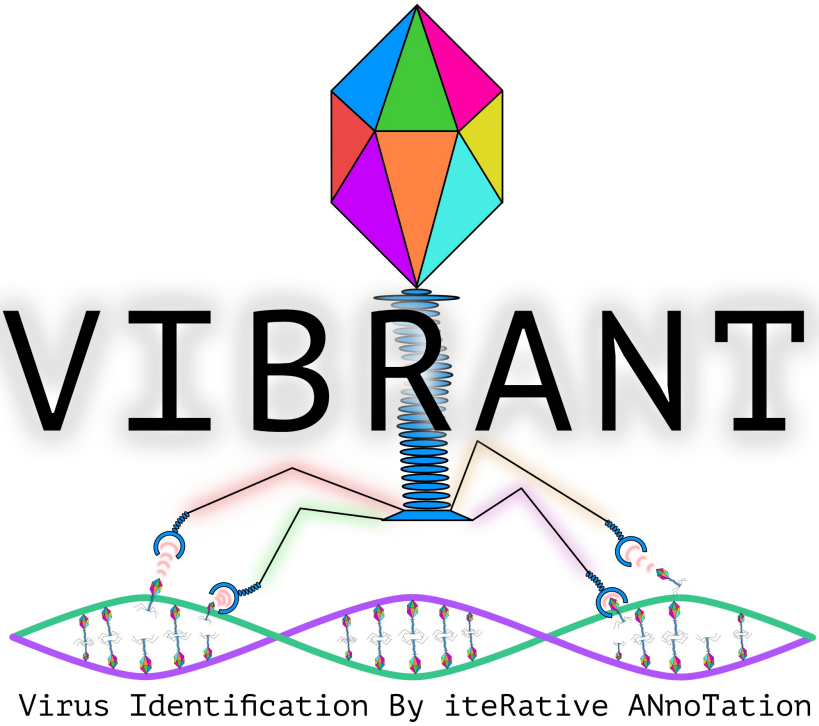


VIBRANT

Virus Identification By iteRative ANnoTation

10/22/2019
Kristopher Kieft
Anantharaman Lab
University of Wisconsin-Madison
kieft@wisc.edu

Version

VIBRANT v1.0.1

Citation

If you find VIBRANT useful please consider citing our preprint on bioRxiv:
Kieft, K., Zhou, Z., and Anantharaman, K. (2019). VIBRANT: Automated recovery, annotation and curation of microbial viruses, and evaluation of virome function from genomic sequences.

BioRxiv 855387.

Updates for v1.0.1:

Summary:

- Issue: some extracted proviruses were given the wrong genome sequence. Issue is now fixed.
- Update: minimum sequence length is now 1000bp which greatly increases virus identifications from some metagenomes.
- Update: metrics for quality analysis of scaffolds has been changed.
- Note: there is no need to re-download or re-compile the databases.

Explanations:

- Fixed a bug that was causing issues with extracting the correct genomic region of an integrated lysogenic virus (provirus). Briefly, the issue was causing some extracted proviruses (those with "*_fragment_#*" in the name) to have either the wrong genomic sequence or a sequence length of zero. The respective proteins and virus identification metrics for these scaffolds remains identical.
- The minimum scaffold length is now set to 1000bp (previous 3000bp). This greatly increases total virus identification within metagenomes that contain many sequences less than 3kb. This has no effect on false discovery since the minimum open reading frame requirement is still 4. For example, a 1kb and 3kb scaffold each encoding 4 open reading frames will be considered identically since neither sequence length nor sequence features impact virus identification.
- The quality analysis of scaffolds was updated. The category *fragment quality draft* was removed. The categories are now *complete circular*, *high*, *medium* and *low quality draft*. The update was made to better represent the completeness of *Caudovirales* scaffolds which are likely to be the most abundant viral population in a metagenome. Baseline tests indicate the new metrics linearly represent the completeness of *Caudovirales* genomes.

Program Description

VIBRANT is a tool for automated recovery and annotation of bacterial and archaeal viruses, determination of genome completeness, and characterization of virome function from metagenomic assemblies. VIBRANT uses neural networks of protein annotation signatures and genomic features to maximize identification of highly diverse partial or complete viral

genomes as well as excise integrated proviruses.

- Uses neural network machine learning of protein annotation signatures
- Asigns novel 'v-score' for determining the virus-like nature of all annotations
- Determines genome completeness
- Characterizes virome function by metabolic analysis
- Identifies auxiliary metabolic genes (AMGs)
- Excises integrated viral genomes from host scaffolds
- Performs well in diverse environments
- Recovers novel and abundant viral genomes
- Built for dsDNA, ssDNA and RNA viruses

VIBRANT uses three databases for identifying viruses and characterizing virome metabolic potential:

- KEGG (March release): https://www.genome.jp/kegg/ (FTP: ftp://ftp.genome.jp/pub/db/kofam/archives/2019-03-20/)
- Pfam (v32): https://pfam.xfam.org (FTP: ftp://ftp.ebi.ac.uk/pub/databases/Pfam/releases/Pfam32.0/)
- VOG (release 94): http://vogdb.org/ (FTP: http://fileshare.csb.univie.ac.at/vog/vog94/)

Requirements

System Requirements: VIBRANT has been tested and successfully run on Mac, Linux and Ubuntu systems.
Program Dependancies: Python3, Prodigal, HMMER3, gzip, tar, wget (see section below)
Python Dependancies: BioPython, Pandas, Matplotlib, Seaborn, Numpy, Scikit-learn, Pickle (see section below)

Don't worry, these should all be easy and quick to install if you do not already have the requirements satisfied. Suggested methods of installation are with [1] pip (pip3) (https://pypi.org/project/pip/), [2] conda (https://anaconda.org/anaconda/conda), [3] homebrew (https://brew.sh/) or [4] apt (you make need to use 'apt-get' or 'sudo') (https://help.ubuntu.com/lts/serverguide/apt.html). The method will depend on your operating system and setup.

Program Dependancies: Installation

Please ensure the following programs are installed and in your machine's PATH. Note: most downloads will automatically place these programs in your PATH.

Programs:

1. Python3: https://www.python.org (version >= 3.5)
2. Prodigal: https://github.com/hyattpd/Prodigal
3. HMMER3: https://github.com/EddyRivasLab/hmmer
4. gzip: http://www.gzip.org/
5. tar: https://www.gnu.org/software/tar/
6. wget: https://www.gnu.org/software/wget/

Example Installations:

1. Python3: see Python webpage. You can check your current version using `python --version`. Required version >= 3.5.
2. Prodigal: `conda install -c bioconda prodigal` or `brew install prodigal` or `apt-get install prodigal` or GitHub clone
3. HMMER3: `conda install -c bioconda hmmer` or `brew install hmmer` or `apt install hmmer` or GitHub clone
4. gzip: you likely already have this installed
5. tar: you likely already have this installed
6. wget: you likely already have this installed or `brew install wget` or `apt install wget` or `pip install wget`

Python3 Dependancies: Installation

There are several Python3 dependancies that must be installed as well. You may already have some of these installed.

Packages

1. BioPython: https://biopython.org/wiki/Download
2. Pandas: https://pandas.pydata.org/pandas-docs/stable/install.html
3. Matplotlib: https://matplotlib.org/
4. Seaborn: https://seaborn.pydata.org/
5. Numpy: https://numpy.org/
6. Scikit-learn: https://scikit-learn.org/stable/
7. Pickle: https://docs.python.org/3/library/pickle.html

Example Installations:

1. BioPython: `pip install biopython` or `apt-get install python-biopython` or `conda install -c conda-forge biopython`
2. Pandas: `pip install pandas` or `conda install -c anaconda pandas` or `apt-get install python-pip`
3. Matplotlib: `pip install matplotlib` or `conda install matplotlib` or `apt-get install python-matplotlib`

4. Seaborn: `pip install seaborn` or `conda install seaborn`
5. Numpy: `pip install numpy` or `conda install -c anaconda numpy` or `apt-get install python-numpy`
6. Scikit-learn: `pip install -U scikit-learn` or `conda install scikit-learn`
7. Pickle: this should already come with Python3. If you have issues try `pip install pickle-mixin`

Running VIBRANT

VIBRANT is built for efficiently running on metagenomes but can also run on individual or small groups of genomes. Each scaffold is considered individually, so results will *not* vary whether the scaffold is run as part of a metagenome or by itself.

Quick Start

There are two different routes to downloading VIBRANT: Anaconda install or GitHub clone/download.

Anaconda

1) Install dependancies. See *Requirements* section above.
2) Install directly to PATH using conda.
`conda install -c bioconda vibrant`
3) Download and setup databases. This will take some time due to file sizes, but it only needs to be run once. This step requires 20GB of temporary storage space and ~11.2GB of final storage space. To do this, identify the location of VIBRANT scripts using `which VIBRANT_run.py`. Copy the path to bin (e.g., `/Users/kristopher/anaconda3/bin/`) and replace with `your_path` in the example below (e.g., `Users/kristopher/anaconda3/bin/download-db.sh`).
`your_path/download-db.sh`

GitHub

Note: if at any time you are given a "permission denied" error you can run `chmod 777 <file_name>` or `chmod -R 777 <folder_name>`. Simply repace `<file_name>` or `<folder_name>` with the file/folder that you would like to add permissions to.

1) Install dependancies. See *Requirements* section above.
2) Download VIBRANT using git clone or download zip file. *Note*: if you download the zip file

you will have the parent folder `VIBRANT-master` instead of `VIBRANT`.

`git clone https://github.com/AnantharamanLab/VIBRANT`

3) You may want to add permissions to all files.

`chmod -R 777 VIBRANT`

4) Move parent folder (`VIBRANT`) to desired location. VIBRANT will function no matter where the parent folder is located or moved to, but hierarchy of files and folders within the parent folder must remain constant.

5) Move into databases folder for setup.

`cd VIBRANT/databases`

6) Download and setup databases. This will take some time due to file sizes, but it only needs to be run once. This step requires 20GB of temporary storage space and ~11.2GB of final storage space. Run the following:

`python3 VIBRANT_setup.py` or `./VIBRANT_setup.py`

7) VIBRANT can automatically verify that downloads, setup and installation of dependancies was completed properly. You may choose to skip this, but it's very quick. This script can be called as many times as necessary to run verifications.

`python3 VIBRANT_test_setup.py` or `./VIBRANT_test_setup.py`

Testing VIBRANT

Note: VIBRANT does not write to standard out (command prompt screen) while running or when it finishes (i.e., not verbose). However, VIBRANT will write to standard out in the event that it encounters an error, such as incorrect use of optional arguments or incorrect input file format. See step 11 below for an example.

Optional: run VIBRANT on test datasets (steps 8-11). *Note*: you may want to try using the `-t` flag to increase VIBRANT's speed. See *Arguments and Flags* section for details.

8) Test out a small dataset of mixed viral and non-viral scaffolds in nucleotide format. See `example_output/VIBRANT_mixed_example` for how the results should look.

`python3 ../VIBRANT_run.py -i mixed_example.fasta` from within the `example_data/` folder

or

`python3 VIBRANT_run.py -i example_data/mixed_example.fasta` from within the parent folder

9) Test out a single viral scaffold in protein format.

`python3 ../VIBRANT_run.py -i Microviridae_MH552510.2.faa -f prot` from within the `example_data/` folder

10) Test out a set of non-viral scaffolds in nucleotide format.

`python3 ../VIBRANT_run.py -i no_phages.fna` from within the `example_data/` folder

11) Test out a scaffold in nucleotide format that does not meet the minimum requirement of

1kb in length. VIBRANT will exit because the input sequence is not 1kb. *Note:* Exiting only occurs if *all* the scaffolds do not meet the minimum requirements.
`python3 ../VIBRANT_run.py -i short_scaffold.fsa` from within the `example_data/` folder

Arguments and Flags

The only required flag is the input (`-i`) FASTA file which can either be in nucleotide or protein format. The file extension (e.g., fasta, fna, faa, fsa) does not matter. For nucleotide input, the definition lines can be in any format. The only exception is that the phrase "fragment" should not appear in the definition line because this term is used by VIBRANT during analysis and output parsing. For protein input, proteins must be grouped by genome and in decending order by protein number. Also, the definition lines for proteins must be in Prodigal format. Please see `example_data/Microviridae_MH552510.2.faa` for an example of Prodigal format definition lines. The required items are the scaffold name, protein number, start site, end site and strand. It is suggested to use nucleotide input unless inputting proteins will fit best with your current/downstream analyses.

VIBRANT comes with a couple of very simple optional arguments. At any point you can see the help menu with `VIBRANT_run.py -h`.

Common optional arguments

- `-t`: increase the number of VIBRANT parallel runs (similar to threads). The integer entered here will have no impact on results but may impact runtime. To parallelize VIBRANT, the `-t` flag will indicate how many separate files to split the input file into; these will all be run in tandem. For example, an input file with 10 scaffolds that has invoked `-t 5` will run VIBRANT 5 separate times simultaneously with each run consisting of 2 scaffolds each. Default = `1`.

- `-f`: identify either nucleotide or protein input. This flag is only required when inputting proteins (`-f prot`) but can be added with any nucleotide input (`-f nucl`). Default = `nucl`.

Uncommon optional arguments

- `-l`: increase the minimum scaffold length requirement. The minimum is 3000 basepairs (`-l 3000`). Default = `3000`. For example, if `-l 5000` is invoked VIBRANT will only consider scaffolds greater than or equal to 5000bp.
- `-o`: increase the minimum number of open readings frames (ORFs, or proteins) per scaffold requirement. The minimum is 4 ORFs (`-o 4`). Default = `4`. For example, if `-o 8` is invoked VIBRANT will only consider scaffolds encoding at least 8 proteins.
- `-virome`: This flag should be used cautiously. This will edit VIBRANT's sensitivity if the

input dataset is a virome and not mixed metagenome. That is, if you expect the vast majority of your input scaffolds to be viruses then `-virome` can be used to remove obvious non-viral scaffolds. This will have no effect on runtime. Default = off.
- `-no_plot` : This flag can be used to skip generation of the graphical representations of the VIBRANT's results. You may wish to use this flag if you simply have no use for the output figures or if for some reason this function of VIBRANT is causing the program to break. This will have little effect on runtime. Default = off.

Unused optional arguments
There are several additional flags that will only be used if the hierarchy of the parent folder is modified. This will likely never apply.

Output Explanations

VIBRANT outputs a lot of files and folders. Please see `output_explanations.pdf` within the parent folder for information regarding each file/folder that is generated by VIBRANT. Each file and folder will have a prefix or suffix respective to the name of the input file.
Note: some scaffolds will be proviruses that have been extracted from a host scaffold. These viral sequences will be given a new name. Specifically, the term "*_fragment_#*" will be appended to the name to indicate that it is a fragment of a larger scaffold. The number associated with the fragment is, for the most part, arbitrary.

Useful Outputs

- FASTA file of identified virus genomes:
`VIBRANT_phages_<input_file>/<input_file>.phages_combined.fna`
- List of identified virus genomes:
`VIBRANT_phages_<input_file>/<input_file>.phages_combined.txt`
- GenBank file of identified virus genomes (if `-f nucl`):
`VIBRANT_phages_<input_file>/<input_file>.phages_combined.gbk`

Note: integrated viruses that have been excised from a scaffold will have 'fragment_#' appended to the genome and protein names.

General Overview

Briefly, the folder `VIBRANT_phages_<input_file>` will contain FASTA, GenBank and list files for the identified viruses; the folder `VIBRANT_results_<input_file>` will contain annotation, metabolic and summary information for the identified viruses; the folders `VIBRANT_HMM_tables_parsed_<input_file>` and `VIBRANT_HMM_tables_unformatted_<input_file>` will contain raw HMM tables used for

analyses; the folder `VIBRANT_figures_<input_file>` will contain summary figures for the dataset and identified viruses; the file `VIBRANT_log_<input_file>` will contain the log summary and run information. All outputs will be contained within a folder named `VIBRANT_<input_file>`.

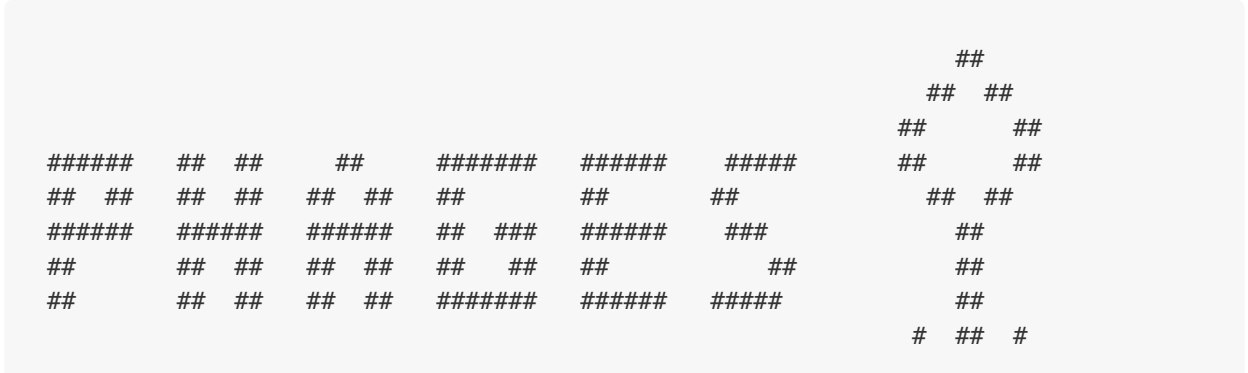
VIBRANT Files and Folders

VIBRANT comes with several folders, files and scripts that are used during analysis. You will not need to interact with any of these, but knowing what they contain may be useful. The folder `databases` contains two scripts (`VIBRANT_test_setup.py` and `VIBRANT_setup.py`) that are only used during initial setup of VIBRANT. There is also a folder `profile_names` that contains text files with lists of all HMM profiles used per database. The folder `example_data` has a standard setup for testing VIBRANT and validating the correct outputs. The folder `files` contains several very useful documents: `VIBRANT_AMGs.tsv` - list of all KEGG KOs designated as auxiliary metabolic genes (AMGs); `VIBRANT_categories.tsv` - list of accession numbers for VOG, KEGG and Pfam with their respective "v-score" which is an essential metric used to identify viruses; `VIBRANT_KEGG_pathways_summary.tsv` - list of KEGG map pathways, and their associated metabolic and KO information which is used to summarize virome metabolism; `VIBRANT_machine_model.sav` - the neural network model used for virus classification; `VIBRANT_names.tsv` - list of names associated with each VOG, KEGG and Pfam accession number. The folder `scripts` contains three auxiliary scripts used to run VIBRANT. The script `VIBRANT_run.py` within the parent folder is actually a wrapper script to faciliate parallelization of VIBRANT and perform final metric analyses, whereas the script `VIBRANT_annotation.py` is what does the real virus identification. The two scripts `VIBRANT_extract_nucleotide.py` and `VIBRANT_extract_protein.py` are used when splitting the input file for parallelization.

Contact

Please contact Kristopher Kieft (kieft@wisc.edu) with any questions, concerns or comments.

Thank you for using VIBRANT!



```
                                                  # # ## # #
                                                #           #
```

Copyright