PanTools

Release 3.4.0

Sandra Smit

Dec 01, 2022

CONTENTS

PanTools is a toolkit for comparative analysis of large number of genomes. It is developed in the Bioinformatics Group of Wageningen University, the Netherlands. Please cite the relevant publication(s) from the list of publications if you use PanTools in your research.

CHAPTER ONE

LICENCE

PanTools has been licensed under GNU GENERAL PUBLIC LICENSE version 3.

CHAPTER

TWO

PUBLICATIONS

- PanTools: representation, storage and exploration of pan-genomic data.
- Efficient inference of homologs in large eukaryotic pan-proteomes
- Pan-genomic read mapping
- The Pectobacterium pangenome, with a focus on Pectobacterium brasiliense, shows a robust core and extensive exchange of genes from a shared gene pool
- Pantools v3: functional annotation, classification, and phylogenomics

CHAPTER

THREE

FUNCTIONALITIES

PanTools currently provides these functionalities:

- Construction of a panproteome
- Adding new genomes to the pangenome
- Adding structural/functional annotations to the genomes
- Detecting homology groups based on similarity of proteins
- Optimization of homology grouping using BUSCO
- Read mapping
- Gene classification
- Phylogenetic methods

CHAPTER

FOUR

REQUIREMENTS

- **Java Virtual Machine** version 1.8 or higher, Add path to the java executable to your OS path environment variable.
- **KMC**: A disk-based k-mer counter, After downloading the appropriate version (linux, macos or windows), add path to the *kmc* and *kmc_tools* executables to your OS path environment variable.
- **MCL**: The Markov Clustering Algorithm, After downloading and compiling the software, add path to the *mcl* executable to your OS path environment variable.

For installing and configuring all required software, please see our *Installing and configuring the required software* page.

CHAPTER

FIVE

RUNNING THE PROGRAM

Add the path to the java archive of PanTools, located in the *pantools/target* subdirectory, to the OS path environment variable. Then run PanTools from the command line by:

```
$ java <JVM options> -jar pantools-3.4.0.jar <subcommand> <arguments>
```

Useful JVM options
- **-server** : To optimize JIT compilations for higher performance
- **-Xmn(a number followed by m/g)** : Minimum heap size in mega/giga bytes
- **-Xmx(a number followed by m/g)** : Maximum heap size in mega/giga bytes

CHAPTER

SIX

CONTENTS

6.1 Installing and configuring the required software

1. *Download PanTools*
2. *Install Neo4j*
3. *Install dependencies, either manually or through conda.*

For PanTools developers:

4. *Installing pre-commit hooks*

6.1.1 Download PanTools

The preferred option is to download the .jar file from https://git.wur.nl/bioinformatics/pantools/-/releases and put it in a directory named "pantools/target".

Alternatively, follow the installation and compilation instructions from the README.md file in the desired version (e.g. https://git.wur.nl/bioinformatics/pantools/-/tree/v3.4.0).

Test if PanTools is executable:

```
$ java -jar /YOUR_FULL_PATH/pantools/target/pantools-3.4.0.jar
```

If the help page does not appear this (likely) means you don't have a properly working Java version 8. Java is included in the PanTools conda environment, please consider to first install the environment. To manually download Java, follow the instructions at https://www.java.com/en/download.

Set PanTools alias

To avoid typing long command line arguments every time, we suggest setting an alias to your profile. Set an alias in your ~/.bashrc using the following command. Always include the **full** path to PanTools' .jar file.

If Java is set to your $PATH.

```
$ echo "alias pantools='java -Xms20g -Xmx50g -jar /YOUR_FULL_PATH/pantools/target/
↪pantools-3.4.0.jar'" >> ~/.bashrc
```

If Java is not set to your $PATH, include the **full** path in the alias. Replace 'YOUR_PATH' 2x with the correct directory structure.

```
$ echo "alias pantools='/YOUR_PATH/jdk1.8.0_161/bin/java -Xms20g -Xmx50g -jar /YOUR_PATH/
↪pantools/target/pantools-3.4.0.jar'" >> ~/.bashrc
```

Source your profile and test if the alias works.

```
$ source ~/.bashrc
pantools version
```

6.1.2 Install Neo4j

Although Neo4j is not needed for any of the PanTools functionalities, it is required to be able to start up a database and use cypher queries. In the PanTools versions up to 3.2 we use Neo4j 3.5.3 libraries, whereas newer releases use Neo4j 3.5.30. Neo4j version 3.5.30 is compatible with all earlier PanTools versions.

Download the Neo4j 3.5.30 community edition from the Neo4j website or download the binaries directly from our server.

```
$ wget http://www.bioinformatics.nl/pangenomics/tutorial/neo4j-community-3.5.30-unix.tar.
↪gz
$ tar -xvzf neo4j-community-*

# Edit your ~/.bashrc to include Neo4j to your $PATH
$ echo "export PATH=/YOUR_PATH/neo4j-community-3.5.30/bin:\$PATH" >> ~/.bashrc #replace␣
↪YOUR_PATH with the correct path on your computer
$ source ~/.bashrc
$ neo4j status # test if Neo4j is executable
```

Official Neo4j 3.5 manual: https://neo4j.com/docs/operations-manual/3.5/

6.1.3 Dependencies

Some of PanTools functionalities require additional software to be installed. Installing every dependency will take a considerate amount of time, therefore we highly recommend to use Mamba. Mamba efficiently manages Conda environments allowing the installation of all required tools into a separate environment. Instructions for creating the Mamba environment or installing the tools manually are found in the sections below.

Install dependencies using Conda

Instructions on how to install and use conda can be found in the **conda manual page**. Once conda is installed, we suggest to install Mamba into the Conda base environment to enable much faster dependency solving.

To install every dependency, download **pantools.yaml** and include it in the installation command.

```
$ wget http://www.bioinformatics.nl/pangenomics/manual/pantools.yaml
$ conda install mamba -n base -c conda-forge
$ mamba env create -n pantools --file pantools.yaml

$ conda activate pantools # activate the environment before using PanTools
$ conda deactivate # deactivate when you are done
```

Run the following commands when you do not want to install every dependency, but only specific ones for the analysis that you're interested in.

```
$ conda create -n pantools python=3.6 kmc=3.0 mcl # Creates an environment that is able␣
↪to construct the pangenome and cluster protein sequences
$ conda install -n pantools mafft iqtree fasttree blast mash fastani busco=5.2.2 r-
↪ggplot2 r-ape graphviz # include tools you want to install via conda
```

Manual installation of dependencies

All tools must be set to your $PATH so PanTools is able to use them on any location. The instructions below are based on a linux machine.

Install KMC

PanTools requires **KMC v2.3** or **3.0** for k-mer counting during the constructing of the pangenome graph. KMC v3.0 is fastest, but v2.3 should also be compatible with PanTools. The KMC3 binaries can be downloaded from https://github.com/refresh-bio/KMC/releases.

```
$ tar -xvzf KMC* #uncompress the KMC binaries

# Edit your ~/.bashrc to include KMC to your PATH
$ echo "export PATH=/YOUR_PATH/KMC/:\$PATH" >> ~/.bashrc #replace YOUR_PATH with the␣
↪correct path on your computer
$ source ~/.bashrc
$ kmc # test if KMC is executable
$ kmc_tools # test if kmc_tools is executable
```

Install MCL

The MCL (Markov clustering) algorithm is required for the homology grouping of PanTools. The software can be found on https://micans.org/mcl under License & software.

```
$ wget https://micans.org/mcl/src/mcl-14-137.tar.gz
$ tar -xvzf mcl-*
$ cd mcl-14-137
$ ./configure --prefix=/YOUR_PATH/mcl-14-137/shared #replace YOUR_PATH with the correct␣
↪path on your computer
$ make install

# Edit your ~/.bashrc to include MCL to your PATH
$ echo "export PATH=/YOUR_PATH/mcl-14-137/bin/:\$PATH" >> ~/.bashrc #replace YOUR_PATH␣
↪with the correct path on your computer
$ source ~/.bashrc
$ mcl -h # test if MCL is executable
```

Install BUSCO

BUSCO v3 to v5 can be run against the pangenome to estimate annotation completeness. The versions require a different Python release and need to be installed in a different way. We suggest to install BUSCO v5, follow the instructions at https://gitlab.com/ezlab/busco/.

Install FastTree

FastTree is used to infer approximately-maximum-likelihood phylogenetic trees from the alignments of nucleotide or protein sequences which are extracted from the pangenome. An executable can be found on the FastTree website: http://www.microbesonline.org/fasttree/.

```
$ wget http://www.microbesonline.org/fasttree/FastTree
$ chmod +x FastTree
$ ./FastTree # test if FastTree is executable

# Edit your ~/.bashrc to include FastTree to your PATH
$ echo "export PATH=/YOUR_PATH:\$PATH" >> ~/.bashrc #replace YOUR_PATH with the correct path on your computer
$ source ~/.bashrc
```

Install R

R and some additional R packages are required to execute R scripts (files with .R extension) that create plots and construct Neighbor-Joining phylogenies. In most cases, R is already installed on a server. If this is not the case, install it through the instructions on the website https://cran.r-project.org/, or compile it by using following steps.

```
mkdir R
mkdir R/R_LIBS
cd R
wget https://cran.r-project.org/src/base/R-4/R-4.0.2.tar.gz #version number might have changed already
tar -xvf R-4.0.2.tar.gz
cd R-4.0.2/
./configure --prefix=/YOUR_PATH/R/  #replace YOUR_PATH with the correct path on your computer
make

# Edit your ~/.bashrc to include R to your PATH
$ echo "export PATH=/YOUR_PATH/R/bin/:\$PATH" >> ~/.bashrc #replace YOUR_PATH with the correct path on your computer
$ source ~/.bashrc
$ R --help # test if R is executable
```

When **R_LIB** is set to your $PATH, R scripts know the location of the libraries and are able to install additional R packages to the selected directory.

```
$ echo "R_LIBS=/YOUR_PATH/R/R_LIBS/" >> ~/.bashrc
$ echo "export R_LIBS" >> ~/.bashrc
$ echo $R_LIBS # validate if the path to the R libraries can be found
```

Install MAFFT

MAFFT is required for all the alignment functionalities, such as the alignment of homology groups and inferring the core SNP phylogeny. The full manual is available at https://mafft.cbrc.jp/alignment/software/.

```
$ git clone https://github.com/GSLBiotech/mafft.git
$ cd mafft/core

# Edit the first line of Makefile to change the desired install location, from 'PREFIX = /
↪usr/local' to 'PREFIX = /YOUR_DESIRED_PATH/mafft/'
# Make sure the 'ENABLE_MULTITHREAD = -Denablemultithread' line is uncommented, to enable␣
↪multithreading

# Edit your ~/.bashrc to include MAFFT to your $PATH
$ echo "export PATH=/YOUR_PATH/mafft/bin/:\$PATH" >> ~/.bashrc #replace YOUR_PATH with␣
↪the correct path on your computer
$ source ~/.bashrc
$ mafft --help # test if MAFFT is executable
```

Install IQ-tree

Using IQ-tree we infer phylogenetic trees by maximum likelihood. Information about the tool can found on their webpage https://github.com/ebi-pf-team/interproscan/wiki/HowToDownload

```
wget https://github.com/Cibiv/IQ-TREE/releases/download/v1.6.12/iqtree-1.6.12-Linux.tar.
↪gz
tar -xvf iqtree-1.6.12-Linux

# Edit your ~/.bashrc to include IQ-tree to your $PATH
$ echo "export PATH=/YOUR_PATH/iqtree-1.6.12-Linux/bin/:\$PATH" >> ~/.bashrc #replace␣
↪YOUR_PATH with the correct path on your computer
$ source ~/.bashrc
$ iqtree -h # test if IQ-tree is executable
```

Install fastANI or MASH

To be able to construct a Neighbor-Joining phylogeny using ANI-scores, either **fastANI** or **MASH** is required. The manual for **fastANI** is available at https://github.com/ParBLiSS/FastANI/. The manual for **MASH** can be found at https://mash.readthedocs.io/en/latest/.

```
$ wget https://github.com/marbl/Mash/releases/download/v2.2/mash-Linux64-v2.2.tar
$ tar -xvf mash-Linux64-v2.2.tar
$ mv mash-Linux64-v2.2/mash .

$ wget https://github.com/ParBLiSS/FastANI/releases/download/v1.32/fastANI-Linux64-v1.32.
↪zip #
$ unzip fastANI-Linux64-v1.32.zip

# Edit your ~/.bashrc to include MASH and FastANI to your $PATH
$ echo "export PATH=/YOUR_PATH/:\$PATH" >> ~/.bashrc #replace YOUR_PATH with the correct␣
↪path on your computer
$ source ~/.bashrc
$ mash -h # test if MASH is executable
$ fastANI -h # test if FastANI is executable
```

Install BLAST

BLAST is only required by one function, where the sequences are blasted against a database to obtain their COG category. Information about BLAST can be found at https://www.ncbi.nlm.nih.gov/books/NBK279690/?report=classic.

```
$ wget https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.10.1+-
↪x64-linux.tar.gz
$ tar -xvf ncbi-blast-2.10.1+-x64-linux.tar.gz

# Edit your ~/.bashrc to include BLAST to your $PATH
$ echo "export PATH=/YOUR_PATH/ncbi-blast-2.10.1+/bin/:\$PATH" >> ~/.bashrc #replace␣
↪YOUR_PATH with the correct path on your computer
$ source ~/.bashrc
$ blastp -help # test if BLAST is executable
```

Install InterProScan

Not required by any function, but the .GFF3 output of **InterProScan** can be read to include functional annotations to the database. The installation itself can be quite tricky as it uses many different third-party binaries and each having their own dependencies. Please check https://github.com/ebi-pf-team/interproscan/wiki/HowToDownload and take a look at the install requirements as well. Installation of the Panther models is not required.

Phobius via InterProScan

Phobius predictions can be performed during the InterProScan analysis but it is not part of the standard set of predictions. To allow these predictions, https://phobius.sbc.su.se/, place the entire directory in the InterProScan/bin/ directory and edit the **interproscan.properties** configuration file. More information about including Phobius into the InterProScan analysis is found at https://interproscan-docs.readthedocs.io/en/latest/ActivatingLicensedAnalyses.html.

Install eggNOGmapper

Not required by any function, but the .annotations output of **eggNOG-mapper** can be read to include functional annotations to the database. Information about this tool can be found on http://eggnog-mapper.embl.de/

```
git clone https://github.com/eggnogdb/eggnog-mapper.git
```

6.1.4 Installing pre-commit hooks

First install the pre-commit Python package by following the installation instructions.

Then, inside the root directory of the repository, run:

```
pre-commit install
```

This step you will need to run only once after cloning the repository. The hooks will be installed in your local repository's configuration under `.git/hooks/pre-commit`.

After installation of the hooks they will be triggered at each commit if any Java files have changed. Should any of the pre-commit hooks fail, git will not allow you to create the commit. The output of the pre-commit hooks should tell you what failed, allowing you to fix any problems and to re-add the affected files for another commit attempt.

Pre-commit hooks can be run manually as well with:

```
pre-commit run
```

6.2 Construct pangenome

6.2.1 Build pangenome

Build a pangenome out of a set of genomes.

Required software

KMC 2.3 or 3.0

Required arguments

`--database-path/-dp` Path to the pangenome database.

`--genomes-file/-gf` A text file containing paths to FASTA files of genomes to be added to the pangenome; each on a separate line.

Optional arguments

`--kmer-size/-ks` Size of *k*-mers, allowed to be 6 <= K_SIZE <= 255. By not giving this argument, the most optimal *k*-mer size is calculated automatically.

Example input file

```
/always/genome1.fasta
/use_the/genome2.fasta
/full_path/genome3.fasta
```

Example command

```
$ pantools build_pangenome -dp tomato_DB -gf tomato_3.txt
```

Relevant literature

- PanTools: representation, storage and exploration of pan-genomic data

6.2.2 Add annotations

Construct or expand the annotation layer of an existing pangenome. The layer consists of genomic features like genes, mRNAs, proteins, tRNAs etc. PanTools is only able to read General Feature Format (**GFF**) files.

Multiple annotations can be assigned to a single genome; however, only one annotation a time can be included in an analysis. The most recently included annotation of a genome is included as default, unless a different annotation is specified via `--annotations-file`, see the explanation *below*

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--annotations-file`/`-af` A text file with on each line a genome number and the full path to the corresponding annotation file, separated by a space.

Optional arguments

`--connect-annotations`/`-ca` Connect the annotated genomic features to nucleotide nodes in the DBG.

Example command

```
$ pantools add_annotations -dp tomato_DB -af annotations.txt
```

Output

The annotated features are incorporated in the graph. Output files are written to the database directory.

- **annotation_overview.txt**, a summary of the GFF files incorporated in the pangenome
- **annotation.log**, a list of misannotated feature identifiers.

Example input file

Each line of the file starts with the genome number followed by the full path to the annotation file. The genome numbers match the line number of the file that you used to construct the pangenome.

```
1 /always/genome1.gff
2 /use_the/genome2.gff
3 /full_path/genome3.gff
```

GFF3 file format

The GFF format consists of one line per feature, each containing 9 columns of data, plus optional track definition lines, that must be tab separated. Please use the proper hierarchy for the feature: **gene** -> **mRNA** -> **CDS**. Where *gene* is the parent of *mRNA* and *mRNA* is the parent of the *CDS* feature. When a *gene* consists of multiple *CDS* features but is missing *mRNA*, only the last *CDS* feature is annotated in the pangenome. The following example from *Saccharomyces cerevisiae* YJM320 (GCA_000975885) displays a correctly formatted gene entry:

```
CP004621.1      Genbank gene    44836   45753   .       -       .       ID=gene99;
↪Name=RPL23A;end_range=45753,.;gbkey=Gene;gene=RPL23A;gene_biotype=protein_coding;locus_
↪tag=H754_YJM320B00023;partial=true;start_range=.,44836
CP004621.1      Genbank mRNA    44836   45753   .       -       .       ID=rna99;
↪Parent=gene99;gbkey=mRNA;gene=RPL23A;product=Rpl23ap
CP004621.1      Genbank exon    45712   45753   .       -       .       ID=id112;
↪Parent=rna99;gbkey=mRNA;gene=RPL23A;product=Rpl23ap
CP004621.1      Genbank exon    44836   45207   .       -       .       ID=id113;
↪Parent=rna99;gbkey=mRNA;gene=RPL23A;product=Rpl23ap
```

(continues on next page)

(continued from previous page)

```
CP004621.1      Genbank CDS     45712   45753   .       -       0       ID=cds92;
↪Parent=rna99;Dbxref=SGD:S000000183,NCBI_GP:AJQ01854.1;Name=AJQ01854.1;Note=corresponds␣
↪to s288c YBL087C;gbkey=CDS;gene=RPL23A;product=Rpl23ap;protein_id=AJQ01854.1
CP004621.1      Genbank CDS     44836   45207   .       -       0       ID=cds92;
↪Parent=rna99;Dbxref=SGD:S000000183,NCBI_GP:AJQ01854.1;Name=AJQ01854.1;Note=corresponds␣
↪to s288c YBL087C;gbkey=CDS;gene=RPL23A;product=Rpl23ap;protein_id=AJQ01854.1
```

Select specific annotations for analysis

Only **one** annotation per genome is considered by any PanTools functionality. When multiple annotations are included, the last added annotation of a genome is automatically selected unless an `--annotations-file` is included specifying which annotations to use. This annotation file contains only annotation identifiers, each on a separate line. The most recent annotation is used for genomes where no annotation number is specified in the file. Below is an example where the third annotation of genome 1 is selected and the second annotation of genome 2 and 3.

```
1_3
2_2
3_2
```

6.2.3 Grouping proteins

Group

Generate homology groups based on similarity of protein sequences. The resulting homology groups connect similar sequences in the pangenome database. Homology groups contain not only orthologous pairs, but also pairs of homologs duplicated after the speciation of the two species, so-called in-paralogs. The sizes of the groups are controlled by the `--relaxation` parameter that can be set very strict or more lenient, depending on the evolutionary distance of the genomes. When you are unsure which relaxation setting is most suitable for your dataset, running the *optimal_grouping* functionality is recommended.

Be aware that not every sequence within a homology group has to be similar to the other sequences. For example, two non-similar protein sequences each have a high-similarity hit with the same protein sequence but align to a different region, one at the start and one near the end of the sequence.

When you want to run **group** another time but with different parameters, the currently active grouping must first either be moved or removed. This can be achieved with the *move- or remove_homology_groups* functions.

Method

Here, we explain a simplified version of the original algorithm, please take a look at our publication for an extensive explanation. First, potential similar sequences are identified by counting shared *k*-mer (protein) sequences. Similarity between the selected protein sequences is calculated through (local) Smith-Waterman alignments. When the (normalized) similarity score of two sequences is above a given threshold (controlled by `--relaxation`), the proteins are connected with each other in the similarity graph. Every similarity component is then passed to the MCL (Markov clustering) algorithm to be possibly broken into several homology groups.

Required software

MCL

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

`--threads`/`-tn` The number of parallel working threads. Default and minimum required threads is 3.

`--longest-transcript` Only cluster the longest protein-coding transcript of genes.

`--annotations-file`/`-af` A text file with the identifiers of annotations to be included, each on a separate line. The most recent annotation is selected for genomes without an identifier.

Optional arguments that influence the clustering sensitivity

`--relaxation`/`-rn` The relaxation in homology calls. Should be in range [1-8], from strict to relaxed (default 1). **IMPORTANT!** This argument automatically sets the four remaining arguments, stated here below.

`--intersection-rate`/`-ir` The fraction of *k*-mers that needs to be shared by two intersecting proteins. Should be in range [0.001, 0.1] (default = 0.08).

`--similarity-threshold`/`-st` The minimum normalized similarity score of two proteins. Should be in range [1-99] (default = 95).

`--mcl-inflation`/`-mi` The MCL inflation. Should be in range [1-19] (default = 10.8).

`--contrast`/`-cn` The contrast factor. Should be in range [0-10] (default = 8).

Example commands

```
$ pantools group -dp tomato_DB
$ pantools group -dp tomato_DB -tn 12 -rn 4
```

Output

- **pantools_homology_groups.txt**, overview of the created homology groups. Each line represents one homology group, starting with the homology group (database) identifier followed by a colon (:) and mRNA identifiers (from GFF) that are separated by a space. To ensure all identifiers are unique in this file, the mRNA ids are extended by a hash symbol (#) and a genome number. The following line is example output of an homology group with two genes from genome 1 and 146:

```
14001754: DLACAPHP_00001_mRNA#1 OPJEMMMF_03822_mRNA#146
```

Relevant literature

- Efficient inference of homologs in large eukaryotic pan-proteomes

6.2.4 Optimal grouping

Finding the most suitable settings for *group* can be difficult and is always dependent on evolutionary distance of the genomes in the pangenome. This functionality runs **group** on all eight `--relaxation` settings, from strictest (d1) to the most relaxed (d8). To find the optimal setting, complete and **non-duplicated BUSCO** genes that are present in all genomes are used to validate each setting.

Method

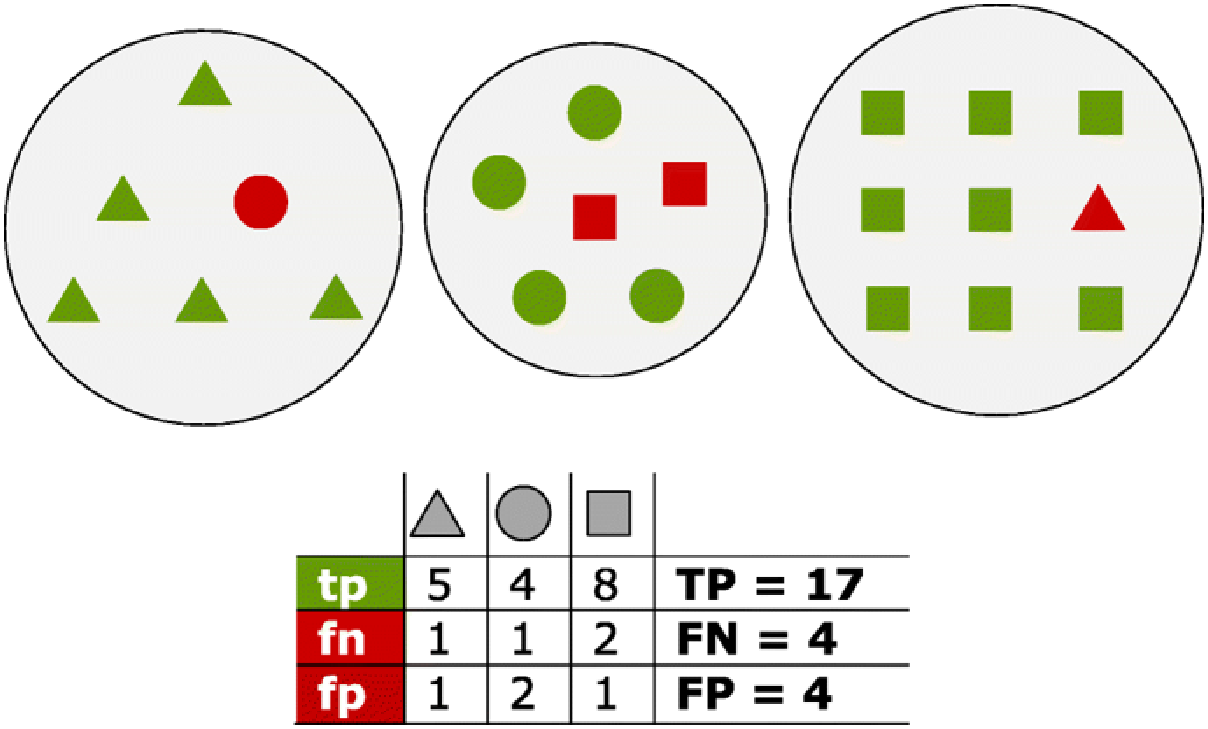
A perfect clustering of the sequences would place each BUSCO in a separate homology group with one representative protein per genome. When BUSCO is run against the pangenome, the proteins corresponding to the BUSCO HMMs have been identified. For each BUSCO, the representative proteins are checked whether these are clustered into a single or multiple groups. These groups are searched to identify sequences other than the current BUSCO. The highest number of correctly clustered BUSCOs present in one group are true positives (**tp**). Any other gene clustered inside this group is considered a false positive (**fp**) The remaining BUSCO genes outside this best group are counted as false negative (**fn**). The summation of tps fps and fns are defined as **TP**, **FP** and **FN**, respectively. From these scores recall, precision and F-score measures are calculated as follows:

$$
\begin{aligned}
Recall &= \frac{TP}{TP+FN}\\
Precision &= \frac{TP}{TP+FP}\\
F-score &= 2\frac{Recall * Precision}{Recall + Precision}
\end{aligned}
$$

Choosing the optimal setting

Choosing the correct setting is usually a trade-off between TPs and FNs. The most strict grouping results in a significantly higher number of clusters as the more relaxed settings. With stringent settings, related proteins could get separated; however, a high number of false positives is (usually) prevented (FN > FP). When you would go for a more loose setting, the related proteins are likely to part of the same group, but other sequences could be included as well (FN < FP).

No grouping is active after running this function. Use the generated output files to identify a suitable grouping. Activate this grouping using *change_grouping*. An overview of the available groupings and used settings is stored in the 'pangenome' node (inside the database), or can be created by running *grouping_overview*.



	△	○	□	
tp	5	4	8	TP = 17
fn	1	1	2	FN = 4
fp	1	2	1	FP = 4

Fig. 6.1: *Proteins of three distinct homology groups are represented as triangles, circles and squares. Green shapes are true positives (tp) which have been assigned to the true group; red shapes are false positives (fp) for the group they have been incorrectly assigned to, and false negatives (fn) for their true group*

Required software

MCL

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--input-file`/`-if` The output directory created by the *busco_protein* function. This directory is found **inside** the pangenome database, in the *busco* directory.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

`--threads`/`tn` Number of threads. The default and minimum required threads is 3.

`--value` Only consider a selection of relaxation settings (1-8 allowed).

`--fast` Assume the optimal grouping is found when the F1-score drops compared to the previous clustering round.

`--longest-transcript` Only cluster protein sequences of the largest transcript per gene.

`--annotations-file`/`-af` A text file with the identifiers of annotations to be included, each on a separate line. The most recent annotation is selected for genomes without an identifier.

Example commands

```
$ pantools optimal_grouping -dp bacteria_DB -if bacteria_DB/busco/bacteria_odb9
$ pantools optimal_grouping -dp bacteria_DB -if bacteria_DB/busco/bacteria_odb9 -tn 12 --
↪fast
$ pantools optimal_grouping -dp bacteria_DB -if bacteria_DB/busco/bacteria_odb9 -tn 12 --
↪fast --longest-transcript
$ pantools optimal_grouping -dp bacteria_DB -if bacteria_DB/busco/bacteria_odb9 -tn 12 --
↪value 1,2,3,4

$ Rscript optimal_grouping.R
```

Output

After each clustering round, homology groups are incorporated in the graph. A text file with homology group and gene identifiers is stored in the **group** directory in the pangenome database. This file is named after the used sequence similarity threshold (25-95). Each line represents one homology group, starting with the homology group (database) identifier followed by a colon (:) and mRNA identifiers (from GFF) that are separated by a space. The mRNA identifiers are extended by a hash (#) and their genome number. The following line is example output of an homology group with two genes from genome 1 and 146:

```
14001754: DLACAPHP_00001_mRNA#1 OPJEMMMF_03822_mRNA#146
```

Output files are written to **optimal_grouping** directory inside the database.

- **grouping_overview.csv**, a summary of the benchmark statistics. Use this file to find the most suitable grouping for your pangenome.
- **optimal_grouping.R**, Rscript to plot FN and FP values per grouping.
- **counts_per_busco.info**, a log file of the scoring. Shows in which homology groups the BUSCO genes were placed for the different groupings.

6.2.5 Change grouping

Only a single homology grouping can be active in the pangenome. Use this function to change the active grouping version. Information of the available groupings and used settings is stored in the 'pangenome' node (inside the database) and can be created by running *grouping_overview*.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--version` The version of homology grouping to become active.

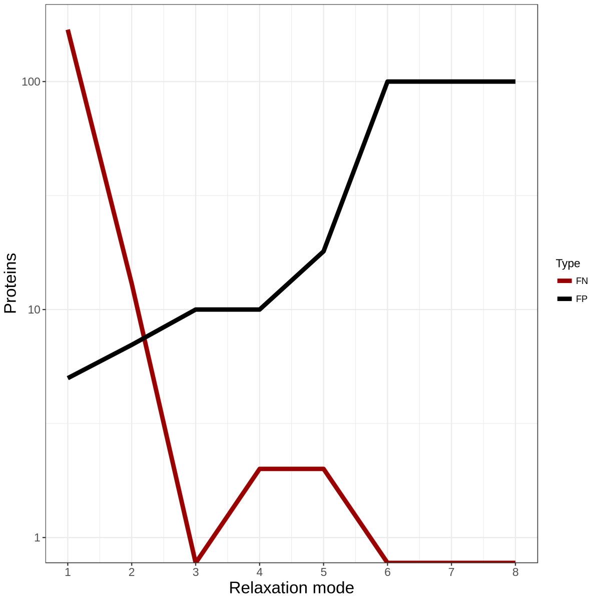


Fig. 6.2: **:italic:`Example output of **optimal_grouping.R**. The number of FN and FP for all eight relaxation settings.`**

Example command

```
$ pantools change_grouping -dp tomato_DB --version 5
```

6.2.6 Build panproteome

Build a panproteome out of a set of proteins. By only including protein sequences, the usable functionalities are limited to a protein-based analysis, please see *differences pangenome and panproteome*. No additional proteins can be added to the panproteome, it needs to be rebuilt completely.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--proteomes-file`/`-pf` A text file containing paths to FASTA files of proteins to be added to the panproteome; each on a separate line.

Example input file

```
/always/proteins1.fasta
/use_the/proteins2.fasta
/full_path/proteins3.faa
```

Example command

```
$ pantools build_panproteome -dp proteome_DB -pf proteins.txt
```

6.2.7 Add genomes

Include additional genomes to an already available pangenome.

Required software

KMC 2.3 or 3.0

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--genomes-file`/`-gf` A text file containing paths to FASTA files of genomes to be added to the pangenome; each on a separate line.

Example input file

```
/use_the/genome4.fasta
/full_path/genome5.fasta
```

Example command

```
$ pantools add_genomes -dp pangenome_DB -gf extra_genomes.txt
```

6.2.8 Add phenotypes

Including phenotype data to the pangenome which allows the identification of phenotype specific genes, SNPs, functions, etc.. Altering the data is done by rerunning the command with an updated CSV file.

Data types

Each phenotype node contains a genome number and can hold the following data types: **String**, **Integer**, **Float** or **Boolean**.

- Values recognized as round number are converted to an **Integer** and to a **Double** when having one or multiple decimals.
- **Boolean** types are identified by checking if the value matches 'true' or 'false', ignoring capitalization of letters.
- **String** values remain completely unaltered except for spaces and quotes characters. Spaces are changed into an underscore ('_') character and quotes are completely removed.

Bin numerical values

When using numerical values, two genomes are only considered to share a phenotype if the value is identical. PanTools creates an alternative version for these phenotypes by binning the values. Taking 'Pathogenicity' from the example below we see the integers between 3 and 15. Using these two extreme values three bins are created for a new phenotype 'Pathogenicity_binned': 3-6.33, 6.34-11.66 and 11.67-15. The number of bins is controlled through `--value`. For skewed data, consider making the bins manually and include this as string phenotype.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--phenotype`/`-ph` A CSV file containing the phenotype information.

Optional argument

`--append` Do not remove existing phenotype nodes but only add new properties to it. If a property already exists, values from the new file will overwrite the old.

`--value` Number of bins used to group numerical values of a phenotype.

Example input file

The input file needs to be in .CSV format, a plain text file where each value is separated by a comma. The first **row** should start with 'Genome,' followed by the phenotype names and/or identifiers. The first **column** must start with genome numbers corresponding to the one in your pangenome. Phenotypes and metadata must be placed on the same line as their genome number. A field can remain empty when the phenotype for a genome is missing or unknown. Here below is an example of five genomes contains six phenotypes:

```
Genome,Gram,Region,Pathogenicity,Boolean,float,species
1,+,NL,3,True,0.1,Species
2,+,BE,,False,0.1,Species3
3,+,LUX,7,true,0.1,Species3
4,+,NL,9,false,0.1,Species3
5,+,BE,15,TRUE,0.1,Species1
```

Example command

```
$ pantools add_phenotype -dp tomato_DB --phenotype pheno.csv
$ pantools add_phenotype -dp tomato_DB -ph pheno.csv --append
```

Output

Phenotype information is stored in 'phenotype' nodes in the graph. An output file is written to the database directory.

- **phenotype_overview.txt**, a summary of the available phenotypes in the pangenome

6.2.9 BUSCO

BUSCO attempts to provide a quantitative assessment of the completeness in terms of expected gene content of a genome assembly. Proteins are placed into categories of Complete and **single-copy** (S), Complete and **duplicated** (D), **fragmented** (F), or **missing** (M). This function is able to run BUSCO **v3**, **v4** or **v5** against protein sequences of the pangenome.

The number of reported duplicated genes in eukaryotes is often to high as different protein isoforms are counted multiple times. To adjust the imprecise duplication score, include the `--longest-transcripts` argument to the command.

You don't have a benchmark set?

- When using BUSCO v3, go to https://busco.ezlab.org, download a odb9 set, and untar it with `tar -xvzf`. Include the entire directory in the command using the `--input-file` argument.
- For BUSCO v4 and v5, you only have to provide the odb10 database name with the `--input-file` argument, the database is downloaded automatically. To get a full list of the available datasets, run `busco --list-datasets`.

Required software

BUSCO must be set to your $PATH. For v3, test if the `which run_BUSCO.py` command displays the full path so it can accessed anywhere. For v4 and v5, test if `busco` is executable.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.
`--input-file`/`-if` A BUSCO benchmark dataset.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.
`--name` A string with questionable BUSCOs. Completeness (%) is recalculated by excluding these genes.
`--version` The BUSCO version. Select either 'busco3', 'busco4' or 'busco5' (default).
`--longest-transcript` Only search against the longest protein-coding transcript of genes.
`--annotations-file`/`-af` A text file with the identifiers of annotations to be included, each on a separate line. The most recent annotation is selected for genomes without an identifier.

Example commands

```
$ pantools busco_protein -dp bacteria_DB -if bacteria_odb10
$ pantools busco_protein -dp bacteria_DB -if busco_sets/bacteria_odb9/ --version busco3
$ pantools busco_protein -dp bacteria_DB -if busco_sets/bacteria_odb9/ --version busco3 -
↪-name POG093P01OY,POG093P0009,POG093P022K,POG093P027M,POG093P00Z2,POG093P013J
$ pantools busco_protein -dp bacteria_DB -if bacteria_odb10 --version busco4 --longest-
↪transcript
```

Output

The BUSCO scores are stored inside **BUSCO** nodes of the pangenome graph. Output files are written to the **busco** directory inside the database.

- **busco_scores.txt**, overview of the BUSCO scores per genome. Average and median statistics are calculated per category.
- **busco_overview.csv**, a table which combines the completeness scores per genome together with the duplicated, fragmented and missing BUSCO genes.
- **hmm_overview.txt**, a list of BUSCO genes showing the assigned categories per genome.

6.2.10 Add functional annotations

PanTools is able to incorporate functional annotations into the pangenome by reading output from various functional annotation tools.

Add functions

This function can integrate different functional annotations from a variety of annotation files. Currently available functional annotations: **Gene Ontology**, **Pfam**, **InterPro**, **TIGRFAM**, **Phobius**, **SignalP** and **COG**. The first time this function is executed, the Pfam, TIRGRAM, GO, and InterPro databases are integrated into the pangenome. Phobius, SignalP and COG annotations do not have separate nodes and are directly annotated on 'mRNA' nodes in the pangenome.

Gene names (or identifiers) from the input file are used to identify gene nodes in the pangenome. Only genes with an exactly matching name/identifier can be connected to functional annotation nodes! Use the same FASTA and GFF3 files that were used to construct the pangenome database.

Functional databases

Database versions in v3.4.0 repository

	Version	Download date (dd-mm-yyyy)
Gene ontology	2021-12-15	20-12-2021
Pfam	35.0	20-12-2021
TIGRFAM	15.0	01-10-2020
InterPro	87+	Not included in repository

We regularly check and update the four functional database. To update the functional database manually, download the following files and replace the old ones in the */pantools/addons/* directory. The TIGRFAM.info files are bundled in the TIGRFAMs_15.0_INFO.tar.gz file; download the file to addons/tigrfam and uncompress the tarball first. The first time running this function .INFO files are combined into a new file **COMBINATION_INFO_FILES** and removed afterwards.

File	Database type	Required directory	Download link
go.basic.obo	GO	addons	http://purl.obolibrary.org/obo/go/go-basic.obo
gene_ontology.txt	Pfam	addons	ftp://ftp.ebi.ac.uk/pub/databases/Pfam/releases//Pfam35.0/database_files/gene_ontology.txt.gz
Pfam-A.clans.tsv	Pfam	addons	ftp://ftp.ebi.ac.uk/pub/databases/Pfam/releases//Pfam35.0/Pfam-A.clans.tsv.gz
interpro.xml	InterPro	addons	https://ftp.ebi.ac.uk/pub/databases/interpro/current_release/interpro.xml.gz
TIGRFAMS_GO_LINK	TIGRFAM	addons/tigrfam	https://ftp.ncbi.nlm.nih.gov/hmm/TIGRFAMs/release_15.0/TIGRFAMS_GO_LINK
TIGRFAMS_ROLE_LINK	TIGRFAM	addons/tigrfam	https://ftp.ncbi.nlm.nih.gov/hmm/TIGRFAMs/release_15.0/TIGRFAMS_ROLE_LINK
TIGR_ROLE_NAMES	TIGRFAM	addons/tigrfam	https://ftp.ncbi.nlm.nih.gov/hmm/TIGRFAMs/release_15.0/TIGR_ROLE_NAMES
TIGR00001.INFO to TIGR04571.INFO	TIGRFAM	addons/tigrfam	https://ftp.ncbi.nlm.nih.gov/hmm/TIGRFAMs/release_15.0/TIGRFAMs_15.0_INFO.tar.gz

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--input-file`/`-if` A text file with on each line a genome number and the full path to the corresponding annotation file, separated by a space.

Optional arguments

`--annotations-file`/`-af` A text file with the identifiers of annotations to be included, each on a separate line. The most recent annotation is selected for genomes without an identifier.

Example command

```
$ pantools add_functions -dp tomato_DB -if f_annotations.txt
$ pantools add_functions -dp tomato_DB -if f_annotations.txt -af annotations.txt
```

Output

Functional annotations are incorporated in the graph. A log file is written to the **log** directory.

- **add_functional_annotations.log**, a log file with the the number of added functions per type and the identifiers of functions that could not be included.

Example input files

The `--input-file` requires to be formatted like an annotation input file. Each line of the file starts with the genome number followed by the full path to an annotation file.

File type	Recognized by pattern in file name
InterProScan	interpro & .gff
eggNOG-mapper	eggnog
Phobius	phobius
SignalP	signalp
Custom file	custom

```
1 /mnt/scratch/interpro_results_genome_1.gff
1 /mnt/scratch/custom_annotation_1.txt
1 /mnt/scratch/phobius_1.txt
2 /mnt/scratch/signalp.txt
2 /mnt/scratch/eggnog_genome_2.annotations
2 /mnt/scratch/transmembrane_annotations.txt phobius
3 /mnt/scratch/ipro_results_genome_3.annot custom
```

Annotation file types

PanTools can recognize functional annotations in different output formats.

Phobius and SignalP are not standard analyses of the InterProScan pipeline and require some additional steps during the InterProScan installation. Please take a look at *our InterProScan install instruction* to verify if the tools are part of the prediction pipeline. Phobius 1.01

Function type	Allowed annotation file
GO	InterProscan .gff & custom annotation file
Pfam	InterProscan .gff & custom annotation file
InterPro	InterProscan .gff & custom annotation file
TIGRFAM	InterProscan .gff & custom annotation file
Phobius	InterProscan .gff & Phobius 1.01 output
SignalP	InterProscan .gff, signalP 4.1 output, signalP 5.0 output
COG	eggNOG-mapper

InterProScan gff file:

```
##gff-version 3
##interproscan-version 5.52-86.0
AT4G21230.1   ProSiteProfiles protein_match 333 620 39.000664   +   .   date=06-10-2021;
↪Target=mRNA.AT4G21230.1 333 620;Ontology_term="GO:0004672","GO:0005524","GO:0006468";
↪ID=match$42_333_620;signature_desc=Protein kinase domain profile.;Name=PS50011;
↪status=T;Dbxref="InterPro:IPR000719"
AT3G08980.5   TIGRFAM protein_match         25   101 3.7E-14     +   .   date=06-10-2021;
↪Target=mRNA.AT3G08980.5 25 101;Ontology_term="GO:0006508","GO:0008236","GO:0016020";
↪ID=match$66_25_101;signature_desc=sigpep_I_bact: signal peptidase I;Name=TIGR02227;
↪status=T;Dbxref="InterPro:IPR000223"
AT2G17780.2   Phobius protein_match         338 354 .            +   .   date=06-10-2021;
↪Target=AT2G17780.2 338 354;ID=match$141_338_354;signature_desc=Region of a membrane-
↪bound protein predicted to be embedded in the membrane.;Name=TRANSMEMBRANE;status=T
AT2G17780.2   Phobius protein_match         1    337 .           +   .   date=06-10-2021;
↪Target=AT2G17780.2 1 337;ID=match$142_1_337;signature_desc=Region of a membrane-bound␣
↪protein predicted to be outside the membrane, in the extracellular region.;Name=NON_
↪CYTOPLASMIC_DOMAIN;status=T
AT3G11780.2   SignalP_EUK protein_match     1    24   .          +   .   date=06-10-2021;
↪Target=mRNA.AT3G11780.2 1 24;ID=match$230_1_24;Name=SignalP-noTM;status=T
AT1G04300.2   CDD protein_match             40   114 1.54717E-13 +   .   date=06-10-2021;
↪Target=mRNA.AT1G04300.2 40 114;Ontology_term="GO:0005515";ID=match$212_40_114;
↪signature_desc=MATH;Name=cd00121;status=T;Dbxref="InterPro:IPR002083"
```

eggNOG-mapper (tab separated) file:

```
#query_name     seed_eggNOG_ortholog seed_ortholog_evalue seed_ortholog_score best_tax_
↪level Preferred_name GOs EC KEGG_ko KEGG_Pathway KEGG_Module KEGG_Reaction KEGG_rclass␣
↪BRITE KEGG_TC CAZy BiGG_Reaction taxonomic scope eggNOG OGs best eggNOG OG COG␣
↪Functional cat. eggNOG free text desc.
ATKYO-2G54530.1 3702.AT2G35130.2     1.9e-179             636.0              ␣
↪Brassicales     GO:0003674,GO:0003676,GO:0003723,GO:0003824,GO:0004518,GO:0004519,
↪GO:0005488,GO:0005575,GO:0005622,GO:0005623,GO:0006139,GO:0006725,GO:0006807,
↪GO:0008150,GO:0008152,GO:0009451,GO:0009987,GO:0016070,GO:0016787,GO:0016788,
↪GO:0034641,GO:0043170,GO:0043226,GO:0043227,GO:0043229,GO:0043231,GO:0043412,
↪GO:0044237,GO:0044238,GO:0044424,GO:0044464,GO:0046483,GO:0071704,GO:0090304,
↪GO:0090305,GO:0097159,GO:1901360,GO:1901363                                    ␣
↪Viridiplantae   37R67@33090,3GAUT@35493,3HNDD@3699,KOG4197@1,KOG4197@2759   NA|NA|NA  ␣
↪ E   Pentacotripeptide-repeat region of PRORP
```

(continues on next page)

(continued from previous page)

```
ATKYO-UG22500.1 3712.Bo02269s010.1   7.5e-35              153.7                 ␣
↪Brassicales                                              Viridiplantae   29I9W@1,
↪2RRH4@2759,383W6@33090,3GWQZ@35493,3I1A9@3699   NA|NA|NA
ATKYO-1G60060.1 3702.AT1G48090.1     0.0                  6241.0                ␣
↪Brassicales            ko:K19525                  ko00000             Viridiplantae ␣
↪ 37IJB@33090,3GAN0@35493,3HQ90@3699,COG5043@1,KOG1809@2759   NA|NA|NA    U   Vacuolar␣
↪protein sorting-associated protein
ATKYO-3G74720.1 3702.AT3G52120.1     7.2e-245             852.8                 ␣
↪Brassicales            ko:K13096                  ko00000,ko03041              ␣
↪Viridiplantae   37QYY@33090,3G9VU@35493,3HRDK@3699,KOG0965@1,KOG0965@2759   NA|NA|NA  ␣
↪ L   SWAP (Suppressor-of-White-APricot) surp domain-containing protein D111 G-patch␣
↪domain-containing protein
ATKYO-4G41660.1 3702.AT4G16340.1     0.0                  3392.1                ␣
↪Brassicales     GO:0003674,GO:0005085,GO:0005088,GO:0005089,GO:0005488,GO:0005515,
↪GO:0005575,GO:0005622,GO:0005623,GO:0005634,GO:0005737,GO:0005783,GO:0005829,
↪GO:0005886,GO:0006810,GO:0008064,GO:0008150,GO:0008360,GO:0009605,GO:0009606,
↪GO:0009628,GO:0009629,GO:0009630,GO:0009958,GO:0009966,GO:0009987,GO:0010646,
↪GO:0010928,GO:0012505,GO:0016020,GO:0016043,GO:0016192,GO:0017016,GO:0017048,
↪GO:0019898,GO:0019899,GO:0022603,GO:0022604,GO:0023051,GO:0030832,GO:0031267,
↪GO:0032535,GO:0032956,GO:0032970,GO:0033043,GO:0043226,GO:0043227,GO:0043229,
↪GO:0043231,GO:0044422,GO:0044424,GO:0044425,GO:0044432,GO:0044444,GO:0044446,
↪GO:0044464,GO:0048583,GO:0050789,GO:0050793,GO:0050794,GO:0050896,GO:0051020,
↪GO:0051128,GO:0051179,GO:0051234,GO:0051493,GO:0065007,GO:0065008,GO:0065009,
↪GO:0070971,GO:0071840,GO:0071944,GO:0090066,GO:0098772,GO:0110053,GO:1902903    ␣
↪ko:K21852                  ko00000,ko04131             Viridiplantae   37QIM@33090,
↪3G8RK@35493,3HSFN@3699,KOG1997@1,KOG1997@2759   NA|NA|NA    T   Belongs to the DOCK␣
↪family
```

A custom input file must consist of two tab or comma separated columns. The first column should contain a gene/mRNA id, the second an identifier from one of four functional annotation databases: GO, Pfam, InterPro or TIGRFAM.

```
AT5G23090.4,GO:0046982
AT5G23090.4,IPR009072
AT1G27540.2,PF03478
AT2G18450.1,TIGR01816
```

Phobius 1.01 ‘short’ (tab separated) input file:

```
SEQENCE ID                     TM SP PREDICTION
mRNA-YPR204W                    0  0 o
mRNA-ndhB-2_1                   6  Y n5-16c21/22o37-57i64-83o89-113i134-156o168-189i223-
↪246o
```

Phobius 1.01 ‘long’ (tab separated) input file:

```
ID   mRNA-YPR204W
FT   DOMAIN        1   1032       NON CYTOPLASMIC.
//
ID   mRNA-ndhB-2_1
FT   SIGNAL        1     21
FT   DOMAIN        1      4       N-REGION.
FT   DOMAIN        5     16       H-REGION.
FT   DOMAIN       17     21       C-REGION.
```

(continues on next page)

(continued from previous page)

```
FT   DOMAIN        22     36       NON CYTOPLASMIC.
FT   TRANSMEM      37     57
FT   DOMAIN        58     63       CYTOPLASMIC.
FT   TRANSMEM      64     83
FT   DOMAIN        84     88       NON CYTOPLASMIC.
FT   TRANSMEM      89    113
FT   DOMAIN       114    133       CYTOPLASMIC.
FT   TRANSMEM     134    156
FT   DOMAIN       157    167       NON CYTOPLASMIC.
FT   TRANSMEM     168    189
FT   DOMAIN       190    222       CYTOPLASMIC.
FT   TRANSMEM     223    246
FT   DOMAIN       247    253       NON CYTOPLASMIC.
//
```

SignalP 4.1 ‘short’ (tab separated) input file:

```
# name                     Cmax  pos  Ymax  pos  Smax  pos  Smean   D     ?  Dmaxcut   Networks-used
mRNA-rpl2-3                0.148  20  0.136  20  0.146   3  0.126   0.131 N  0.450     SignalP-noTM
mRNA-cox2                  0.107  25  0.132  12  0.270   4  0.162   0.148 N  0.450     SignalP-noTM
mRNA-cox2_1                0.850  17  0.776  17  0.785   2  0.717   0.753 Y  0.500     SignalP-TM
```

SignalP 5.0 ‘short’ (tab separated) input file:

```
# SignalP-5.0 Organism:   Eukarya     Timestamp: 20211122233246
# ID          Prediction  SP(Sec/SPI) OTHER    CS Position
AT3G26880.1   SP(Sec/SPI) 0.998803    0.001197 CS pos: 21-22. VYG-KK. Pr: 0.9807
mRNA-rpl2-3   OTHER       0.001227    0.998773
```

Relevant literature

- Expansion of the Gene Ontology knowledgebase and resources
- InterPro in 2019: improving coverage, classification and access to protein sequence annotations
- TIGRFAMs and Genome Properties in 2013
- A Combined Transmembrane Topology and Signal Peptide Prediction Method
- Expanded microbial genome coverage and improved protein family annotation in the COG database

Add antiSMASH gene clusters

Read antiSMASH output and incorporate **Biosynthetic Gene Clusters** (BGC) nodes into the pangenome database. A 'bgc' node holds the gene cluster product, the cluster address and has a relationship to all gene nodes of the cluster. For this function to work, antiSMASH should be performed with the same FASTA and GFF3 files used for building the pangenome. antiSMASH output will not match the identifiers of the pangenome when no GFF file was included.

As of PanTools v3.3.4 the required antiSMASH version is 6.0.0. Gene cluster information is parsed from the .JSON file that is generated in each run. We try to keep the parser updated with newer versions but please contact us when this is no longer the case.

	Version	Version date
antiSMASH	6.0.0	21-02-2021

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

`--input-file`/`-if` A text file with on each line a genome number and the full path to the corresponding antiSMASH output file, separated by a space.

Optional arguments

`--annotations-file`/`-af` A text file with the identifiers of annotations to be included, each on a separate line. The most recent annotation is selected for genomes without an identifier.

Example input file

The `--input-file` requires to be formatted like a regular annotation input file. Each line of the file starts with the genome number followed by the full path to the **JSON** file.

```
1 /mnt/scratch/IPO3844/antismash/IPO3844.json
4 /home/user/IPO3845/antismash/IPO3845.json
```

Example command

```
$ pantools add_antismash -dp tomato_DB -if clusters.txt
```

6.2.11 Removing data

The following functionalities allow the removal of large sets of nodes and relationships from the pangenome. These functions will first ask for a confirmation before the nodes are actually removed. Be careful, the data is not backed up and removing nodes or properties means it is permanently gone.

Remove nodes

Remove a selection of nodes and their relationships from the pangenome. For a pangenome database the following nodes cannot be removed: *nucleotide*, *pangenome*, *genome*, *sequence*. When using a panproteome, *mRNA* nodes cannot be removed.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Requires either one of the following arguments

`--node` one or multiple node identifiers, separated by a comma.
`--label` a node label, all nodes matching the label are removed.

Optional arguments

Both optional arguments can only be used in combination with `--label`.

`--skip`/`-sk` Do not remove nodes of the selected genomes.
`--reference`/`-ref` Only remove nodes of the selected genomes.

Example commands

```
$ pantools remove_nodes -dp tomato_DB --node 10348734,10348735,10348736
$ pantools remove_nodes -dp tomato_DB --label pfam
$ pantools remove_nodes -dp tomato_DB --label interpro --reference 2-6
```

Remove phenotypes

Delete **phenotype** nodes or remove specific phenotype information from the nodes. The specific phenotype property needs to be specified with `--phenotype`. When this argument is not included, *phenotype* nodes are removed.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--phenotype`/`-ph` name of the phenotype. All information of the given phenotype is removed from 'phenotype' nodes.
`--skip`/`-sk` Do not remove nodes of the selected genomes.
`--reference`/`-ref` Only remove nodes of the selected genomes.

Example commands

```
$ pantools remove_phenotype -dp tomato_DB
$ pantools remove_phenotype -dp tomato_DB --phenotype color
$ pantools remove_phenotype -dp tomato_DB --phenotype color --skip 11,12
```

Remove annotations

Remove all the genomic features that belong to annotations, such as *gene*, *mRNA*, *exon*, *tRNA*, and *feature* nodes. Functional annotation nodes are not removed with this function but can be removed with *remove_nodes*. Removing annotations can be done in two ways:

1. Selecting genomes with `--reference` or `--skip`, for which all annotation features will be removed.
2. Remove specific annotations by providing a text file with identifiers via the `--annotations-file` argument.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Requires either one of the following arguments

`--skip`/`-sk` a selection of genomes excluded from the removal of annotations.
`--reference`/`-ref` a selection of genomes for which all annotations will be removed.
`--annotations-file`/`-af` A text file with the identifiers of annotations to be removed, each on a separate line.

Example input file

The input file should be a single line with annotation identifiers separated by a comma. The following example will remove the first annotations of genome 1, 2 and 3 and the second annotation of genome 1.

```
1_1
1_2
2_1
3_1
```

Example command

```
$ pantools remove_annotations --skip 3,4,5
$ pantools remove_annotations -af annotations.txt
```

Move or remove grouping

As only one grouping can be active at the time, the currently active grouping needs to be removed or inactivated before *group* can be run again.

- **remove_grouping** deletes all 'homology_group' nodes and 'is_similar' relations between 'mRNA' nodes from the database.
- **move_grouping** relabels 'homology_group' nodes to 'inactive_homology_group'. The moved grouping can be activated again with change_grouping.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments for remove_grouping

`--version` Select a specific grouping version to be removed. Two additional options: 'all' to remove all groupings and 'all_inactive' to remove all inactive groupings.
`--fast` Do not remove the 'is_similar' relationships between mRNA nodes. This does not influence the next grouping.

Example command

```
$ pantools move_grouping -dp tomato_DB

$ pantools remove_grouping -dp tomato_DB
$ pantools remove_grouping -dp tomato_DB --version 1
$ pantools remove_grouping -dp tomato_DB --version all --fast
$ pantools remove_grouping -dp tomato_DB --version all_inactive
```

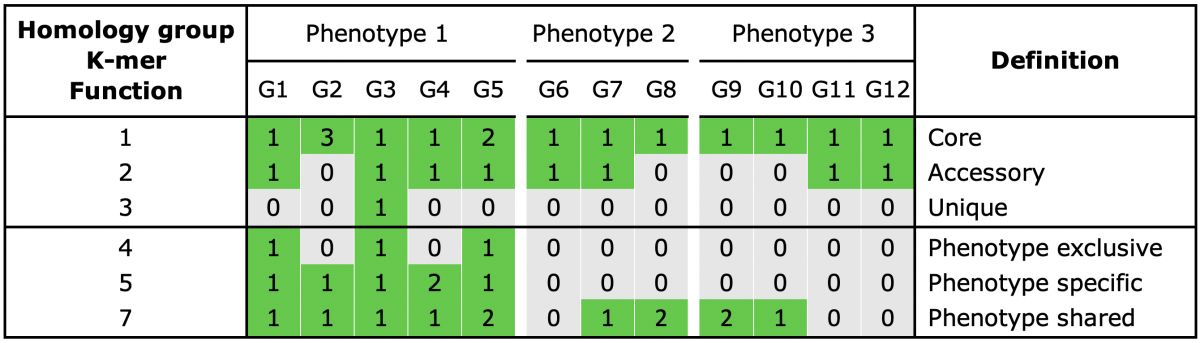
6.3 Pangenome characterization

Functionalities for characterization a pangenome based on genes, *k*-mer sequences and functions. In this manual we use several pangenome related terms with the following definitions:

- **Core**, an element is present in all genomes
- **Unique**, an element is present in a single genome
- **Accessory**, an element is present in some but not all genomes

When phenotype information is used in the analysis, three additional categories can be assigned:

- **Shared**, an element present in all genomes of a phenotype
- **Exclusive**, an element is only present in a certain phenotype
- **Specific**, an element present in all genomes of a phenotype and is also exclusive



Homology group **K-mer** **Function**	Phenotype 1					Phenotype 2			Phenotype 3				**Definition**
	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	
1	1	3	1	1	2	1	1	1	1	1	1	1	Core
2	1	0	1	1	1	1	1	0	0	0	1	1	Accessory
3	0	0	1	0	0	0	0	0	0	0	0	0	Unique
4	1	0	1	0	1	0	0	0	0	0	0	0	Phenotype exclusive
5	1	1	1	2	1	0	0	0	0	0	0	0	Phenotype specific
7	1	1	1	1	2	0	1	2	2	1	0	0	Phenotype shared

Fig. 6.3: *The possible classification categories for genes, k mers and functions. Additional copies of an element are assigned to the same category.*

6.3.1 Pangenome metrics

Generates relevant metrics of the pangenome and the individual genomes and sequences.

- On the pangenome level: the number of genomes, sequences, annotations, genes, proteins, homology groups, *k*-mers, and database nodes and edges.
- On the genome and sequence level: assembly statistics and metrics about functional elements. The assembly statistics consists of genome size, N25-N95, L25-L95, BUSCO scores and GC content. An overview of the functional elements is created by summarizing the functional annotations per genome (and sequence) and reporting the shortest, longest, average length and density per MB for genome features such as genes, exons and CDS.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes

`--reference`/`-ref` Only include a selection of genomes.

`--annotations-file`/`-af` A text file with the identifiers of annotations that should be used. The most recent annotation is selected for genomes without an identifier.

Example commands

```
$ pantools metrics -dp tomato_DB
$ pantools metrics -dp tomato_DB --skip 1,2,5
```

Output

Output files are written to the **metrics** directory in the database. Note: the percentage a genome or sequence is covered by a genes, repeats etc., (currently) does not consider overlap between features!

- **metrics.txt**, overview of the metrics calculated on the pangenome and genome level.
- **metrics_per_genome.csv**, summary of the metrics that are calculated on a genome level. The output is formatted as table.
- **metrics_per_sequence.csv**, summary of metrics that are calculated on a sequence (contig/scaffold) level. The output is formatted as table. This file is **not** created when using a panproteome.

6.3.2 Homology groups

The following functions require the protein sequences to be clustered by *group*.

Gene classification

Classification of the pangenome's gene repertoire. Homology groups are utilized to identify shared genes between genomes. The default criteria for defining the category of a gene is shown in Fig. 6.3.

To identify soft core and cloud genes, the core and unique thresholds (%) can be relaxed by `--core-threshold` and `--unique-threshold`, respectively. The `--phenotype-threshold` argument can be used to lower the threshold for phenotype specific and shared homology groups.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--phenotype`/`-ph` A phenotype name, used to find genes specific to the phenotype.
`--skip`/`-sk` Exclude a selection of genomes. This automatically lowers the threshold for core genes.
`--reference`/`-ref` Only include a selection of genomes. This automatically lowers the threshold for core genes.
`--core-threshold`/`-ct` Threshold (%) for (soft) core genes. Default is 100% of genomes.
`--unique-threshold`/`-ut` Threshold (%) for unique/cloud genes. Default is a single genome, not a percentage.
`--phenotype-threshold`/`-pt` Threshold (%) for phenotype specific/shared genes. Default is 100% of genomes with phenotype.
`--mode MLSA` Finds suitable single-copy groups for a *MLSA*.

Example command

```
$ pantools gene_classification -dp tomato_DB
$ pantools gene_classification -dp tomato_DB --unique-threshold 5 --core-threshold 95
$ pantools gene_classification -dp tomato_DB --phenotype resistance --skip 2,3 --
↪phenotype-threshold 95
```

Output

Output files are written to the **gene_classification** directory in the database.

1. **gene_classification_overview.txt**, statistics of the core, accessory, unique groups of the pangenome and individual genomes.
2. **classified_groups.csv**, the classified homology groups formatted as the table in the example table above.
3. **cnv_core_accessory.txt**, core and accessory groups with genomes that have additional copies compared to the lowest number (at least 1) in the group.
4. **group_size_occurrence.txt**, number of times a group of a certain size occurs in the pangenome. The homology group sizes can be based on the number of proteins or the number of genomes.
5. **gene_distance_tree.R**, an R script to cluster genomes based on gene distance (absence/presence). For more information, see the *Gene distance tree* manual.
6. **shared_unshared_gene_count.csv**, six tables with the number of shared and unshared genes between genomes: all genes, distinct genes and informative distinct genes. To get the number of distinct genes, additional copies of a gene within a homology group are ignored. Genes are considered informative when shared by at least two genomes.

Additional files are generated when the `--phenotype` argument is included.

1. **gene_classification_phenotype_overview.txt**, the number of identified phenotype shared and specific groups.
2. **phenotype_disrupted.txt**, this file shows which proteins prevented phenotype shared groups to be specific.
3. **phenotype_cnv**, homology groups where all members of a phenotype have at least one additional copy of a gene compared to one of the other phenotypes.

4. **phenotype_association.csv**, results of performed Fisher exact tests on homology groups with an unequal proportion of phenotype members.

The following files contain homology group node identifiers.

1. **all_homology_groups.csv**, the node identifiers of all homology groups.
2. **core_groups.csv**, the node identifiers of the core homology groups.
3. **single_copy_orthologs.csv**, the node identifiers of single-copy ortholog groups. This is a subset of the core set where each genome is only allowed to have a one copy of a gene.
4. **accessory_groups.csv**, the node identifiers of accessory homology groups. The groups are ordered (in descending order) by the group size based on the total number of genomes present.
5. **accessory_combinations.csv**, the node identifiers of accessory homology groups, ordered by the combination of genomes by which they are shared.
6. **unique_groups.csv**, the node identifiers of unique homology groups ordered by genome.
7. **phenotype_specific_groups.csv**, the node identifiers of phenotype specific homology groups.
8. **phenotype_shared_groups.csv**, the node identifiers of phenotype shared homology groups.
9. **phenotype_exclusive_groups.csv**, the node identifiers of phenotype exclusive homology groups.

When `--mode MLSA` is included

1. **mlsa_suggestions.txt**, a list of single copy ortholog genes all having the same gene name. This file cannot be created when using a panproteome.

Core unique thresholds

Runs a simplified version of the **gene_classification** function to test the effect of different `--core-threshold` and `--unique-threshold` cut-offs between 1 and 100%.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes. This automatically lowers the threshold for core genes.

`--reference`/`-ref` Only include a selection of genomes. This automatically lowers the threshold for core genes.

Example command

```
$ pantools core_unique_thresholds -dp tomato_DB
$ pantools core_unique_thresholds -dp tomato_DB --skip 1,2,5-10
$ R script tomato_DB/R_scripts/core_unique_thresholds/core_unique_thresholds.R
```

Output

Output files are written to **core_unique_thresholds** directory in the database.

- **core_unique_thresholds.csv**, the number of (soft) core unique/cloud homology groups for all tested thresholds.
- **core_unique_thresholds.R**, the R script plots the number of (soft) core unique/cloud homology groups for all tested thresholds.

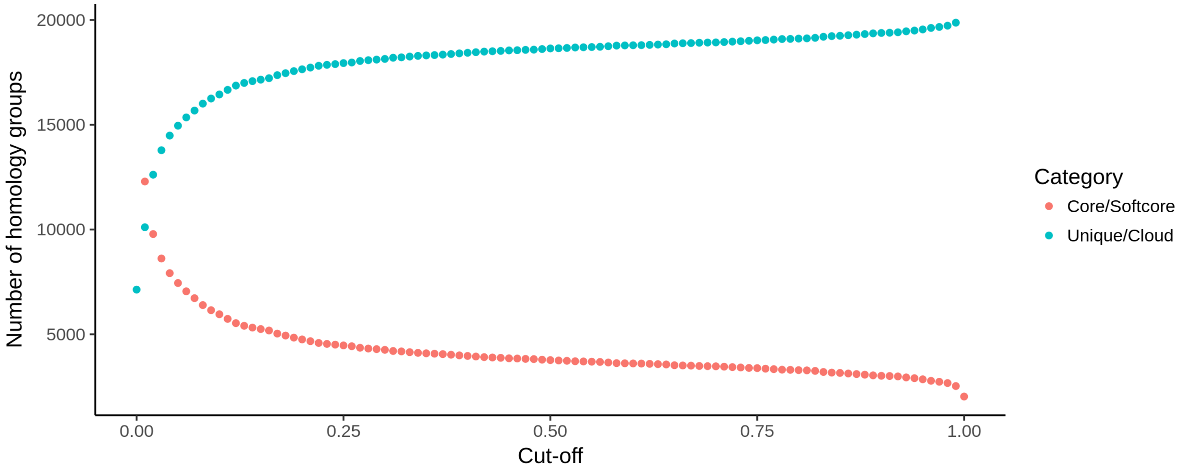


Fig. 6.4: *Example output of* **core_unique_thresholds.R** *on a pangenome of 197 Pectobacterium genomes demonstrates the effect of loosening the thresholds. The number of (soft) core (orange) homology groups slightly increases when the cut-off for this category is lowered from 100% (200 genomes) to 1% (2) in steps of 1%. Unique/cloud (blue) start at 0.00 which represents a single genome. Using a 0.01 cut-off, groups are unique/cloud having 2 genomes or less. The threshold is further increased to 100% (200) in steps of 1%.*

Grouping overview

Reports the content of all (active & inactive) homology groups for the different groupings in the pangenome. Include `--mode fast` into the command to get a quick overview of the available groupings and the settings that were used.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--mode fast` Only show which grouping is active and which groupings can be activated.

Example commands

```
$ pantools grouping_overview -dp tomato_DB
$ pantools grouping_overview -dp tomato_DB --mode fast
```

Output

Output files are written to */database_directory/group/*

- **grouping_overview.txt**, all homology groups in the pangenome. For each homology group, the total number of members and the number of members per per genome is reported.
- **current_pantools_homology_groups.txt**, overview of the active homology groups. Each line represents one homology group. The line starts with the homology group (database) identifier followed by a colon and the rest are mRNA IDs (from gff/genbank) seperated by a space.

6.3.3 Pangenome structure

Iterations of random genome combinations according to the models proposed by Tettelin et al.[*] in 2005 are used to determine the contribution of new accessions with respect to the increase in core, accessory, and unique. Each iteration starts with three random genomes from which core, accessory and unique homology groups are identified. Subsequently, random genomes are added and group reclassified until the maximum number of genomes is reached. To simulate the overall pangenome-size increase and core-genome decrease, we suggest to use at least 10,000 iterations. Additional copies of a gene are ignored in the simulation.

Heaps' law (a power law) can be fitted to the number of new genes observed when increasing the pangenome by one random genome. The formula for the power law model is $n = k * N^{-a}$, where n is the newly discovered genes, N is the total number of genomes, and k and a are the fitting parameters. A pangenome can be considered open when $a < 1$ and closed if $a > 1$.

Pangenome size genes

Pangenome size estimation based on homology groups. This function requires the sequences to be already clustered by *group*.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--threads`/`-tn` (**default value**: 1) : The number of parallel working threads.
`--skip`/`-sk` Exclude a selection of genomes
`--reference`/`-ref` Only include a selection of genomes.
`--value` Number of loops (default is 10.000).

Example commands

```
$ pantools pangenome_structure_genes -dp tomato_DB
$ pantools pangenome_structure_genes -dp tomato_DB --value 1000 --skip 1-3,5

$ R script pangenome_growth.R
$ R script gains_losses_median_or_average.R
$ R script gains_losses_median_and_average.R
$ R script heaps_law.R
```

Output

Output files are written to */database_directory/pangenome_size/gene/*

- **pangenome_size.txt**, various statistics on the number core, accessory, and unique homology groups for the different pangenome sizes.
- **gains_losses.txt**, the average group gain and loss between different pangenome sizes. First the average number (core, accessory, and unique) groups for each pangenome size is calculated. The average gain and loss of groups is then found by subtracting the averages of a certain size to the averages of one genome larger (e.g. pangenome size of 5 is compared to 6).
- **gains_losses_last_genome.txt**, the number of (core, accessory, and unique) groups that are gained or lost when including one of the genomes to a pangenome of the remaining genomes.
- **pangenome_growth.R**, an R script to plot the number of core, accessory and unique groups for the different genome combinations. Second option is to only plot a core and accessory curve by including unique groups to the accessory.
- **gains_losses_median_and/or_average.R**, R scripts to plot the average and median group gain and loss between pangenome sizes.
- **heaps_law.R**, an R script to perform Heaps' law.

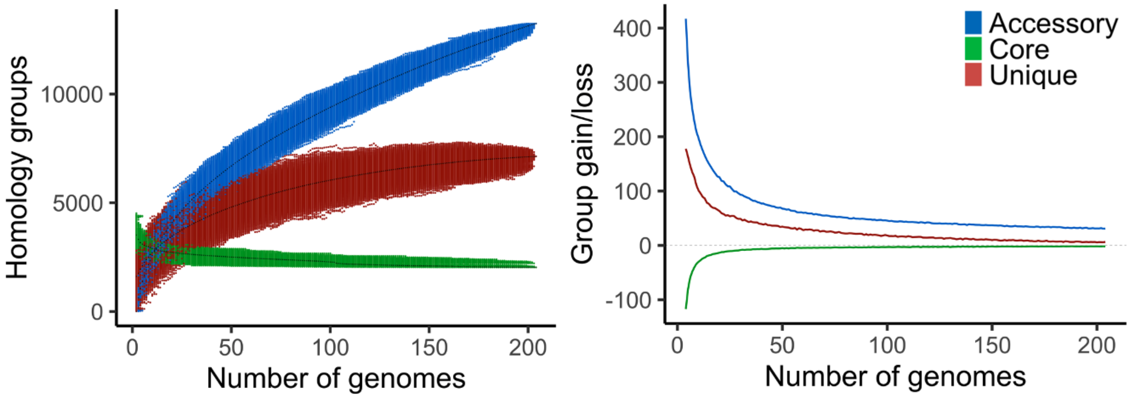


Fig. 6.5: *Example output of* **pangenome_growth.R** *(left) and* **gains_losses_median_and_average.R** *(right) on a pangenome of 204 bacteria.*

Relevant literature

- Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: Implications for the microbial "pan-genome"
- Comparative genomics: the bacterial pan-genome

Pangenome size *k*-mers

The same simulation as **pangenome_size_genes**, but performed on *k*-mer sequences instead of homology groups. As the number of *k*-mers is significantly higher than the number of homology groups, the runtime is much longer and the (default) number of loops is set to only 100.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--threads`/`-tn` (**default value**: 1) : The number of parallel working threads.
`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.
`--value` Number of loops (default is 100).

Example command

```
$ pantools pangenome_size_kmer -dp tomato_db
$ pantools pangenome_size_kmer -dp tomato_db --skip 4,5-9 --value 500
$ R script core_access_unique.R
```

Output

Output files are written to */database_directory/pangenome_size/kmer/*

- **pangenome_size_kmer.txt**, statistics of the number of *k*-mers with different pangenome sizes.
- **core_access_unique.R**, an R script to plot the number core, accessory, unique *k*-mers for the different genome combinations.
- **core_access.R**, an R script to plot the number of core and accessory (including unique) *k*-mers for the different genome combinations.

6.3.4 K-mer classification

Calculate the number of core, accessory, unique, (and phenotype specific) *k*-mer sequences. Because *k*-mer sequences of non-branching paths of the DBG graph are collapsed into a single node, *k*-mers are first uncompressed before they are counted. When `--mode compressed` is included, sequences are not uncompressed and considered as a single *k*-mer. Nucleotide nodes with a 'degenerate' label contain letters other than the four non-ambiguous ones (A, T, C, G). and are ignored by this function.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional arguments

`--phenotype`/`-ph` A phenotype name, used to identify phenotype specific *k*-mers.
`--skip`/`-sk` Exclude a selection of genomes. This automatically lowers the threshold for core *k*-mers.
`--reference`/`-ref` Only include a selection of genomes. This automatically lowers the threshold for core *k*-mers.
`--core-threshold`/`-ct` Threshold (%) for (soft) core *k*-mers. Default is 100% of the genomes.
`--unique-threshold`/`-ut` Threshold (%) for unique/cloud *k*-mers. Default is a single genome, not a percentage.
`--phenotype-threshold`/`-pt` Threshold (%) for phenotype specific/shared *k*-mers. Default is 100% of genomes with phenotype. `--mode compressed` Do not uncompress collapsed non-branching *k*-mers for *k*-mer counting.

Example commands

```
$ pantools kmer_classification -dp tomato_DB
$ pantools kmer_classification -dp tomato_DB --phenotype resistant --skip 2,3,4
$ pantools kmer_classification -dp tomato_DB --mode compressed --core-threshold 95 --
↪unique-threshold 5
```

Output

Output files are written to */database_directory/kmer_classification/*

- **kmer_classification_overview.txt**, some general statistics and percentages about the core, accessory unique *k*-mers per genome.
- **kmer_occurrence.txt**, the occurrence of *k*-mers per genome and total occurrence in the pangenome.
- **kmer_distance_tree.R**, an R script to cluster genomes with four different *k*-mer distances to choose from. For more information, see The *k*-mers are ordered from high to low by the total number of genomes the *k*-mer is found.
- **unique_kmers.csv**, the node identifiers of unique *k*-mers ordered by genome.
- **phenotype_specific_kmers.csv**, the node identifiers of phenotype specific *k*-mers.
- **phenotype_shared_kmers.csv**, the node identifiers of phenotype shared *k*-mers.

6.3.5 Functional annotations

The following functions can only be used when any type of functional annotation is *added to the database*.

Functional classification

Similar to **gene** and **k-mer classification**, this function identifies core, accessory, unique functional annotations in the pangenome. Only the following functions are considered for this analysis: biosynthetic gene clusters from antiSMASH, GO, PFAM, InterPro, TIGRFAM.

Required arguments

`--database-path`/`-dp` Path to the pangenome database.

Optional commands

`--phenotype`/`-ph` A phenotype name, used to find functions specific to a phenotype.
`--skip`/`-sk` Exclude a selection of genomes. This automatically lowers the threshold for core genes.
`--reference`/`-ref` Only include a selection of genomes. This automatically lowers the threshold for core genes.
`--core-threshold`/`-ct` Threshold (%) For (soft) core functions (default is 100%).
`--unique-threshold`/`-ut` Threshold (%) For unique/cloud functions (default is a single genome, not a percentage).
`--annotations-file`/`-af` A text file with the identifiers of annotations that should be used. The most recent annotation is selected for genomes without an identifier.

Example command

```
$ pantools functional_classification -dp tomato_DB
$ pantools functional_classification -dp tomato_DB -ph flowering_time
```

Output

Output files are written to */database_directory/function/functional_classification/*

- **functional_annotation_overview**, number of core, accessory, and unique functions. Holds the number of phenotype shared and specific functions when a phenotype is included.
- **core_functions.txt**, functional annotations found in every genome of the pangenome.
- **accessory_functions.txt**, functional annotations labeled as accessory.
- **unique_functions.txt**, functional annotations unique to a single genome.

When a `--phenotype` is included

- **phenotype_shared_functions.txt**, functional annotations shared by all phenotype members.
- **phenotype_specific_functions.txt**, functional annotations specific to certain phenotypes.

Functional annotation overview

Creates several summary files for each type of functional annotation present in the database: GO, PFAM, InterPro, TIGRFAM, COG, Phobius, and biosynthetic gene clusters from antiSMASH. In addition to the functions that must be added via *add_functional_annotations*, this function also requires proteins to be clustered by *group*.

Required argument

`--database-path`/`-dp` Path to the pangenome database.

Optional commands

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

`--annotations-file`/`-af` A text file with the identifiers of annotations that should be used. The most recent annotation is selected for genomes without an identifier.

Example command

```
$ pantools function_overview -dp tomato_DB
$ pantools function_overview -dp tomato_DB --reference 2-4
```

Output

Output files are written to *function* directory in the database. The overview CSV files are tables with on each row a function identifier with the frequency of per genome and.

- **functions_per_group_and_mrna.csv**, overview of all homology groups and the associated functions.
- **function_counts_per_group.csv**,
- **go_overview.csv**, overview of the GO terms in the pangenome.
- **pfam_overview.csv**, overview of the PFAM domains in the pangenome.
- **tigrfam_overview.csv**, overview of the TIGRFAMs in the pangenome.
- **interpro_overview.csv**, overview of the InterPro domains in the pangenome.
- **bgc_overview.csv**, overview of the added biosynthetic gene clusters from antiSMASH in the pangenome.
- **phobius_signalp_overview.csv**, overview of the included Phobius transmembrane topology and signal peptide predictions in the pangenome.
- **cog_overview.csv**, overview of the functional COG categories in the pangenome.
- **cog_per_class.R**, an R script to plot the distribution of COG categories over the core, accessory, unique homology groups.

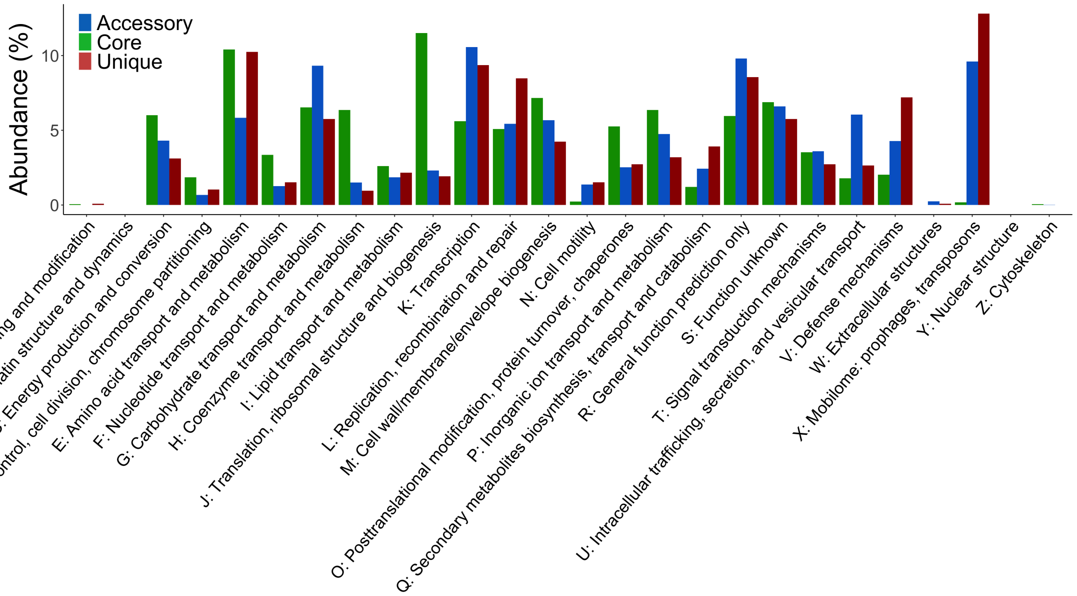


Fig. 6.6: *Example output of* **cog_per_class.R**. *The proportion of COGs functional categories assigned to homology groups.*

GO enrichment

For a given set of mRNA's or homology groups, this function identifies over or underrepresented GO terms by using a hypergeometric distribution.

The p-value is calculated from the hypergeometric distribution

$$P(X=k)=\frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

- Parameter **N** = size of the population (Universe of genes).
- Parameter **n** = size of the sample (signature gene set)
- Parameter **K** = successes in population (enrichment gene set)
- Parameter **k** = successes in sample (intersection of both gene sets)
- Return the **p-value** of the Hypergeometric Distribution for P(X=k)

Prepare input for hypergeometric tests

The size and number of successes of the sample (n, k) and background (N, K) is prepared for each genome individually. Per genome, loops over every mRNA and checks for connected GO nodes. Each GO node connected to the mRNA is used to move up in the GO hierarchy via '*is_a*' relations until the **molecular_function**, **biological_process** or **cellular_component** node is reached. Each GO term is counted only once per mRNA and a mRNA needs at least one GO term to be included in the sample and background sets. mRNA nodes which are part of the input homology groups are included into the sample set.

Multiple testing correction

Critical p-value using Bonferroni

For a GO germ to be significant, the p-value should be below 0.05 divided by number of tests per genome. For example, when 100 tests were performed, each p-value must be below 0.05/100 = 0.0005 to be considered significant.

Critical p-value using Benjamini-Hochberg procedure

1. Individual p-values are put in ascending order.
2. Ranks are assigned to the p-values. The lowest value has a rank of 1, the second lowest gets rank 2, etc..
3. The individual p-values Benjamini-Hochberg critical value is calculated using the formula $(i/m)Q$, where i is the individual p-values rank, m = total number of tests and Q is the false discovery rate.
4. Compare your original p-values to the critical B-H from Step 3; find the largest p value that is smaller than the critical value.

The critical p-value for the first rank for a total of 100 GO terms (tests) with a 5% false discovery rate is $(1/100)*0.05 = 0.0005$. For the second and third rank this will be 0.0010 and 0.0015, respectively.

Required software

- dot. Although this function still works when dot is not (properly) installed, no visualizations of the GO hierarchy can be created.

Required arguments

`--database-path`/`-dp` Path to the database.

Requires either **one** of the following arguments

`--homology-groups`/`-hm` A text file with homology group node identifiers, seperated by a comma `--node` mRNA node identifiers, seperated by a comma on the command line

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.
`--value` The false discovery rate (percentage), default is 5%.

Example command

```
$ pantools go_enrichment -dp tomato_DB -hm unique_groups.txt
$ pantools go_enrichment -dp tomato_DB -hm pheno_specific.txt --value 1 -ref 1-3,5
```

Output

Output files are stored in */database_directory/function/go_enrichment/*.

- **go_enrichment.csv**, overview of all GO terms, p-values and the significance of enrichment. The output is formatted as a table.
- **go_enrichment_overview_per_go.txt**, results of the analysis are ordered by GO term.
- **function_overview_per_mrna.txt**, all functional annotations connected to the input sequences, ordered per mRNA.
- **function_overview_per_genome.txt**, all functional annotations connected to the input sequences, ordered per genome.

Additional files are generated per individual genome and placed in */results_per_genome/*.

- **go_enrichment.txt**, list of GO terms, p-values and the critical p-values of Benjamin-Hochberg and Bonferroni.
- **revigo.txt**, a list of GO terms and p-values that can be visualized on http://revigo.irb.hr
- **bio_process.pdf**, dot visualisation of the Biological Process GO hierarchy.
- **cell_comp.pdf**, dot visualisation of the Cellular Component GO hierarchy.
- **mol_function.pdf**, dot visualisation of the Molecular Function GO hierarchy.

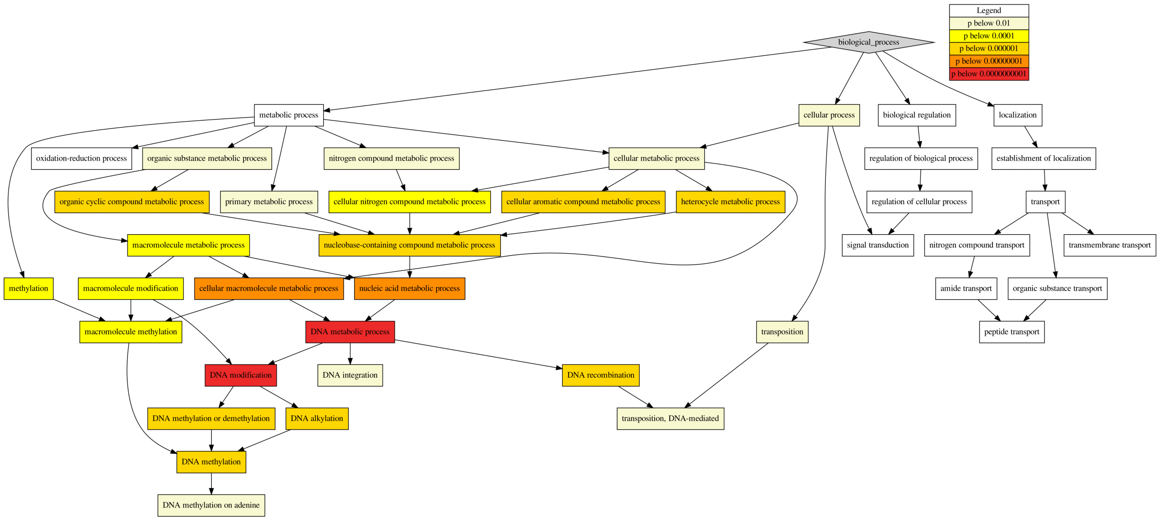

Legend
p below 0.01
p below 0.0001
p below 0.000001
p below 0.00000001
p below 0.0000000001
biological_process
metabolic process
cellular process
biological regulation
localization
oxidation-reduction process
organic substance metabolic process
nitrogen compound metabolic process
cellular metabolic process
regulation of biological process
establishment of localization
organic cyclic compound metabolic process
primary metabolic process
cellular nitrogen compound metabolic process
cellular aromatic compound metabolic process
heterocycle metabolic process
regulation of cellular process
transport
macromolecule metabolic process
nucleobase-containing compound metabolic process
signal transduction
nitrogen compound transport
transmembrane transport
methylation
macromolecule modification
cellular macromolecule metabolic process
nucleic acid metabolic process
amide transport
organic substance transport
macromolecule methylation
DNA metabolic process
transposition
peptide transport
DNA modification
DNA integration
DNA recombination
DNA methylation or demethylation
DNA alkylation
transposition, DNA-mediated
DNA methylation
DNA methylation on adenine

Fig. 6.7: *Visualization of GO hierarchy by dot*

6.4 Phylogeny

There are six different methods implemented which can create phylogenetic trees. The consensus tree method creates a Maximum per-Locus Quartet-score Species Tree (MLQST) from a set of gene trees. The other five methods use a Neighbour-joining (**NJ**) or Maximum Likelihood **(ML**) algorithm to infer the phylogeny.

- *Core phylogeny* **(ML)**
- *K-mer distance tree* **(NJ)**
- *Consensus tree*
- *Gene distance tree* **(NJ)**
- *ANI tree* **(NJ)**
- *MLSA* **(ML)**

All functions produce tree files in Newick format that can be visualized with iTOL or any other phylogenetic tree visualization software.

- *Rename phylogeny*
- *Reroot phylogeny*
- *Create tree template*

6.4.1 Core phylogeny

Infer a Maximum likelihood (ML) or Neighbour-Joining (NJ) phylogeny from SNPs identified from single copy orthologous genes. This function requires single-copy homology groups which are automatically detected if *gene_classification* was run before. The homology groups are aligned in two consecutive rounds with *msa*.

When using `--clustering-method ML`, parsimony informative positions are extracted from the trimmed alignments and concatenated into single continuous sequence per genome. IQ-tree infers the ML tree with minimum of 1000 bootstrap iterations.

The `--clustering-method NJ` method counts the total and shared number of variable sites between two genomes in the alignment and calculates a Jaccard distance (0-1):

$$D_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Required software

Please cite the appropriate tool(s) when using the core phylogeny in your research.

- MAFFT
- IQ-tree (Only required for ML)

Required arguments

`--database-path`/`-dp` Path to the database.

Optional arguments

`--homology-groups`/`-hm` A file with homology group node identifiers of single copy groups. Default is single_copy_orthologs.csv, generated in the previous *gene_classification* run.

`--clustering-method ML`/`--clustering-method NJ` Maximum likelihood (default) or Neighbour joining.

`--mode protein` Use proteins instead of nucleotide sequences.

`--threads`/`-tn` Number of threads (default is 1).

`--phenotype`/`-ph` Include phenotype information in the resulting phylogeny.

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

Example commands

```
$ pantools core_phylogeny -dp tomato_DB -tn 24
$ pantools core_phylogeny -dp tomato_DB -tn 24 --clustering-method NJ --mode protein
$ pantools core_phylogeny -dp tomato_DB -tn 24 --clustering-method ML --phenotype␣
↪resistance
```

Output

Output files are written to the **core_snp_tree** directory in the database.

- **sites_per_group.csv**, number of parsimony informative and variable sites per homology group.

When `--clustering-method NJ` is included

- **core_snp_NJ_tree.R**, Rscript to create NJ tree from distances based on shared sites. Two distances can be selected, based on variable sites and parsimony informative sites.
- **shared_informative_positions.csv**, table with total number of shared parsimony informative sites between genomes.
- **shared_variable_positions.csv**, table with total number of shared variable sites between genomes.

When `--clustering-method ML` is included

- **informative.fasta**, nucleotides from parsimony informative sites of the alignments, concatenated into a single sequences per genomes.
- **variable.fasta**, nucleotides from variable sites of the alignment, concatenated into a single sequences per genomes.

A command is generated which can be used to execute IQ-tree and infer the phylogeny on **informative.fasta**.

- **informative.fasta.iqtree**, IQ-tree log file.
- **informative.fasta.treefile**, the ML phylogeny.
- **informative.fasta.splits.nex**, the splits graph. With ideal data, this file is a tree, whereas data with conflicting phylogenetic signals will result in a tree-like network. This type of tree/network can be visualized with a tool like SplitsTree

6.4.2 K-mer distance tree

A NJ phylogeny of *k*-mer distances can be created by executing the Rscript generated by *k-mer_classification*.

Three types of distances can be selected to infer the phylogeny. The first two distances are Jaccard distances (0-1): one considering only distinct *k*-mers and the other using all *k*-mers. The distance from distinct *k*-mers ignores additional copies of a *k*-mer.

$$D_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

$$D_J(A,B) = 1 - J(A,B) = \frac{|A \uplus B| - |A \cap B|}{|A \uplus B|}$$

We observed an exponential increase in the *k*-mer distance as the evolutionary distance between two genomes increases. So in the case of more distant genomes, the depicted clades are still correct but the extreme long branch lengths make the tree hard to decipher. To normalize the numbers, we implemented the MASH distance. Distance = 1/ * ln(J), where k is the *k*-mer length; J is the jaccard index (of distinct *k*-mers).

```
$ Rscript genome_kmer_distance_tree.R
```

Output file

The phylogenetic tree **genome_kmer_distance_tree.tree** is written to the *kmer_classification* directory in the database.

6.4.3 Consensus tree

Create a consensus tree by combining gene trees from homology groups using ASTRAL-Pro. Gene trees are created from all sequences in an homology groups, no genomes can be skipped.

Required software

Please cite MAFFT, FastTree and ASTRAL-Pro when using the consensus tree in your research.

- MAFFT
- FastTree
- ASTRAL-Pro

Required arguments

`--database-path`/`-dp` Path to the database.

Optional arguments

`--threads`/`-tn` Number of threads (default is 1).

`--homology-groups`/`-hm` A file with homology group node identifiers. Default is all_homology_groups.csv, generated in the previous *gene_classification* run.

Example commands

```
$ pantools consensus_tree -dp apple_DB -tn 24
$ pantools consensus_tree -dp apple_DB -tn 24 -hm apple_DB/gene_classification/group_
↪identifiers/all_homology_groups.csv
$ pantools consensus_tree -dp apple_DB -tn 24 -hm apple_DB/gene_classification/group_
↪identifiers/core_homology_groups.csv
$ pantools consensus_tree -dp apple_DB -tn 24 -hm apple_DB/gene_classification/group_
↪identifiers/accessory_homology_groups.csv
```

Output

Output files are written to the **consensus_tree** directory in the database.

- **all_trees.hmgroups.newick**, all gene trees of homology groups included in the analysis, combined into a single file.
- **consensus_tree.astral-pro.newick**, the output consensus tree from ASTRAL-Pro.

Relevant literature

- ASTRAL-Pro: quartet-based species-tree inference despite paralogy. Molecular biology and evolution

---

6.4.4 Gene distance tree

A NJ phylogeny of gene distances is created by executing the Rscript generated by *gene_classification*.

Shared genes between genomes are identified through homology groups. Two Jaccard distance (0-1) can be used to infer a tree: one considering only distinct genes and the other using all genes. The distance from distinct genes ignores additional gene copies in an homology group.

$$D_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

$$D_J(A,B) = 1 - J(A,B) = \frac{|A \uplus B| - |A \cap B|}{|A \uplus B|}$$

```
$ Rscript gene_distance_tree.R
```

Output file

The phylogenetic tree **gene_distance_tree.tree** is written to the *gene_classification* directory in the database.

6.4.5 ANI tree

Average Nucleotide Identity (ANI) is a measure of nucleotide-level genomic similarity between the coding regions of two prokaryotic genomes. Two very fast ANI estimation tools (**fastANI** and **MASH**) are implemented and are able to perform the pairwise comparisons between genomes in the pangenome. To convert the ANI score into a distance (0-1), the scores are transformed by $1 - (ANI/100)$.

Required software

The required software depends on the tool you want to use. Please cite the appropriate tool when using the ANI tree in your research.

- fastANI
- MASH

Required argument

`--database-path`/`-dp` Path to the database.

Optional arguments

`--mode mash`/`--mode fastani` Software to calculate ANI score (default is MASH)
`--phenotype`/`-ph` Include phenotype information in the phylogeny.
`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.
`--threads`/`-tn` Number of threads used by FastANI (default is 1). MASH is single threaded (and currently not parallelized yet).

Example command

```
$ pantools ani -dp pecto_DB
$ pantools ani -dp pecto_DB --phenotype species_name --mode fastani
$ pantools ani -dp pecto_DB --skip 4,5,6 --mode mash
```

Output

Output files are written to the **ANI** directory in the database.

- **ANI_scores.csv**, a table with ANI scores for all genome pairs.
- **ANI_distance_matrix.csv**, a table with the ANI distances (1-ANI). This matrix is read by ANI_tree.R.
- **ANI_tree.R**, Rscript to generate NJ tree from ANI distances

Find closest typestrain

Compares bacterial strains to the typestrain when this information is available in a pangenome database.

1. Add the 'typestrain' phenotype to the pangenome with *add_phenotypes*. You only have to include typestrains names, other genomes can be left empty as shown in the example below, five genomes with three different typestrains.
2. Run the **ANI** function
3. The 'typestrain' phenotype is recognized, and **typestrain_comparison.csv** is created. This file contains the highest score of each genome(5) against all the included typestrains and states whether the score is above 95%.

```
Genome,typestrain
1,Salmonella choleraesuis NCTC 5735
2,Salmonella enteritidisi NCTC 12694
3,
4,Salmonella paratyphi NCTC 5702
5,
```

Relevant literature

- High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries
- Mash: fast genome and metagenome distance estimation using MinHash

6.4.6 MLSA

Within PanTools you can perform a Multilocus sequence analysis (**MLSA**) by running three consecutive functions:

1. *mlsa_find_genes*
2. *mlsa_concatenate*
3. *mlsa*

Step 1 Search for genes

Find your genes of interest in the pangenome and extract their nucleotide and protein sequence. A regular search is not case sensitive but the gene names must exactly match the given input name. For example, searching a gene with 'sonic1' as query will not be able find 'sonic', but is able to find Sonic1, SONIC1 or sOnIc1. Including the `--mode extensive` argument allows a more relaxed search and using 'sonic' will now also find gene name variations as 'sonic1', 'sonic3' etc.. For this function it is important that genomes are annotated by a method that follow the rules for genetic nomenclature, so there are no differences in the naming of genes.

To gain insight in which genes are appropriate for this analysis, run *gene_classification* with the `--mode mlsa` argument. This method creates a list of genes that have same gene name, are present in all (selected) genomes and are placed in the single-copy homology group. Using genes from this list guarantees a successful MLSA.

Possible generated warnings during gene search

When a gene is included that is not on the list of suitable genes, it is not necessarily unusable but possibly requires manual . This function generates a log file with the issues and explains the user what to do.

- Gene is not found in every genome. Consider using `--mode extensive`. The gene is not suitable with the current genome selection when this argument was already included.
- The found genes are placed in different homology groups. A directory named the gene name is created where sequences are stored in a separate file per homology group. When one of the groups is single copy orthologous, it is automatically selected. With multiple correct single-copy groups, the first is selected. If no single-copy groups are found, this gene is probably not a suitable candidate based on the high divergence. If you are determined to use the gene, align and infer a gene tree on **all_sequences.fasta** to identify appropriate sequences.
- At least one gene has an additional copy. The extra copies must be removed from the output file if you want to include this gene in the analysis. Find the copies that stand out by aligning and inferring a gene tree of the homology group.

Required arguments

`--database-path`/`-dp` Path to the database.
`--name` One or multiple gene names, seperated by a comma.

Optional arguments

`--mode extensive` Perform a more extensive gene search.
`--skip`/`-sk` Do not search for genes in this selection of genomes.
`--reference`/`-ref` Only search for genes in a selection of genomes.

Example command

```
$ pantools mlsa_find_genes -dp bacteria_DB --name dnaX,gapA,recA
$ pantools mlsa_find_genes -dp bacteria_DB --name gapA --mode extensive
```

Output

Output files are written to the **mlsa/input/** directory in the database. For each gene name that was included, a nucleotide and protein and FASTA file is created that holding the sequences found in all genomes.

- **mlsa_find_genes.log**, when one or multiple warnings are given they are placed in this log file. File is not created when there aren't any warnings.

Step 2 Concatenate genes

Concatenate sequences obtained by *mlsa_find_genes* into a single sequence per genome. The `--name` argument is required, but the selection of gene names is allowed to be a sub-selection of the earlier selection.

1. Proteins are aligned with MAFFT
2. The longest gap at the start and end of each protein alignment is identified.
3. Nucleotide sequences are trimmed accordingly
4. Trimmed nucleotide sequence are concatenated into a single sequence per genome.

Required software

- MAFFT

Required arguments

`--database-path`/`-dp` Path to the database.
`--name` One or multiple gene names, seperated by a comma.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`ref` Only include a of genomes.
`--threads`/`-tn` Number of threads for MAFFT (default is 1).

Example command

```
$ pantools mlsa_concatenate -dp bacteria_DB --name dnaX,gapA
$ pantools mlsa_concatenate -dp bacteria_DB --name dnaX,gapA,recA --skip 1,2,10-25
```

Output

The output file is stored in */database_directory/mlsa/input/*

- **concatenated.fasta**, file holding one concatenated sequence per genome.

Step 3 Run MLSA

Run MAFFT and IQ-tree on the concatenated nucleotide sequences from *mlsa_concatenate* to create an unrooted ML tree with 1,000 bootstrappings.

Required software

Please cite the MAFFT and IQ-tree when using the MLSA in your research.

- MAFFT
- IQ-tree

Required argument

`--database-path`/`-dp` Path to the database.

Optional arguments

`--threads`/`-tn` Select number of threads for MAFFT and IQ-tree (default is 1).

`--phenotype`/`-ph` Add phenotype information/values to the phylogeny. Allows the identification of phenotype specific SNPs in the alignment.

Example commands

```
$ pantools mlsa -dp bacteria_DB
$ pantools mlsa -dp bacteria_DB -tn 24 -ph species
```

Output

Input and output files are written to the **mlsa/output/** directory in the database.

- **mlsa.afa**, the alignment in CLUSTAL format.
- **mlsa.fasta**, the alignment in FASTA format.
- **mlsa.fasta.treefile**, the (ML) phylogeny created by IQ-tree in Newick format.

When a `--phenotype` is included

- **nuc_phenotype_specific_changes.info**, the positions of phenotype specific substitutions in the alignment.

The *var_inf_positions* directory holds files related to the counting variable positions of the alignment.

- **nuc_variable_positions.csv**, a table with the counts of A, T, C, G, or gap for every variable position in the alignment
- **informative_nuc_distance.csv**, a table with distances calculated from parsimony **informative** positions in the alignment.
- **informative_nuc_site_counts.csv**, a table with number of shared parsimony informative positions between genomes.
- **variable_nuc_distance.csv**, a table with distances calculated from **variable** positions in the alignment.
- **variable_nuc_site_counts.csv**, a table with number of shared positions between genomes.

6.4.7 Edit Phylogeny

Rename phylogeny

Update or the terminal nodes (leaves) of a phylogenic tree. This is useful when you already constructed a tree but forgot to include a phenotype or to update the tree with a different phenotype. When no `--phenotype` is included, the node values are changed to genome numbers.

Required arguments

`--database-path`/`-dp` Path to the database.

`--input-file`/`-if` A phylogenetic tree in **newick** or **nexus** format. The tree must be generated by PanTools.

Optional arguments

`--phenotype`/`-ph` The phenotype used to rename the terminal nodes (leaves) of selected tree. `--mode no-numbers` Exclude genome numbers from the terminal nodes (leaves).

Example command

```
$ pantools rename_phylogeny -dp bacteria_DB -if core_snp.tree
$ pantools rename_phylogeny -dp bacteria_DB --phenotype species -if bacteria_DB/ANI/
↪fastANI/ani.tree
```

Output file

A new phylogenetic tree is written to the directory of the selected input tree:

- When the original file is called '*old_tree.newick*', a new tree is created with filename '*old_tree_RENAMED.newick*'.

Reroot phylogeny

All phylogenetic trees that come from the PanTools functionalities are unrooted. This function is able to create a new rooted tree simply by selecting one of the external (terminal) nodes via `--value`. The included number or string should match exactly one node in the phylogeny or the program will not execute.

Required software

- ape 5

Required arguments

`--input-file`/`-if` A phylogenetic tree in **newick** format. The tree must be generated by PanTools.

`--value` The name of the terminal node that will root the tree.

Example command

```
$ pantools reroot_phylogeny -if core_snp.tree --value 1
$ pantools reroot_phylogeny -if core_snp.tree --value 1_A.thaliana
$ pantools reroot_phylogeny -if kmer.tree --value 1_1
$ Rscript reroot.R
```

Output file

A new phylogenetic tree is written to the same location as the provided input file

- When the original tree is called '*tree.newick*', the new file is named '*tree_REROOTED.newick*'.

Create tree template

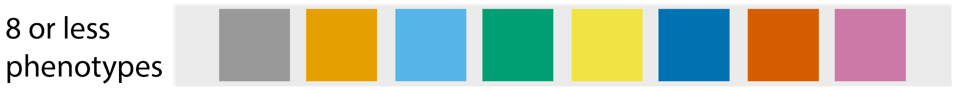
Creates 'ring' and 'colored range' ITOL templates based on phenotypes for the visualization of phylogenies in iTOL. Phenotypes must already be included in the pangenome with the *add_phenotypes* functionality. How to use the template files in iTOL can be found in one of the *tutorials*.

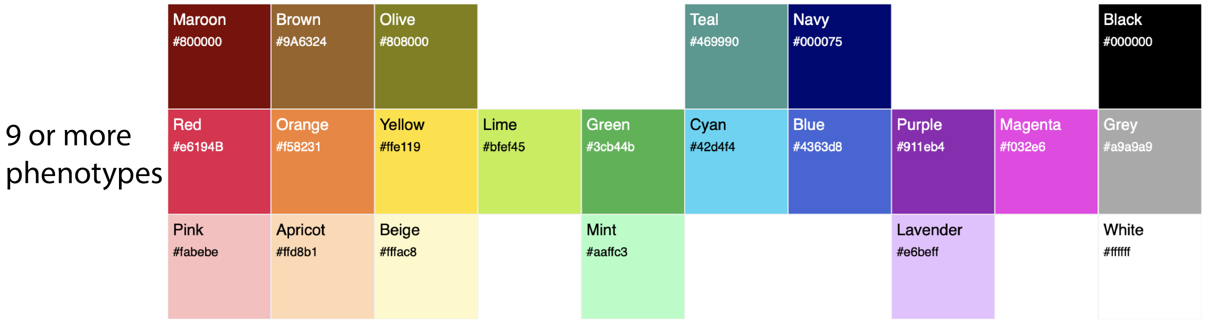
If you run this function without a `--phenotype` argument, templates are created for trees that contain only genome numbers as node labels. When there is a `--phenotype` included, templates are created where the leaves are named according to the selected phenotype but are coloured by one of the other phenotypes in the pangenome. For example, you originally used the 'species name' as a phenotype to construct the phylogeny but want them to be coloured by the 'pathogenicity' phenotype.

More information about ITOL templates can be found on their own webpage.

There is a maximum of 20 possible colors that are used in the following order:

	Color (> 8 phenotypes)	Hexadecimal color	Color (8 phenotypes)	Hexadecimal color
1	Pink	#fabebe	Orange	#E69F00
2	Lime	#bfef45	Sky blue	#56B4E9
3	Cyan	#42d4f4	Bluish green	#009E73
4	Apricot	#ffd8b1	Yellow	#F0E442
5	Mint	#aaffc3	Blue	#0072B2
6	Beige	#fffac8	Vermilion	#D55E00
7	Lavender	#e6beff	Reddish purple	#CC79A7
8	Teal	#469990	Grey	#999999
9	Red	#e6194B		
10	Orange	#f58231		
11	Yellow	#ffe119		
12	Green	#3cb44b		
13	Blue	#4363d8		
14	Purple	#911eb4		
15	Grey	#a9a9a9		
16	Maroon	#800000		
17	Olive	#808000		
18	Brown	#9A6324		
19	Navy	#000075		
20	Magenta	#f032e6		





Figures were copied from:

http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/
https://sashamaps.net/docs/resources/20-colors/

Required argument

`--database-path`/`-dp` Path to the database.

Optional argument

`--phenotype`/`-ph` Use the names from this phenotype.
`--value 1` Assign a color to phenotypes shared by only a single genome. If not set, default is a minimum of two genomes.

Example command

```
$ pantools create_tree_template -dp bacteria_DB
$ pantools create_tree_template -dp bacteria_DB --phenotype flowering --value 1
$ pantools create_tree_template -dp bacteria_DB --phenotype root_morph --value 3
```

Output

Output files are written to the **create_tree_template** directory in the database.

- When **no** phenotype information is included, a directory 'genome_numbers' is created where the templates are stored.
- When a `--phenotype` is included, a directory (named after the phenotype) is created where the templates are stored.

The template files are named after the phenotypes, therefore the colors are based on that phenotype as well.

6.5 Multiple Sequence Alignments

This page is entirely dedicated to performing Multiple Sequence Alignments (MSA) with PanTools.

6.5.1 Sequence alignments

Alignment of homology groups

Performs multiple sequence alignments with **MAFFT** on sets of sequences. These alignments can either be:

- per homology group
- multiple homology groups
- regions
- with all sequences containing a functional domain

The alignment consists of two rounds: After the initial alignment, protein sequences are trimmed based on the longest start and end gap of the alignment. The number of trimmed amino acids is multiplied by three to trim the correct number of nucleotides. If only nucleotide sequences are aligned, the nucleotide sequence alignment is used for trimming. The trimmed sequences are aligned a second time to identify variable and parsimony informative sites. For each round, a ML phylogeny will be created with **FastTree**.

Select a method

By default, this function will make a MSA per homology group. (It can still be specified with `--method per_group`.) Using another option from the list above requires use of the `--method` argument. For aligning multiple homology groups, please use `--method multiple_groups` with the homology groups specified in a csv file on a single line (can be added with `-hm /path/to/hm.csv`. For aligning regions, please use `--method regions` with a regions file that is added with `-rf /path/to/rf.txt`. For aligning sequences based on a functional domain, please use `--method functions` together with a functional domain that is added with `--name <domain>`.

Other options

In case you are only interested in the alignment of the nucleotide or protein sequences, use `--mode nucleotide` or `--mode protein`. When the `--no-trimming` argument is included, the variable and parsimony informative sites are identified from the initial alignment and no trimming is performed (thus, only one round of aligning). The option `--fast` can be used to skip running FastTree, which is used for generating a tree from the alignment.

Identify phenotype shared or specific variation

Shared SNPs or amino acid substitutions can be found among the members of a phenotype when `--phenotype <phenotype>` is included. As homology groups can highly differ in size, the threshold for a phenotype shared or specific SNP/substitution is based on the number of sequences (from a certain phenotype) of an homology group instead of the number of genomes in the pangenome. For example, the pangenome holds 500 genomes but the homology group consists of only 100 sequences. The threshold can be lowered by including `--phenotype-threshold <threshold>`, which lowers the original threshold by multiplying it to a given percentage.

Sequence identity and similarity

- The percentage **identity** of two sequences is calculated based on the number of exactly matching characters divided by the alignment length minus the positions were both sequences have a gap.

- The **similarity** (protein only) is calculated from the number of identical matches, increased by the number of similar amino acids (according to the BLOSUM 62 matrix), divided by the alignment length minus the shared gap positions. The calculated percentage of similarity is dependant on the BLOSUM matrix set by `--blosum`. Choose a larger BLOSUM number BLOSUM less divergent sequences.

Required software

- MAFFT
- FastTree

Required arguments

`--database-path`/`-dp` Path to the database

Optional arguments

`--method` The kind of alignment to make. Can be either `per_group`, `multiple_groups`, `regions` or `functions`.
`--homology-groups`/`-hm` Text file with homology group node identifiers. Default is **all** groups!
`--phenotype`/`-ph` a phenotype name, used to identify phenotype specific SNPs/substitutions.
`--phenotype-threshold` Threshold for phenotype specific SNPs (default is 100%).
`--skip` and `--reference`/`-ref` Skip over a selection of genomes.
`--threads-number`/`-tn` The number of parallel working threads for MAFFT and FastTree (Highly recommended! default is 1).
`--mode nucleotide` or `--mode protein` Choose to only align **nucleotide** or **protein** sequences (default is both).
`--no-trimming` Align the sequences only once.
`--fast` Don't run FastTree.
`--name` For specifying one or multiple functional domains. (Only used when `--method functions`.)
`--regions-file`/`-rf` Regions file for aligning regions. (Only used when `--method regions`.)
`--blosum` a BLOSUM matrix number to control MAFFT's sensitivity and the similarity calculation. Allowed values: 45, 62 (default), 80.

Example regions file

Each line must have a genome number, sequence number, begin and end positions that are separated by a space. Place a minus symbol behind a region to extract the reverse complement sequence.

```
1 1 1 10000
195 1 477722 478426
71 10 17346 18056 -
138 47 159593 160300 -
```

Example commands

```
$ pantools msa -dp tomato_DB
$ pantools msa -dp tomato_DB -hm hmgroups.txt --mode protein
$ pantools msa -dp tomato_DB -hm hmgroups.txt --mode nucleotide --no-trimming
$ pantools msa -dp tomato_DB -hm hmgroups.txt --phenotype resistance --phenotype-
↪threshold 99
$ pantools msa --method multiple_groups -dp tomato_DB
$ pantools msa --method multiple_groups -dp tomato_DB -hm hmgroups.txt --mode protein
$ pantools msa --method multiple_groups -dp tomato_DB -hm hmgroups.txt --phenotype␣
↪resistance --phenotype-threshold 95
$ pantools msa --method regions -dp tomato_DB -rf regions.txt
$ pantools msa --method functions -dp tomato_DB --name PF10137
```

Output files

Output files are stored in *database_directory/alignments/msa_/grouping_v?/* A separate directory is created for each alignment which holds the input and output files.

The 'input' directory contains the input files for the alignments.

- **nuc/prot(_trimmed).fasta**, original and trimmed input sequences.
- **trimmed.info**, number of trimmed positions per sequence.
- **sequences.info**, relevant gene information of sequences in group: gene names, mRNA node id, address, strand orientation.

The alignments and output files are written to the 'output' directory.

- **nuc/prot(_trimmed).afa**, the initial and second (trimmed) alignment in CLUSTAL format.
- **nuc/prot(_trimmed).fasta**, the initial and second (trimmed) alignment in FASTA format.
- **nuc/prot(_trimmed).newick**, FastTree ML tree inferred from the initial and second (trimmed) alignment.
- **nuc/prot(_trimmed)_alignment.info**, some statistics about the initial and second (trimmed) alignment: alignment length, number of conserved, variable and parsimony informative sites

Sequence identity and similarity output files.

- **nuc/prot(_trimmed)_identity.csv**, table with the sequence identity scores.
- **prot(_trimmed)_similarity.csv**, table with similarity of the protein sequences.

Variable and parsimony informative sites output files.

- **informative_nuc/prot(_trimmed)_distance.csv**, table with distances between sequences based on parsimony informative sites in the alignment.
- **variable_nuc/prot(_trimmed)_distance.csv**, table with distances between sequences based on variable sites in the alignment.
- **informative_nuc/prot(_trimmed)_sites.csv**, table with the number shared parsimony informative sites between sequences.
- **variable_nuc/prot(_trimmed)_sites.csv**, table with the number of shared variable sites between sequences.

When a `--phenotype` is included.

- **phenotype_specific_changes_nuc/prot_groups.csv**, the node identifiers of homology groups with phenotype specific substitutions.
- **phenotype_specific_changes_nuc/prot.txt**, the positions of phenotype specific substitutions in the alignments.
- **phenotype_disrupted_nuc/prot.txt**, shows how many sequences of different phenotypes prevented a SNP/substitution from becoming phenotype specific.

6.6 Explore the pangenome

The functionalities on this page allow to actively explore the pangenome.

- Retrieve regions from the pangenome
- Retrieve sequences and functional annotations from homology groups
- Search for genes using a gene name, functional annotation or database node identifier
- Align homology groups or genomic regions
- GO enrichment analysis

6.6.1 Gene locations

Identify and compare gene clusters of neighbouring genes based on a set of homology groups. First, identifies the genomic position of genes in homology groups, retrieves the order of genes per genome and based on this construct the gene clusters. If homology groups with multiple genomes were selected, the gene cluster composition is compared between genomes. When a `--phenotype` is included, gene clusters can be found that only consist of groups of a certain phenotype.

For example, 100 groups were predicted as core in a pangenome of 5 genomes. The gene clusters are first identified per genome, after which it compares the gene order of one genome to all the other genomes. The result could be 75 groups with genes that are not only homologous but also share their gene neighbourhood. Another example, when accessory (present 2 in to 4 genomes) groups are given to this function in combination with a `--phenotype` (assigned to only two genomes), the function can return clusters that can only be found in the phenotype members.

Required arguments

`--database-path`/`-dp` Path to the database.

`--homology-groups`/`-hm` A text file with homology group node identifiers, seperated by a comma.

Optional arguments

`--phenotype`/`-ph` A phenotype name, used to identify gene clusters shared by all phenotype members.

`--value` The number of allowed nucleotides between two neighbouring genes (default is 1 MB).

`--gap-open`/`-go` When constructing the clusters, allow a number of genes for each cluster that are not originally part of the input groups (default is 0).

`--core-threshold` Lower the threshold (%) for a group to be considered (soft) core (default is the total number of genomes found in the groups, not a percentage).

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

`--mode ignore-copies` Duplicated and co-localized genes no longer break up clusters.

Example command

```
$ pantools locate_genes -dp tomato_DB -hm phenotype_groups.csv
$ pantools locate_genes -dp tomato_DB -hm unique_groups.csv --value 5000 -go 1
$ pantools locate_genes -dp tomato_DB -hm accessory_groups.csv --core-threshold 95 -go 1
```

Output files

Output files are stored in *database_directory/locate_genes/*

- **gene_clusters_by_position.txt**, the identified gene clusters ordered by their position in the genome.
- **gene_clusters_by_size.txt**, the identified gene clusters ordered from largest to smallest.
- **compare_gene_clusters**, the composition of found gene clusters is compared to the other genomes. For each cluster, it shows which parts match other clusters and which parts do not. The file is not created when homology groups only contain proteins of a single genome (unique).

When a `--phenotype` is included

- **phenotype_clusters**, homology group node identifiers from phenotype shared and specific clusters.
- **compare_gene_clusters_PHENOTYPE.txt**, the same information as **compare_gene_clusters** but now the gene cluster comparison is only done between phenotype members.

6.6.2 Find genes

Find genes by name

Find your genes of interest in the pangenome by using the gene name and extract the nucleotide and protein sequence. To be able to find a gene, every letter of the given input must match a gene name. The search is not case sensitive. Performing a search with 'sonic1' as query will not be able find 'sonic', but is able to find Sonic1, SONIC1 or sOnIc1. Including the `--mode 1` argument allows a more relaxed search and using 'sonic' will now also find gene name variations as 'sonic1', 'sonic3' etc..

Be aware, for this function to work it is important that genomes are annotated by a method that follows the rules for genetic nomenclature. Gene naming can be inconsistent when different tools are used for genome annotation, making this functionality ineffective.

This function is the same as *mlsa_find_genes* but uses a different output directory. Several warnings (shown in the other manual) can be generated during the search. These warning are less relevant for this function as the genes are not required to be single copy-orthologous.

Required arguments

`--database-path`/`-dp` Path to the database.
`--name` One or multiple gene names, seperated by a comma.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Exclude a selection of genomes.
`--mode extensive` Perform a more extensive gene search.

Example command

```
$ pantools find_genes_by_name -dp tomato_DB --name dnaX,gapA,recA
$ pantools find_genes_by_name -dp tomato_DB --name gapA --mode extensive
```

Output files

Output files are stored in */database_directory/find_genes/by_name/*. For each gene name that was included, a nucleotide and protein and .FASTA file is created with sequences found in all genomes.

- **find_genes_by_name.log**, relevant information about the extracted genes: node identifier, gene location, homology group etc..

Find genes by annotation

Find genes of interest in the pangenome that share a functional annotation node and extract the nucleotide and protein sequence.

Required arguments

`--database-path`/`-dp` Path to the database.

Requires either **one** of the following arguments

`--node` One or multiple identifiers of function nodes (GO, InterPro, PFAM, TIGRFAM), seperated by a comma.
`--name` One or multiple function identifiers (GO, InterPro, PFAM, TIGRFAM), seperated by a comma.

Optional arguments

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.

Example command

```
$ pantools find_genes_by_annotation -dp tomato_DB --node 14928,25809
$ pantools find_genes_by_annotation -dp tomato_DB --name PF00005,GO:0000160,IPR000683,
↪TIGR02499
```

Output files

Output files are stored in */database_directory/find_genes/by_annotation/*. For each function (node) that was included, a nucleotide and protein and .FASTA file is created with sequences from the genes that are connected to the node.

- **find_genes_by_annotation.log**, relevant information about the extracted genes: node identifier, gene location, homology group etc..

Find genes in region

Find genes of interest in the pangenome that can be (partially) found within a given region (partially). For each found gene, relevant information, the nucleotide sequence and protein sequence is extracted.

Required arguments

`--database-path`/`-dp` Path to the database.

`--regions-file`/`-rf` A text file containing genome locations with on each line: a genome number, sequence number, begin and end position, separated by a space.

Optional arguments

`--mode partial` Also retrieve genes that only partially overlap the input regions.

Example input file

Each line must have a genome number, sequence number, begin and end positions that are separated by a space.

```
195 1 477722 478426
71 10 17346 18056
138 47 159593 160300
```

Example command

```
$ pantools find_genes_in_region -dp tomato_DB -rf regions.txt
$ pantools find_genes_in_region -dp tomato_DB -rf regions.txt --mode partial
```

Output files

Output files are stored in */database_directory/find_genes/in_region/*. For each region that was included, a nucleotide and protein and .FASTA file is created with sequences from the genes that are found within the region.

- **find_genes_in_region.log**, relevant information about the extracted genes: node identifier, gene location, homology group etc..

6.6.3 Functional annotations

The following functions can only be used when any type of functional annotation is *added to the database*.

Show GO

For a selection of '**GO**' nodes, retrieves connected 'mRNA' nodes, child and all parent GO terms that are higher in the GO hierarchy. This function follows the 'is_a' relationships of GO each node to their parent GO term until the 'biological process', 'molecular function' or 'cellular location' node is reached. This can be is useful in case InterProScan annotations were included, as these only add the most specific GO terms of the hierarchy to a sequence.

Required arguments

`--database-path`/`-dp` Path to the database

Requires either **one** of the following arguments

`--node` One or multiple identifiers of 'GO' nodes, seperated by a comma.
`--name` One or multiple GO term identifiers, seperated by a comma.

Example commands

```
$ pantools show_go -dp tomato_DB --node 15078,15079
$ pantools show_go -dp tomato_DB --name GO:0000001,GO:0000002,GO:0008982
```

Output file

- **show_go.txt**, information of the selected GO node(s): the connected 'mRNA' nodes, the GO layer below, and all layers above.

Compare GO

Check if and how similar two given GO terms are. For both nodes, follows the 'is_a' relationships up to their parent GO terms until the 'biological process', 'molecular function' or 'cellular location' node is reached. After all parent terms are found, the shared GO terms and their location in the hierarchy is reported.

Required arguments

`--database-path`/`-dp` Path to the database.

Requires either **one** of the following arguments

`--node` Two node identifiers of 'GO' nodes, seperated by a comma.

`--name` Two GO identifiers, seperated by a comma.

Example command

```
$ pantools compare_go -dp tomato_DB --name GO:0032775,GO:0006313
$ pantools compare_go -dp tomato_DB --node 741487,741488
```

Output file

Output files are stored in *database_directory/function/*

- **compare_go.txt**, information of the two GO nodes: the connected 'mRNA' nodes, the GO layer below, all layers above and the shared GO terms between the two nodes.

6.6.4 Homology group information

Report all available information of one or multiple homology groups.

Required arguments

`--database-path`/`-dp` Path to the database.

`--homology-groups`/`-hm` A text file with homology group node identifiers, seperated by a comma

Optional arguments

`--label` Name of function identifiers from GO, PFAM, InterPro or TIGRAM. To find Phobius (P) or SignalP (S) annotations, include: ‘secreted’ (P/S), ‘receptor’ (P/S), or ‘transmembrane’ (P).

`--name` One or multiple gene names, seperated by a comma.

`--skip`/`-sk` Exclude a selection of genomes.

`--reference`/`-ref` Only include a selection of genomes.

Example command

```
$ pantools group_info -dp yeast_DB -hm core_groups.txt
$ pantools group_info -dp yeast_DB -hm core_groups.txt --label GO:0032775,GO:0006313 --
↪name budC,estP
```

Output files

Output files are stored in *database_directory/alignments/grouping_v?/groups/*. For each homology group that was included, a nucleotide and protein and .FASTA file is created with sequences found in all genomes.

- **group_info.txt**, relevant information for each homology group: number of copies per genome, gene names, mRNA node identifiers, functions, protein sequence lengths, etc..
- **group_functions.txt**, full description of the functions found in homology groups

When function identifiers are included via `--label`

- **groups_with_function.txt**, homology group node identifiers from groups that match one of the input functions.

When gene names are included via `--name`

- **groups_with_name.txt**, homology group node identifiers from groups that match one of the input gene ames.

6.6.5 Sequence alignments

The manual for PanTools’ sequence alignment functionalities moved to a standalone page - *Multiple Sequence Alignments*.

6.6.6 Matrix files

Several functions generate tables in a CSV file format. as tables that the following functions can work with. For example, ANI scores, *k*-mer and gene distance used for constructing the Neighbour Joining *phylogenetic trees*, and the identity and protein sequence similarity tables created by the *alignment functions*.

Order matrix

Transforms the CSV table to easy to read file by ordering the values in ascending order from low to high or descending order when `--mode desc` is included in the command. If phenotype information is included in the header, a separate file with the range of found values is created for each phenotype. If this information is not present (only genome numbers in the header), use *rename_matrix* to change the headers.

Required argument

`--database-path`/`-dp` Path to the database.
`--input-file`/`-af` A CSV formatted matrix file.

Optional argument

`--skip`/`-sk` Skip over the values of a selection of genomes.
`--reference`/`-ref` Only include the values from a selection of genomes.
`--mode asc` or `--mode desc` Order the matrix in ascending or descending order (ascending is default).

Example command

```
$ pantools order_matrix -dp bacteria_DB -if bacteria_DB/ANI/fastANI/ANI_distance_matrix.
↪csv
$ pantools order_matrix -dp bacteria_DB -if bacteria_DB/ANI/fastANI/ANI_distance_matrix.
↪csv --mode desc
```

Output file

Output is written to the same directory as the selected input file

- '*old file name*' + '*_ORDERED*', ordered values of the original matrix file.

When phenotype information is present in the header

- '*old file name*' + '*_PHENOTYPE*', range of values per phenotype.

Rename matrix

Rename the headers (first row and leftmost column) of CSV formatted matrix files. If no `--phenotype` is included, headers are changed to only contain genome numbers.

Required arguments

`--database-path`/`-dp` Path to the database.
`--input-file`/`-af` a matrix file with numerical values.

Optional arguments

`--phenotype`/`-ph` A phenotype name, used to include phenotype information into the headers.
`--skip`/`sk` Exclude a selection of genomes from the new matrix file. `--reference`/`-ref` Only include a selection of genomes in the new matrix file.
`--mode no-numbers` Exclude genome numbers from the headers.

Example command

```
$ pantools rename_matrix -dp pecto_DB -phenotype species -if pecto_DB/ANI/fastANI/ANI_
↪distance_matrix.csv
```

Output file

Output is written to the same directory as the selected input file.

- '*old file name*' + '*_RENAMED*', the original matrix file with changed headers.

6.6.7 Retrieve regions, genomes or features

The two following functions allow users to retrieve genomic regions from the pangenome.

Retrieve regions

Retrieve the full genome sequence or genomic regions from the pangenome.

Required arguments

`--database-path`/`-dp` Path to the database.
`--regions-file`/`-rf` A text file containing genome locations with on each line: a genome number, sequence number, begin and end positions separated by a space.

Example command

```
$ pantools retrieve_regions -dp pecto_DB -rf regions.txt
```

Example input

To extract:

- Complete genome - Include a genome number
- An entire sequence - Include a genome number with sequence number
- A genomic region - Include a genome number, sequence number, begin and end positions that are separated by a space. Place a minus symbol behind the regions to extract the reverse complement sequence of the region.

```
1
1 1
1 1 1 10000
1 1 1000 1500 -
195 1 477722 478426
71 10 17346 18056 -
138 47 159593 160300 -
```

Output file

A single FASTA file is created for all given locations and is stored in the database directory.

Retrieve features

To retrieve the sequence of annotated features from the pangenome.

Required arguments

`--database-path`/`-dp` Path to the database.
`--feature-type` or `-ft` The feature name; for example ‘gene’, ‘mRNA’, ‘exon’, ‘tRNA’, etc.

Optional arguments

Use one of the following arguments to limit the sequence retrieval to a selection of genomes.

`--skip`/`-sk` Exclude a selection of genomes.
`--reference`/`-ref` Only include a selection of genomes.

Example command

```
$ pantools retrieve_features -dp pecto_DB --feature-type gene
$ pantools retrieve_features -dp pecto_DB --ft mRNA
```

Output files

For each genome a FASTA file containing the retrieved features will be stored in the database directory. For example, genes.1.fasta contains all the genes annotated in genome 1.

6.7 Read mapping

6.7.1 Map

Map single or paired-end short reads to one or multiple genomes in the pangenome. One SAM or BAM file is generated for each genome included in the analysis.

Required arguments

`--database_path`/`-dp` Path to the pangenome database.
`-1` The first short-read archive in FASTQ format, which can be gz/bz2 compressed. This file can be precessed interleaved by -il option.
`--genome-numbers`/`-gn` A text file containing genome numbers to map reads against in each line.

Optional arguments

`-2` The second short-read archive in FASTQ format, which can be gz/bz2 compressed.
`--out-format`/`-of` `SAM` `BAM` `none` Writes the alignment files in BAM or SAM format or don't write any output files.
`--output-path`/`-op` (**default value**: Database path determined by *-dp*) : Path to the output files.
`--threads`/`-tn` (**default value**: 1) : The number of parallel working threads.
`--interleaved`/`-il` Process the fastq file as an interleaved paired-end archive.
`--raw-abundance-file`/`-raf` The *mapping_summary.txt* file from a previous mapping run (random-best competitive mode) for a better estimation of coverage in a metagenomic setting.
`--alignment-mode` or `-am` The alignment mode:

-1 : Competitive, none-bests
-2 : Competitive, random-best

-3 : Competitive, all-bests
1 : Normal, none-bests
2 : Normal, random-best (**default**)
3 : Normal, all-bests
0 : Normal, all-hits

Optional arguments that influence the mapping sensitivity

`--very-fast`/`--fast`/`--sensitive`/`--very-sensitive` Four settings that automatically set the parameters controlling the sensitivity, ranging from least to most sensitive.

`--min-mapping-identity*`/`-mmi` (**default value**: 0.5, **valid range**: [0..1)) : The minimum acceptable identity of the alignment.

`--num-kmer-samples`/`-nks` (**default value**: 15, **valid range**: [1..r-k+1]) : The number of kmers sampled from read.

`--min-hit-length`/`-mhl` (**default value**: 13, **valid range**: [10..100]) : The minimum acceptable length of alignment after soft-clipping.

`--max-alignment-length`/`-mal` (**default value**: 1000, **valid range**: [50..5000]) : The maximum acceptable length of alignment.

`--max-fragment-length`/`-mfl` (**default value**: 2000, **valid range**: [50..5000]) : The maximum acceptable length of fragment.

`--max-num-locations`/`-mnl` (**default value**: 15, **valid range**: [1..100]) : The maximum number of location of candidate hits to examine.

`--alignment-band`/`-ab` (**default value**: 5, **valid range**: [1..100]) : The length of bound of banded alignment.

`--clipping-stringency`/`-ci` (**default value**: 1) : The stringency of soft-clipping.

0 : no soft clipping
1 : low
2 : medium
3 : high

Example input files

FASTQ file

```
@SRR13153715.1 1/1
TGGTCATACAGCAAAGCATAATTGTCACCATTACTATGGCAATCAAGCCAGCTATAAAACCTAGCCAAATGTACCATGGCCATTTTATATACTGCTCATACTTT
+
EEEEEEEEEEEEEEEEAEEEE/EEEEE/AEEEEEEEEEEEEEE/EE/EEE/<EEEEEEE/
↪EEEEEEEEEEEEEAEEEEEAEEEEEAEEAEEEEEEEA<AAAEEAEEA<EE/EEEEAEAEA/EEAA/
```

Genome numbers file

```
1
2
5
```

Example commands

```
$ pantools map -dp arabidopsis_DB -1 ERR031564_1.fastq --reference 1-5
$ pantools map -dp arabidopsis_DB -1 ERR031564_1.fastq -gn genome_numbers.txt
$ pantools map -dp arabidopsis_DB -1 interleaved_reads.fastq --interleaved -gn genome_
↪numbers.txt
$ pantools map -dp arabidopsis_DB -1 ERR031564_1.fastq -2 ERR031564_2.fastq -gn genome_
↪numbers.txt
```

Output files

- **mapping_summary.txt**, number of mapped and unmapped reads per genome
- One SAM or BAM file is generated for each genome included in the analysis.

6.8 Querying the pangenome

Cypher is Neo4j's graph query language that lets you ask specific questions or retrieve data from the graph database. The Cypher query language depicts patterns of nodes and relationships and filters those patterns based on labels and properties. While using node and relationship patterns in databases queries may seem a little daunting, it is easy to pick up! This page contains some example queries to help you get started. Feel free to email us if you have any question regarding Cypher queries.

More information on Neo4j and the Cypher language:

Neo4j Cypher Manual v3.5
Neo4j Cypher Refcard
Neo4j API

Match and return 100 nucleotide nodes

```
MATCH (n:nucleotide) RETURN n LIMIT 100
```

Find all the genome nodes

```
MATCH (n:genome) RETURN n
```

Retrieve the pangenome node

```
MATCH (n:pangenome) RETURN n
```

Match and return 100 genes

```
MATCH (g:gene) RETURN g LIMIT 100
```

Match and return 100 genes and order them by length

```
MATCH (g:gene) RETURN g ORDER BY g.length DESC LIMIT 100
```

The same query as before but results are now returned in a table

```
MATCH (g:gene) RETURN g.name, g.address, g.length ORDER BY g.length DESC LIMIT 100
```

Return genes which are between 100 and 250 bp. This can also be applied to other features such as exons introns or CDS.

```
MATCH (g:gene) where g.length > 100 AND g.length < 250 RETURN * LIMIT 100
```

Find genes located on first genome

```
MATCH (g:gene) WHERE g.address[0] = 1 RETURN * LIMIT 100
```

Find genes located on first genome and first sequence

```
MATCH (g:gene) WHERE g.address[0] = 1 AND g.address[1] = 1 RETURN * LIMIT 100
```

Obtain genes between 100 and 250 nucleotides

```
MATCH (g:gene) where g.length > 100 AND g.length < 250 RETURN *
```

Return pfam identifiers for genes between 100 and 250 nucleotides long

```
match (n:mRNA)--(m:pfam) where n.length > 100 and n.length < 150 return m.id
```

Return all genes for a specific contig and count them

```
MATCH (n:gene) WHERE n.address[0] = 1 and n.address[1] = 1 RETURN count(n)
```

Return all genes genes between 1000-1500 nucleotides and order them by length

```
MATCH (n:gene) WHERE n.length > 1000 and n.length < 1500 RETURN n order by n.length DESC
```

Returns the homology group matching your gene of interest

```
MATCH (n:homology_group)--(m:mRNA)--(g:gene) WHERE g.name = 'GENE\_NAME' RETURN *
```

Returns the genes of genome 1 that don't have a homolog in a the other genome

```
MATCH (n:homology_group)--(m:mRNA)--(g:gene) where n.num_members = 1 and g.genome = 1␣
↪RETURN g
```

Retrieve unique GO identifiers for mRNA's with a signal peptide

```
MATCH (m:mRNA)--(g:GO) where m.signalp_signal_peptide = true RETURN DISTINCT m.id, g.id
```

Return all sequence nodes for a specific contig

```
MATCH (n)-[r]->() WHERE exists (r.'a1\_1') and (n:degenerate or n:node) RETURN id(n), n.
↪sequence , r.'a1\_1'
```

Return all sequence nodes for a specific contig within the range of position 1000 and 2000

```
MATCH (n)-[r]->() WHERE exists (r.'a1\_1') and (n:degenerate or n:node) and r.'a1'\_1[0]␣
↪> 1000 and r.'a1\_1'[0] < 2000 RETURN id(n), n.sequence, r.'a1\_1'
```

Find SNP bubbles in the graph. For simplification we only use the FF relation

```
MATCH p= (n:nucleotide) -[:FF]-> (a1)-[:FF]->(m:nucleotide) <-[:FF]-(b1) <-[:FF]- (n)␣
↪return * limit 50
```

6.9 Differences between pangenome and panproteome

PanTools offers functionalities to build and analyze a pangenome or panproteome.

A **pangenome** is constructed from genome and annotation files. First, genome sequences are k-merized and compressed into a De Bruijn graph. Genes and other annotation features from annotation files are integrated into the pangenome as 'gene', 'mRNA' and 'CDS' nodes. Gene start and stop positions are annotated in the graph as relationships and connect the annotation layer to the nucleotide layer. The protein sequences can be clustered into homology groups and connect homologous proteins from different genomes.

A **panproteome** is built from protein sequences only, ignoring the underlying genome structure. Again, the protein sequences are clustered into homology groups which serve as main input for many functionalities.

In addition to the single layer in panproteomes and three layers in pangenomes, a functional layer can be included in both databases. This layer consists of multiple functional annotation databases (e.g. GO, PFAM) and connects proteins with a shared function.

Since there is only a protein layer and functional layer present in panproteomes, not all functions can be utilized. See the table below for which functions can be used for pangenomes and panproteomes.

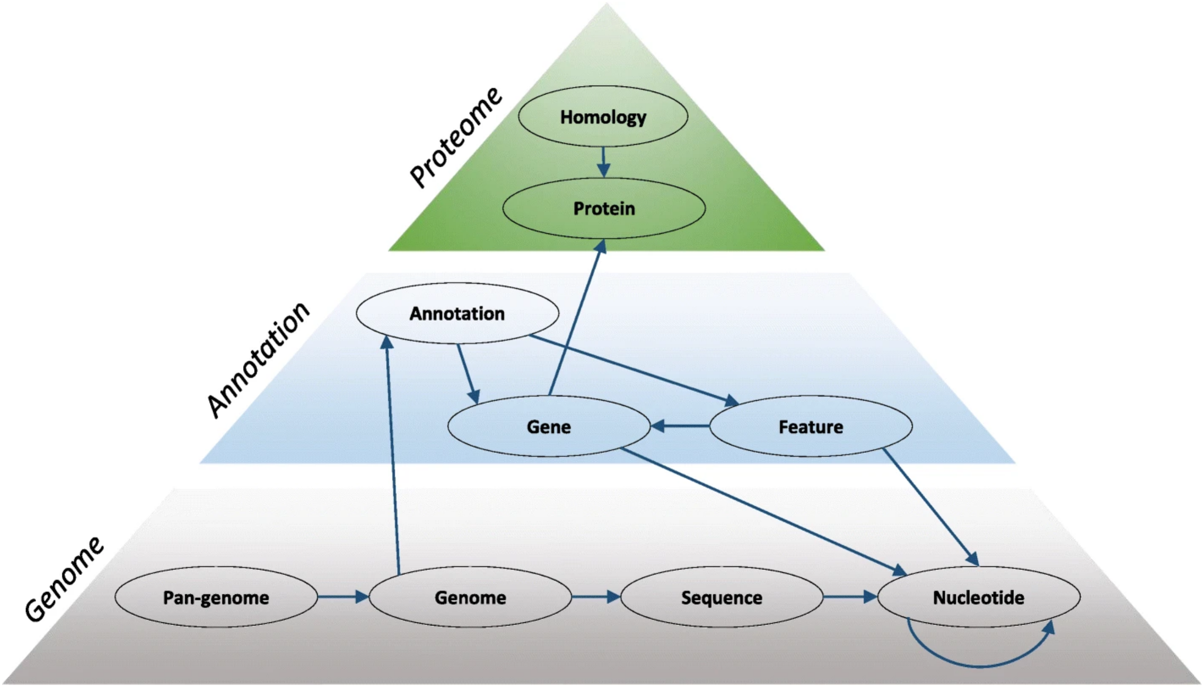


Fig. 6.8: *Schematic of genome, annotation, and protein layer of a pangenome database. Figure taken from Efficient inference of homologs in large eukaryotic pan-proteomes*

6.9.1 Available functions

Construct pangenome

Function	Pangenome	Panproteome
Build pangenome	YES	NO
Build panproteome	NO	YES
Add annotations	YES	NO
Add genomes	YES	NO
Group	YES	YES
Optimal grouping	YES	YES
Change grouping	YES	YES
BUSCO protein	YES	YES
Add phenotype	YES	YES
Add functional annotations	YES	YES
Add antiSMASH	YES	NO
Remove nodes	YES	YES
Move or remove grouping	YES	YES

Pangenome characterization

Function	Pangenome	Panproteome
Statistics	YES	YES
Gene classification	YES	YES
Core unique thresholds	YES	YES
Grouping overview	YES	YES
Pangenome size genes	YES	YES
Pangenome size k-mers	YES	NO
K-mer classification	YES	NO
Functional classification	YES	YES
Functional annotation overview	YES	YES

Explore the pangenome

Function	Pangenome	Panproteome
Locate genes	YES	NO
mRNAs connected to function	YES	NO
Find gene	YES	NO
GO enrichment	YES	YES
Show GO	YES	YES
Compare GO	YES	YES
Compare BGC	YES	NO
Alignment of homology group	YES	YES
Alignment of multiple homology groups	YES	YES
Alignment of genomic regions	YES	NO
Order matrix	YES	YES
Rename matrix	YES	YES
Retrieve genomes	YES	NO
Retrieve regions	YES	NO
Retrieve features	YES	NO

Phylogeny

Function	Pangenome	Panproteome
Core SNP tree	YES	YES
K-mer distance tree	YES	NO
Gene distance tree	YES	YES
ANI tree	YES	NO
MLSA	YES	NO
Rename phylogeny	YES	YES
Create tree template	YES	YES

Read mapping

Function	Pangenome	Panproteome
Map	YES	NO

6.10 Part 1. Install PanTools

For instructions on how to install PanTools, see *Installing and configuring the required software*.

6.11 Part 2. Build your own pangenome using PanTools

To demonstrate the main functionalities of PanTools we use a small chloroplasts dataset to avoid long construction times.

Genome	Chloroplast genome	Accession	Length	Genes	tRNAs
1	Cucumis sativus (cucumber)	NC_007144.1	155,293 bp	85	37
2	Oryza sativa Indica 93-11 (rice)	NC_008155.1	134,496 bp	100	40
3	Solanum lycopersicum (tomato)	NC_007898.3	155,461 bp	87	45
4	Solanum tuberosum (potato)	NC_008096.2	155,296 bp	84	45
5	Zea mays (maize)	NC_001666.2	140,384 bp	111	38

Download the chloroplast fasta and gff files here or via wget.

```
$ wget http://bioinformatics.nl/pangenomics/tutorial/chloroplasts.tar.gz
$ tar -xvzf chloroplasts.tar.gz #unpack the archive
```

We assume a PanTools alias was set during the *installation*. This allows PanTools to be executed with `pantools` rather than `pantools/target/pantools-3.4.0.jar`. If you don't have an alias, either set one or replace the pantools command with the full path to the .jar file in the tutorials.

6.11.1 BUILD, ANNOTATE and GROUP

We start with building a pangenome using four of the five chloroplast genomes. For this you need a text file which directs PanTools to the FASTA files. Call your text file **genome_locations.txt** and include the following lines:

```
YOUR_PATH/C_sativus.fasta
YOUR_PATH/O_sativa.fasta
YOUR_PATH/S_lycopersicum.fasta
YOUR_PATH/S_tuberosum.fasta
```

Make sure that '*YOUR_PATH*' is the full path to the input files! Then run PanTools with the *build_pangenome* function and include the text file

```
$ pantools build_pangenome -dp chloroplast_DB -gf genome_locations.txt
```

Did the program run without any error messages? Congratulations, you've built your first pangenome! If not? Make sure your Java version is up to date and kmc is executable. The text file should only contain full paths to FASTA files, no additional spaces or empty lines.

Adding additional genomes

PanTools has the ability to add additional genomes to an already existing pangenome. To test the function of PanTools, prepare a text file containing the path to the Maize chloroplast genome. Call your text file **fifth_genome_location.txt** and include the following line to the file:

```
YOUR_PATH/Z_mays.fasta
```

Run PanTools on the new text file and use the *add_genomes* function

```
$ pantools add_genomes -dp chloroplast_DB -gf fifth_genome_location.txt
```

Adding annotations To include gene annotations to the pangenome, prepare a text file containing paths to the GFF files. Call your text file **annotation_locations.txt** and include the following lines into the file:

```
1 YOUR_PATH/C_sativus.gff3
2 YOUR_PATH/O_sativa.gff3
3 YOUR_PATH/S_lycopersicum.gff3
4 YOUR_PATH/S_tuberosum.gff3
5 YOUR_PATH/Z_mays.gff3
```

Run PanTools using the *add_annotations* function and include the new text file

```
$ pantools add_annotations -dp chloroplast_DB -af annotation_locations.txt -ca
```

PanTools attached the annotations to our nucleotide nodes so now we can cluster them.

Homology grouping

PanTools can infer homology between the protein sequences of a pangenome and cluster them into homology groups. Multiple parameters can be set to influence the sensitivity but for now we use the *group* functionality with default settings.

```
$ pantools group -dp chloroplast_DB
```

6.11.2 Adding phenotypes (requires PanTools v3)

Phenotype values can be Integers, Double, String or Boolean values. Create a text file **phenotypes.txt**.

```
Genome,Solanum
1,false
2,false
3,true
4,true
5,false
```

And use *add_phenotypes* to add the information to the pangenome.

```
$ pantools add_phenotypes -dp chloroplast_DB -ph phenotypes.txt
```

6.11.3 RETRIEVE functions

Now that the construction is complete, lets quickly validate if the construction was successful and the database can be used. To retrieve some genomic regions, prepare a text file containing genomic coordinates. Create the file **regions.txt** and include the following for each region: genome number, contig number, start and stop position and separate them by a single space

```
1 1 200 500
2 1 300 700
3 1 1 10000
3 1 1 10000 -
4 1 9999 15000
5 1 100000 110000
```

Now run the *retrieve_regions* function and include the new text file

```
$ pantools retrieve_regions -dp chloroplast_DB --regions-file regions.txt
```

Take a look at the extracted regions that are written to the **chloroplast_DB/retrieval/regions/** directory.

To retrieve entire genomes, prepare a text file **genome_numbers.txt** and include each genome number on a separate line in the file

```
1
3
5
```

Use the **retrieve_regions** function again but include the new text file

```
$ pantools retrieve_regions -dp chloroplast_DB -rf genome_numbers.txt
```

Genome files are written to same directory as before. Take a look at one of the three genomes you have just retrieved.

In *part 3* of the tutorial we explore the pangenome you just built using the Neo4j browser and the Cypher language.

6.12 Part 3. Explore the pangenome using the Neo4j browser

Did you skip *part 2* of the tutorial or were you unable to build the chloroplast pangenome? Download the pre-constructed pangenome here or via wget.

```
$ wget http://bioinformatics.nl/pangenomics/tutorial/chloroplast_DB.tar.gz
$ tar -xvzf chloroplast_DB.tar.gz
```

6.12.1 Configuring Neo4j

Set the full path to the chloroplast pangenome database by opening neo4j.conf ('*neo4j-community-3.5.30/conf/neo4j.conf*') and include the following line in the config file. Please make sure there is always only a single uncommented line with 'dbms.directories.data'.

```
#dbms.directories.data=/YOUR_PATH/any_other_database
dbms.directories.data=/YOUR_PATH/chloroplast_DB
```

Allowing non-local connections

To be able to run Neo4j on a server and have access to it from anywhere, some additional lines in the config file must be changed.

- **Uncomment** the four following lines in neo4j-community-3.5.30/conf/neo4j.conf.
- Replace 7686, 7474, and 7473 by three different numbers that are not in use by other people on your server. In this way, everyone can have their own database running at the same time.

```
#dbms.connectors.default_listen_address=0.0.0.0
#dbms.connector.bolt.listen_address=:7687
#dbms.connector.http.listen_address=:7474
#dbms.connector.https.listen_address=:7473
```

Lets start up the Neo4j server!

```
$ neo4j start
```

Start Firefox (or a web browser of your own preference) and let it run on the background.

```
$ firefox &
```

In case you did not change the config to allow non-local connections, browse to *http://localhost:7474*. Whenever you did change the config file, go to *server_address:7474*, where 7474 should be replaced with the number you chose earlier.

If the database startup was successful, a login terminal will appear in the webpage. Use '*neo4j*' both as username and password. After logging in, you are requested to set a new password.

6.12.2 Exploring nodes and edges in Neo4j

Go through the following steps to become proficient in using the Neo4j browser and the underlying PanTools data structure. If you have any difficulty trouble finding a node, relationship or any type of information, download and use this visual guide.

1. Click on the database icon on the left. A menu with all node types and relationship types will appear.
2. Click on the '*gene*' button in the node label section. This automatically generated a query. Execute the query.
3. The **LIMIT** clause prevents large numbers of nodes popping up to avoid your web browser from crashing. Set LIMIT to 10 and execute the query.
4. Hover over the nodes, click on them and take a look at the values stored in the nodes. All these features (except ID) were extracted from the GFF annotation files. ID is an unique number automatically assigned to nodes and relationships by Neo4j.
5. Double-click on the **matK** gene node, all nodes with a connection to this gene node will appear. The nodes have distinct colors as these are different node types, such as **mRNA**, **CDS**, **nucleotide**. Take a look at the node properties to observe that most values and information is specific to a certain node type.
6. Double-click on the *matK* mRNA node, a **homology_group** node should appear. These type of nodes connect homologous genes in the graph. However, you can see this gene did not cluster with any other gene.
7. Hover over the **start** relation of the *matK* gene node. As you can see information is not only stored in nodes, but also in relationships! A relationship always has a certain direction, in this case the relation starts at the gene node and points to a nucleotide node. Offset marks the location within the node.
8. Double-click on the **nucleotide** node at the end of the 'start' relationship. An in- and outgoing relation appear that connect to other nucleotide nodes. Hover over both the relations and compare them. The relations holds the genomic coordinates and shows this path only occurs in contig/sequence 1 of genome 1.
9. Follow the outgoing **FF**-relationship to the next nucleotide node and expand this node by double-clicking. Three nodes will pop up this time. If you hover over the relations you see the coordinates belong to other genomes as well. You may also notice the relationships between nucleotide nodes is always a two letter combination of F (forward) and R (reverse) which state if a sequence is reverse complemented or not. The first letter corresponds to the sequence of the node at the start of the relation where the second letters refers to the sequence of the end node.
10. Finally, execute the following query to call the database scheme to see how all node types are connected to each other: *CALL db.schema()*. The schema will be useful when designing your own queries!

6.12.3 Query the pangenome database using CYPHER

Cypher is a declarative, SQL-inspired language and uses ASCII-Art to represent patterns. Nodes are represented by circles and relationships by arrows.

- The **MATCH** clause allows you to specify the patterns Neo4j will search for in the database.
- With **WHERE** you can add constraints to the patterns described.
- In the **RETURN** clause you define which parts of the pattern to display.

Cypher queries

Match and return 100 nucleotide nodes

```
MATCH (n:nucleotide) RETURN n LIMIT 100
```

Find all the genome nodes

```
MATCH (n:genome) RETURN n
```

Find the pangenome node

```
MATCH (n:pangenome) RETURN n
```

Match and return 100 genes

```
MATCH (g:gene) RETURN g LIMIT 100
```

Match and return 100 genes and order them by length

```
MATCH (g:gene) RETURN g ORDER BY g.length DESC LIMIT 100
```

The same query as before but results are now returned in a table

```
MATCH (g:gene) RETURN g.name, g.address, g.length ORDER BY g.length DESC LIMIT 100
```

Return genes which are longer as 100 but shorter than 250 bp (this can also be applied to other features such as exons introns or CDS)

```
MATCH (g:gene) where g.length > 100 AND g.length < 250 RETURN * LIMIT 100
```

Find genes located on first genome

```
MATCH (g:gene) WHERE g.address[0] = 1 RETURN * LIMIT 100
```

Find genes located on first genome and first sequence

```
MATCH (g:gene) WHERE g.address[0] = 1 AND g.address[1] = 1 RETURN * LIMIT 100
```

Homology group queries

Return 100 homology groups

```
MATCH (h:homology_group) RETURN h LIMIT 100
```

Match homology groups which contain two members

```
MATCH (h:homology_group) WHERE h.num_members = 2 RETURN h
```

Match homology groups and 'walk' to the genes and corresponding start and end node

```
MATCH (h:homology_group)-->(f:feature)<--(g:gene)-->(n:nucleotide) WHERE h.num_members =␣
↪2 RETURN * LIMIT 25
```

Turn off autocomplete by clicking on the button on the bottom right. The graph falls apart because relations were not assigned to variables.

The same query as before but now the relations do have variables

```
MATCH (h:homology_group)-[r1]-> (f:feature) <-[r2]-(g:gene)-[r3]-> (n:nucleotide) WHERE␣
↪h.num_members = 2 RETURN * LIMIT 25
```

When you turn off autocomplete again only the '*is_similar_to*' relation disappears since we did not call it

Find homology group that belong to the rpoC1 gene

```
MATCH (n:homology_group)--(m:mRNA)--(g:gene) WHERE g.name = 'rpoC1' RETURN *
```

Find genes on genome 1 which don't show homology

```
MATCH (n:homology_group)--(m:mRNA)--(g:gene) WHERE n.num_members = 1 and g.genome = 1␣
↪RETURN *
```

Structural variant detection

Find SNP bubbles (for simplification we only use the FF relation)

```
MATCH p= (n:nucleotide) -[:FF]-> (a1)-[:FF]->(m:nucleotide) <-[:FF]-(b1) <-[:FF]- (n)␣
↪return * limit 50
```

The same query but returning the results in a table

```
MATCH (n:nucleotide) -[:FF]-> (a1)-[:FF]->(m:nucleotide) <-[:FF]-(b1) <-[:FF]- (n)␣
↪return a1.length,b1.length, a1.sequence, b1.sequence limit 50
```

Functions such as **count()**, **sum()** and **stDev()** can be used in a query.

The same SNP query but count the hits instead of displaying them

```
MATCH p= (n:nucleotide) -[:FF]-> (a1)-[:FF]->(m:nucleotide) <-[:FF]-(b1) <-[:FF]- (n)␣
↪return count(p)
```

Hopefully you know have some feeling with the Neo4j browser and cypher and you're inspired to create your own queries!

When you're done working in the browser, close the database (by using the command line again).

```
$ neo4j stop
```

More information on Neo4j and the cypher language:
Neo4j Cypher Manual v3.5
Neo4j Cypher Refcard
Neo4j API

In *part 4* of the tutorial we explore some of the functionalities to analyze the pangenome.

6.13 Part 4. Characterization

6.13.1 Part 4 preparation

PanTools v3 is required to follow this part of the tutorial. In addition, MAFFT and R (and a few packages) need to be installed and set to your $PATH. Everything should already be correctly installed if you use the conda environment. Validate if the tools are executable by using the following commands.

```
$ Rscript --help
$ mafft -h
```

We assume a PanTools alias was set during the *installation*. This allows PanTools to be executed with `pantools` rather than `pantools/target/pantools-3.4.0.jar`. If you don't have an alias, either set one or replace the pantools command with the full path to the .jar file in the tutorials.

6.13.2 Input data

Genome	Name	Accession	Length	Sequences	Genes
1	P. odoriferum Q166	GCF_002904195.1	5.09 Mb	66	4510
2	P. fontis M022	GCF_000803215.1	4.15 Mb	107	3723
3	P. polaris S4.16.03.2B	GCF_003595035.1	4.86 Mb	65	4442
4	P. brasiliense S2	GCF_000808375.1	4.84 Mb	37	4367
5	P. brasiliense Y49	GCF_000808115.1	4.70 Mb	31	4231
6	D. dadantii 3937	GCF_000147055.1	4.92 Mb	1	4281

To demonstrate how to use the PanTools functionalities we use a small dataset of six bacteria to avoid long runtimes. Download a pre-constructed pangenome or test your new skills and construct a pangenome yourself using the fasta and gff files.

Option 1: Download separate genome and annotation files

```
$ wget http://bioinformatics.nl/pangenomics/tutorial/pecto_dickeya_input.tar.gz
$ tar -xvzf pecto_dickeya_input.tar.gz
$ gzip -d pecto_dickeya_input/annotations/*
$ gzip -d pecto_dickeya_input/genomes/*
$ gzip -d pecto_dickeya_input/functions/*

$ pantools build_pangenome -dp pecto_dickeya_DB -gf pecto_dickeya_input/genomes.txt
$ pantools add_annotations -dp pecto_dickeya_DB -af pecto_dickeya_input/annotations.txt -
↪ca
$ pantools group -dp pecto_dickeya_DB -rn 4 -tn 10
```

Option 2: Download the pre-constructed pangenome

```
$ wget http://bioinformatics.nl/pangenomics/tutorial/pecto_dickeya_DB.tar.gz
$ tar -xvzf pecto_dickeya_DB.tar.gz
```

6.13.3 Adding phenotype/metadata to the pangenome

Before starting with the analysis, we will add some phenotype data to the pangenome. Phenotypes allow you to find similarities for a group of genomes sharing a phenotype as well as identifying variation between different phenotypes. Below is a textfile with data for three phenotypes. The third phenotype, *low_temperature*, is in this case a made up example! It states whether the strain is capable of growing on (extreme) low temperatures. The phenotype file can be found inside the database directory or create a new file using the text from the box below. Add the phenotype information to the pangenome using *add_phenotype*.

```
Genome, species, strain_name, low_temperature
1,P. odoriferum,P. odoriferum Q166, false
2,P. fontis, P. fontis M022, true
3,P. polaris,P. polaris S4.16.03.2B, false
4,P. brasiliense, P. brasiliense S2, true
5,P. brasiliense, P. brasiliense Y49, false
6,D. dadantii, D. dadantii 3937,?
```

```
$ pantools add_phenotype -dp pecto_dickeya_DB -ph pecto_dickeya_input/phenotypes.txt
```

6.13.4 Metrics and general statistics

After building or uncompressing the pangenome, run the metrics functionality to produce various statistics that should verify an errorless construction.

```
$ pantools metrics -dp pecto_dickeya_DB
```

Open **metrics_per_genome.csv** with a spreadsheet tool (Excel, Libreoffice, Google sheets) and make sure the columns are split on commas. You may easily notice the many empty columns in this table as these type of annotations or features are not included in the database (yet). Functional annotations are incorporated later in this tutorial. Columns for features like exon and intron will remain empty as bacterial coding sequences are not interrupted.

6.13.5 Gene classification

With the *gene_classification* functionality you are able to organize the gene repertoire into the core, accessory or unique part of the pangenome.

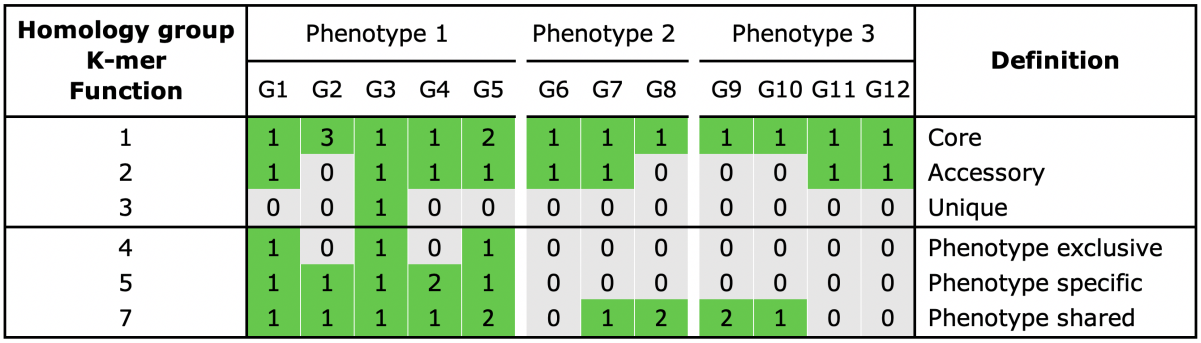
- **Core**, a gene is present in all genomes
- **Unique**, a gene is present in a single genome
- **Accessory**, a gene is present in some but not all genomes

```
$ pantools gene_classification -dp pecto_dickeya_DB
```

Take a look in **gene_classification_overview.txt**. Here you can find the number of classified homology groups and genes on a pangenome level but also for individual genomes.

Open **additional_copies.csv** with a spreadsheet tool. This file can be useful to identify duplicated genes in relation to other genomes.

The default criteria to call a group core is presence in all genomes where unique is only allowed to be present in one genome. These two categories are highly influenced by annotation quality, especially in large pangenomes. Luckily,



Homology group K-mer Function	Phenotype 1					Phenotype 2			Phenotype 3				Definition
	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	
1	1	3	1	1	2	1	1	1	1	1	1	1	Core
2	1	0	1	1	1	1	1	0	0	0	1	1	Accessory
3	0	0	1	0	0	0	0	0	0	0	0	0	Unique
4	1	0	1	0	1	0	0	0	0	0	0	0	Phenotype exclusive
5	1	1	1	2	1	0	0	0	0	0	0	0	Phenotype specific
7	1	1	1	1	2	0	1	2	2	1	0	0	Phenotype shared

the threshold for core and unique groups can easily be adjusted. Let's consider genes to be core when present in only five of the six genomes by setting the `--core-threshold` argument.

```
$ pantools gene_classification -dp pecto_dickeya_DB --core-threshold 85
```

Look in **gene_classification_overview.txt** again to observe the increase of core groups/genes at the cost of accessory groups.

For this pangenome, the *Dickeya* genome is considered an outgroup to the five *Pectobacterium* genomes. While this outgroup is needed to root and analyze phylogenetic trees (tutorial part 5), it affects the number classified groups for the all other genomes. Use `--reference` or `--skip` to exclude the *Dickeya* genome.

```
$ pantools gene_classification -dp pecto_dickeya_DB --reference 1,2,3,4,5
$ pantools gene_classification -dp pecto_dickeya_DB --skip 6
```

Take a look in **gene_classification_overview.txt** one more time to examine the effect of excluding this genome. The total number of groups in the analysis is lower now but the number of core and unique genes have increased for the five remaining genomes.

When phenotype information is used in the analysis, three additional categories can be assigned to a group:

- **Shared**, a gene present in all genomes of a phenotype
- **Exclusive**, a gene is only present in a certain phenotype
- **Specific**, a gene present in all genomes of a phenotype and is also exclusive

Include a `--phenotype` argument to find genes that are exclusive for a certain species.

```
$ pantools gene_classification -dp pecto_dickeya_DB --phenotype species
```

Open **gene_classification_phenotype_overview.txt** to see the number of classified groups for the species phenotype.

Open **phenotype_disrupted.csv** in a spreadsheet tool. This file explains exactly why a homology groups is labeled as phenotype shared and not specific.

Open **phenotype_additional_copies.csv** in a spreadsheet tool. Similarly to *phenotype_additional.csv* this file shows groups where all genomes of a certain phenotype have additional gene copies to (at least one of) the other phenotypes.

Each time you run the *gene_classification* function, multiple files are created that contain node identifiers of a certain homology group category. These files can be given to other PanTools functions for a downstream analysis, for example, sequence alignment, phylogeny, or GO enrichment. We will use one of the files later in this tutorial.

6.13.6 Pangenome structure

With the previous functionality we identified the core, accessory and unique parts of the pangenome. Now we will use the pangenome_size_genes function to observe how these numbers are reached by simulating the growth of the pangenome. Simulating the growth helps explaining if a pangenome should be considered open or closed. An pangenome is called open as long as a significant number of new (unique) genes are added to the total gene repertoire. The openness of a pangenome is usually tested using Heap's law. Heaps' law (a power law) can be fitted to the number of new genes observed when increasing the pangenome by one random genome. The formula for the power law model is n = k x N-a, where n is the newly discovered genes, N is the total number of genomes, and k and a are the fitting parameters. A pangenome can be considered open when a < 1 and closed if a > 1.

The outcome of the function can again be controlled through command line arguments. Genomes can be excluded from the analysis with `--skip`. You can set the number of iterations with `--value`. Because iterations can be assigned to different threads, including multiple threads with `--threads` is recommended.

```
$ pantools pangenome_structure_genes -dp pecto_dickeya_DB -tn 4
```

The current test set of six bacterial genomes is not representative of a full-sized pangenome. Therefore we prepared the results for the structure simulation on a set of 197 *Pectobacterium* genomes. The runtime of the analysis using 10.000 loops and 24 threads was 1 minute and 54 seconds. Download the files here, unpack the archive and take a look at the files.

```
$ wget wget http://bioinformatics.nl/pangenomics/tutorial/pectobacterium_structure.tar.gz
$ tar -xvf pectobacterium_structure.tar.gz
```

Normally you still have to run the R scripts to create the output figures and determine the openness of the pangenome.

```
cd pectobacterium_structure
$ Rscript pangenome_growth.R
$ Rscript gains_losses_median_and_average.R
$ Rscript heaps_law.R
```

Take a look at the plot. In **core_accessory_unique_size.png**, the number of classified groups are plotted for any of the genome combination that occured during the simulation. For the **core_accessory_size.png** plots, the number of unique groups is combined with accessory groups.

The **gains_losses.png** files display the average and mean group gain and loss between different pangenome sizes. The line of the core starts below zero, meaning for every random genome added, the core genome decreases by a number of *X* genes.

6.13.7 Functional annotations

PanTools is able to incorporate functional annotations into the pangenome by reading output of various functional annotation tools. In this tutorial we only include annotations from InterProScan. Please see the *add_functions* manual to check which other tools are available. To include the annotations, create a file **functions.txt** using text from the box below and add it to the command line argument.

```
1 YOUR_PATH/GCF_002904195.1.gff3
2 YOUR_PATH/GCF_000803215.1.gff3
3 YOUR_PATH/GCF_003595035.1.gff3
4 YOUR_PATH/GCF_000808375.1.gff3
5 YOUR_PATH/GCF_000808115.1.gff3
6 YOUR_PATH/GCA_000147055.1.gff3
```

```
$ pantools add_functions -dp pecto_dickeya_DB -if functions.txt
```

PanTools will ask you to download the InterPro database. Follow the steps and execute the program again.

The complete GO, PFAM, Interpro and TIGRFAM, databases are now integrated in the graph database after. Genes with a predicted function have gained a relationship to that function node. Retrieving a set of genes that share a function is now possible through a simple cypher query. If you would run **metrics** again, statistics for these type functional annotations are calculated. To create a summary table for each type of functional annotation, run *function_overview*.

```
$ pantools function_overview -dp pecto_dickeya_DB
```

In **function_overview_per_group.csv** you can navigate to a homology group or gene to see the connected functions. You can also search in the opposite direction, use one of the created overview files for a type of functional annotation and quickly navigate to a function of interest to find which genes are connected.

GO enrichment

We go back to the output files from gene classification that only contain node identifiers. We can retrieve group functions by providing one the files to *group_info* with the `--homology-groups` argument. However, interpreting groups by assessing each one individually is not very practical. A common approach to discover interesting genes from a large set is GO-enrichment. This statistical method enables the identification of genes sharing a function that are significantly over or under-represented in a given gene set compared to the rest of the genome. Let's perform a *GO enrichment* on homology groups of the core genome.

```
Phenotype: P._brasiliense, 2 genomes, threshold of 2 genomes
1278537,1282642,1283856,1283861,1283862,1283869,1283906,1283921,1283934,1283941,1283945,
↪1283946
```

```
$ pantools group_info -dp pecto_dickeya_DB -hm brasiliense_groups.csv
$ pantools go_enrichment -dp pecto_dickeya_DB -hm brasiliense_groups.csv
```

Open **go_enrichment.csv** with a spreadsheet tool. This file holds GO terms found in at least one of the genomes, the p-value of the statistical test and whether it is actually enriched after the multiple testing correction. as this is different for each genome a function might enriched in one genome but not in another.

A directory with seperate output files is created for each genome, open **go_enrichment.csv** for the genome 4 or 5 in a spreedsheet. Also take a look at the PDF files that visualize part of the Gene ontology hierarchy.

Classifying functional annotations

Similarly to classifying gene content, functional annotations can be categorized using *functional_classification*. This tool provides an easy way to identify functions shared by a group of genomes of a certain phenotype but can also be used to identify core or unique functions. The functionality uses the same set of arguments as **gene_classification**. You can go through the same steps again to see the effect of changing the arguments.

```
$ pantools functional_classification -dp pecto_dickeya_DB
$ pantools functional_classification -dp pecto_dickeya_DB -ct 85
$ pantools functional_classification -dp pecto_dickeya_DB --skip 6
$ pantools functional_classification -dp pecto_dickeya_DB -ph species
```

6.13.8 Sequence alignment

In the final part of this tutorial we will test the alignment function by aligning homology groups. PanTools is able to align genomic regions, genes and proteins to identify SNPs or amino acid changes with *msa*.

Start with the alignment of protein sequences from the 12 *P. brasiliense* specific homology groups.

```
$ pantools msa -dp pecto_dickeya_DB -hm brasiliense_groups.csv --mode protein --method␣
↪per_group
```

Go to the **pecto_dickeya_DB/alignments/grouping_v1/groups/** directory and select one of homology groups and check if you can find the following files

- The alignments are written to **prot_trimmed.fasta** and **prot_trimmed.afa**.
- A gene tree is written to **prot_trimmed.newick**
- **prot_trimmed_variable_positions.csv** located in the **var_inf_positions** subdirectory. This matrix holds every variable position of the alignment; the rows are the position in the alignment and the columns are the 20 amino acids and gaps.
- The identity and similarity (for proteins) is calculated between sequences and written to tables in the **similarity_identity** subdirectory.

Run the function again but include the `--no_trimming` argument.

```
$ pantools msa -dp pecto_dickeya_DB -hm brasiliense_groups.csv --mode protein --method␣
↪per_group --no-trimming
```

The output files are generated right after the first alignment without trimming the sequences first. The file names differ from the trimmed alignments by the ’*_trimmed*’ substring.

Run the function again but exclude the `--mode protein` and `--no_trimming` arguments. When no additional arguments are included to the command, both nucleotide and protein sequences are aligned two consecutive times.

```
$ pantools msa -dp pecto_dickeya_DB -hm brasiliense_groups.csv --method per_group
```

Again, the same type of files are generated but the output files from nucleotide sequence can be recognized by the ‘*nuc_*’ substrings. The matrix in **nuc_trimmed_variable_positions.csv** now only has columns for the four nucleotides and gaps.

Finally, run the function one more time but include a phenotype. This allows you to identify phenotype specific SNPs or amino acid changes.

```
$ pantools msa -dp pecto_dickeya_DB -hm brasiliense_groups.csv --no-trimming -ph low_
↪temperature
```

Open the **nuc**- or **prot_trimmed_phenotype_specific_changes.info** file inside one of the homology group output directories.

Besides the functionalities in this tutorial, PanTools has more useful functions that may aid you in retrieving more specific information from the pangenome.

- Identify shared k-mers between genomes with *kmer_classification*.
- Find co-localized genes in a set of homology groups: *locate_genes*.
- Mapping short reads against the pangenome with *map*.

In *part 5* of the tutorial we explore some of the phylogenetic methods implemented in PanTools.

6.14 Part 5. Phylogeny

6.14.1 Part 5 preparation

Pantools v3 is required to follow this part of the tutorial. In addition, MAFFT, FastTree, IQ-tree, R (and the ape R package) need to be installed and set to your $PATH. Validate if the tools are executable by using the following commands.

```
pantools version
Rscript --help
mafft -h
iqtree -h
fasttree -h
```

If you did not follow part 4 of the tutorial, download the pre-constructed pangenome here.

```
$ wget http://bioinformatics.nl/pangenomics/tutorial/pecto_dickeya_DB.tar.gz
$ tar -xvzf pecto_dickeya_DB.tar.gz
```

6.14.2 Adding phenotype/metadata to the pangenome

Before we construct the trees, we will add some phenotype data to the pangenome. Once the we have a phylogeny, the information can be included or be used to color parts of the tree. Below is a textfile with data for three phenotypes. The third phenotype, *low_temperature*, is in this case a made up example! It states whether the strain is capable of growing on (extreme) low temperatures. The phenotype file can be found inside the database directory, add the information to the pangenome by using *add_phenotype*.

```
Genome, species, strain_name, low_temperature
1,P. odoriferum,P. odoriferum Q166, false
2,P. fontis, P. fontis M022, true
3,P. polaris,P. polaris S4.16.03.2B, false
4,P. brasiliense, P. brasiliense S2, true
5,P. brasiliense, P. brasiliense Y49, false
6,D. dadantii, D. dadantii 3937,?
```

```
$ pantools add_phenotype -dp pecto_dickeya_DB/ -ph pecto_dickeya_DB/phenotypes.txt
```

6.14.3 Constructing a phylogeny

In this tutorial we will construct three phylogenies, each based on a different type of variation: SNPs, genes and k-mers. Take a look at the phylogeny manuals to get an understanding how the three methods work and how they differ from each other.

1. phylogeny:core snp tree>
2. phylogeny:gene distance tree>
3. phylogeny:k-mer distance tree>

6.14.4 Core SNP phylogeny

The core SNP phylogeny will run various Maximum Likelihood models on parsimony informative sites of single-copy orthologous sequences. A site is parsimony-informative when there are at least two types of nucleotides that occur with a minimum frequency of two. The informative sites are automatically identified by aligning the sequences; however, it does not know which sequences are single-copy orthologous. You can identify these conserved sequences by running *gene_classification*.

```
$ pantools gene_classification -dp pecto_dickeya_DB/ -ph species
```

Open **gene_classification_overview.txt** and take a look at statistics. As you can see there are 2134 single-copy ortholog groups. Normally, all of these groups are aligned to identify SNPs but for this tutorial we'll make a selection of only a few groups to accelerate the steps. You can do this in two different ways:

Option 1: Open **single_copy_orthologs.csv** and remove all node identifiers after the first 20 homology groups and save the file.

```
$ pantools core_snp_tree -dp pecto_dickeya_DB/ --mode ML -tn 4
```

Option 2: Open **single_copy_orthologs.csv** and select the first 20 homology_group node identifiers. Place them in a new file sco_groups.txt and include this file to the function.

```
$ pantools core_snp_tree -dp pecto_dickeya_DB/ --mode ML -tn 4 -hm sco_groups.txt
```

The sequences of the homology groups are being aligned two consecutive times. After the initial alignment, input sequences are trimmed based on the longest start and end gap of the alignment. The parsimony informative positions are taken from the second alignment and concatenated into a sequence. When opening **informative.fasta** you can find 6 sequences, the length of the sequences being the number of parsimony-informative sites.

```
$ iqtree -nt 4 -s pecto_dickeya_DB/alignments/grouping_v1/core_snp_tree/informative.
↪fasta -redo -bb 1000
```

IQ-tree generates several files, the tree that we later on in the tutorial will continue with is called **informative.fasta.treefile**. When examining the **informative.fasta.iqtree** file you can find the best fit model of the data. This file also shows the number of sites that were used, as sites with gaps (which IQ-tree does not allow) were changed into singleton or constant sites.

Gene distance tree

To create a phylogeny based on gene distances (absence/presence), we can simply execute the Rscript that was created by *gene_classification*.

```
$ Rscript pecto_dickeya_DB/gene_classification/gene_distance_tree.R
```

The resulting tree is called **gene_distance.tree**.

K-mer distance tree

To obtain a k-mer distance phylogeny, the k-mers must first be counted with the *kmer_classification* function. Afterwards, the tree can be constructed by executing the Rscript.

```
$ pantools kmer_classification -dp pecto_dickeya_DB/
$ Rscript pecto_dickeya_DB/kmer_classification/genome_kmer_distance_tree.R
```

The resulting tree is written to **genome_kmer_distance.tree**.

6.14.5 Renaming tree nodes

So far, we used three different types of distances (SNPs, genes, k-mers), and two different methods (ML, NJ) to create three phylogenetic trees. First, lets take a look at the text files. The **informative.fasta.treefile** only contain genome numbers, bootstrap values and branch lengths but is lacking the metadata. Examining **gene_distance.tree** file also shows this information but the species names as well, because we included this as a phenotype during *gene_classification*.

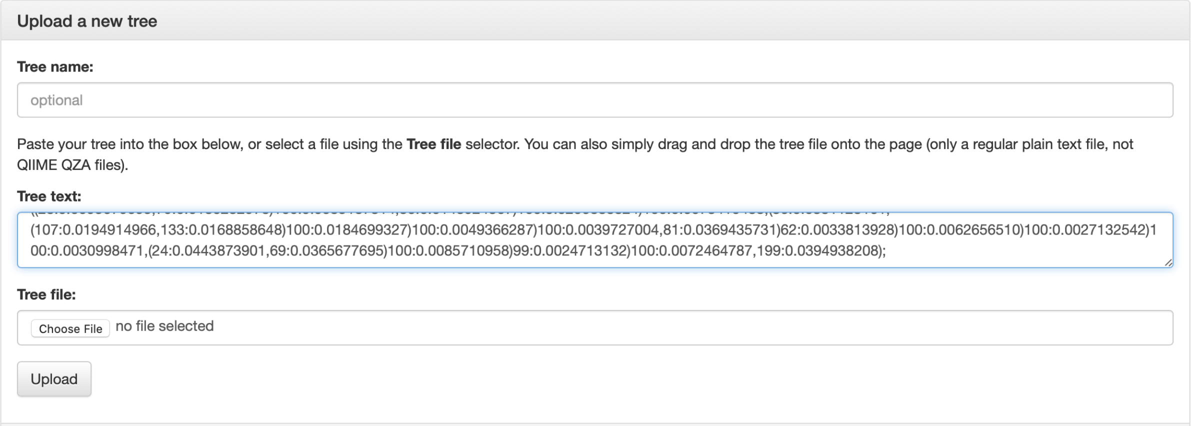
Let's include the strain identifiers to the core snp tree to make the final figure more informative. Use the *rename_phylogeny* function to rename the tree nodes.

```
$ pantools rename_phylogeny -dp pecto_dickeya_DB --phenotype strain_name -if pecto_
↪dickeya_DB/alignments/grouping_v1/core_snp_tree/informative.fasta.treefile
```

Take a look at **informative.fasta_RENAMED.treefile**, strain identifiers have been added to the tree.

6.14.6 Visualizing the tree in iTOL

Go to https://itol.embl.de and click on "Upload a tree" under the **ANNOTATE** box. On this page you can paste the tree directly into the **tree text:** textbox or can click the button to upload the .newick file.



Upload a new tree

Tree name:

optional

Paste your tree into the box below, or select a file using the **Tree file** selector. You can also simply drag and drop the tree file onto the page (only a regular plain text file, not QIIME QZA files).

Tree text:

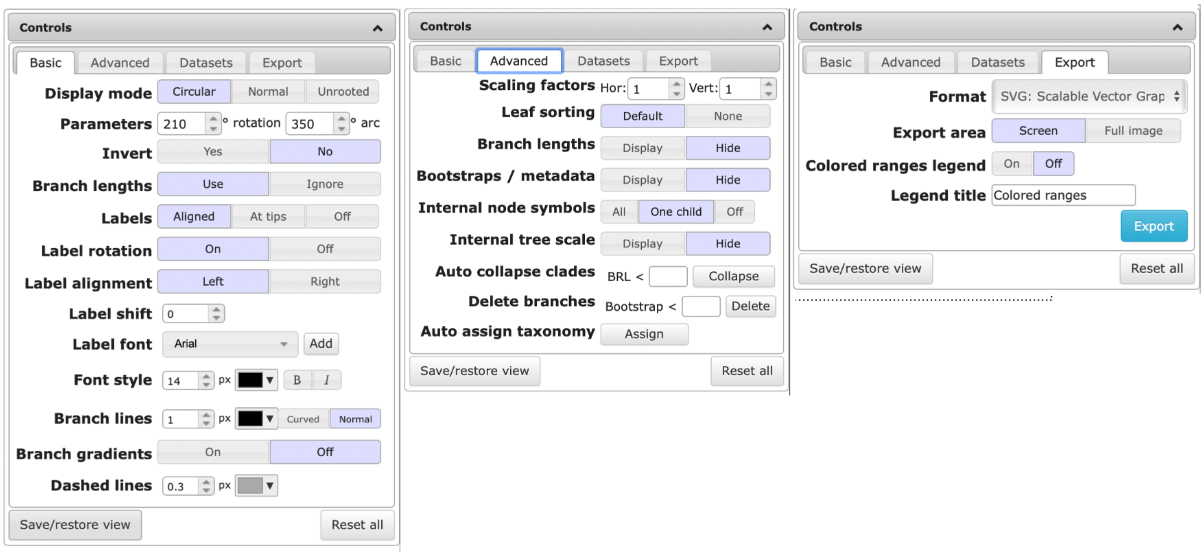
```
(107:0.0194914966,133:0.0168858648)100:0.0184699327)100:0.0049366287)100:0.0039727004,81:0.0369435731)62:0.0033813928)100:0.0062656510)100:0.0027132542)1
00:0.0030998471,(24:0.0443873901,69:0.0365677695)100:0.0085710958)99:0.0024713132)100:0.0072464787,199:0.0394938208);
```

Tree file:

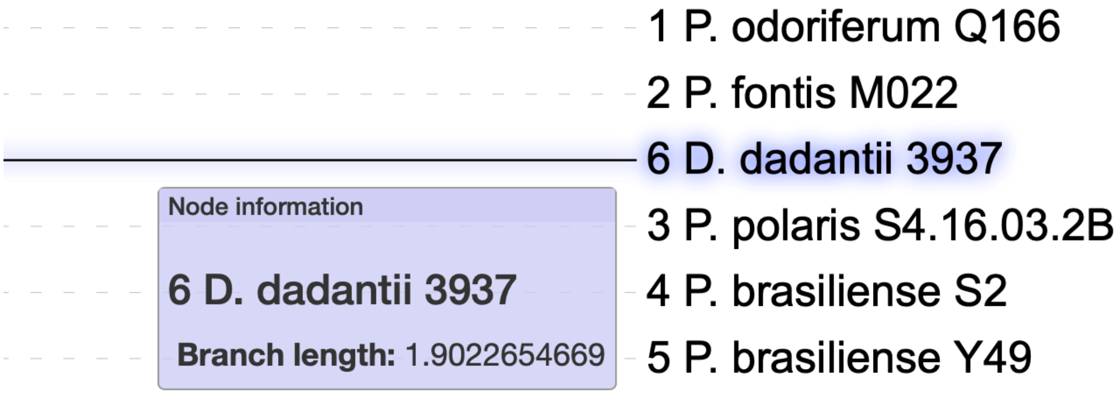
Choose File no file selected

6.14.7 Basic controls ITOL

- The default way of visualizing a tree is the rectangular view. Depending on the number of genomes, the circular view can be easier to interpret. You can the view by clicking on the "Display Mode" buttons.
- Increase the font size and branch width to improve readability
- When visualizing a Maximum likelihood (ML) tree, bootstrap values can be displayed by clicking the "Display" button next to **Bootstrap/metadata** in the Advanced tab of the Control window. This enables you to visualize the values as text or symbol on the branch. or by coloring the branch or adjusting the width.



- When you have a known outgroup or one of the genomes is a clear outlier in the tree, you should reroot the tree. Hover over the name, click it so a pop-up menu appears. Click "tree structure" followed by "Reroot the tree here".



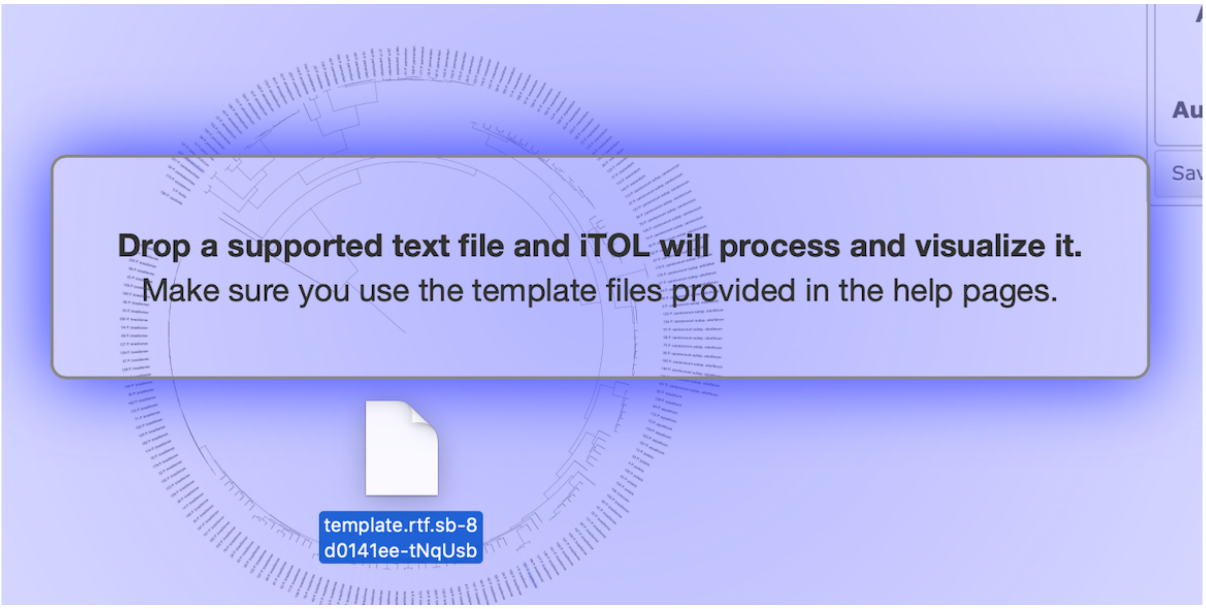
- Clicking on the name of a node in the tree allows you to color the name, branch, or background of that specific node.
- When you're happy the way your tree looks, go to the Export tab of the Control window. Select the desired output format, click on the "Full image" button and export the file to a figure.
- Refresh the webpage to go back to the default view of your tree.

6.14.8 Create iTOL templates

In iTOL it is possible to add colors to the tree by coloring the terminal nodes or adding an outer ring. The PanTools function *create_tree_template* is able to create templates that allows for easy coloring (with maximum of 20 possible colors). If the function is run without any additional argument, templates are created for trees that only contain genome numbers (e.g. k-mer distance tree). Here we want to color the (renamed) core SNP tree with the 'low_temperature' phenotype. Therefore, the `--phenotype` strain_name must be included to the function.

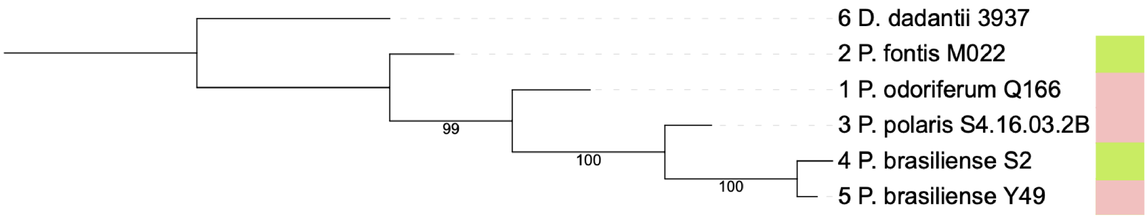
```
$ pantools create_tree_template -dp pecto_dickeya_DB # Run this command when the tree contains genome numbers only
$ pantools create_tree_template -dp pecto_dickeya_DB -ph strain_name
```

Copy the two low_temperature.txt files from the label/strain_name/ and ring/strain_name/ directories to your personal computer. Click and move the ring template file into the tree visualization webpage.



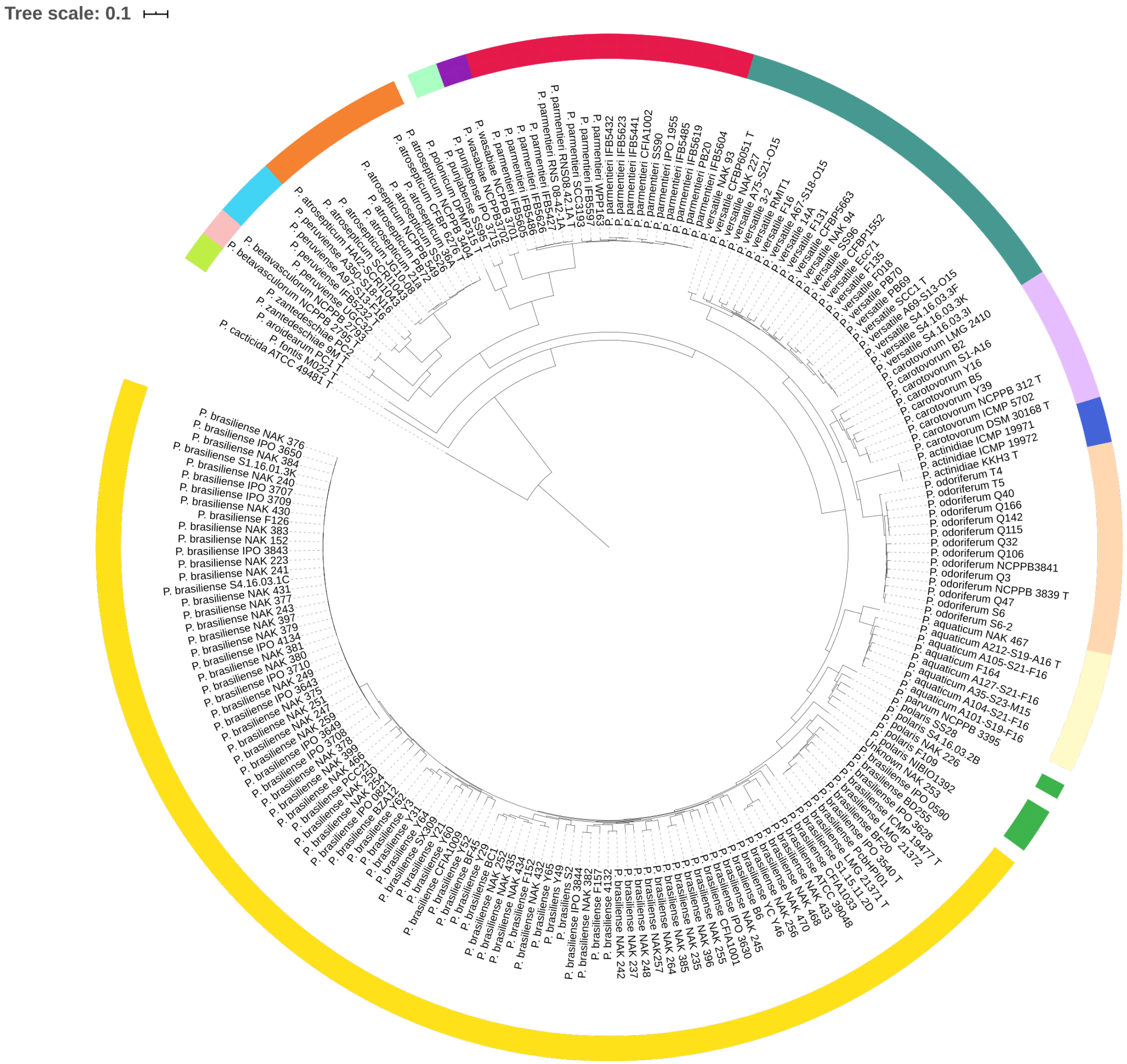
Drop a supported text file and iTOL will process and visualize it.
Make sure you use the template files provided in the help pages.

The resulting tree should look this when: the tree is rooted with the *Dickeya* genome, bootstrap values are displayed as text and the ring color template was included.



Tree coloring is especially useful for large datasets. An example is shown in the figure below, where members of the same species share a color.

PanTools has its documentation hosted on Read the Docs.


Tree scale: 0.1
P. cacticida ATCC 49481 T
P. fontis M022 T
P. aroidearum PC1 T
P. zantedeschiae 9M T
P. zantedeschiae PC2
P. betavasculorum NCPPB 2795 T
P. betavasculorum NCPPB 2793
P. peruviense UGC32
P. peruviense IFB5232 T
P. peruviense A97-S13-F16
P. peruviense A350-S18-N16
P. atrosepticum HAI2-SCRI1043
P. atrosepticum SCRI1043
P. atrosepticum JG10-08
P. atrosepticum 21a
P. atrosepticum PB72
P. atrosepticum NCPPB 549
P. atrosepticum SS26
P. atrosepticum 36A
P. atrosepticum CFBP 6276 T
P. polonicum NCPPB 3404
P. punjabense DPMP315 T
P. punjabense SS95 T
P. wasabiae IPO 3715
P. wasabiae NCPPB3702
P. parmentieri NCPPB 3701
P. parmentieri IFB5605
P. parmentieri IFB5486
P. parmentieri IFB5626
P. parmentieri IFB5427
P. parmentieri RNS 08-42-1A
P. parmentieri RNS08.42.1A T
P. parmentieri SCC3193
P. parmentieri IFB5597
P. parmentieri WPP163
P. parmentieri IFB5432
P. parmentieri IFB5623
P. parmentieri IFB5441
P. parmentieri CFIA1002
P. parmentieri SS90
P. parmentieri IPO 1955
P. parmentieri IFB5485
P. parmentieri IFB5619
P. parmentieri PB20
P. parmentieri IFB5604
P. versatile NAK 93
P. versatile CFBP6051 T
P. versatile NAK 227
P. versatile A75-S21-O15
P. versatile 3-2
P. versatile RMIT1
P. versatile F16
P. versatile A67-S18-O15
P. versatile 14A
P. versatile F131
P. versatile CFBP5663
P. versatile NAK 94
P. versatile SS96
P. versatile CFBP1552
P. versatile Ecc71
P. versatile F135
P. versatile F018
P. versatile PB70
P. versatile PB69
P. versatile SCC1 T
P. versatile A69-S13-O15
P. versatile S4.16.03.3F
P. versatile S4.16.03.3K
P. versatile S4.16.03.3I
P. carotovorum LMG 2410
P. carotovorum B2
P. carotovorum S1-A16
P. carotovorum Y16
P. carotovorum B5
P. carotovorum Y39
P. carotovorum NCPPB 312 T
P. carotovorum ICMP 5702
P. carotovorum DSM 30168 T
P. actinidiae ICMP 19971
P. actinidiae ICMP 19972
P. actinidiae KKH3 T
P. odoriferum T4
P. odoriferum T5
P. odoriferum Q40
P. odoriferum Q166
P. odoriferum Q142
P. odoriferum Q115
P. odoriferum Q32
P. odoriferum Q106
P. odoriferum NCPPB3841
P. odoriferum Q3
P. odoriferum NCPPB 3839 T
P. odoriferum Q47
P. odoriferum S6
P. odoriferum S6-2
P. aquaticum NAK 467
P. aquaticum A212-S19-A16 T
P. aquaticum A105-S21-F16
P. aquaticum F164
P. aquaticum A127-S21-F16
P. aquaticum A35-S23-M15
P. aquaticum A104-S21-F16
P. aquaticum A101-S19-F16
P. parvum NCPPB 3395
P. polaris SS28
P. polaris S4.16.03.2B
P. polaris NAK 226
P. polaris F109
P. polaris NIBIO1392
Unknown NAK 253
P. brasiliense IPO 0590
P. brasiliense BD255
P. brasiliense IPO 3628
P. brasiliense ICMP 19477 T
P. brasiliense LMG 21372
P. brasiliense BF20
P. brasiliense IPO 3540 T
P. brasiliense PcbHPI01
P. brasiliense LMG 21371 T
P. brasiliense S1.15.11.2D
P. brasiliense CFIA1033
P. brasiliense ATCC 39048
P. brasiliense NAK 433
P. brasiliense NAK 468
P. brasiliense NAK 470
P. brasiliense NAK 256
P. brasiliense YC Y46
P. brasiliense B6
P. brasiliense NAK 245
P. brasiliense IPO 3630
P. brasiliense CFIA1001
P. brasiliense NAK 255
P. brasiliense NAK 396
P. brasiliense NAK 235
P. brasiliense NAK 385
P. brasiliense NAK 264
P. brasiliense NAK257
P. brasiliense NAK 248
P. brasiliense NAK 237
P. brasiliense NAK 242
P. brasiliense 4132
P. brasiliense F157
P. brasiliense NAK 382
P. brasiliense IPO 3844
P. brasiliens S2
P. brasiliens Y49
P. brasiliense Y65
P. brasiliense NAK 432
P. brasiliense F152
P. brasiliense NAK 434
P. brasiliense NAK 435
P. brasiliense NAK 252
P. brasiliense BC1
P. brasiliense Y29
P. brasiliense BF45
P. brasiliense Y52
P. brasiliense CFIA1009
P. brasiliense Y60
P. brasiliense Y21
P. brasiliense SX309
P. brasiliense Y64
P. brasiliense Y31
P. brasiliense Y3
P. brasiliense Y62
P. brasiliense BZA12
P. brasiliense IPO 0821
P. brasiliense NAK 254
P. brasiliense NAK 250
P. brasiliense PCC21
P. brasiliense NAK 466
P. brasiliense NAK 399
P. brasiliense NAK 378
P. brasiliense IPO 3708
P. brasiliense IPO 3649
P. brasiliense NAK 259
P. brasiliense NAK 247
P. brasiliense NAK 251
P. brasiliense NAK 375
P. brasiliense IPO 3643
P. brasiliense NAK 249
P. brasiliense IPO 3710
P. brasiliense NAK 380
P. brasiliense NAK 381
P. brasiliense IPO 4134
P. brasiliense NAK 379
P. brasiliense NAK 397
P. brasiliense NAK 243
P. brasiliense NAK 377
P. brasiliense NAK 431
P. brasiliense S4.16.03.1C
P. brasiliense NAK 241
P. brasiliense NAK 223
P. brasiliense IPO 3843
P. brasiliense NAK 152
P. brasiliense NAK 383
P. brasiliense F126
P. brasiliense NAK 430
P. brasiliense IPO 3709
P. brasiliense IPO 3707
P. brasiliense NAK 240
P. brasiliense S1.16.01.3K
P. brasiliense NAK 384
P. brasiliense IPO 3650
P. brasiliense NAK 376