Panakeia – User Guide

v. 2021-1

Sina Beier

September 15, 2021

Introduction

Panakeia provides a detailed view of the pangenome structure which can efficiently be utilised for discovery, or further in-depth analysis, of features of interest. It analyses synteny and multiple structural patterns of the pangenome, giving insights into the biological diversity and evolution of the studied taxon. Panakeia provides both broad and detailed information on the structure of a pangenome, for diverse and highly clonal populations of bacteria.

Contents

1 Overview

Panakeia is a tool which strives to be easy to use and providing a detailed view of the pangenome structure which can efficiently be utilised for discovery, or further in-depth analysis, of features of interest. It analyses synteny and multiple structural patterns of the pangenome, giving insights into the biological diversity and evolution of the studied taxon. Panakeia hence provides both broad and detailed information on the structure of a pangenome, for diverse and highly clonal populations of bacteria. Development of new pan-genome analysis tools is important, as the pangenome of a microbial species has become an important method to define the diversity of a selected taxon, most commonly a species, in the last years. This enables comparison of strains from different ecological niches and can be used to define the functional potential in a bacterial population. It gives us a much better view of microbial genomics than can be gained from singular genomes which after all are just single representatives of a much more varied population.
Previously published pangenome tools often reduce the information to a presence/absence matrix of unconnected genes or generate massive hard to interpret output graphs. However, Panakeia includes synteny and structural information and presents it in a way that can readily be used for further analysis. Basic post-processing scripts are also available.

References provided in this User Guide are URLs to make it easier to find the tools and dependancies. For more detailed references, please see the Panakeia paper on BioarXiv [1].

2 Using Panakeia

2.1 Before running Panakeia

2.1.1 Prerequisites

Panakeia is written in Python3 and uses the NetworkX [2] and library [version 2.4 and above] for graph generation. A few of the postprocessing scripts also require Matplotlib [3] The clustering step uses CD-HIT [4].

2.1.2 Input data

To run Panakeia, a set of input genomes with gene annotations is needed. The files necessary to run Panakeia are

- a GFF3 file for each of the input genomes (all of these saved in one directory)
- a multi-fastA file consisting of all protein sequences from the input genomes (one file where all proteins from these genomes are found, if necessary separate files from each genome have to be concatenated into one file

The headers of the protein multi-fastA must contain the locus_tag as found in the matching GFF3 file before the first space. Functional annotation and gene names will be extracted from the GFF3 files if available. If only sequence information is available, Prokka [5] can be used to functionally annotate the sequences. It will create both the GFF3 and protein sequence files necessary. The protein sequence files will have to be concatenated to be used in the clustering step of Panakeia.

2.2 Clustering

The first step in a Panakeia analysis is to cluster all proteins found in the pangenome int groups of similar proteins.
The clustering script `Clustering.py` requires a FastA file including all protein sequences from any used input genome. If Prokka [5] was used to annotate the input genomes, this means concatenating all of the *.faa files that Prokka has generated into one file. Further, the clustering script requires the number of input genomes.

```
Usage: Clustering.py [-h] infasta count

positional arguments:
  infasta               Input file of protein sequences
  count                 Number of sequence origins (strains)

optional arguments:
  -h, --help            show this help message and exit
```

Clustering is done using CD-HIT [4]. The clustering step is run iteratively with increasingly relaxed parameters to generate as many clusters as possible which include proteins from all input genomes, as we expect most proteins to be part of the hard core in the pangenome. Iterative clustering allows for different levels of variability in the protein sequences, going from 90% to 80% and finally 75%. Clusters including a number of proteins that is at least the number of input genomes are kept and the included proteins removed from the next clustering iteration. Only in the last iteration all clusters are kept.

Clustering returns intermediate files for each iteration and two important output files: `protein_clusters.txt` which includes the final protein clusters used for the analysis and `representative_seq.faa` which includes one representative protein sequence and functional annotation for each cluster. These two files will serve as input for the next step. Running `Clustering.py` on proteins from 42 *Streptococcus equi* input genomes on a M1 Mac, MacOS BigSur and a CD-HIT installation without OpenMP finishes in less than five seconds.

2.3 Pangenome graph generation

The main step of the Panakeia pipeline is in the `Panakeia.py` script. Here strain graphs from all input genomes and one pangenome graph for the full pangenome are generated. Input for this step is the path to the output directory, the input directory including all GFF3 files,and the `protein_clusters.txt` and `representative_seq.faa` files generated from the clustering step. Additionaly, parameters can be set to define the hard core, soft core and shell percentages, which are set by default to the most commonly used values for Pangenome analysis on one species. The `weightcutoff` parameter is an integer cutoff for the minimal weight of an edge to be drawn, meaning the minimal number of strains this neighborhood relation has to occur in for it to be accepted into the pangenome graph. This can be used when a large input set is available, as it will then exclude rare conncetions. Connections occurring only in one of many input genomes are likely to be caused by misassembly or annotation errors and their influence on the further analysis can be reduced this way.

```
Usage: Panakeia.py [-h] [-hardcore HARDCORE] [-softcore SOFTCORE]
                   [-shell SHELL] [-weightcutoff WEIGHTCUTOFF]
                   output gffs clust rep

positional arguments:
  output                Output directory
  gffs                  Directory with input gffs
  clust                 File of clustered proteins
  rep                   File of (annotated) representative sequences

optional arguments:
  -h, --help            show this help message and exit
  -hardcore HARDCORE    Percentage of genomes to count as hard core,
                        defaults to 0.99
  -softcore SOFTCORE    Percentage of genomes to count as soft core,
                        defaults to 0.95
  -shell SHELL          Percentage of genomes to count as shell,
                        defaults to 0.15, everything less is cloud
  -weightcutoff WEIGHTCUTOFF
                        Integer cutoff for minimal weight of an edge
                        to be drawn
```

Running `Panakeia.py` on 42 *Streptococcus equi* input genomes on a M1 Mac, MacOS BigSur finishes in less than three minutes.

2.4 Highlighting and Strain graphs

Strain graphs are the graphs generated for each input genome. A strain graph includes all proteins of the genome annotation as nodes and neighborhood and paralogy relations as edges. Edges marked as "local" are added between each pair of proteins which are direct neighbors on the respective input sequence. A protein node can have maximally two direct neighbors (its local predecessor and successor on the sequence). Protein nodes can have any number of edges marked as "paralogs", which denote a paralogy relationship of the protein sequences. Paralogous proteins are not necessarily local neighbors (although they can be), but as they indicate a high similarity of the underlying DNA sequences they can be locations of assembly errors, hence their local neighbors could be incorrect.
A sequence of proteins connected by local edges which is matched by a sequence of paralogous proteins which have the same order in a different part of the genome can also hint at a repeat sequence which can either have biological relevance or have been caused by misassembly, as is shown in Figure 1.

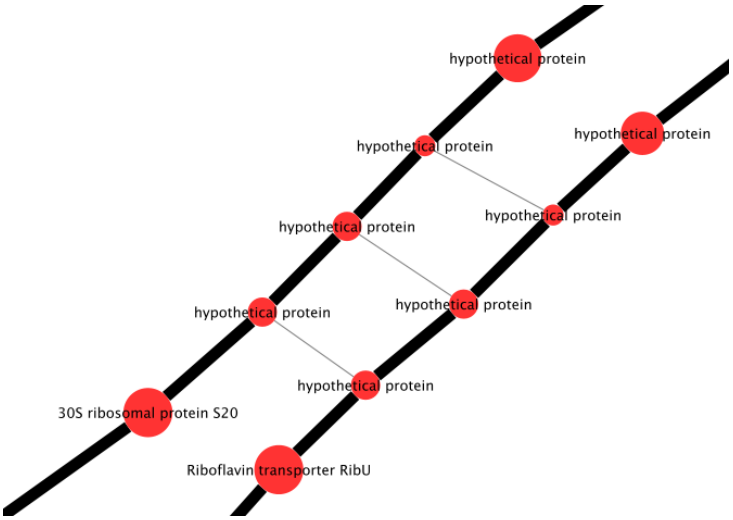

Figure 1: Closeup of a sequence of proteins (red nodes) in synteny with a sequence of paralogous proteins. Paralogous relationship is marked by grey, thin edges; local neighbors are connected by black, wide edges.
As the paralogs could not be functionally annotated and the annotated neighbors are not hinting at a specific biological mechanism for the repeated sequence, this could be either a real sequence of paralogs or an assembly error.

2.4.1 Highlighting groups of genomes

If a set of input genomes should be highlighted in a visualisation, users can add an overlay feature for this group to the pangenome graph. As input a list of the selected strain graph files (ideogenomes) (one file per line) has to be provided in a text file.

```
Usage: HighlightStrains.py [-h] pangenome ideogenomes highlighted

positional arguments:
  pangenome    Pangenome GraphML
  ideogenomes  File with list of Ideogenome GraphMLs (one per line)
  highlighted  Highlighted GraphML

optional arguments:
  -h, --help   show this help message and exit
```

This will create a second version of the pangenome graph with the overlay feature added which can be used in visualisation. The overlay is gray for nodes (protein clusters) and edges (local neighborhood relation) that are not included in the input set, orange for nodes end edges which are included and yellow for nodes and edges which are in the highlighted genomes, but had been filtered out from the full pangenome graph though setting filtering parameters in `Panakeia.py`, respectively.

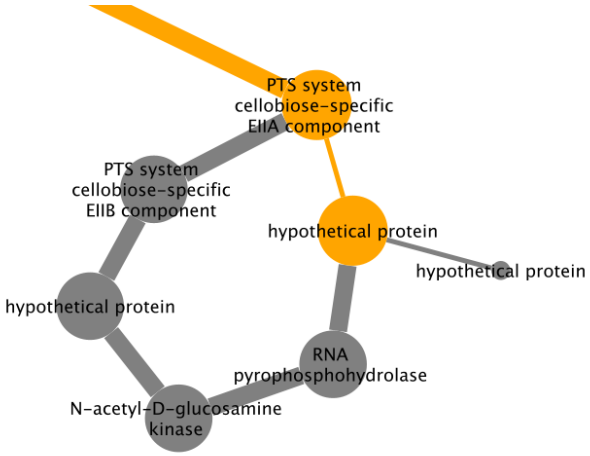

Figure 2: Closeup of a deletion of multiple proteins in two strains highlighted on the pangenome. The orange nodes and edges represent proteins and local neighborhood in the highlighted input genomes, while grey nodes and edges are part of the pangenome, but not this particular subset.Parts of the PTS system have been deleted in these geneomes (*S. equi* from Argentina).

2.5 Chromosomizing graphs

Bacterial genomes can have multiple chromosomes (like *Vibrio* do) or carry plasmids. If there are finished or high quality draft genomes available which contain complete sequences of the chromosomes and or plasmids, these can be used to predict the location of genes in less complete assemblies. To do that, the finished genomes (templates) have to be added to the Panakeia analysis and the location of the strain graphs for these genomes have to be written into a text file (one path per line). This file can then be used to determine chromosomal location for the protein clusters found in the selected template genomes. The more genomes are available, the more proteins will be covered and the more accurate they will be.
The number of chromosomes and plasmids is determined by the number of connected components found in the first file of the list of selected genomes. If it is necessary to have multiple plasmids separated out, an artificial "genome" including the chromosome and each of the plasmids would have to be created. If it is just required to detect if a protein is chromosomal or not, multiple genomes carrying different plasmids can be added.
Protein clusters which always occur in the biggest connected component of the given strain graphs will be assigned to the chromosome (or chromosome 1), clusters always found in the next connected component will be assigned as plasmid (or chromosome 2). If a cluster is found on different connected components it will be marked as "undecided".
Protein clusters that are not included in any of the templates it will be annotated as "unknown" chromosomal location.

2.5.1 Chromosomizing the pangenome graph

To add the predicted chromosomal location onto the Pangenome graph,
`ChromosomizePangenome.py` can be used. It requires a path to the output GraphML file, the `protein_clusters.txt`, the directory including all straingraphs and a text file including a list of filenames in that directory denoting the finished genome templates to be used to

define the chromosomal location for the protein clusters.

```
usage: ChromosomizePangenome.py [-h] pangenome
                                 outfile clusters straingraphs template

positional arguments:
  pangenome     Pangenome GraphML
  outfile       Chromosomized pangenome GraphML
  clusters      Cluster file from Panakeia
  straingraphs  Directory including the template straingraphs
  template      File with names of templates in straingraph directory

optional arguments:
  -h, --help    show this help message and exit
```

2.5.2 Chromosomizing strain graphs

The respective script to add the same information to each strain graph is the `Chrmosomize.py` script. It needs similar input to `ChromosomizePangenome.py`, but instead of one output file it will generate new graphs for each input genome and save them in the given output directory.

```
usage: Chromosomize.py [-h] straingraphs outdir clusters template

positional arguments:
  straingraphs  Directory with Straingraph GraphMLs
  outdir        Directory with Chromosomized Straingraph GraphMLs
  clusters      Cluster file from Panakeia
  template      File with names of templates in straingraph directory

optional arguments:
  -h, --help    show this help message and exit
```

An example of chromosomized pangenome and strain graphs will be added to this guide in the future.

2.6 Pattern analysis

TODO

```
usage: Patterns.py [-h] pangenome directory patterns

positional arguments:
  pangenome   Pangenome GraphML
  directory   Output directory
  patterns    Comma separated list of pattern types to search for

optional arguments:
  -h, --help  show this help message and exit
```

Running `Patterns.py` on the pangenome generated from 42 *Streptococcus equi* input genomes detecting all patterns on a M1 Mac, MacOS BigSur finishes in less than a minute.

3 Panakeia Output

3.1 Output files and formats

TODO

3.2 Visualisation

All graphs generated by Panakeia are provided in GraphML format, which can be read by most network visualisation tools. Visualisations for this user guide and the Panakeia paper have been done using Cytoscape[6]. The layout used is usually the yWorks Organic Layout algorithm [7] and Cytoscape style XML files for strain graphs and the pangenome graph – both with and without chromosomization – are available in the Panakeia GitHub repository.

4 Panakeia and Pantagruel

Panakeia and Pantagruel [8] output can be connected by using the scripts in the `PantagruelConnect` directory. This includes four scripts to be used to generate statistics from Pantagruel output and connect the predicted clades and their specific proteins with the pangenome graph. A detailed description of this will be part of a publication which is currently in preparation and the user guide will be updated with examples as soon as this is published. The four scripts are shortly described below. The first step is to add the Pantagruel output to the pangenome graph. The necessary Pantagruel output files are the clade file and the clade specific genes file as well as a mapping file which matches the clusterIDs used by Panakeia to the orthologous groups used by Pantagruel.
To reduce the runtime, in case a only a specific set of clades has to be considered but the full analysis has been run already, an existing pangenome file including the clade information can be loaded to add new clade specific genes.

```
usage: MatchClades.py [-h] [-cladepan CLADEPAN]
                      pangenome outdir clusters map clades spec

positional arguments:
  pangenome           Pangenome GraphML
  outdir              Output directory
  clusters            Cluster file from Panakeia
  map                 Mapping file for clusterID to Pantagruel orthologous
                      groups
  clades              Clade file
  spec                Clade specific genes file

optional arguments:
  -h, --help          show this help message and exit
  -cladepan CLADEPAN  Exisiting cladified pangenome file
```

The output directory will include a pangenome grapML with clade information and a set of clade graphs, which are subgraphs of the pangenome for each clade (a clade specific pangenome, so to speak).

The `MarkClade.py` script will highlight a clade after the clade information has been added to the pangenome graph.

```
Usage: MarkClade.py [-h] pangenome outfile clade

positional arguments:
  pangenome   Pangenome GraphML
  outfile     Highlighted pangenome GraphML
  clade       CladeID

optional arguments:
  -h, --help  show this help message and exit
```

To generate simple statistics on the Pantagruel clades, `CladeSize.py` can be run.

```
usage: CladeSize.py [-h] clades out

positional arguments:
  clades      Clade definition
  out         Output table

optional arguments:
  -h, --help  show this help message and exit
```

The last script in the set is `TabularCladegraphs.py` which will generate tabular information from each clade graph generated by `MatchClades.py`

```
usage: TabularCladegraphs.py [-h] indir

positional arguments:
  indir       Cladegraph Directory

optional arguments:
  -h, --help  show this help message and exit
```

5 Examples

Examples will be added soon, including a step-by-step example with an example dataset provided in the Panakeia GitHub.

References

[1] https://www.biorxiv.org/content/10.1101/2021.03.02.433540v1

[2] https://networkx.org/

[3] https://matplotlib.org/

[4] http://weizhong-lab.ucsd.edu/cd-hit/

[5] https://github.com/tseemann/prokka

[6] https://cytoscape.org/

[7] https://www.yworks.com/products/yfiles-layout-algorithms-for-cytoscape

[8] https://github.com/flass/pantagruel