Plugin Based Microbiome Analysis (PLUMA , formerly MIAMI) Version 1.0 - User Guide

Trevor Cickovski and Giri Narasimhan
School of Computing and Information Sciences
Florida International University
tcickovs@fiu.edu, giri@fiu.edu

`http://biorg.cis.fiu.edu/PluMA/`

Other Contributors from Florida International University:

Vanessa Aguiar-Pulido
Michael Campos
Mitch Fernandez
Wenrui Huang
Shamsed Mahmoud
Jingan Qu
Juan Daniel Riveros
Victoria Suarez-Ulloa
Camilo Valdez

June 14, 2017

Abstract

We present PLUMA, a lightweight and flexible package for constructing general software pipelines. PLUMA is designed to be infinitely extensible, allowing researchers to select a specific set of dynamically loaded plugins as sequential stages of their pipelines. These plugins can be implemented by either themselves or other users, in their language of choice.

We begin by introducing the key features of PLUMA, and follow with a discussion of how to download and install the latest version, compile, and run the software. We also include information on setting up configuration files that specify desired plugins for a pipeline, and how to extend PLUMA with new plugins in various programming languages. Finally, we conclude with a full pipeline example and a brief discussion of our envisioned future of PLUMA.

Acknowledgements

We acknowledge support from the following on the PLUMA project:

- The College of Engineering and Computer Science, Florida International University
- Florida Department of Health (FDOH 09KW-10)
- Alpha-One Fundation
- The Natural Sciences Collegium, Eckerd College (Faculty Development Grant)
- NVIDIA (CUDA Teaching Center Program)

Contents

List of Figures

Chapter 1

Introduction

Software pipelines are applicable to any field of study and involve a framework where control flow executes as a series of *stages*, with an output of a given stage serving as an input to the next. As an example, Figure 1.1 shows a common pipeline in metagenomics analysis, with an initial set of DNA sequences passing through a denoising stage that improves the quality of reads, followed by clustering through some similarity or compositional metric, and finally labelling sequence clusters with the closest matching taxonomic unit [2]. Each stage takes as input the output of the previous stage, and produces new output.

Many pipelines have been developed by independent teams as standalone tools, although stages of individual pipelines could potentially be reused amongst one another. Particularly in the field of metagenomics analysis, there are many software pipelines that perform similar tasks and may even share stages that are still constructed independently, because there is no standardized framework for developing, testing and particularly *integrating* these stages. PLUMA is designed to address this need by providing a lightweight and generic back end that can execute these stages as dynamically loaded *plugins*, specified by a user through a configuration file or GUI.

We show the conceptual design of PLUMA in Figure 1.2. The design of PLUMA follows that of a Problem-Solving Environment (PSE, [21]) using three tiers: a compiled machine layer, a middle scripted layer, and an upper user layer. PSEs are specifically designed with extensibility in mind, but often involve only two layers. The first is what we refer to as the machine layer (also often called a *back end*), and the other is what we refer to as the user layer (also often called a *front end*). The user layer of PLUMA consists of an easy-to-use configuration file, and we plan on adding a graphical interface later. The machine layer consists of compiled plugin extensions to the software that become integrated with the back end through application of various software design patterns [22].

PSEs are hierarchical, so that any upper layer can invoke any of the layers beneath. What distinguishes PLUMA from two-tier packages is the middle (scripting) layer between the user and computational layers. This improves extensibility by allowing new plugins to be built using *components of existing plugins*, which

Figure 1.1: An example metagenomics analysis pipeline. Each stage gets executed sequentially, with the output of a specific stage serving as input to a later stage of the pipeline.

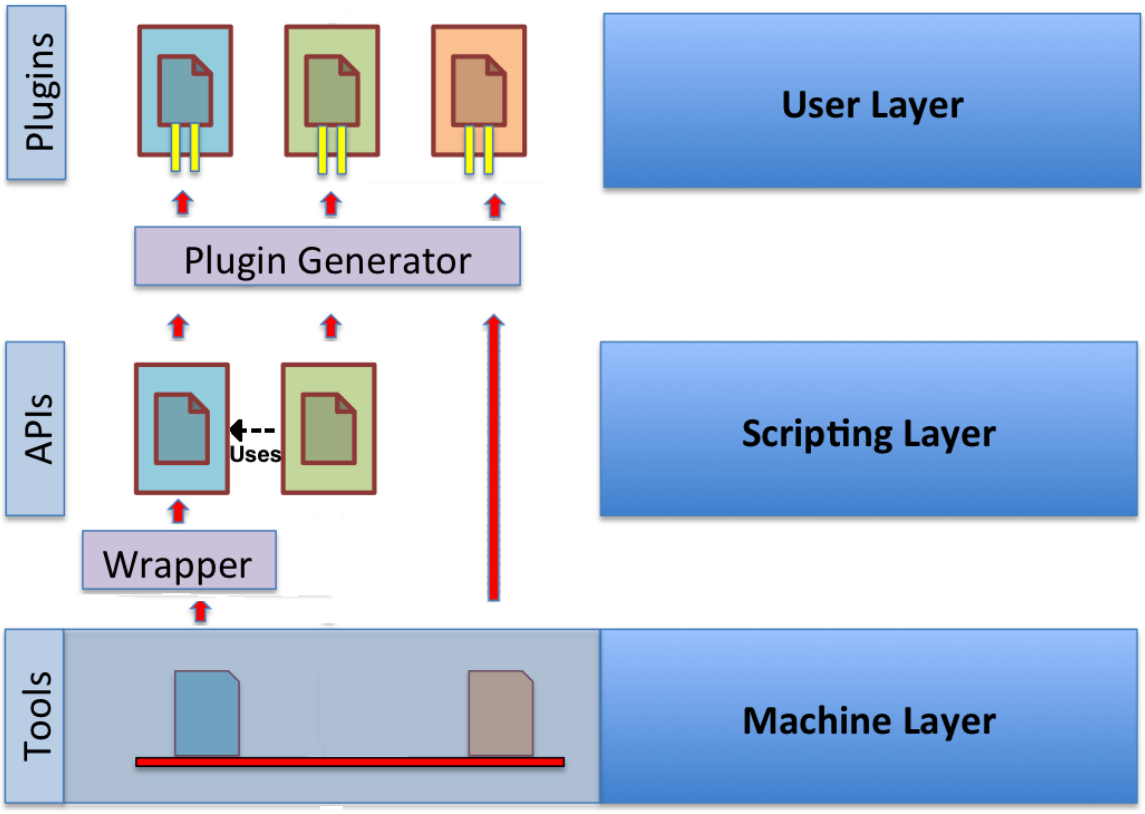

Figure 1.2: Conceptual design of PLUMA, with the goal of offering infinite extensibility at the level and language of choice. We offer three layers to interact with the framework: Machine, Scripted and User. From [11].

facilitates development of new plugins while minimizing having to reinvent the wheel. It has been shown through case studies [14] that this three-tier design offers improved extensibility for both biologists and computer scientists.

Moreover, this design allows users to develop new plugins in the language of their choice. Currently, PLUMA supports C++, Compute Unified Device Architecture (CUDA, [43]) for Graphics Processing Units (GPUs), Python, Perl and R for plugin development. A pipeline can be set up in the user layer using plugins that are heterogenous in their underlying implementation language, allowing users to have installed only the plugins and tools required for their particular task. We distribute PLUMA open source under the GNU General Public License (GPL), version 3.

1.1 Useful Features

1.1.1 Compatibility with a Wide Range of Languages

The computational back end of PLUMA has been written in C++ and is a simple, lightweight package that loads and runs plugins. However, plugin stages can be built in one of many possible languages. Currently, PLUMA supports plugins for both the CPU and GPU. On the CPU side they can be compiled using C/C++

or scripted using Python, Perl or R. For the GPU side, PLUMA supports CUDA. Because of the uniform back end that executes these plugins, a user need not worry about the particular implementation details of the plugin including its language, but can simply download the plugin and include it in their pipeline. Since plugins are dynamically loaded a user can just download and install the plugins they need. For example, a user may not have an NVIDIA graphics card, but they can still run any pipeline that uses CPU plugins. If they do not have R installed, they can still use C++ and/or Python plugins. The goal of PLUMA is thus to maximize both flexibility and extensibility in terms of both language and feature integration.

1.1.2 Minimal External Dependencies

As a result of this flexibility and the tool being lightweight, PLUMA requires only one external dependency for installation. This tool is SCons [30], which is available open-source and is a cross-platform compilation tool implemented in Python. The tool is easy to download and run through a single command `scons`.

1.1.3 Plugin Generator

A user may wish to integrate a standalone tool (i.e., Mothur [54]) as an inner stage of a larger pipeline. PLUMA includes a `PluginGenerator` module that can be used for this purpose. The `PluginGenerator` will produce a C++ wrapper plugin for the desired package, which can in turn be compiled and dynamically loaded as a stage in any general pipeline.

1.1.4 Log Files

Log files provide a useful method for tracking pipeline progress, particularly when timestamped with a date and a time. PLUMA outputs log files automatically for each pipeline execution, sending them to a file conveniently assigned a name that includes the date and time. Particularly for a pipeline that contains many stages, this can be a convienent way to determine a point of failure, or an intermediate interesting result [52]. Either can work with PLUMA's *Restart* feature (described next), to start a pipeline from a middle point.

1.1.5 Restarting

Users can restart a pipeline from a specific stage, as opposed to the first stage, without having to change the configuration file. This is convenient if for example a log file (described above) displayed an error message at a particular stage of the pipeline, as the pipeline can be started from the erroneous stage as opposed to having to rerun from the beginning. Depending on the computational intensity of each stage, this could save a great deal of execution time.

Chapter 2

Availability and Installation of PLUMA

2.1 How to Download PLUMA

The download site for PLUMA is hosted by the Bioinformatics Research Group (BioRG) at Florida International University: `http://biorg.cis.fiu.edu/pluma`. Source code is available on this site as a tarball. An agreement to a Non-Exclusive, Non-Commercial Use License is required (this exact license can be found in the appendix of this user guide). We also make available a plugin pool of dynamically loadable plugin libraries implemented in various languages, as well as example data sets and configuration files.

2.2 Compiling PLUMA

PLUMA uses the SCons (`http://www.scons.org`) open source software construction tool to compile its back end. Please download and install SCons before compiling PLUMA. Once SCons is installed, the steps to compile PLUMA are:

1. Change to the main `pluma` directory.
2. Run the command "`scons /`". If you run only "`scons /`" and pass no flags, PLUMA will assume you have tools installed for all supported languages. At a minimum, PLUMA will assume that you have C++ and Python installed (these are necessary anyway to compile the back end and run `scons`, respectively). For the other three supported languages (CUDA, R, and Perl) you can turn off their compilation by passing flags `cuda=0`, `r=0` and `perl=0`, respectively. This information is also outlined in the `README` file inside the PLUMA root directory.

2.2.1 Environment Variables

If your installations for R and/or Perl are not in default locations, you may need to specify these as environment variables. Below we provide a set of environment variables that PLUMA uses when determining the location of these libraries.

Environment Variable	Description	Default
`RHOME`	R root directory	`/usr/share/R`
`RSITELIBHOME`	Root directory of `site-library` directory for R	`/usr/local/lib/R/site-library`
`PERLHOME`	Perl root directory	`/usr/`
`PTHREADHOME`	PThread root directory	`/usr/local/`

Running `scons` compiles the PLUMA back end and all prepackaged plugins in the `plugins/` directory. Repeated runs of `scons` will only recompile those portions of the code that changed. By running `scons -c`, all object files and dynamically loaded libraries will clear recursively throughout the directory tree.

After everything has compiled successfully, the user may run PLUMA from the root directory. This is described in the next chapter.

2.3 Documentation

All available documentation for PLUMA can be downloaded from:

`http://biorg.cs.fiu.edu/pluma/`

Chapter 3

Getting Started

In this chapter, we introduce the commands needed to run PLUMA. These include the command line formats on UNIX-based machines, followed by a description of the configuration file formats. In the next section we show example configuration files that are included in the PLUMA installation, to help illustrate concepts.

3.1 Command Line

The application file for the PLUMA package is conveniently named `pluma`. To run the application, type `pluma` at a UNIX prompt followed by a configuration file and, optionally, a restart point. Thus the PLUMA execution command takes the following format:

```
./pluma (config file) (optional restart point)
```

The second argument can be an absolute or relative path to the configuration file. The restart point must be a valid plugin that has been specified in the configuration file provided in the second argument. If you provide a restart point, the pipeline will start from the stage specified by that plugin.

Below we show three example PLUMA executions, along with their meanings, assuming PLUMA has been installed in the location `/home/johndoe/` and the file `/home/johndoe/examples/myconfig.txt` is our configuration file with three plugins *Stage1*, *Stage2* and *Stage3*.

`./pluma /home/johndoe/examples/myconfig.txt`	Run using the configuration file */home/johndoe/examples/myconfig.txt*
`./pluma examples/myconfig.txt`	Same
`./pluma examples/myconfig.txt Stage1`	Same
`./pluma examples/myconfig.txt Stage2`	Same, but start with second stage
`./pluma examples/myconfig.txt Stage3`	Same, but only run last stage

3.1.1 Plugin Location

The directory containing the plugins for PLUMA is assumed to be the directory of the `pluma` executable, followed by `plugins/`. So for example the above case would assume all plugins are in the folder `/home/johndoe/plugins`, which would be the case for all plugins that come with the PLUMA package. You can also set this location through an environment variable called `PLUMA_PLUGIN_PATH`. This can be used to load plugins from multiple directories, separated by the colon `:` character:

```
export PLUMA_PLUGIN_PATH=/usr/lib/plugins:/home/johndoe/johnsplugins
```

Setting this variable would load plugins from the PLUMA root directory, followed by `/usr/lib/plugins`, followed by `/home/johndoe/johnsplugins`. If two plugins in two different folders have the same name, PLUMA will use the second plugin.

3.1.2 Help Option

A user may also type one of the following two possibilities at a UNIX prompt for help with the PLUMA command line:

1. `./pluma` (with no arguments)
2. `./pluma help`

3.1.3 Version Option

A user may provide the argument `version` to get the release number of PLUMA that they are running:

1. `./pluma version`

3.1.4 Plugins Option

To get a list of all currently installed plugins, a user can provide the `plugins` option to `pluma`:

1. `./pluma plugins`

3.2 Configuration File

The configuration file is the only required command-line input to the PLUMA executable. This section discusses the structure of the PLUMA configuration file. In addition for guidance, PLUMA has a set of example configuration files already available in the `examples/` directory. We provide one of these here, which runs a complete metagenomics analysis pipeline (`FullPipeline`):

 # Metagenomics analysis pipeline
Prefix data/Stability
Plugin Mothur **inputfile** Stability.mothur **outputfile** none
Plugin CountTableProcessing **inputfile** stability.trim.abund.pick.an.unique_list **outputfile** input.Stability.csv
Plugin CSVNormalize **inputfile** input.Stability.csv **outputfile** input.Stability.normalized.csv
Plugin Correlation **inputfile** input.Stability.normalized.csv **outputfile** correlations.pvalued.csv
Plugin CSVPad **inputfile** correlations.pvalued.csv **outputfile** Stability.pvalued.csv
Plugin GPUATria **inputfile** Stability.pvalued.csv **outputfile** Stability.pvalued.ATria.noa
 # Plugin ATria **inputfile** *Stability.pvalued.csv* **outputfile** *Stability.pvalued.ATria.noa*
Plugin CSV2GML **inputfile** Stability.pvalued.csv **outputfile** Stability.pvalued.gml
Plugin Cytoscape **inputfile** Stability.pvalued.gml **outputfile** none

This particular pipeline is eight stages long. As can be seen, it is quite easy to assemble and is composed of three different components: comments, prefixes, and plugins. We discuss each of these now.

3.2.1 Comments

PLUMA will ignore all characters on a line that follow the pound (`#`) sign. Comments are useful for writing well-documented configuration files, or also to quickly swap out a plugin and test a different one. For example in the case above we included a plugin for the *Ablatio Triadum* (ATria, [13]) algorithm, but then implemented a more efficient version on the GPU. To test for accuracy and performance improvement, we could simply comment out the CPU version of ATria and add the GPU version.

3.2.2 Prefix

The **Prefix** keyword allows you to set a relative or absolute path to use for data files (input and output). There will often be situations where you will use multiple input and output files that are in the same directory. This option allows you to specify that directory once and then assume it, as opposed to having to retype it for every `inputfile` and `outputfile` in the configuration file. In the case above for example, all of our data files were contained in `data/Stability`, so we specified that as the first line of the PLUMA configuration file. You are allowed to use multiple lines that start with **Prefix**, if at a later stage you want the prefix to change. For a given `inputfile` or `outputfile`, PLUMA will always append the last **Prefix** specified (or none if one was never specified).

3.2.3 Plugins

The core component of the PLUMA configuration file is a sequential collection of unique plugin identifiers (one for each stage of the pipeline), their input files, and their output files:

Plugin *plugin1* **inputfile** *inputfilename1* **outputfile** *outputfilename1*
Plugin *plugin2* **inputfile** *inputfilename2* **outputfile** *outputfilename2*
Plugin *plugin3* **inputfile** *inputfilename3* **outputfile** *outputfilename3*
.
.
.

The plugin identifiers (*plugin1*, *plugin2*, *plugin3*, etc.) must refer to names of plugins installed within the `PLUMA_PLUGIN_PATH` or the `plugins/` folder in the PLUMA source tree. Output files will be written by their respective plugins, and input files will be read. If a particular plugin does not read and/or write a file, specify `none` as the value of `inputfile` and/or `outputfile`, respectively. Input files must exist at the point when a plugin is executed, but not necessarily at the start of the pipeline execution. Since the configuration file is sequential, an input file could be an output file of a previous plugin. Note while it will often be the case for the input file of a particular plugin to be the same as the output file name of its immediately preceding plugin, that need not necessarily be the case. In our above example, both `GPUATria` (Stage 6) and `CSV2GML` (Stage 7) took the same file `input.Stability.csv` as input, which was produced as an output of `CSVPad` (Stage 5).

3.2.4 PLUMA prepackaged plugins

Upon installing PLUMA, you receive a set of prepackaged plugins along with some example configuration files that use them. The plugins are contained within the `plugins/` subdirectory of the PLUMA source tree, and the configuration files are under `examples/`. Input and output files referenced in the configuration files will be in the `data/` subdirectory.

Below we provide a description of each of these plugins, that includes their plugin identifier, source code language, and input and output file formats. Note that many of the plugins convert between file formats, which can be useful for intermediate stages of a PLUMA pipeline.

Plugin ID	Language	Description	Input Format	Output Format
AffinityPropagation	Python	Runs *Affinity Propagation* [20], an algorithm for clustering that uses message-passing between nodes.	CSV	CSV
ATria	C++	Runs the *Albatio Triadum* (ATria) algorithm for centrality (CPU version).	CSV	NOA
AutoCorrelation	R	Computes the autocorrelation value of a time series variable or set of variables.	TXT	TXT
BiasedPageRank	Python	Runs a biased [62] version of Google PageRank [45] that favors nodes in the same cluster. Built with components of two other plugins.	prefix	NOA
Binomial	R	Distance function for a network that uses Binomial Deviance [35].	CSV	CSV
Bray	R	Bray-Curtis [8] distance function.	CSV	CSV
CalcMeanStd	Python	Calculate and print the mean and standard deviation of a datset.	NOA	none
Canberra	R	Canberra [33] distance function.	CSV	CSV
Chao	R	Chao's Method [10].	CSV	CSV
Classify	Python	Properly classify bacterial OTUs at the lowest level in the phylogentic tree.	NOA	NOA
ClusterizeCSV2NOA	Python	Takes a clusterized CSV file and converts it to NOA format. This in turn can be used by Cytoscape to color nodes by cluster.	CSV	NOA
Clusterize	Python	Takes a network and a set of clusters, and output a modifies adjacency matrix that includes only edges between neighbors in the same cluster.	prefix	CSV
CountTableProcessing	R	Takes a set of abundances and OTUs from Mothur, and computes an abundance matrix.	prefix	CSV
CrossCorrelation	R	Cross-correlate two univariate time series.	prefix	TXT
CSV2GML	Python	Converts a CSV file to GML.	CSV	GML
CSVMapRange	Python	Takes two files of abundances and maps the values of the first into the range of the second.	prefix	CSV
CSVNeg2Zero	Python	Remove all negative edges.	CSV	CSV
CSVNormalize	Python	Takes a CSV file and normalizes its values across rows.	CSV	CSV
CSVPad	Python	Pads the top row of a CSV file with a null string. Useful when reading as a table.	CSV	CSV
CSVScale	Python	Sets zero elements to the minimum value, and rescales.	CSV	CSV
CSVTranspose	Python	Takes a matrix and computes its tranpose.	CSV	CSV
CSVZero2Min	Python	Sets zero elements to the minimum value but does not rescale.	CSV	CSV
Cytoscape	C++	Runs Cytoscape [15] on the input network. Assumes Cytoscape is in your `PATH`.	GML	none
CytoViz	Perl	Scripted version of Cytoscape plugin.	GML	none
Degree	C++	Runs weighted degree [24] centrality.	CSV	NOA
Detrend	R	Least-squares fitting of time-series data.	CSV	CSV

Plugin ID	Language	Description	Input Format	Output Format
Dickey-Fuller	R	Augmented Dickey-Fuller Test [53] for time-series data.	TXT	TXT
DistanceCorrelation	R	Distance Correlation Function [57].	CSV	CSV
EM	C++	K-Group Community Detection [41].	GML	TXT
Euclidean	R	Euclidean distance function.	CSV	CSV
Exponential	Python	Exponential centrality [7].	CSV	NOA
Gower	R	Computes Gower index.	CSV	CSV
GPUATria	CUDA	Runs the *Albatio Triadum* algorithm for centrality (GPU version). Useful for large networks.	CSV	NOA
Horn	R	Horn's overlap index.	CSV	CSV
InverseSimpson	Python	Inverse Simpson diversity measure.	CSV	none
Jaccard	R	Jaccard index [25].	CSV	CSV
Kendall	R	Kendall correlation [28].	CSV	CSV
Kulczynski	R	Kulczynski similarity [32].	CSV	CSV
LSA	Python	Local Similarity Analysis [51].	CSV	CSV
MakeCliques	Python	Create a network with a certain number of cliques.	TXT	CSV
MakeSynthetic	Python	Create a synthetic network.	TXT	CSV
Manhattan	R	Manhattan distance.	CSV	CSV
Map2Positive	Python	Maps a [-1,1] range to [0,2], useful for correlations and visualization.	CSV	EDA
MappedWeight	Python	Sets negative edges to zero and produces an edge file for Cytoscape (this allows the original weights to be kept).	CSV	EDA
MCL	R	Runs Markov clustering [59].	CSV	CSV
MetaBAT	C++	Runs the MetaBAT [26] software, if installed.	TXT	prefix
MIC	R	Computes maximal information coefficient [49].	CSV	CSV
ModularityMaximization	Python	Detects communities using the Modularity Maximization [40] algorithm.	prefix	none
Morisita	R	Computes Morisita overlap [37].	CSV	CSV
Mothur	C++	Runs the Mothur software, if installed.	MOTHUR	none
Mountford	R	Computes the Mountford Dissimilarity index [38].	CSV	CSV
NetworkViz	Python	Takes a prefix for a CSV and cluster CSV file, and generates appropriate files for Cytoscape visualization.	prefix	COM
NormScoreTransform	Python	Normalizes across columns.	CSV	CSV
PageRank	Python	Runs Google's PageRank centrality algorithm.	CSV	NOA
PCL2CSV	Python	Converts a PCL file to CSV.	PCL	CSV
Pearson	R	Pearson [46] correlation.	CSV	CSV
Prestige	C++	Calculates Katz prestige [27] centrality.	CSV	NOA
PyATria	Python	Scripted version of *Albatio Triadum*.	CSV	NOA
QGraph	R	Visualizes a network using the R QGraph library.	prefix	PNG
Raup	R	Runs the Raup-Crick [48] dissimilarity algorithm.	CSV	CSV

Plugin ID	Language	Description	Input Format	Output Format
ReactionPathway	Python	Takes a set of input files and determines statistics based on how often two metabolites are on the same reaction pathway, and in the same cluster.	prefix	TXT
RemoveNegative	Python	Takes a network and makes all negative edges positive.	CSV	CSV
SIMLR	R	Similarity Learning algorithm for clustering [60].	prefix	prefix
Spearman	R	Spearman [56] correlation.	CSV	CSV
Spectral	Python	Runs spectral clustering [4].	CSV	CSV
Variance	Python	Takes a CSV file and computes variance in weights across rows.	CSV	TXT

PLUMA prepackaged plugins use the following file formats:

1. **Comma-Separated Value (CSV):** A tabulated data format consisting of a header followed by rows, with each data cell separated by commas. Microsoft Excel commonly uses CSV files.
2. **Graph Modeling Language (GML):** Textual network format with modules for nodes and edges, used by Gephi [5] and Cytoscape.
3. **Node Attribute File (NOA):** A simple table of node names and attribute values, accepted by Cytoscape.
4. **Edge Attribute File (EDA):** A table of edge names and attribute values, also accepted by Cytoscape.
5. **Pre-CLustered File (PCL):** Describes a gene (often regulatory) network, used by the Stanford Microarray Database [16].
6. **Text File (TXT):** Standard plain textual representation.
7. **Mothur Program (MOTHUR):** A program written in Mothur's domain-specific language.
8. **Portable Network Graphics (PNG):** Image format.
9. **Cytoscape COMmands (COM):** Cytoscape scripting language.
10. **prefix:** Indicates that the plugin will be reading/writing multiple files, in this situation just accept the prefix.

New PLUMA plugin extensions need not use one of these file formats. The important matter is that whenever you use a plugin, a file in its accepted format must either exist or be produced by an earlier pipeline stage.

Chapter 4

Extending PLUMA

Since as we mentioned our primary goal in developing PLUMA is infinite extensibility, its most important component does not lie in the features that exist, but those that will exist in the future as new plugins. We thus now provide an outline on how to develop a new plugin for PLUMA.

Plugins can be built using existing tools or even existing plugins, or be constructed entirely from scratch in your language of choice. Our `PluginGenerator` module can be used if you want to include an existing, working software tool as part of the PLUMA pipeline. We describe that first. Next, we give an outline on how to develop a plugin in all languages supported by PLUMA. Finally, we illustrate how to take components of an existing plugin and use them as part of a new one. Examples of all of these exist in the PLUMA prepackaged plugins, and we will refer to those examples as we describe these processes below.

4.1 Generating a New Plugin for An Existing Software Package

In many situations, you may have an existing tool that is already fully developed and tested, and you would like to include that tool as a part of your PLUMA pipeline. PLUMA includes a *plugin generator* tool for this purpose, which is available in the `PluginGenerator/` subdirectory. The generated plugin will be in C++, will assume that this existing tool is within your system `PATH`, and will be automatically placed in the `plugins/` subdirectory of the PLUMA source tree. The advantage of placing it here is that no modifications to any PLUMA compile scripts will be required. These scripts automatically check for new or modified plugins within `plugins/`, since that is the default location for all PLUMA plugins. We chose C++ as the language for the generated plugin, since it is the same language as the PLUMA back end and thus will require no additional language tools to be installed.

The plugin generator is automatically compiled as a part of the PLUMA package when running SCons. To run the plugin generator, change to the `PluginGenerator/` subdirectory.

4.1.1 Option 1: Single Command

For a package that accepts no or predefined command line arguments, the command to generate the plugin is straightforward:

`./generate` *plugin_name* *command_to_run*

Where *plugin_name* is the name of the new PLUMA plugin, and *command_to_run* is the command that should be used to run the software. Be sure that, unless you intend to replace an existing plugin in the `plugins/` subdirectory, that *plugin_name* does not name an existing plugin. Otherwise the existing plugin that shares the same name will be automatically overwritten. As an example, to generate a plugin that runs Mothur with no command line arguments, you can run:

```
./generate Mothur mothur
```

since `mothur` is the name of the executable file for Mothur.

4.1.2 Option 2: Input and Output File

Most software tools (including Mothur) will take some type of input parameter(s), however. In a simple case, the only dependencies can be the `inputfile` and `outputfile` that are specified in the configuration file. Recall that our earlier configuration file specification for our *Mothur* plugin looked like this:

Plugin `Mothur` **inputfile** `Stability.mothur` **outputfile** `none`

For the generator, this same `inputfile` keyword can also act as a placeholder for the argument that is specified in the configuration file. For example, this *Mothur* plugin accepts one input file and produces no output. We generated this plugin with the command:

```
./generate Mothur mothur inputfile
```

So that if the user provided `Stability.mothur` as the `inputfile` in the configuration file, this will automatically be plugged in as `inputfile` here, and the command "`mothur Stability.mothur`" will be executed by this plugin. Thus the `PluginGenerator` can automatically produce plugins that can be customized by the user at runtime.

4.1.3 Option 3: Fully Customized

Other software tools such as Cytoscape will take multiple input parameters. Since PLUMA plugins in general can only accept a single input file, in this situation the generated plugin will accept one plaintext file, in the following format:

```
keyword1 value1
keyword2 value2
keyword3 value3
...
```

Where the keyword values are specified in the command to `generate`. For example, our Cytoscape plugin was generated using the following command:

```
./generate Cytoscape cytoscape -N networkfile -s sessionfile
```

We also include a file in `data/Stability` called `Stability.visualization.txt`, with the following contents:

```
networkfile Stability.pvalued.gml
sessionfile default.cys
```

By providing `Stability.visualization.txt` as the `inputfile` argument to the `Cytoscape` plugin in the PLUMA configuration file, this will run the following command:

```
cytoscape -N Stability.pvalued.gml -s default.cys
```

Thus the user-specified values in the input plain text file automatically get plugged in to their corresponding keyword in the execution command. Note that command line arguments that start with a dash (−) are copied directly. The output file can still be used as before in the command line. If `inputfile` is detected in the arguments to `generate` the PluginGenerator will automatically assume Option 2, otherwise it will use Option 3. Therefore you must make sure to not use `inputfile` as a keyword if you desire Option 3.

4.2 Building a New Plugin From Scratch

We now describe how to build a new plugin entirely from scratch, using one of the supported languages of PLUMA. These languages have a wide variety of syntax and semantics, some are object-oriented and some are not, some are compiled and some are scripted, some run on the CPU and some run on the GPU. As one can thus imagine, the process of setting these up will vary by the language, however it is important to remember that from the perspective of running PLUMA, nothing will be different. When specifying a plugin in the configuration file, you do not need to know in which language it is written, nor do you need to worry about parsing the configuration file differently depending on the language. You only need to be concerned with writing the functionality of the plugin itself.

One programming unit that all of these languages do have in common is procedures. There are three procedures that you will need to setup, independent of the language you choose:

1. An `input()` procedure that accepts the name of an input file, reads it, and performs initialization.
2. A `run()` procedure that accepts zero parameters, and runs the plugin.
3. An `output()` procedure that accepts the name of an output file, finalizes the plugin, and writes the file.

We now review the five supported languages, examples of how to write those procedures in the respective languages, and any particular nuances that you need to follow given your language of choice. With respect to extensibility however, no one language has advantages over the other in terms of ease. We included this flexibility to accomodate a wide range of user preferences with respect to programming languages.

4.2.1 C++

For our C++ example we use the `ATria` plugin, which takes as input a signed and weighted network in CSV format, runs the Ablatio Triadum algorithm for centrality (importance), and outputs a list of the most important nodes in the network in NOA format. This latter format makes it convenient to import the file into Cytoscape and perform a useful visualization such as coloring the nodes in the network by centrality value.

When constructing a new plugin, you must include it inside a subdirectory of either the default PLUMA `plugins` directory or another directory within the `PLUMA_PLUGIN_PATH`. The subdirectory should uniquely define the plugin, and your C++ source files should use that name followed by `Plugin`. This also should be the name of your new C++ class, which will inherit from the parent class `Plugin`. As an example, assuming we are currently in the `plugins` directory - we would first make a directory `ATria` to hold all source files. Then, we can setup the header file for this plugin `ATria/ATriaPlugin.h` as shown in Program 1.

As is typical in a C++ header file, preprocessor macros (`#ifndef`, `#define` and `#endif`) should be used to avoid including the file multiple times. New C++ plugins must include two files from PLUMA: `Plugin.h` and `PluginProxy.h`, as well as any other files that need to be included (i.e. from the Standard Template Library [39], we included its `string` class above). Our new class will be named `ATriaPlugin`, which as mentioned inherits from `Plugin`. Within the class, the only requirements are that there must be three procedures: `input`, `run`, and `output`. The `input` and `output` procedures must accept one `std::string` parameter for the input and output file, respectively. Note these are required *even if* they do not do anything for this particular plugin, you just can leave their definitions empty. From there, you can feel free to include any other necessary member variables or procedures to accomplish your particular task. In Program 1, we have variables for the network (an array of `float`) and bacteria

Program 1 `ATria/ATriaPlugin.cpp`

```
#ifndef ATRIAPLUGIN_H
#define ATRIAPLUGIN_H

#include ''Plugin.h''
#include ''PluginProxy.h''
#include <string>
// Other necessary includes...

class ATriaPlugin : public Plugin
{
  public:
  // These are required
  void input(std::string file);
  void run();
  void output(std::string file);

  // Other member procedures...

  private:
     float* OrigGraph;
     std::string* bacteria;
     // Other member variables...
};


#endif
```

(the names of nodes in this network, an array of `std::string`). This is so named because ATria was originally tested on bacterial co-occurence networks [18].

The corresponding source file, `ATria/ATriaPlugin.cpp` is setup as shown in Program 2.

Program 2 `ATria/ATriaPlugin.cpp`

```
#include ''PluginManager.h''
#include ''ATriaPlugin.h''
// Other necessary includes...

void ATriaPlugin::input(std::string file) {
   // Read file, and initialize member variables...
}

void ATriaPlugin::run() {
   // Run the algorithm...
}

void ATriaPlugin::output(std::string file) {
   // Perform any final operations, and write file...
}

// Other member procedure definitions...

// Required, connects the plugin to the PluMA back end
PluginProxy<ATriaPlugin> ATriaPluginProxy
      = PluginProxy<ATriaPlugin>(''ATria'', PluginManager::getInstance());
```

There are two necessary includes at the top of the file. `PluginManager.h` is required because at the end of the file we make use of PLUMA's `PluginManager` to interface the new plugin with the PLUMA back end. Here we create a `PluginProxy` which uses the Proxy [3] design pattern to *register* the new plugin with the `PluginManager`. As arguments to its constructor, it accepts the unique plugin name `ATria` as a string, along with the `PluginManager`. This will correspond to its name in a PLUMA configuration file. The other required include is standard C++ practice, the corresponding header file for this class. Any other necessary includes can then follow.

The source file now must define `input`, `run` and `output`. We provide these three procedures for convenience in terms of separating functionality. The one requirement is that if your plugin will require an `inputfile` in the PLUMA configuration file it must be read in `input`, and similarly an `outputfile` must be written in `output`. From there as the new plugin developer you have control over what functionality goes where, but the convention is to initialize member variables and prepare to run the new algorithm in `input`, actually run the algorithm in `run`, and perform any final operations before writing the outputfile in `output`. As an example, our implementation of `ATriaPlugin` reads the CSV file and populates the graph and bacteria arrays in `input`, executes the centrality algorithm in `run`, and sorts the list of nodes by centrality value before writing the NOA file in `output`.

The SCons scripts of PLUMA will automatically detect and compile the new plugin as a shared object (`so`) file if it is stored in the `plugins/` subdirectory of the PLUMA root. We do not compile everything in the `PLUMA_PLUGIN_PATH` for now, to avoid potentially accessing installation folders for which a user may not have permission. Shared objects are loaded at runtime, which will help to keep both the executable and execution environment of PLUMA lightweight.

4.2.2 CUDA

We provide CUDA as an option for constructing a Graphics Processing Unit (GPU) plugin that can run on an NVIDIA graphics card. The large number of cores and multithreading within each core have made the GPU a solid option for many bioinformatics applications [9]. As an example, applying the GPU to ATria yielded an order of magnitude improvement in speed for large networks [12], allowing us to complete analysis of a 3000-node fruit fly network in a few hours instead of a few days.

CUDA is an extension of C and recent additions to the NVIDIA compiler allow GPU functionality to be invoked from C++ classes, so building a CUDA plugin will not be much different from building a C++ plugin. The first difference is in the filename; while the header file can still end in a `.h` extension the source file must now end in `.cu`. This extension is recognizable by the NVIDIA compiler, and is what tells the PLUMA SCons scripts to use this compiler as opposed to the standard GNU compiler.

The second difference is that while GPU functionality can be invoked from a C++ class, it cannot be contained in a member procedure of a C++ class. Rather, it must be implemented as a standalone *kernel* procedure where all parameters and variables are assumed to be allocated on the GPU. As an example, see our new header file for our plugin `GPUATria` in Program 3.

Note that the preprocessor directives, required includes, and C++ class template are the same. However, at the bottom of the file we include two GPU kernel procedure headers. In CUDA these always begin with `__global__` and accept parameters that have been allocated on the GPU. In our implementation we run a GPU version of Floyd-Warshall [19], [44] which runs the all-pairs shortest path algorithm, and then another kernel procedure to compute centrality, which we call *pay* as ATria was economically-based. Any number of GPU kernels can be declared outside of the C++ class.

The source file `GPUATria/GPUATriaPlugin.cu` now must include, in addition to the member procedure definitions, the kernel procedure definitions. These then will be called from the member procedures. We show this template in Program 4. Otherwise, there is nothing different from defining a C++ plugin.

Since GPU parallelism will be used for computationally expensive tasks it is likely that you will invoke your GPU kernels from the `run` procedure, though this is not required. CUDA procedures are called with a certain number of thread *blocks* and *threads per block*, which are provided as arguments to the kernel within the triple carat <<< and >>> operators. Their values can have critical effects on the efficiency of GPU code [34]. For more information, please view the CUDA Programmer's Guide [42].

As with C++ plugins, the SCons scripts of PLUMA will compile any CUDA plugins as shared objects.

4.2.3 Python

The major difference with the rest of the PLUMA supported languages is that they are scripted. This difference is critical in terms of system architecture. First, any scripted plugins by definition will not require compilation at all, and thus there is no mechanism built into the PLUMA SCons scripts to detect their presence (though they still will be detected at runtime). Another big difference is that you can contain the entire plugin within one file. You can still use multiple files if you choose (perhaps you want to borrow an existing procedure or module from someplace else), however the functionality of the plugin should all be placed in the same file.

As our Python example, we will use the `PageRank` plugin - which runs Google's PageRank algorithm and like ATria computes a centrality score for every node in the network. The location and name of this file should follow the same conventions up to this point - from a location in the `PLUMA_PLUGIN_PATH` create a new directory that corresponds to the name of the new plugin (in this case, `PageRank`) and use this to name the file followed by `Plugin` (in this case, `PageRank/PageRankPlugin.py`). Once again because there is no compilation, we no longer have a need to create a proxy to link the new plugin module with the back end. Program 5 shows our template for `PageRankPlugin`.

This particular plugin is a good example of some advantages of scripting. Our implementation uses the NetworkX [23] libraries for running PageRank, PythonDS [36] to construct the graph, and Numerical Python [58] to perform useful and efficient calculations. All of these are easily importable at the top of the file. We follow these with global variables, and any procedure definitions (here we define a procedure that takes a CSV file and produces a graph that NetworkX accepts).

Program 3 `GPUATria/GPUATriaPlugin.h`

```
#ifndef GPUATRIAPLUGIN_H
#define GPUATRIAPLUGIN_H

#include ''Plugin.h''
#include ''PluginProxy.h''
#include <string>
// Other necessary includes...

class GPUATriaPlugin : public Plugin
{
  public:
  // These are required
  void input(std::string file);
  void run();
  void output(std::string file);

  // Other member procedures...

  private:
     float* OrigGraph;
     std::string* bacteria;
     // Other member variables...

};

__global__ void _GPU_Floyd_kernel(int k, float *G, int N);
__global__ void _GPU_Pay_kernel(float* D, float* P, int N);
// Other GPU kernels...

#endif
```

Program 4 GPUATria/GPUATriaPlugin.cu

```
#include ''PluginManager.h''
#include ''GPUATriaPlugin.h''
// Other necessary includes...

void GPUATriaPlugin::input(std::string file) {
   // Read file, and initialize member variables...
}

void GPUATriaPlugin::run() {
   // Run part of the algorithm...

   // At some point call first kernel
   _GPU_Floyd_kernel<<<dimGrid,BLOCK_SIZE>>>(k,dG,N);

   // ...

   // At some point call second kernel
   _GPU_Pay_kernel<<<numblocks, BLOCK_SIZE>>>(dG, dPay, (N/2));

   // Finish the algorithm....
}

void GPUATriaPlugin::output(std::string file) {
   // Perform any final operations, and write file...
}

// Other member procedure definitions...

__global__ void _GPU_Pay_kernel(float* D, float* P, int N) {
   // GPU code...
}


__global__ void _GPU_Floyd_kernel(int k, float *G, int N){
   // GPU code...
}

// Other kernel procedure definitions...



// Required, connects the plugin to the PluMA back end
PluginProxy<GPUATriaPlugin> GPUATriaPluginProxy
      = PluginProxy<GPUATriaPlugin>(''GPUATria'', PluginManager::getInstance());
```

Program 5 `PageRank/PageRankPlugin.py`

```
import numpy
import networkx
from pythonds.graphs import PriorityQueue, Graph, Vertex
# Any other imports...

numdiff = 0
ALPHA=0.5
def buildNetworkXGraph(myfile):
    # Read myfile, build and return graph

# Any other global variables or procedures...

class PageRankPlugin:
    def input(self, file):
        self.bacteria, self.graph = buildNetworkXGraph(file)
    def run(self):
        self.U = networkx.pagerank(self.graph, alpha=ALPHA, max_iter=100)
    def output(self, file):
      UG = []
      for key in self.U:
          UG.append((self.U[key], key))
      UG.sort()
      UG.reverse()

      # Write file...
```

Like C++, Python is object-oriented. Thus similarly, we create a class `PageRankPlugin` and define three procedures `input`, `run`, and `output`. In `input`, we call our procedure which reads the input CSV file and builds the graph. In `run` we execute the PageRank algorithm, which consists of a call to the NetworkX functionality. Finally in `output` we sort the centrality values produced by PageRank and output the NOA file for Cytoscape. Note that member procedures in Python must include `self` as their first parameter, and member variables can be created on the fly by appending `self` to the front of their identifiers. There are thus no declarations of variables or plugins in Python, which is also the case in most scripting languages.

4.2.4 R

The final two scripting languages, R and Perl, are not object-oriented by default. However, there are importable packages available in both cases that facilitate object-oriented programming. To uphold compatibility with the standalone tools for each language, we do not use classes to represent plugins in either of these options.

In the case of R we use the same naming scheme, but instead of defining three member procedures `input`, `run` and `output` they will be defined as standalone procedures. Program 6 shows our R implementation of a Correlation plugin, which takes as input a CSV file of abundances and outputs a CSV file of correlations.

The plugin file should start with any global variables. In our case we have a `p_value` that we use to threshold our correlations, and anything below this we assume a value of zero. Any external libraries that should be imported (in our case `Hmisc` will give us access to the R procedure `rcorr`) also should be placed at the top of the file. We then define the three required procedures and as before, with `input` and `output` taking one parameter for their respective files. In our case the input procedure reads the CSV file of abundances and stores them in a matrix `pc`. The `run` procedure does some processing to `pc` (removing headers, converting to numbers) before computing correlations and thresholding them using our `p_value`. Finally, `output` writes its CSV file as a table of correlations. You can define other helper procedures in addition to these. Also since there is no longer a class interface, variables that need to be accessible across procedure calls must be global. In R, these are assigned using `<<-` instead of the standard `<-` for local variables. In our case, `pc` and `cn` (the graph and the headers) are global.

4.2.5 Perl

Building a plugin in Perl follows closely the steps to building a plugin in R, with three standalone procedures `input`, `run` and `output`. The naming strategies are also the same as those used when developing plugins for the other languages. We have developed a `CytoViz` plugin that takes an GML file and visualizes the corresponding network using Cytoscape, in Program 7.

This particular plugin will automatically launch Cytoscape assuming that its install directory is stored in the `CYTOSCAPE_HOME` environment variable. Note that in Perl, procedure parameters are not specified within parentheses but their values are automatically stored in the predefined array `@_`. Since this plugin is only launching Cytoscape, there is nothing to output and we simply return from that procedure. Global variables that should be accessible across procedures should be declared using the `my` keyword at the top of the file. In our case the sole global variable is the input file `$gmlfile`. In `run` we either launch Cytoscape using the `-N` flag to pass a network input file, or provide a message to the user to set `CYTOSCAPE_HOME` properly.

4.3 Building a New Plugin using Existing Plugins

In our Python section, we designed a plugin to implement Google's PageRank algorithm for centrality. The standard PageRank algorithm works by simulating a random walk through a network. The walker always resides on a particular node in the network and upon every move has a probability α of going to a neighbor node, and a probability $1 - \alpha$ of teleporting to somewhere random in the network. Centrality values then depend on the amount of times a node is visited, which will be large for nodes with lots of connections and those connected to other central nodes. PageRank is an eigenvector-based approach, which iteratively solves a system of equations with the leading eigenvector converging to the vector of centrality values.

Program 6 `Correlation/CorrelationPlugin.R`

```
p_value <- 0.01;
# Any other global variables...

libs <- c(''Hmisc'');
# Any other libraries to import

# Import libraries
lapply(libs, require, character.only=T);

# Required
input <- function(inputfile) {
  pc <<- read.csv(inputfile, header = TRUE);
}

# Required
run <- function() {
  cn <<- colnames(pc);
  cn <<- cn[2:length(cn)];
  pc <<- pc[,-1];
  pc <<- apply(pc, 1, as.numeric);
  pc <<- t(pc);
  correlations <<- rcorr(pc[,]);
  pc <<- as.matrix(correlations$r);
  pc[is.na(pc)] <<- 0;
  empty <- c('''');
  pc[which(correlations$P>p_value)] <<- 0;
}

# Required
output <- function(outputfile) {
   write.table(pc, file=outputfile, sep='','', append=FALSE,
        row.names=unlist(cn), col.names=unlist(cn), na='''');
}

# Other helper procedures...
```

Program 7 `CytoViz/CytoVizPlugin.pl`

```
my $gmlfile;
# Any other global variables...

# Required
sub input {
    $gmlfile = @_[0];
    return;
}

# Required
sub run {
    $cytohome = $ENV{''CYTOSCAPE_HOME''};
    length($cytohome) != 0 or die ''Please set CYTOSCAPE_HOME\n'';
    @args = ($cytohome . ''/cytoscape.sh'', ''-N'', $gmlfile);
    system(@args) == 0 or die ''system @args failed: $?\n'';
    return;
}

# Required
sub output {
    return;
}
```

Later works have shown that the PageRank algorithm can be artificially *biased*, to favor nodes with certain properties over others. For example, one way of biasing PageRank is to use multiple values of α. We could have a higher α value (call it α_1) for neighboring nodes in the same cluster, and a lower value (call it α_2) for neighboring nodes in different clusters. The probability of teleporting then becomes $1 - \alpha_1 - \alpha_2$. We have found such an approach to more clearly isolate cluster leaders in their centrality scores, as the walker will tend to stay in the same cluster for a longer period of time and thus visit its leader node more often.

We have already shown the `PageRankPlugin` that has been implemented in Python. We have another plugin `ClusterizePlugin`, which takes as input a network and associated cluster identifiers (one per node) and has a function `inSameCluster()` which returns true if the node pair are in the same cluster, and false otherwise. To build our new `BiasedPageRankPlugin`, we will borrow functionality from both plugins. Program 8 shows our Python implementation.

Program 8 `BiasedPageRank/BiasedPageRankPlugin.py`

```
import numpy
import networkx as nx

import plugins.Clusterize.ClusterizePlugin
import plugins.PageRank.PageRankPlugin


def biasedpagerank(G, clusters, alpha1=0.5, alpha2=0.35, max_iter=100,
                   tol=1.0e-8, nstart=None):

    # Biased Algorithm...

        # Use approriate alpha value
        for nbr in W[n]:
            if (ClusterizePlugin.inSameCluster(n, nbr, clusters)):
             x[nbr]+=alpha1*xlast[n]*W[n][nbr]['weight']
            else:
             x[nbr]+=alpha2*xlast[n]*W[n][nbr]['weight']

    # Biased Algorithm...



############################################################

class BiasedPageRankPlugin(PageRankPlugin.PageRankPlugin):
  def input(self, filename):
    PageRankPlugin.input(self,filename+''.csv'')
    self.clusters = ClusterizePlugin.readClusterFile(filename+''.clusters.csv'')
  def run(self):
    self.U = biasedpagerank(self.graph, self.clusters, alpha1=0.5,
                            alpha2=0.35, max_iter=100)
```

Note that we were able to import our `ClusterizePlugin` and `PageRankPlugin` to gain access to their functionality. We have our `BiasedPageRankPlugin` inherit from `PageRankPlugin`, since it works almost the same way but with a slight modification. This relationship follows the IS-A relationship that justifies using inheritance in object-oriented programming. The call to the `inSameCluster` function of the `ClusterizePlugin` enables us to choose the correct α value to use when updating the vector of centrality values computed by PageRank in our `biasedpagerank` procedure. We also can reuse the `input` procedure of the `PageRank` plugin when reading the CSV file (since it also reads a CSV file), and use the `readClusterFile` procedure of `ClusterizePlugin` to read the file of cluster identifiers for every node. The `run` procedure now calls the new `biasedpagerank` procedure, and the `output` procedure has no differences from that of the `PageRankPlugin` so we do not define a new one and use the inherited procedure.

This method of extending PLUMA will thus minimize the amount of time a user must spend reinventing the wheel, allowing them to focus specifically on their new ideas.

Chapter 5

PLUMA Example Full Pipeline and Configuration File

The PLUMA source release includes a folder of example configurations and selected test data. Examples are organized by purpose, and each folder contains one or more pipeline configuration files. Some configurations include configuration files for both the CPU and GPU, and others run the same algorithm (i.e. ATria) on multiple biological systems.

We take a closer look now at the full pipeline example that we used to describe the structure of the PLUMA configuration file:

 # Metagenomics analysis pipeline
Prefix data/Stability
Plugin Mothur inputfile Stability.mothur outputfile none
Plugin CountTableProcessing inputfile stability.trim.abund.pick.an.unique_list outputfile input.Stability.csv
Plugin CSVNormalize inputfile input.Stability.csv outputfile input.Stability.normalized.csv
Plugin Correlation inputfile input.Stability.normalized.csv outputfile correlations.pvalued.csv
Plugin CSVPad inputfile correlations.pvalued.csv outputfile Stability.pvalued.csv
Plugin GPUATria inputfile Stability.pvalued.csv outputfile Stability.pvalued.ATria.noa
 # Plugin ATria inputfile Stability.pvalued.csv outputfile Stability.pvalued.ATria.noa
Plugin CSV2GML inputfile Stability.pvalued.csv outputfile Stability.pvalued.gml
Plugin Cytoscape inputfile Stability.pvalued.gml outputfile none

This configuration file is a terrific example of the integration of many plugins with different implementation languages, some generated by the PluginGenerator, and others constructed from scratch. Figure 5.1 illustrates this. Mothur and Cytoscape are standalone software tools for which we produced C++ plugins using the PluginGenerator. We implemented three of the plugins (CSVNormalize, CSVPad, and CSV2GML) in Python, two (CountTableProcessing and Correlation) in R and one in CUDA (GPUATria).

We now outline the details of each stage of the pipeline. Some stages are simple file conversions, others are more involved - but the setup is the same with respect to the PLUMA configuration file.

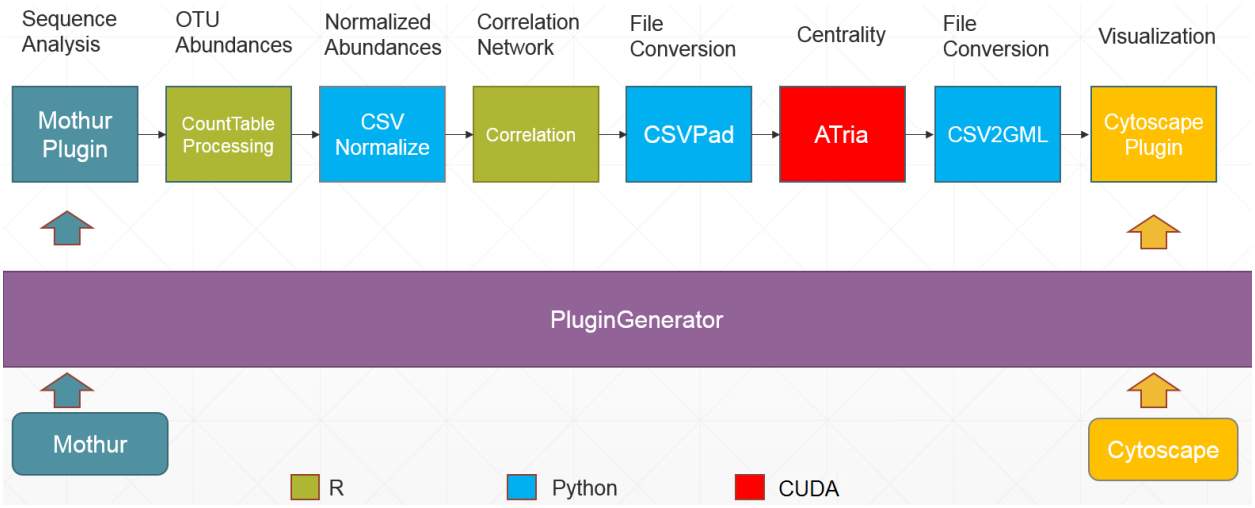

Figure 5.1: Conceptual description of our FullPipeline example. Plugins for Mothur and Cytoscape were generated automatically PLUMA's PluginGenerator module, and the rest were written in one of several different languages.

5.1 Stage 1: Mothur

As mentioned, this plugin was automatically generated through the PluginGenerator module and calls the `mothur` executable assuming it is in your system `PATH`. The input file `Stability.mothur` runs the example from the Mealybugs tutorial [31] for Mothur using input data from Illumina's MiSeq [50] platform. In this case, we start from raw sequence data and perform a series of steps to reduce errors in the reads, followed by a removal of chimeric sequences, and an assessment of error rates (steps 4-7 of the tutorial). This section will end by clustering these final sequences into Operational Taxonomic Unit (OTUs). Mothur does this using similarity-based clustering [63] to compute a distance between sequences. It then references a database (we use SILVA [47]) to take each cluster and map to the most specific phylogenetic classification of microbe.

Mothur produces a number of output files without having to specify them in the PLUMA configuration file. The two with which we are concerned are: (1) the abundance of each OTU (classified by a unique identifier) in each sample (`*.shared`, Program 9), and (2) the mapping of these same OTU identifers to OTU classifications (`*.cons.taxonomy`, Program 10). These each start with the same file prefix, which we feed into the next stage of our pipeline.

Program 9 File of OTU abundances produced by Mothur (`.shared` file).

```
bel    Group     numOtus  Otu001  Otu002  Otu003  Otu004  Otu005  ...
0.03     F3D0      243      500     307     392     404     631   ...
0.03     F3D1      243      350     309     187     63      73    ...
0.03     F3D141    243      387     335     297     483     426   ...
...
```

Program 10 Mapping of OTUs to classifications produced by Mothur (`.taxonomy` file).

```
OTU     Size    Taxonomy
Otu001  12283   Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu002  8888    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu003  7789    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu004  7452    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu005  7425    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu006  6583    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
Otu007  6323    Bacteria(100);''Bacteroidetes''(100);''Bacteroidia...
...
```

5.2 Stage 2: CountTableProcessing

The goal of this plugin (written in R) is to take OTU names and abundances in each sample from Mothur and produce an abundance matrix, with rows as samples and columns as OTUs, and $M[i, j]$ corresponds to the abundance of OTU j in sample i.

`CountTableProcessing` produces this output file in CSV format (Program 11), a format recognized by many spreadsheet applications including Microsoft Excel. This plugin can in a sense be viewed as a tool that merges Mothur's output list of OTUs (which consists of identifers, counts over all samples, and classifications at each level of the taxonomic ranking), and abundances for each sample (which consists of sample identifiers, OTU identifiers, and their abundances in a table). The CSV file produced by this plugin will consist of a table similar to the one in this latter file produced by Mothur, but uses OTU names instead of identifiers for columns. For an OTU name, the plugin takes the most specific classification level produced in the Mothur output for each identifier.

Program 11 OTU abundance matrix produced by Stage 2 of the pipeline.

```
'''','Family.Porphyromonadaceae.0001'','Family.Porphyromonadaceae...
''F3D0'',500,307,392,404,631,352,168,163,136,19,26,113,52,296,101,78...
''F3D1'',350,309,187,63,73,115,131,178,84,112,53,34,116,71,247,0,280...
''F3D141'',387,335,297,483,426,279,205,333,117,181,111,99,9,75,15,99...
''F3D142'',244,258,142,151,253,191,203,81,72,45,51,82,104,10,6,47,8...
''F3D143'',189,152,174,206,310,192,117,91,60,83,68,41,37,17,2,35,7...
''F3D144'',344,243,259,316,467,280,133,36,136,269,93,85,13,29,6,105...
...
```

5.3 Stage 3: CSVNormalize

The third stage of this pipeline (written in Python) normalizes the abundance matrix produced by `CountTableProcessing` across columns, so that each sample's OTU counts correspond to a portion of the total number of OTUs classified in that sample. The output of this plugin is also in CSV format, set up identically to its input file but using normalized instead of raw abundances as shown in Program 12.

Program 12 OTU normalized abundance matrix produced by Stage 3 of the pipeline.

```
'''','''Family.Porphyromonadaceae.0001''','''Family.Porphyromonadaceae...
'''F3D0''',0.0814730324263,0.0500244419097,0.0638748574222,0.06583021...
'''F3D1''',0.0760869565217,0.0671739130435,0.040652173913,0.013695652...
'''F3D141''',0.0835672640898,0.072338587778,0.0641330166271,0.1042971...
'''F3D142''',0.101328903654,0.107142857143,0.0589700996678,0.06270764...
'''F3D143''',0.0792120704107,0.0637049455155,0.0729253981559,0.086336...
'''F3D144''',0.101087275933,0.0714075815457,0.07610931531,0.092859241...
...
```

5.4 Stage 4: Correlation

As a next step, we take these normalized abundances and produce a *correlation matrix*, which provides an estimate of how well pairs of OTU abundances *correlate* with each other. These correlations can be positive or negative. A strong positive correlation between OTU i and j would be indicated by a direct relationship between the abundances of i and j in each sample, i.e. if i is higher in sample A compared to sample B, then j is also higher in sample A compared to sample B. A strong negative correlation between OTU i and j would be indicated by an inverse relationship between their abundances. Zero correlation would indicate that there is no definite relationship between their abundances. Correlations can also be weakly positive and negative, based on the strength of their direct (inverse) relationships. The plugin computes correlation by looking at the values of i and j across all sets of samples.

`Correlation` is an R plugin and uses the `rcorr` method of the R `Hmisc` package, which produces a matrix of correlations and P-values. These values will be thresholded at a P-value of 0.01, which is a modifiable global variable as mentioned earlier.

After running `rcorr`, this plugin does the final processing to produce an output CSV file where this time OTU names occupy both rows and columns, and $M[i, j]$ indicates the correlation between OTU i and OTU j. Program 13 shows this structure. Note that this matrix is symmetric, i.e. $M[i, j] = M[j, i]$. It thus can be represented as an signed and weighted undirected network where OTUs are nodes and correlations are edges, as shown in Figure 5.2. We use this approach to perform downstream analysis in Stage 6.

Program 13 OTU correlation matrix produced by Stage 4 of the pipeline.

```
''Family.Porphyromonadaceae.0001'','''Family.Porphyromonadaceae.0002''...
''Family.Porphyromonadaceae.0001'',1,0.820245563983917,0.579995274543...
''Family.Porphyromonadaceae.0002'',0.820245563983917,1,0.67020320892...
''Family.Porphyromonadaceae.0003'',0.579995274543762,0.67020320892...
''Barnesiella.0001'',0,0,0,1,0.779624223709106,0.874419510364532,0,...
''Family.Porphyromonadaceae.0004'',0,0,0,0.779624223709106,1...
''Family.Porphyromonadaceae.0005'',0,0,0.562049210071564,0.8744195...
...
```

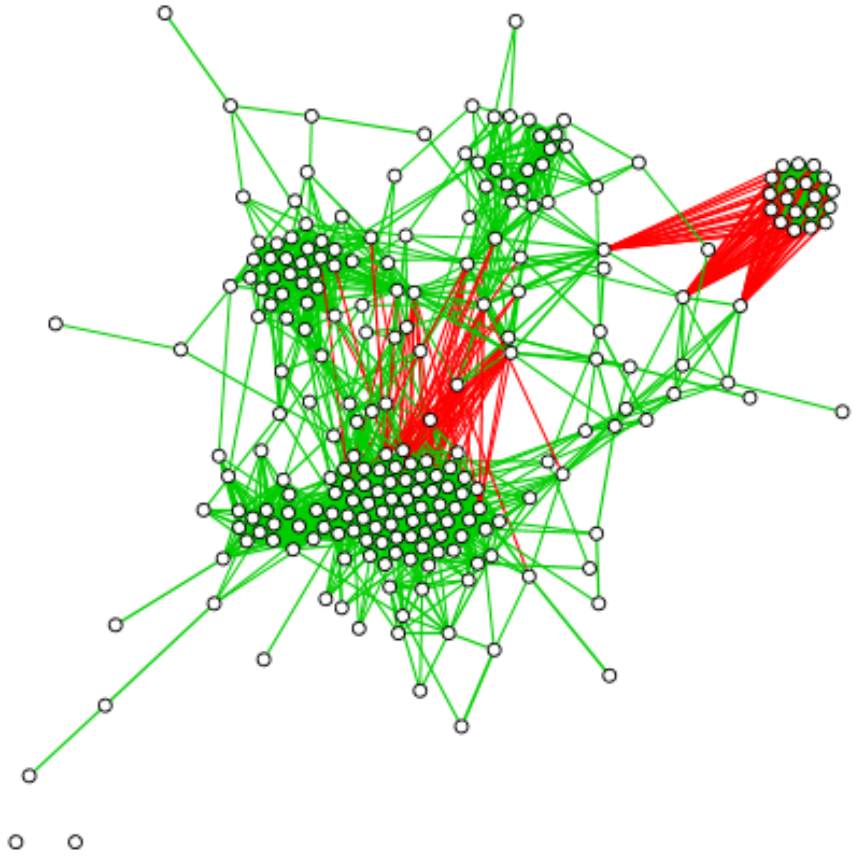

Figure 5.2: The equivalent signed and weighted network corresponding to the correlation matrix in Program 13.

5.5 Stage 5: CSVPad

This plugin (written in Python) is about as simple a file conversion plugin as can be constructed. When assembling this pipeline, we actually created this plugin after the current plugins for Stage 4 and Stage 6. While Stage 6 (`GPUATria`, discussed in further detail next) expects a correlation matrix for input, the format is slightly different than that produced by the `write.table` function of R. The latter function produces a table with row and column labels, but there is no entry for the label where rows and columns intersect. The job of `CSVPad` is simply to append an empty string in that spot, as shown in Program 14.

While a straightforward modification to Stage 4 or (even easier) Stage 6 could have fixed this problem, we felt this was a good demonstration of the extensibility of PLUMA. Since file readers can be added as plugins without loss of generality, no modifications to Stage 4 or Stage 6 were required after testing them. We could simply add a new plugin to "pad" the CSV file output by Stage 4 to be compatible with Stage 6. This problem is now removed for the analysis community, if this situation arises again now a user can simply download this `CSVPad` plugin and drop it into their pipeline. This is much easier than attempting to modify their tools for compatibility with an existing framework.

Program 14 Padded correlation matrix produced by Stage 5 of the pipeline.

```
'''','''Family.Porphyromonadaceae.0001'','''Family.Porphyromonadaceae.00..
'''Family.Porphyromonadaceae.0001'',1,0.820245563983917,0.579995274543...
'''Family.Porphyromonadaceae.0002'',0.820245563983917,1,0.67020320892...
'''Family.Porphyromonadaceae.0003'',0.579995274543762,0.6702032089233...
'''Barnesiella.0001'',0,0,0,1,0.779624223709106,0.874419510364532,0,0...
'''Family.Porphyromonadaceae.0004'',0,0,0,0.779624223709106,1...
'''Family.Porphyromonadaceae.0005'',0,0,0.562049210071564,0.874419510...
...
```

5.6 Stage 6: GPUATria

Once we have the correlation matrix we can perform some downstream network analysis, since as mentioned there is no difference between a correlation matrix and a signed and weighted network. In this network, OTUs are nodes and correlations are edges. Recent work on the microbiome has analyzed these types of networks, which are referred to as bacterial co-occurence networks [29] or also microbial social networks [1]. The concept of a social network naturally arose from the basic property of co-occurence networks, which is a measure of how often two entities appear "in tandem". Biologically this could have several meanings, one microbe may be producing a nutrient that is critical to the other's survival in a positive correlation, or may be producing a toxin that kills off another in a negative correlation.

Viewing these networks from the "social network" perspective, coupled with the fact that our sample co-occurence networks upheld many properties of social networks (such as stability [17]), led to the application of social network algorithms to these networks at later pipeline stages. Ablatio Triadum (ATria) explores the social networking concept of *centrality* (or importance), in these networks. In summary, we would like to produce an ordered list of the most important bacteria in the network. Compared to other social network centrality algorithms, ATria does the best job at finding central nodes in different parts of the network. The three types of nodes it tends to find are: leader nodes of strongly connected components, villain nodes

(common enemies), and bridge nodes between connected components. To improve speed on large networks, this pipeline includes the version of ATria for the GPU (written in CUDA). The output of this plugin is an NOA file, which can be easily imported into Cytoscape. The NOA file consists of a simple table mapping node names to centrality values and ranks, as shown in Program 15.

Program 15 NOA file of nodes and centrality values produced by Stage 6.

```
Name                                Centrality      Rank
Family.Ruminococcaceae.0009         47.1695         243
Oscillibacter.0002                  45.5364         242
Phylum.Firmicutes.0003              41.8464         241
Phylum.Firmicutes.0002              37.3408         240
Family.Lachnospiraceae.0004         33.3711         239
Phylum.Firmicutes.0013              28.8272         238
Order.Clostridiales.0017            25.9752         237
...
```

5.7 Stage 7: CSV2GML

In addition to needing this table, Cytoscape (Stage 8) will also need the network itself, but cannot visualize networks in CSV format. GML is an accepted format by Cytoscape, which is produced by this plugin. `CSV2GML` is thus a simple file converter plugin written in Python that converts an input CSV file into GML format, and its sample output is shown in Program 16. Upon completion of stages 6 and 7, we finally are ready to visualize the network in Cytoscape.

5.8 Stage 8: Cytoscape

The final pipeline stage, `Cytoscape`, was also produced by our PluginGenerator module. This plugin is responsible for visualizing the network produced in Stage 7 using Cytoscape. For this plugin to successfully run, Cytoscape must be in your system `PATH`. The Cytoscape window will automatically open and the network will be displayed, as shown in Figure 5.3. From there, you can also import the table produced by Stage 6 to give each node a "Centrality" attribute. This can in turn be used to color nodes, as shown in Figure 5.4.

Program 16 Equivalent GML version of the CSV file for the correlation network, in Program 14.

```
graph [
node [
id 0
label ''Family.Porphyromonadaceae.0001''
]
node [
id 1
label ''Family.Porphyromonadaceae.0002''
]
...
edge [
source 0
target 1
weight 0.820245563984
]
edge [
source 0
target 2
weight 0.579995274544
]
...
```

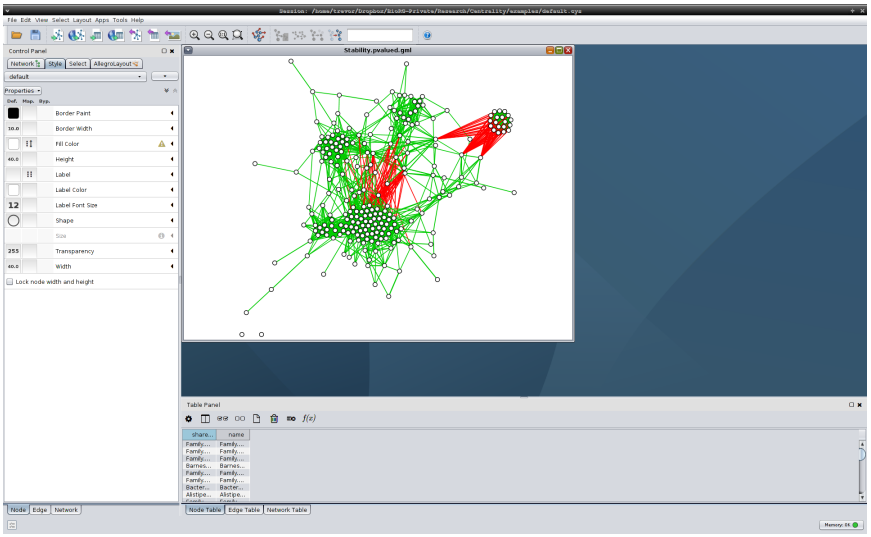

Figure 5.3: Co-occurence network visualized using Cytoscape, opened in stage 8 of our pipeline.

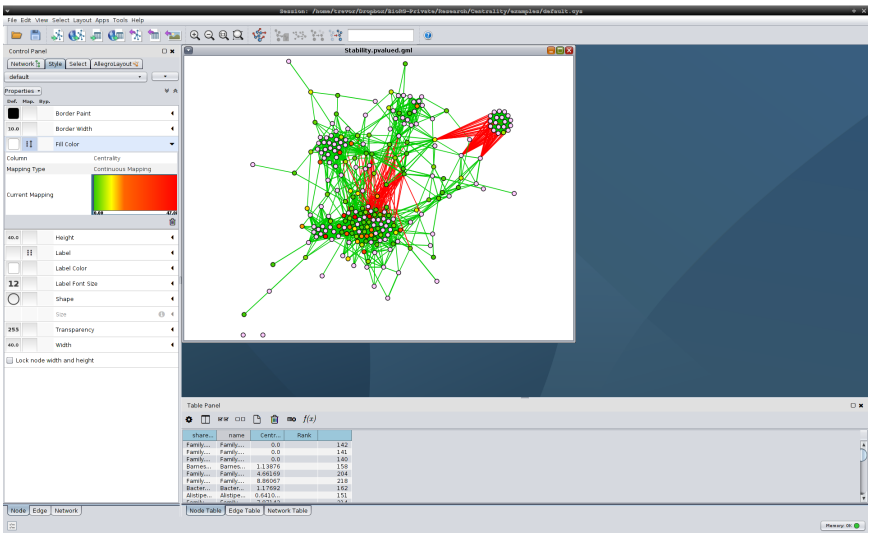

Figure 5.4: Stage 8 co-occurence network, colored by centrality value.

Chapter 6

The Future of PLUMA

Work is already underway to further expand PLUMA. We immediately plan to expand the Wrapper component of the software to include SWIG [6], which will allow compiled plugins to be accessible from scripted ones. This will facilitate compatibility across various programming languages, improving the ability to develop new ideas and reuse necessary existing components without additional overhead.

In addition we have developed an online plugin pool, and in the future would like members of the PLUMA user base will be able to deposit their plugins for other members to download and use in their pipelines. Achieving this will essential to our central purpose of PLUMA: we desire this to serve as a central portal for software pipelines, particularly those for various -omics analyses [55]. Additionally, the next release of PLUMA will hopefully include an easy plug-and-play graphical interface as a part of its User Layer, further improving ease of use. The program is currently being ported to Windows and will be released in the near future.

Appendix A

PLUMA Software License

A.1 Conditions and Regulations

PluMA GNU GENERAL PUBLIC LICENSE

This Software is copyrighted by Florida International University, whom grant you the right to use this software under the terms of the GNU General Public License. Please use the following citation in all research and development works which utilize this software:

T. Cickovski, V. Aguiar-Pulido, W. Huang, S. Mahmoud, and G. Narasimhan. 'Lightweight Microbiome Analysis Pipelines', Proceedings of International Work Conference on Bioinformatics and Biomedical Engineering (IWBBIO16), Granada, Spain, April 2016.

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS
0. Definitions.

This License refers to version 3 of the GNU General Public License.

Copyright also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

The Program refers to any copyrightable work licensed under this License. Each licensee is addressed as you. Licensees and recipients may be individuals or organizations.

To modify a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a modified version of the earlier work or a work based on the earlier work.

A covered work means either the unmodified Program or a work based on the Program.

To propagate a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To convey a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays Appropriate Legal Notices to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.
1. Source Code.

The source code for a work means the preferred form of the work for making modifications to it. Object code means any non-source form of a work.

A Standard Interface means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The System Libraries of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A Major Component, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The Corresponding Source for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.
2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.
3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.
4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.
5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to keep intact all notices.
c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License

will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an aggregate if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A User Product is either (1) a consumer product, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, normally used refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

Installation Information for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

Additional permissions are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose

on those licensors and authors.

All other non-permissive additional terms are considered further restrictions within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.
8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.
9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.
10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An entity transaction is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.
11. Patents.

A contributor is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's contributor version.

A contributor's essential patent claims are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, control includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a patent license is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To grant such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. Knowingly relying means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is discriminatory if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that

contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License or any later version applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the copyright line and a pointer to where the full notice is found.

```
    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
    <program>  Copyright (C) <year>  <name of author>

    This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an about box.

A.2 Contact Information

The best contact path for licensing issues is by e-mail to lead developer Trevor Cickovski at *tcickovs@fiu.edu* or send correspondence to:

PLUMA Team
c/o Prof. Giri Narasimhan
Bioinformatics Research Group (BioRG)
School of Engineering and Computer Science
Florida International University
11200 SW 8th Street Miami, FL 33199 USA

Bibliography

[1] Jennifer Ackerman. The ultimate social network. *Scientific American*, 306(6):36–43, 2012.

[2] V. Aguiar-Pulido, W. Huang, V. Ulloa-Suarez, T. Cickovski, K. Mathee, and G. Narasimhan. Metagenomics, metatranscriptomics and metabolomics approaches for microbiome analysis. *Evolutionary Biology*, 12(1):5–16, 2016.

[3] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Explained*. Addison-Wesley, 2001.

[4] Ery Arias-Castro, Guangliang Chen, and Gilad Lerman. Spectral clustering based on local linear approximations. *Electron. J. Statist.*, 5:1537–1587, 2011.

[5] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*, 2009.

[6] David M. Beazley. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[7] Michele Benzi and Christine Klymko. Total communicability as a centrality measure. *CoRR*, abs/1302.6770, 2013.

[8] J. Roger Bray and J. T. Curtis. An ordination of the upland forest communities of southern wisconsin. *Ecological Monographs*, 27(4):325–349, 1957.

[9] D. A. Carr. Seqnfind: Application of sequencing alignment techniques to CUDA GPU technologies. Presented at *NVIDIA GPU Technology Conference*, 2012.

[10] Anne Chao. Estimating the population size for capture-recapture data with unequal catchability. *Biometrics*, 43:783–791, 1987.

[11] T. Cickovski, V. Aguiar-Pulido, W. Huang, S. Mahmoud, and G. Narasimhan. Lightweight microbiome analysis pipelines. In *Proceedings of International Work Conference on Bioinformatics and Biomedical Engineering (IWBBIO16)*, 2016.

[12] T. Cickovski, V. Aguiar-Pulido, and G. Narasimhan. MATria: A unified centrality algorithm. submitted, 2016.

[13] T. Cickovski, E. Peake, V. Aguiar-Pulido, and G. Narasimhan. ATria: A novel centrality algorithm applied to biological networks. In *International Conference on Computational Advances in Bio and Medical Sciences*, ICCABS '15. IEEE, 2015.

[14] Trevor Cickovski. *Interacting Domain-Specific Languages with Biological Problem Solving Environments*. PhD thesis, University of Notre Dame, 2008.

[15] Barry Demchak, Tim Hull, Michael Reich, Ted Liefeld, Michael Smoot, Trey Ideker, and Jill P. Mesirov. Cytoscape: The network visualization tool for genomespace workflows. *F1000Research 2014*, 3:151–163, 2014.

[16] Janos Demeter, Catherine Beauheim, Jeremy Gollub, Tina Hernandez-Boussard, Heng Jin, Donald Maier, John C. Matese, Michael Nitzberg, Farrell Wymore, Zachariah K. Zachariah, Patrick O. Brown, Gavin Sherlock, and Catherine A. Ball. The Stanford Microarray Database: implementation of new analysis tools and open source release of software. *Nucleic acids research*, 35(Database issue), January 2007.

[17] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[18] K. Faust, J. F. Sathirapongsasuti, J. Izard, N. Segata, D. Gevers, J. Raes, and C. Huttenhower. Microbial co-occurrence relationships in the human microbiome. *PLoS Comput Biol*, 8(7):e1002606, 2012.

[19] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[20] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[21] E. Gallopoulos, E. Houstis, and J.R. Rice. Computer as thinker/doer: problem-solving environments for computational science. *Computational Science Engineering, IEEE*, 1(2):11–23, Summer 1994.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1 edition, 1994.

[23] A. Hagberg, D. Schult, and P. Swart. NetworkX Reference. Available at https://media.readthedocs.org/pdf/networkx/stable/networkx.pdf, 2016.

[24] Vaclav Havel. A remark on the existence of finite graphs. *Casopis pro pestovani matematiky*, 80:477–480, 1955.

[25] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, February 1912.

[26] D. D. Kang, J. Froula, R. Egan, and Z. Wang. Metabat, an efficient tool for accurately reconstructing single genomes from complex microbial communities. *PeerJ*, 3:e1165, 2015.

[27] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[28] M. G. Kendall. *Rank Correlation Methods*. Griffin, 4 edition, 1970.

[29] Pan-Jun Kim and Nathan D. Price. Genetic co-occurrence network across sequenced microbes. *PLoS Comput Biol*, 7(12):e1002340, 12 2011.

[30] Steven Knight. SCons design and implementation. In *Tenth International Python Conference*, 2002.

[31] J. J. Kozich, S. L. Westcott, N. T. Baxter, S. K. Highlander, and P. D. Schloss. Development of a dual-index sequencing strategy and curation pipeline for analyzing amplicon sequence data on the MiSeq Illumina sequencing platform. *Applied and Environmental Microbiology*, 79(17):5112–51120, 2013.

[32] S. Kulczynski. Die pflanzenassoziationen der pieninen. bulletin international de l'academie polonaise des sciences et des lettres. *Classe des Sciences Mathematiques at Naturelles*, B(1):57–203, 1927.

[33] G. N. Lance and W. T. Williams. Computer programs for hierarchical polythetic classification ('similarity analyses'). *The Computer Journal*, 9(1):60–64, May 1966.

[34] J. Luitjens and S. Rennich. CUDA warps and occupancy. GPU Technology Conference, 2011.

[35] Brian H. McArdle and Marti J. Anderson. Fitting multivariate models to community data: A comment on distance-based redundancy analysis. *Ecology*, 82(1):290–297, January 2001.

[36] B. N. Miller and D. L. Ranum. *Problem Solving with Algorithms and Data Structures Using Python*. Franklin-Beedle and Associates, 2013.

[37] M. Morisita. Measuring of the dispersion and analysis of distribution patterns. *Memoires of the Faculty of Science, Kyushu University*, 2:215–235., 1959.

[38] M. D. Mountford. An index of similarity and its application to classification problems. In P. W. Murphy, editor, *Progress in Soil Zoology*, pages 43–50. Butterworths, 1962.

[39] D. Musser. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley, 2001.

[40] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[41] Mark E. J. Newman and Aaron Clauset. Structure and inference in annotated networks. *CoRR*, abs/1507.04001, 2015.

[42] NVIDIA. CUDA-C programming guide. online, 2016.

[43] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, 2014.

[44] Oleg Konengs. Oleg Konengs on Github. online, 2013.

[45] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. *Technical Report*, 1999.

[46] Karl Pearson. Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 187:253–318, 1896.

[47] E. Pruesse, C. Quast, K. Knittel, B. M. Fuchs, W. Ludwig, J. Peplies, and F. O. Glockner. SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Res.*, 35(21):7188–7196, 2007.

[48] D. M. Raup and R. E. Crick. Measurement of faunal similarity in paleontology. *Journal of Paleontology*, 53(5):1213–1227, 1979.

[49] David N. Reshef, Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. Detecting novel associations in large data sets. *Science*, 334(6062):1518–1524, 2011.

[50] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe. Characterizing and measuring bias in sequence data. *Genome Biology*, 14:R51, 2013.

[51] Quansong Ruan, Debojyoti Dutta, Michael S. Schwalbach, Joshua A. Steele, Jed A. Fuhrman, and Fengzhu Sun. Local similarity analysis reveals unique associations among marine bacterioplankton species and environmental factors. *Bioinformatics*, 22(20):2532–2538, 2006.

[52] S. P. Sadedin, B. Pope, and A. Oshlack. Bpipe: A tool for running and managing bioinformatics pipelines. *Bioinformatics*, 28(11):1525–1526, 2012.

[53] Said E. Said and David A. Dickey. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika*, 71:599–607, 1984.

[54] P. D. Schloss, S. L. Westcott, T. Ryabin, J. R. Hall, M. Hartman, E. B. Hollister, R. A. Lesniewski, B. B. Oakley, D. H. Parks, C. J. Robinson, J. W. Sahl, B. Stres, G. G. Thallinger, D. J. Van Horn, and C. F. Weber. Introduction Mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Appl. Environ. Microbial.*, 75(23):7537–7541, 2009.

[55] N. Segata, D. Boernigen, T. L. Tickle, X. C. Morgan, W. S. Garrett, and C. Huttenhower. Computational metaomics for microbial community studies. *Molecular Systems Biology*, 9(1):666–680, 2013.

[56] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[57] Gbor J. Szekely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *Ann. Statist.*, 35(6):2769–2794, 12 2007.

[58] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[59] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[60] Bo Wang, Junjie Zhu, Emma Pierson, Daniele Ramazzotti, and Serafim Batzoglou. Visualization and analysis of single-cell rna-seq data by kernel-based similarity learning. *bioRxiv*, 2016.

[61] Sophie Weiss, Will Van Treuren, Catherine Lozupone, Karoline Faust, Jonathan Friedman, Ye Deng, Li Charlie Xia, Zhengjiang Zech Xu, Luke Ursell, Eric J. Alm, Amanda Birmingham, Jacob A. Cram, Jed A. Fuhrman, Jeroen Raes, Fengzhu Sun, Jizhong Zhou, and Rob Knight. Correlation detection strategies in microbial data sets vary widely in sensitivity and precision. *ISME Journal*, 10(1):1669–1681, 2016.

[62] Wenlei Xie, David Bindel, Alan Demers, and Johannes Gehrke. Edge-weighted personalized pagerank: Breaking a decade-old performance barrier. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1325–1334, New York, NY, USA, 2015. ACM.

[63] M. Yang and K. Wu. A similarity-based robust clustering method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(4):434–448, 2004.