- Developing a tool for automated analysis of viral sequences from raw sequence reads to taxonomic and functional classification;
- Constructing a flexible software that can be customized with the user preferences;
- Reusing existing software as well as implementing new modules to generate a complete viral analysis tool;
- Implementing a software that can be run locally in a reasonable amount of time.

OVERVIEW OF VIRUSLAND

There are four major steps in the pipeline (as indicated by the dashed boxes in Figure 1): Assembly, Gene prediction, Classification and Analysis and visualization.

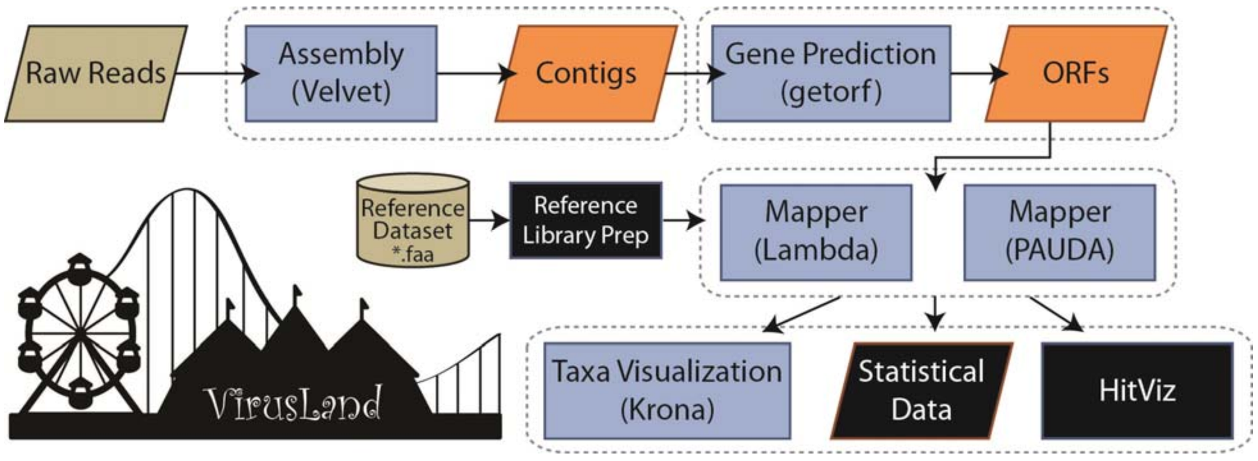

Figure 1. Pipeline for analysis of viral metagenomic sequences using VirusLand.
(Blue boxes indicate incorporation of existing tools; Black shapes indicate results/software developed in this initiative; Tan shapes indicate user supplied data.)

A Python module was developed such that via the use of a single config file, a user can explicitly define parameter values for the analysis. With complete configuration file, this module is able to execute the entire workflow with minimal interruptions. The program runs from beginning to end upon invocation. The program also saves all the intermediate files generated in each step of the process such that the user can access intermediary files (e.g. contigs, ORFs, etc) for independent analysis. Analysis of large datasets can be executed in hours on a 64-bit UNIX operating system with at least 16GB of RAM. (Minimum memory requirements are dependent upon Velvet and a function of *k*, number of reads, and read size.)

TERMINOLOGY

- **Contig:** A contig refers to overlapping sequence data (reads).
- ***k*-mer:** The term *k*-mer typically refers to all the possible substrings, of length *k*, that are contained in a string. The amount of possible *k*-mers within a sequence of a given length, *L*, is $2L-k+1$ (assuming the sequence is double-stranded) whilst the amount of possible *k*-mers given *n* possibilities (4 in the case of DNA) is n^k. Velvet utilizes the parameter *k* in construction of a de Bruijn graph to assemble short reads.

CONFIGURATION MODULE AND CONFIG.VL FILE

The configuration module acts as a parser which loads the input sequences and various parameters as provided by the user in the config.vl file. It is run at the beginning of the modules (preCompiler.py and buildIndexes.py), as well as at the beginning of the VirusLand pipeline. It is invoked by the aforementioned modules and VirusLand script (virusLand.py); the user does not need to run the module.

Essential to the configuration and subsequent execution of VirusLand is the config.vl file. This file is required; a sample text file is included within the /src folder in the installation package. The following illustrates the format of the config file; square brackets indicate the required information and should remain in the config file.

Config file format:

```
[VELVET-FORMAT] <-fa, -fasta, -raw, -fasta.gz -fastq.gz -raw.gz -sam -bam -fmtAuto>
[VELVET-KMER] <kmer size*>
[VELVET-READTYPE] <-shortPaired, -shortPaired2, -short, -short2, -long, -longPaired, -reference>
[FAA-DIR] <directory folder containing reference proteins as .faa files>
[GBK-DIR] <directory folder containing reference viral genomes as .gbk files>
[PAUDA-LOCATION] <directory location of PAUDA software>
[STOP]
```

Each line of the config.vl file should have an entry after the header (bracketed label); if not, the analysis will not be completed. The first three headers are specific to the parameters required for Velvet assembly. *k*-mer size is limited to the MAXKMERLENGTH set while compiling Velvet.

(The default MAXKMERLENGTH is 31; see Velvet documentation for how to change this setting. Implementing tools such as VelvetOptimizer prior to executing VirusLand can assist in identifying if this value should be increased.)

Example Config file:

```
[VELVET-FORMAT] -fasta
[VELVET-KMER] 31
[VELVET-READTYPE] -shortPaired
[FAA-DIR] /home/my/path/virusLand/seqs/
[GBK-DIR] /home/my/path/virusLand/seqs/
[PAUDA-LOCATION] /home/my/path/pauda-1.0.1/pauda/bin
[STOP]
```

DIRECTORY STRUCTURE

/virusland

- /src: This folder contains all of the pipeline modules (python, C++ and Java). (Note, the Velvet, EMBOSS, PAUDA, and Lambda packages should be installed in the /bin directory on the user's computer as noted in the Installation.)
- /uncompiled: This folder contains the C++ and Java source code
- /runs: This folder includes the results of analysis
  - /runName: Each run of VirusLand is written to a folder indicated by the user via a command-line parameter when invoking virusLand.py.
    - /assembled: Velvet assembled contigs
    - /kronaFiles: Statistics file for Krona visualizer (via MS Excel or KronaTools)
    - /lambda: output files from Lambda
    - /ORFs: EMBOSS predicted ORFs
    - /pauda: output files from PAUDA
    - /stats: statistics generated from taxonomic and functional classification; these output files are required for visualization in addition to those generated to assist in user analyses.
- /indexes: Index files generated by VirusLand prior to runs
  - /faa: reference faa index and Lambda index
  - /gbk: reference gbk index
  - /paudaindex: PAUDA index (Note, entire folder must be deleted before recomputing index)
  - /taxindex: taxonomy index
- /seqs: Optional location to put input sequences in; directory location of sequences is still required to be specified as parameters in command-line

Creating Viral Database

A local copy of the reference database must be created.

- *.faa and *.gbk files are required for all sequences/species/genomes included in the database. Such files can be downloaded from NCBI's ftp site: ftp://ftp.ncbi.nlm.nih.gov/genomes/Viruses/ by selecting the current all.faa.tar.gz and all.gbk.tar.gz files.
- Folders must be uncompressed. (Subfolder structure can be maintained although not mandatory.)
- The config file will request the root folder for the faa and gbk files.

Pre-Pipeline Data Preparation

- **preCompiler.py:** The C++ and Java modules provided in the virusLand package must be compiled before the pipeline can be run. A Python script has been provided to perform the compilations, called preCompiler.py. This module goes through the files listed in the /uncompiled folder and creates executables for each in the /src folder.

  Command: `python preCompiler.py`

- **buildIndexes.py:** Some of the programs used in the VirusLand package require reference indexes/datasets to be computed. In order to reduce the time and memory required for each run an indexing script called buildIndexes.py has been created which is run independently of the VirusLand pipeline. This script is required to be run before the analysis can be performed, but the indexes only need to be recomputed if the reference dataset (*.faa and *gbk files) has changed between analyses. In brief, the script creates a concatenated list of GBK files, as well as two files listing the local directories of the FAA and GBK files. These files are then used to create the PAUDA and Lambda indexes from the reference dataset. This reference dataset must be obtained by the user, and placed at the directory referenced by the user in the config.vl file. Lastly another index file, a temporary file, is created to be used for statistical analysis later in the pipeline.

  Command: `python buildIndexes.py config.vl`

VirusLand Modules

Assembly (Velvet): The first step of the VirusLand pipeline is the assembly of the FASTQ files using the Velvet assembler to produce contigs. This module incorporates the existing Velvet assembler tool. Assembly (velvet.py) makes two calls, one to velveth and the second to velvetg. The basic parameters for Velvet are programmable by the user through filling out the config.vl file provided in the /src folder. (These parameters are specific to the velveth process.) The possible user-defined parameters are:

- format: Supported format types include fasta, fastq, fasta.gz, fastq.gz, sam, bam, eland and gerald
- kmer: corresponds to the length (in base pairs) of the words being hashed (refer to the Velvet Manual for further details)
- read type: Supported types include shortPaired, short2 (same as short but for separate insert-size libraries), shortPaired2, long (for Sanger/454/etc), longPaired

The default parameters are used for the call to velvetg (-exp_cov auto). Additional parameters (applicable to both velveth and velvetg) can be specified by the user by adapting the velvet.py script. (For in depth details regarding the Velvet software and parameters, the user is referred to the online manual at https://www.ebi.ac.uk/~zerbino/velvet/.) Velvet output and intermediate files are written to the /assembled folder. All the files except for the file containing the contigs are deleted after assembly in an attempt to save memory (they often represent a large portion of the disk space requirements for Velvet).

Gene Prediction (EMBOSS getorf): After the contigs are generated in the assembly step, they are examined to predict coding regions both in the forward and reverse directions (6 frames). The current implementation of this module employs the getorf function from the EMBOSS package. ORFs are written to the /ORFs folder.

Classification: There are presently two options for classification. Additional modules can easily be plugged in by adapting the python modules and parser code (cFileParse3.cpp).

- **PAUDA:** User provided reference data (in *faa format) is required for taxonomical and functional classification (see **CREATING VIRAL DATABASE**). The PAUDA index is generated by buildIndexes.py. PAUDA relies on the availability of the Bowtie2 software (see **INSTALLATION**). By default VirusLand executes PAUDA using the '--slow' parameter; while slower (albeit significantly faster than BLAST-based searches, still less than 1 hour for 1 million reads) than the alternative '--fast', more reads are assigned. This can be modified by changing the call in the pauda.py script. Output is produced within the /pauda folder. For further details regarding PAUDA, please refer to the software's publication http://www.ncbi.nlm.nih.gov/pubmed/23658416.

- **Lambda:** A second, more expedient means of classification is provided with the addition of the Lambda classification package. Lambda is a local sequence aligner, similar to PAUDA in that it uses read mapping approaches to classify reads in a faster manner that BLAST, while not sacrificing sensitivity. Per Lambda benchmark tests (http://www.ncbi.nlm.nih.gov/pubmed/25161219), Lambda outperforms (produces results more akin to BLAST-based searches yet at a fraction of the time) PAUDA. The index required for running Lambda is generated by buildIndexes.py. Output is produced within the /lambda folder.

Analysis and Visualization: Three analysis tools are included in the VirusLand: Taxa Visualization (Krona), HitViz and Statistical Data.

- **Krona**: Krona (http://sourceforge.net/p/krona/home/krona/) is hierarchical data visualization tool allowing the user to zoom through pie charts. Here Krona is used to visualize relative abundances of metagenomic classifications. VirusLand creates the Krona-formatted file which can be visualized using Microsoft Excel Template for Windows and Mac or via KronaTools for Mac and Linux via the ktImportText script. (Please refer to the Krona site for instructions regarding obtaining as well as generating the interactive chart. Note, Krona is not included in the installation of VirusLand.) The input file for Krona (either template or KronaTools) is contained within the /stats folder of the run and includes the file name indicator of "KronaStats".
- **HitViz:** This tools was developed as part of VirusLand, allowing the user to visualize the coverage of hits to individual genomes. This tool allows the user to create graphs of the presence/absence of a particular species or near-neighbor species. This Java module is automatically executed as part of the complete pipeline; it can, however, also be run post-processing (/src/javaVisuals/virusGraphics.jar). The input for HitViz is a text file containing the species, proteins in the genome and number of hits per annotated coding region. This file, created by VirusLand is contained within the /stats folder of the run and includes the file name indicator of "HV". All species for which one or more hits were found is able to be visualized. Figure 2 shows an example of one such visualization. Charts may be copied to the clipboard or saved in *png format by right clicking on the graph. Furthermore, the graph is interactive; users can click on a bar to identify the number of hits and gene's NCBI accession number. Figure 2 shows an example HitViz visualization. A sample file is included within the /src/javaVisuals folder.

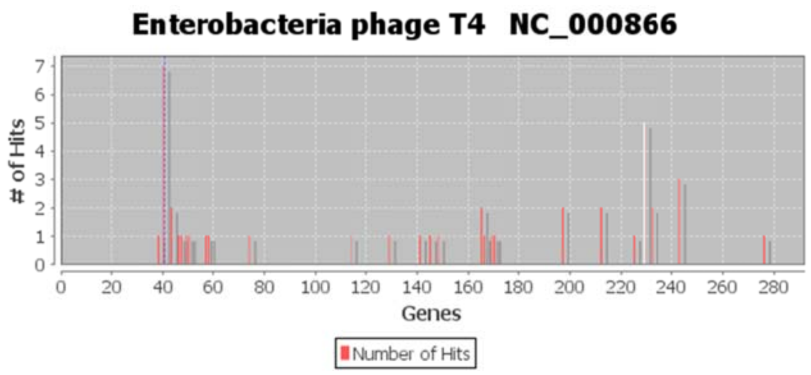

Figure 2. Example of visualization generated by HitViz for hits to a single genome.

- **Statistical Data:** Two tab delimited text files are also generated by VirusLand and contained within the /stats folder of the run. The genome coverage statistics file (file name indicator of "coverage") includes the number of proteins per genome, number of hits total to the genome, and % of the proteins having hits. The hits by protein file (file indicator of "hits_by_protein") includes the number of hits for each gene for each genome included in the reference database; genomes having no hits are not listed.)

RUNNING VIRUSLAND.PY AS A BATCH PROCESS:

It is possible to run multiple analyses back-to-back without interruption via a batch file. This is advantageous for users attempting to analyze multiple datasets at once. The process is simple and involves creating a text file in which each line contains a single command-line call of virusLand.py with the necessary parameters.

Example batch file:

```
python virusLand.py run1 seq1 seq2 configFileLocation
python virusLand.py run2 seq3 seq4 configFileLocation
python virusLand.py run3 seq5 seq6 configFileLocation
```

The run names must be unique for each run; any runs with the same name will be overwritten as VirusLand uses the runName parameter to generate the run folder. The user can create multiple config.vl files with different names and parameters which can then be used for separate runs.

Once this text file has been created and saved, you will have to modify the permissions so it is executable. In the terminal, use the command *chmod 755 <batchFileLocation>* to make the file executable. The file can then be run in the terminal by simply typing the location of the file (or ./*fileName* if you are in the directory containing the batch file).