drep Documentation

Release 2.0.0

Matt Olm

Jan 12, 2018

Contents

CHAPTER 1

Rapid and accurate comparison and de-replication of microbial genomes

The publication is available at ISME and an open-source pre-print is available on bioRxiv.

Source code is available on GitHub.

See links to the left for *Installation* and *Quick Start* instructions

Comments and suggestions can be sent to Matt Olm

Bugs reports and feature requests can be submitted through GitHub.

CHAPTER 2

Contents

2.1 Overview

dRep is a python program which performs rapid pair-wise comparison of genome sets. One of it's major purposes is for genome de-replication, but it can do a lot more.

The publication is available at ISME and an open-source pre-print is available on bioRxiv.

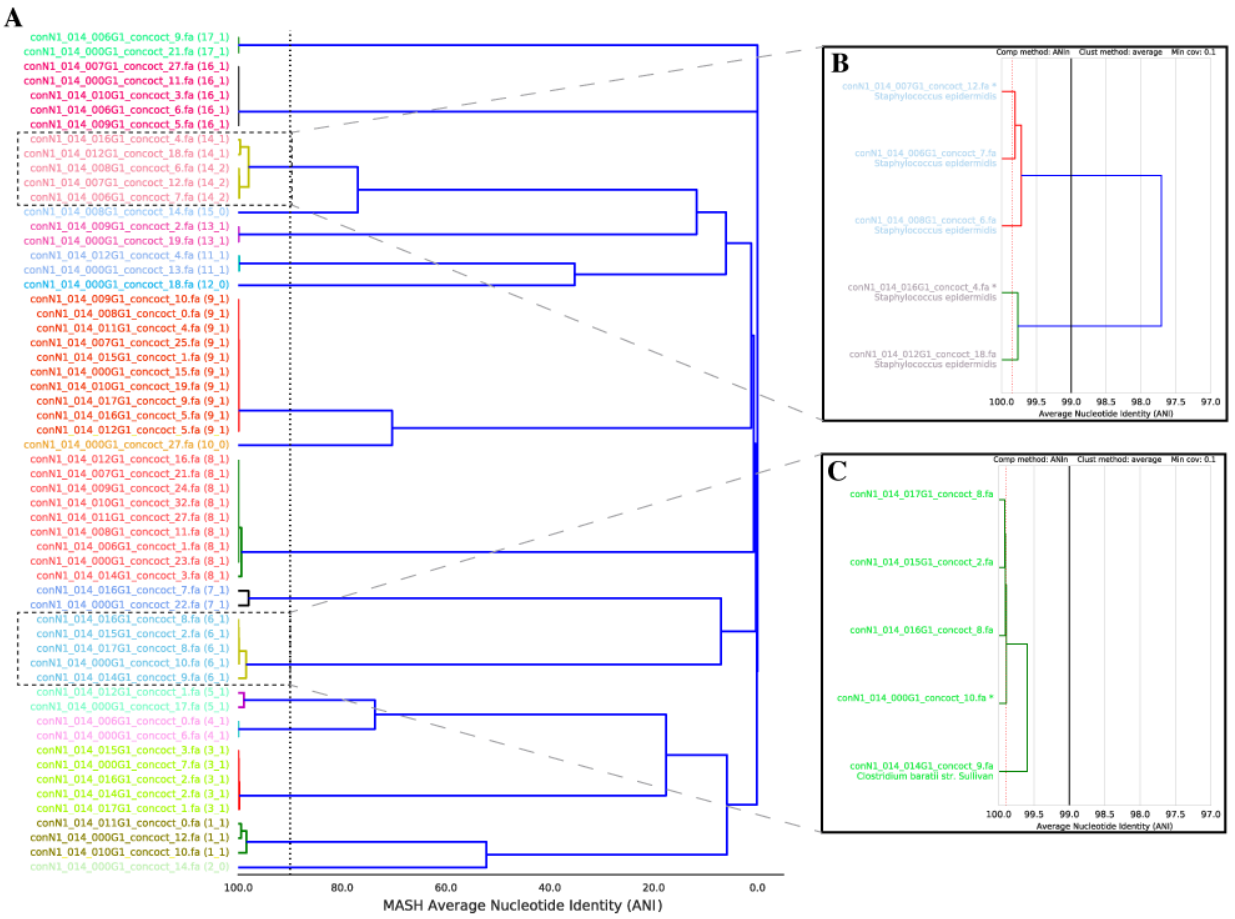
Source code is available on GitHub.

2.1.1 Genome comparison

dRep can rapidly and accurately compare a list of genomes in a pair-wise manner. This allows identification of groups of organisms that share similar DNA content in terms of Average Nucleotide Identity (ANI).

dRep performs this in two steps- first with a rapid primary algorithm (Mash), and second with a more sensitive algorithm (ANIm). We can't just use Mash because, while incredibly fast, it is not robust to genome incompletenss (see *Choosing parameters*) and only provides an "estimate" of ANI. ANIm is robust to genome incompleteness and is more accurate, but too slow to perform pair-wise comparisons of longer genome lists.

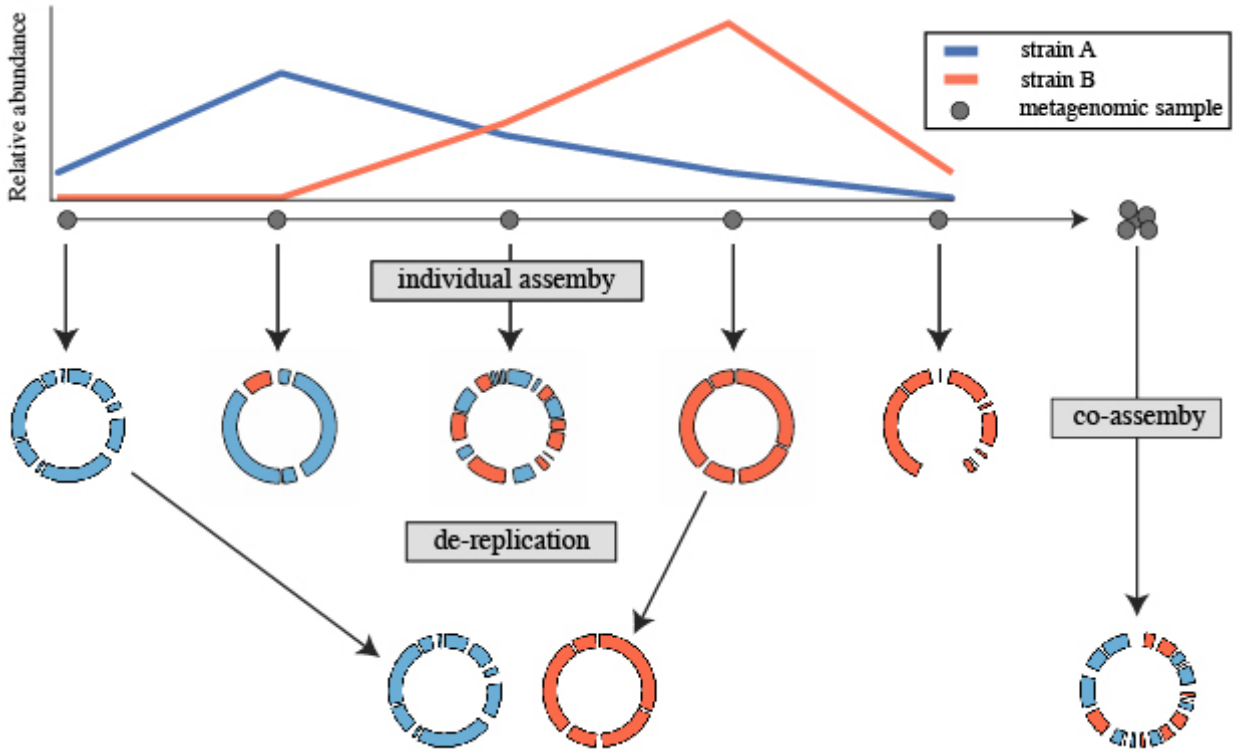
dRep first compares all genomes using Mash, and then only runs the secondary algorithm (ANIm or gANI) on sets of genomes that have at least 90% Mash ANI. This results in a great decrease in the number of (slow) secondary comparisons that need to be run while maintaining the sensitivity of ANIm.



2.1.2 Genome de-replication

De-replication is the process of identifying sets of genomes that are the "same" in a list of genomes, and removing all but the "best" genome from each redundant set. How similar genomes need to be to be considered "same", how to determine which genome is "best", and other important decisions are discussed in *Choosing parameters*

A common use for genome de-replication is the case of individual assembly of metagenomic data. If metagenomic samples are collected in a series, a common way to assemble the short reads is with a "co-assembly". That is, combining the reads from all samples and assembling them together. The problem with this is assembling similar strains together can severely fragment assemblies, precluding recovery of a good genome bin. An alternative option is to assemble each sample separately, and then "de-replicate" the bins from each assembly to make a final genome set.



The steps to this process are:

- Assemble each sample separately using your favorite assembler. You can also perform a co-assembly to catch low-abundance microbes
- Bin each assembly (and co-assembly) separately using your favorite binner
- Pull the bins from all assemblies together and run dRep on them
- Perform downstream analysis on the de-replicated genome list

2.2 Installation

2.2.1 Using pip

To install dRep, simply run

```
$ pip install drep
```

OR

```
$ git clone https://github.com/MrOlm/drep.git

$ cd drep

$ pip install .
```

That's it!

Pip is a great package with many options to change the installation parameters in various ways. For details, see pip documentation

2.2.2 Dependencies

dRep requires other programs to run. Not all dependencies are needed for all operations

To check which dependencies are installed on your system and accessible by dRep, run

```
$ dRep bonus testDir --check_dependencies
```

Near Essential

- Mash - Makes primary clusters (v1.1.1 confirmed works)
- MUMmer - Performs ANIm comparison method (v3.23 confirmed works)

Recommended

- CheckM - Determines contamination and completeness of genomes (v1.0.7 confirmed works)
- gANI (aka ANIcalculator) - Performs gANI comparison method (v1.0 confirmed works)
- Prodigal - Used be both checkM and gANI (v2.6.3 confirmed works)

Accessory

- Centrifuge - Performs taxonomic assignment of bins (v1.0.3 confirmed works)

Programs need to be installed to the system path, so that you can call them from anywhere on your computer.

Note: If you already have information on your genome's completeness and contamination, you can input that to dRep without the need to install checkM (see *Advanced Use*))

2.2.3 pyenv

Because dRep is written in python3 and CheckM is written in python2, you may need to use pyenv to be able to call both.

With CheckM installed in a python2 installation of pyenv, and dRep installed in the python3 version, the following command should set allow both python2 and python3 commands to be called:

```
$ pyenv global 3.5.1 2.7.9
```

Alternatively, you could add python2 to your CheckM shebang line (though I have not confirmed that this works)

2.3 Quick Start

The functionality of dRep is broken up into modules. The modules can be run separately (see *Module Descriptions*), or together in workflows. To see a list of the available modules, check the help:

```
$ dRep -h

               ..::: dRep v2.0.0 :::...

 Choose one of the operations below for more detailed help.
 Example: dRep dereplicate -h

 Workflows:
   dereplicate  -> Combine several of the operations below to de-replicate a genome␣
↪list
```

```
   compare      -> Simply compare a list of genomes

 Single operations:
   filter          -> Filter a genome list based on size, completeness, and/or␣
↪contamination
   cluster         -> Compare and cluster a genome list based on MASH and ANIn/gANI
   choose          -> Choose the best genome from each genome cluster
   evaluate        -> Evaluate genome de-replication
   bonus           -> Other random operations (currently just determine taxonomy)
   analyze         -> Make figures related to the above operations; test alternative␣
↪clustering
```

2.3.1 De-replication

De-replication is the process of identifying groups of genomes that are the "same" in a genome set, and removing all but the "best" genome from each redundant set. How similar genomes need to be to be considered "same", how the "best" genome is chosen, and other options can be adjusted (see *Choosing parameters*)

To de-replicate a set of genomes, run the following command:

```
$ dRep dereplicate outout_directory -g path/to/genomes/*.fasta
```

This will automatically de-replicate the genome list and produce lots of information about it.

See also:

Example Output to view example output

Choosing parameters for guidance changing parameters

2.3.2 Genome comparison

dRep is able to perform rapid genome comparisons for a group of genomes and visualize their relatedness. For example:

```
$ dRep compare_wf output_directory -g path/to/genomes/*.fasta
```

For help understanding the output, see *Example Output*

To change the comparison parameters, see *Choosing parameters*

See also:

Example Output to view example output

Choosing parameters for guidance changing parameters

2.4 Example Output

dRep produces a variety of output in the work directory depending on which operations are run.

To generate the figures below, dRep `dereplicate` was run on a set of 5 randomly chosen *Klebsiella oxytoca* isolate genomes as follows:

```
$ dRep dereplicate complete_only -g *.fna --S_algorithm gANI
```

See also:

Overview for general information on the program

Quick Start for more information on dereplicate_wf

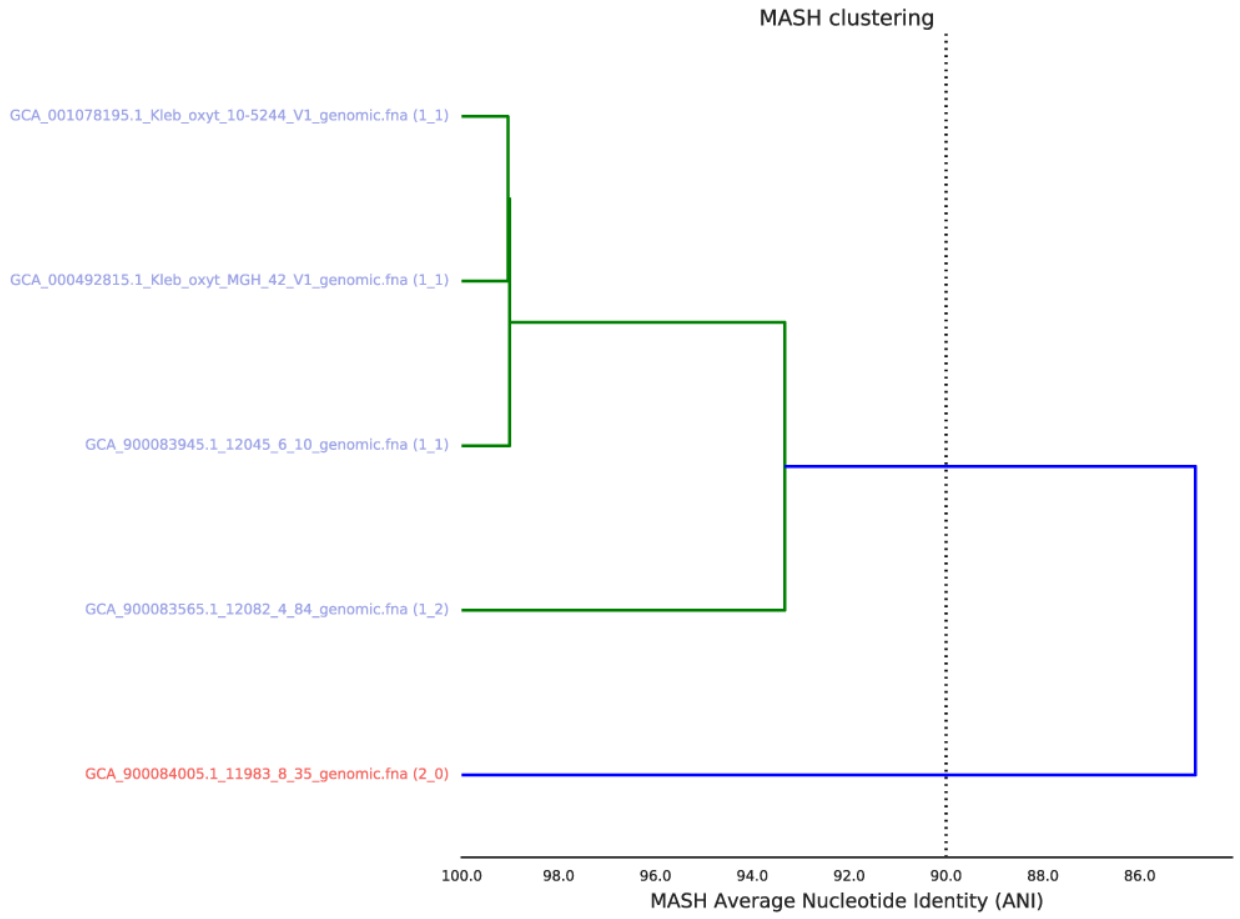
Module Descriptions for a more detailed description of what the modules do

2.4.1 Figures

Figures are located within the work directory, in the folder `figures`:

```
$ ls complete_only/figures/
Clustering_scatterplots.pdf
Cluster_scoring.pdf
Primary_clustering_dendrogram.pdf
Secondary_clustering_dendrograms.pdf
Winning_genomes.pdf
```

Primary_clustering_dendrogram

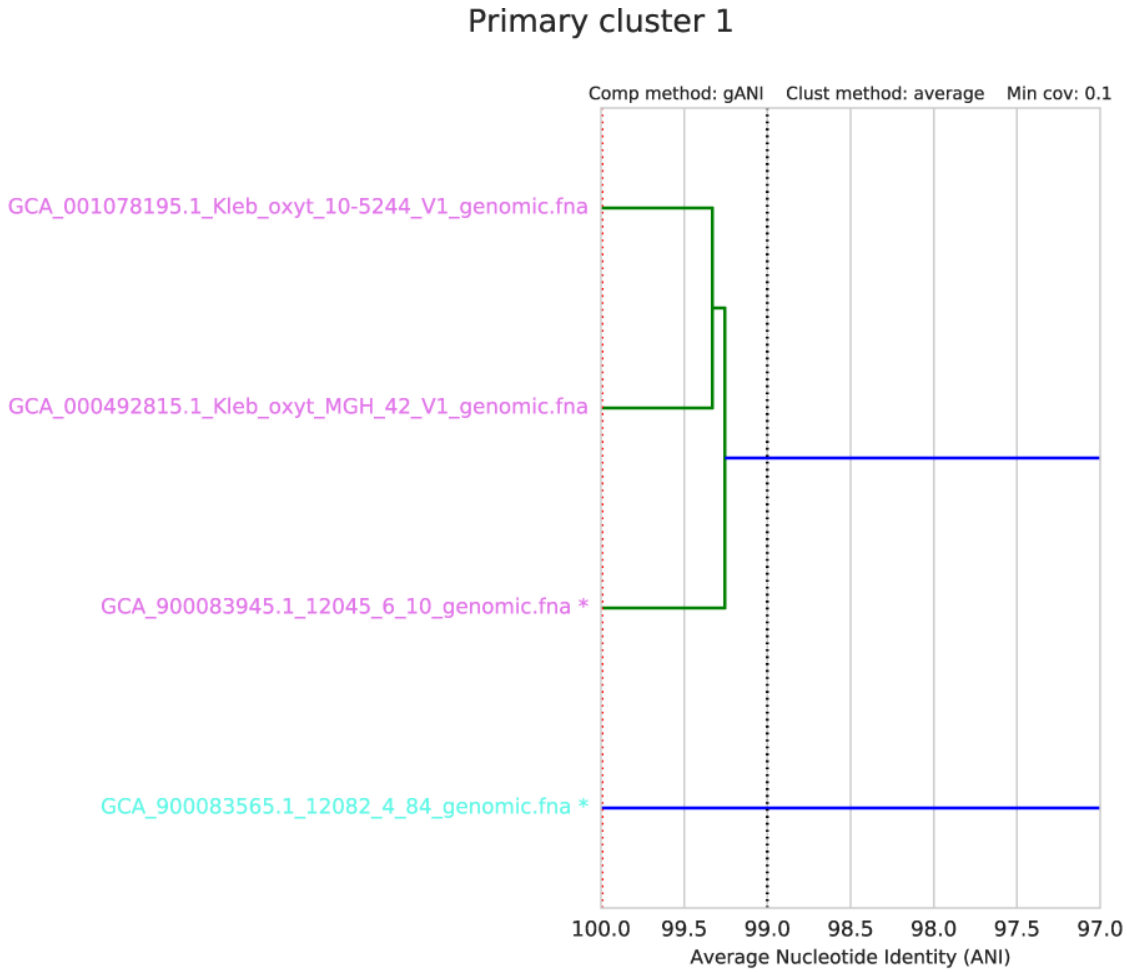


The primary clustering dendrogram summarizes the pair-wise Mash distance between all genomes in the genome list.

The dotted line provides a visualization of the **primary ANI** - the value which determines the creation of primary clusters. It is drawn in the above figure at 90% ANI (the default value). Based on the above figure you can see that two primary clusters were formed- one with genomes colored blue, and one red.

Note: Genomes in the same primary cluster will be compared to each other using a more sensitive algorithm (gANI or ANIm) in order to form secondary clusters. Genomes which are not in the same primary cluster will never be compared again.

Secondary_clustering_dendrograms



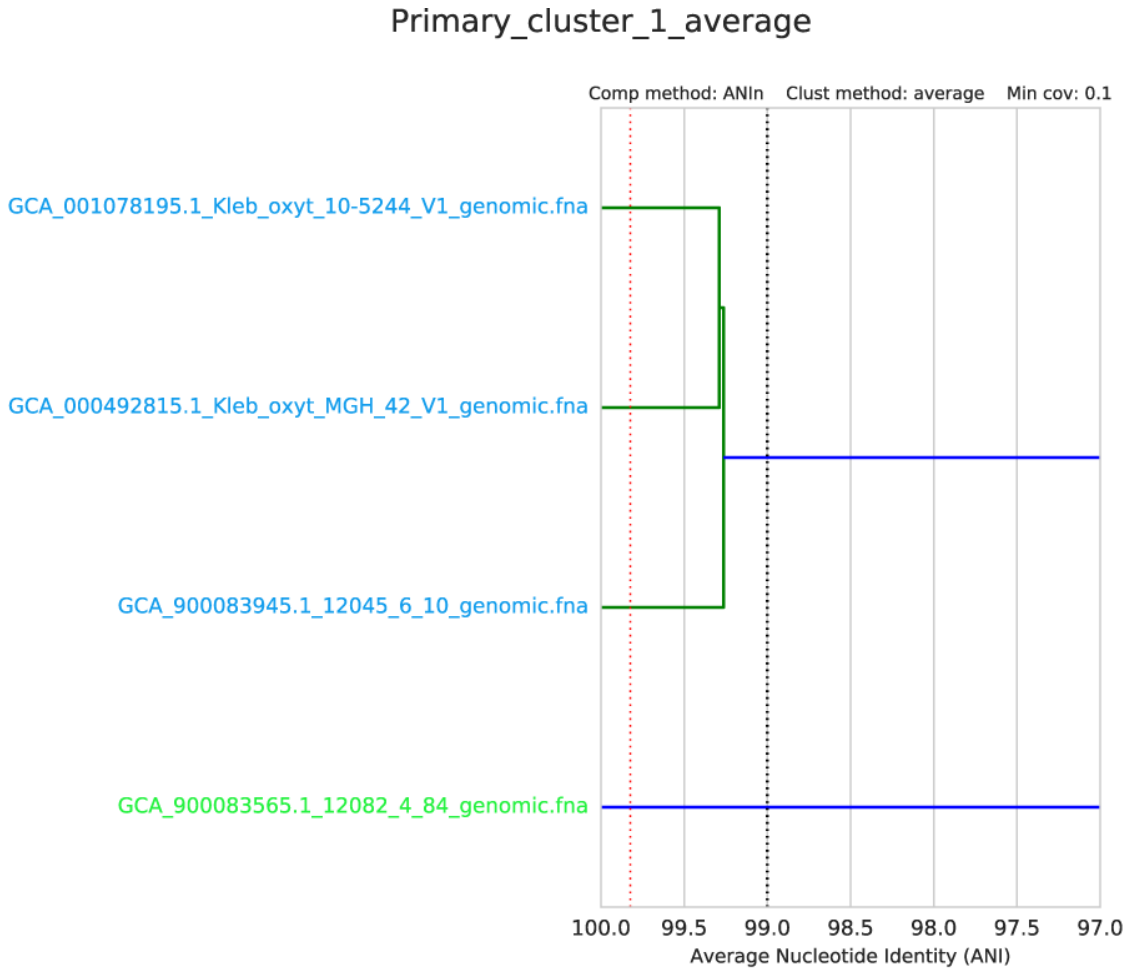
Each primary cluster with more than one member will have a page in the Secondary clustering dendrograms file. In this example, there is only one primary cluster with > 1 member.

This dendrogram summarizes the pair-wise distance between all organisms in each primary cluster, as determined by the secondary algorithm (gANI / ANIm). At the very top the primary cluster identity is shown, and information on the secondary clustering algorithm parameters are shown above the dendrogram itself. You can see in the above figure that this secondary clustering was performed with gANI, the minimum alignment coverage is 10%, and the hierarchical clustering method is average.

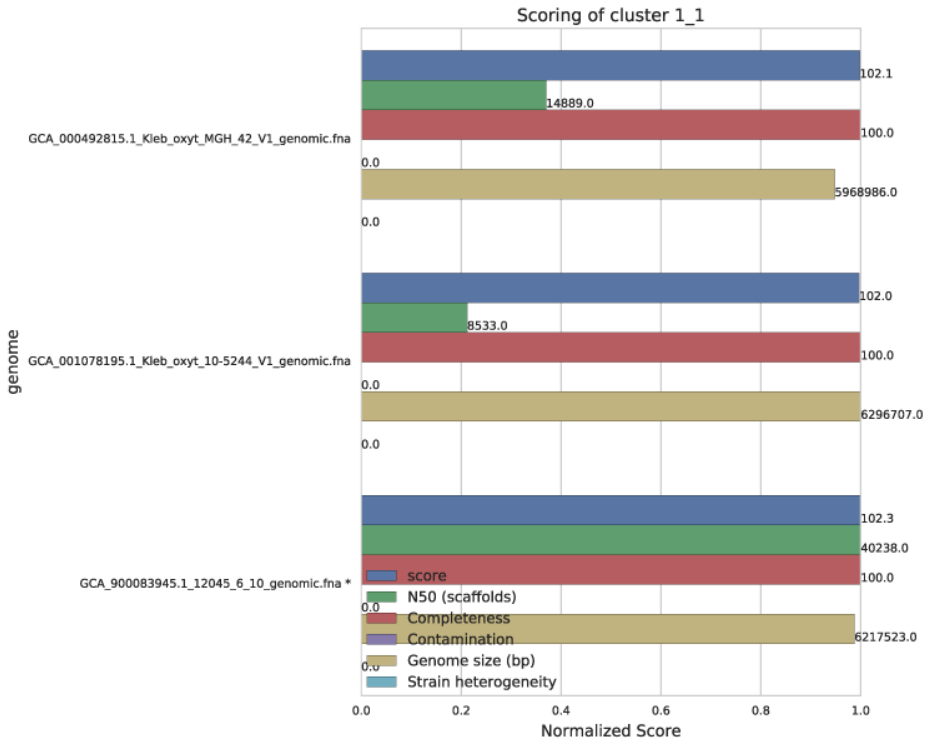
The black dotted line shows the **secondary clustering ANI** (in this case 99%). This value determines which genomes end up in the same secondary cluster, **and thus are considered to be the "same"**. In the above figure, two secondary

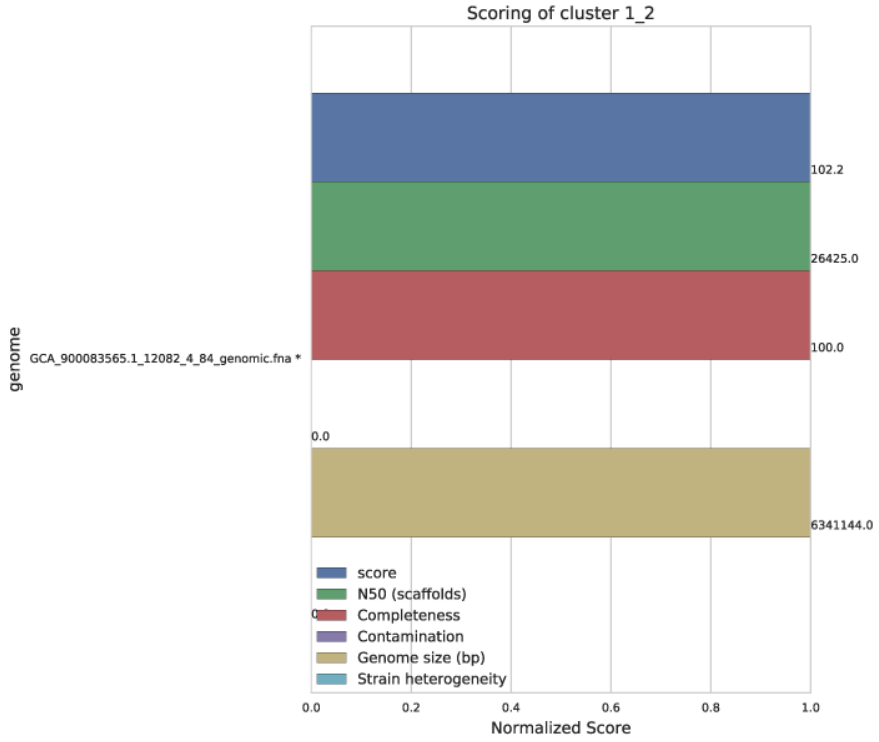
cluster are formed. The "best" genome of each secondary cluster is marked with as asterisk.

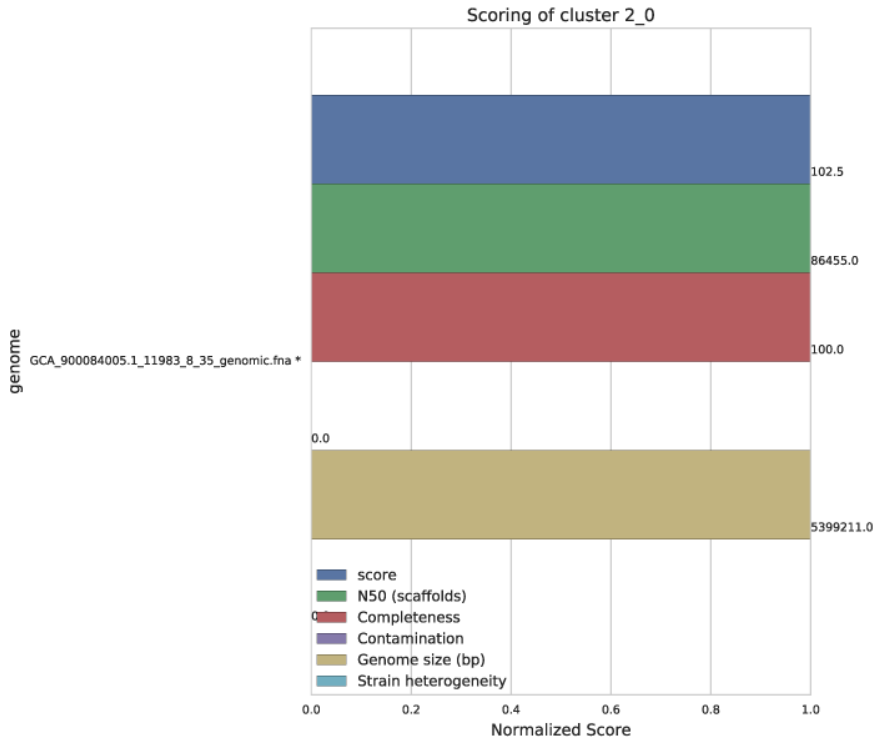
The red line shows the lowest ANI for a "self-vs-self" comparison of all genomes in the genome list. That is, when each genome in this primary cluster is compared to itself, the red line is the lowest ANI you get. This represents a "limit of detection" of sorts. gANI always results in 100% ANI when self-vs-self comparisons are performed, but ANIm does not (as shown in the figure below).



Cluster_scoring




Scoring of cluster 1_2
genome
GCA_900083565.1_12082_4_84_genomic.fna *
102.2
26425.0
100.0
0.0
6341144.0
score
N50 (scaffolds)
Completeness
Contamination
Genome size (bp)
Strain heterogeneity
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Score


Scoring of cluster 2_0
genome
GCA_900084005.1_11983_8_35_genomic.fna *
102.5
86455.0
100.0
0.0
5399211.0
score
N50 (scaffolds)
Completeness
Contamination
Genome size (bp)
Strain heterogeneity
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Score

Each secondary cluster will have its own page in the Cluster scoring figure. There are three secondary clusters in this example- 2 of which came from primary cluster 1, and 1 of which is the only member of primary cluster 2.

These figures show the score of each genome, as well as all metrics that can go into determining the score. This helps the user visualize the quality of all genomes, and ensure that they agree with the genome chosen as "best". The "best" genome is marked with an asterisk, and will always be the genome with the highest score.

One genome will be selected from each secondary cluster to be included in the de-replicated genome set. So in the above example, we will have 3 genomes in the de-replicated genome set. This is because the algorithm decided that all genomes in cluster 1_1 were the "same", and chose GCA_900083945 as the "best".

See also:

See *Module Descriptions* for information on how scoring is done and how to change it

Other figures

Clustering scatterplots provides some information about genome alignment statistics, and **Winning genomes** provides some information about only the "best" genomes of each replicate set, as well as a couple quick overall statistics.

2.4.2 Warnings

Warnings look for two things: **de-replicated genome similarity** and **secondary clusters that were almost different**. **All warnings are located in the log directory within the work directory, in a file titled warnings.txt**

de-replicated genome similarity warns when de-replicated genomes are similar to each other. This is to try and catch cases where similar genomes were split into different primary clusters, and thus failed to be de-replicated.

secondary clusters that were almost different alerts the user to cases where genomes are on the edge between being considered "same" or "different". That is, if a genome is close to one of the differentiating lines in the Primary and Secondary Clustering Dendrograms shown above.

2.4.3 Other data

The folder `dereplicated_genomes` holds a copy of the "best" genome of each secondary cluster

See also:

Almost all data that dRep generates at any point is able to be accessed by the user. This includes the full checkM results of each genome, the value of all genome comparisons, the raw hierarchical clustering files, the primary and secondary cluster identity of each genome, etc.

For information on where all of this is stored, see *Advanced Use*

2.5 Choosing parameters

The values used during de-replication and genome comparison are critical to understanding what the program is actually doing to your genome set.

There are two critical high-level decisions you must make before running dRep to de-replicate a genome set:

1. How similar do genomes need to be for them to be considered the "same"?

2. What is the minimum genome completeness allowed in analysis?

See also:

Module Descriptions For more general descriptions of routine parameters

2.5.1 What defines genomes as being "same"?

There is no standard definition of the average nucleotide identity (ANI) shared between two genomes that are the "same". This is a decision that the user must make on their own, depending on their own specific application. The ANI is determined by the **secondary clustering algorithm**, the **minimum secondary ANI** is the minimum ANI between genomes to be considered the "same", and the **minimum aligned fraction** is the minimum amount of genome overlap to believe the reported ANI value.

Tip: For help choosing this threshold, see the blog post: Are these microbes the "same"?.

Secondary clustering algorithm

The **secondary clustering algorithm** is the program that will calculate the accurate Average Nucleotide Identity (ANI) between genomes. The current options supported by dRep are ANIm (Richter 2009) and gANI (Varghese 2015).

- **ANIm** aligns whole genome fragments and calculates the nucleotide identity of aligned regions
- **gANI** aligns ORFs called by prodigal and calculates the nucleotide identity of aligned ORFs

Neither of these algorithms are perfect, especially in repeat-prone genomes. Regions of the genome which are not homologous can align to each other and artificially decrease ANI. In fact, when a genome is compared to itself, ANIm often reports values <100% for this reason. gANI is better about this, but seems to be more sensitive to genome subsetting.

- **ANImf** is very similar to ANIm, but filters the alignment before calculating ANI. This takes slighty more time, but is much more accurate with repeat regions

Minimum secondary ANI

The **minimum secondary ANI** is the minimum ANI between genomes for them to be considered the "same". For context, genomes of the same species typically have >=96.5% gANI (Varghese 2015).

The default value in dRep is 99%. Preliminary testing suggests that with gANI taking this up to 99.9% is probably safe, but higher than that is beyond the limit of detection. For ANIm you really can't go above 99%, as a comparison of a genome to itself can sometimes get that low. gANI is more thrown by genome incompleteness; ANIm is more thrown by repeat-regions.

Note: Keep in mind that in all cases you are collapsing closely related, but probably not identical, strains / genomes. This is simply the nature of the beast. If desired, you can compare the strains by mapping the original reads back to the de-replicated genome to visualize the strain cloud (Bendall 2016, blog post), or by comparing genomes within a secondary cluster using other methods (like Mauve)

Minimum aligned fraction

The **minimum aligned fraction** is the minimum amount that genomes must overlap to make the reported ANI value "count". This value is reported as part of the ANIm/gANI algorithms.

Imagine a scenario where two genomes of a separate phyla share a single identical transposon. When the ANIm/gANI algorithm is run, the transposon is probably the only part of the genomes that aligns, and the alignment will have 100%

ANI. This will result in a reported ANI of 100%, and reported **aligned fraction** of ~0.1%. The **minimum aligned fraction** is to handle the above scenario- anything with less than the minimum aligned fraction of genome alignment will have the ANI changed to 0. Default value is 10%.

Note: It has been suggested that a minimum aligned fraction of 60% should be applied to species-level taxonomic definitions (Varghese 2015). However, this is probably too stringent when incomplete genomes are being used (as is the case with genome de-replication)

2.5.2 What is the minimum genome completeness allowed in analysis?

This decision is much more complicated than the previous. Essentially, there exists a trade-off between computational efficiency and the minimum genome completeness.

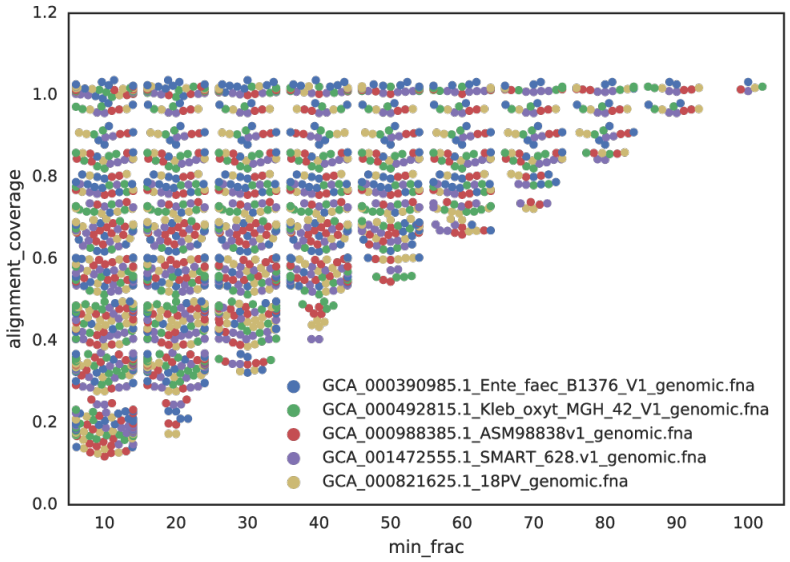


Fig. 2.1: **Figure A**: Five genomes are subset to fractions ranging from 10% - 100%, and fractions from the same genome are compared. The x-axis is the minimum genome completeness allowed. The looser this value is, the wider the range of aligned fractions.

As shown in the above Figure A, the lower the limit of genome completeness, the lower possible aligned fraction of two genomes. This makes sense- if you randomly take 20% of a genome, and than do the same thing again, when you compare these two random 20% subsets you would not expect very much of them to align. This "aligned fraction" really becomes a problem when you consider it's effect on Mash:

As shown in the above Figure B, the lower the aligned fraction, the lower the reported Mash ANI **for identical genomes**.

Remember- genomes are first divided into primary clusters using Mash, and then each primary cluster is divided into secondary clusters of the "same" genomes. Therefore, genomes which fit the definition of "same" **must** end up in the same primary cluster, or the program will never realize they're the same. As more incomplete genomes have lower Mash values (even if the genomes are truly identical; see **Figure B**), the more incomplete of genomes you allow into your genome list, the more you must decrease the **primary cluster threshold**.

Note: Having a lower **primary cluster threshold** which will result in larger primary clusters, which will result in

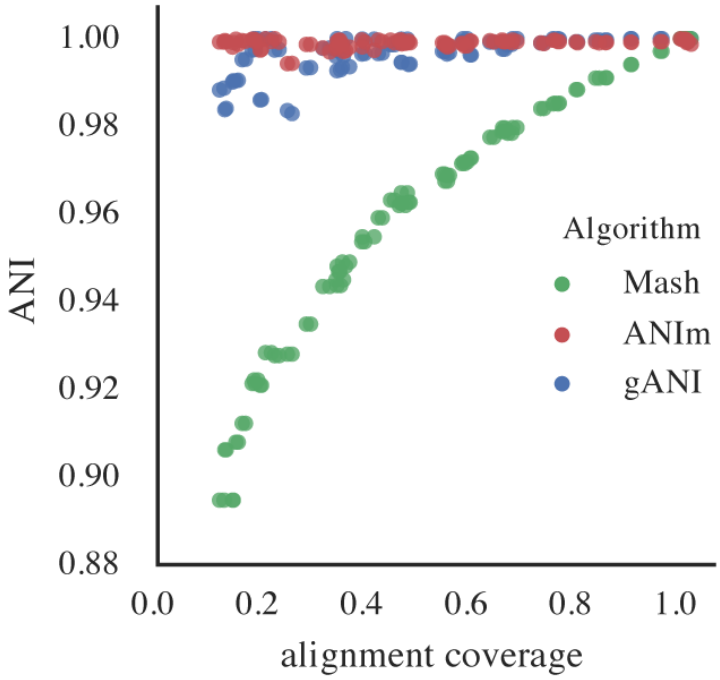


Fig. 2.2: **Figure B**: An identical *E. coli* genome is subset to fractions ranging from 10% - 100% and fractions are compared. When lower amounts of the genome align (due to incompleteness), Mash ANI is severely impacted

more required secondary comparisons. This will result in a longer run-time.

Still with me?

For example, say I set the minimum genome completeness to 50%. If I take an *E. coli* genome, subset it 50% 2 times, and compare those 2 subset genomes together, Mash will report an ANI of 96%. Therefore, the primary cluster threshold must be at least 96%, otherwise the two genomes could end up in different primary clusters, and thus would never have have the secondary algorithm run between them, and thus would not be de-replicated.

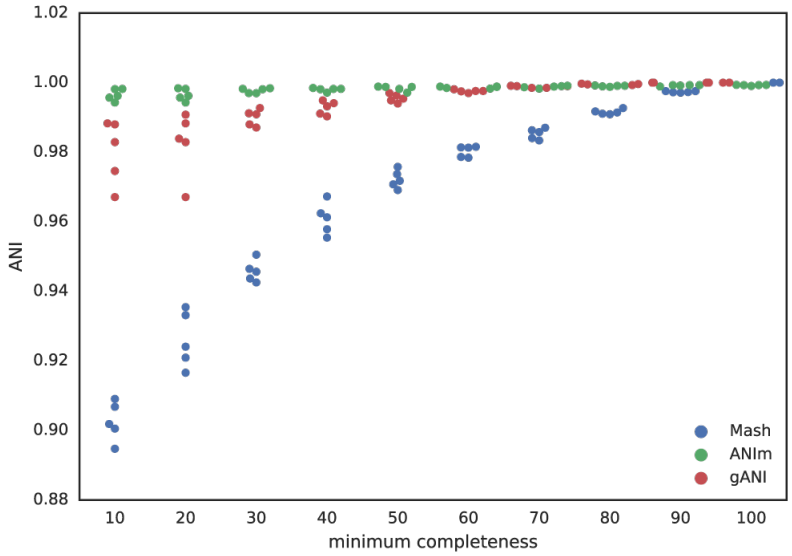
You don't want to set the primary cluster threshold super low, however, as this would result in more genomes being included in each primary cluster, and thus more secondary comparisons (which are slow), and thus a higher run-time.

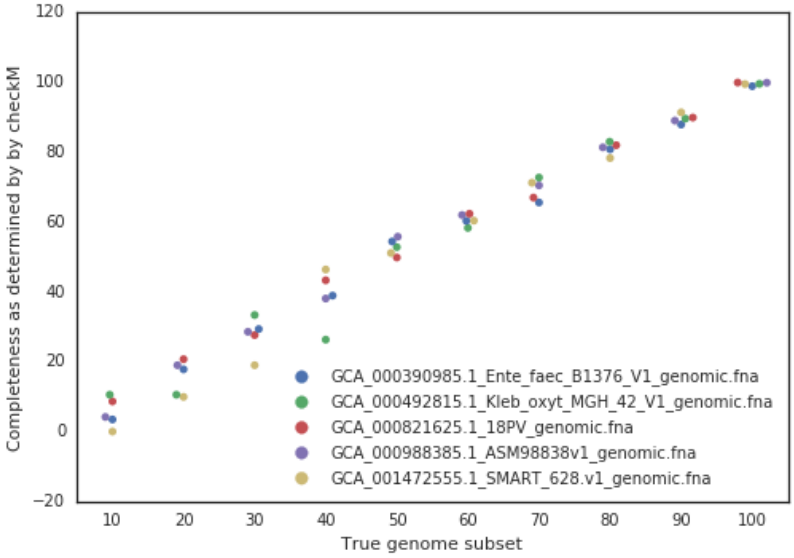
Putting this altogether gives us a figure with the lowest reported ANI of identical genomes being subset to different fractions. This figure only takes into account 5 different genomes, but gives a rough idea of the limits.

A final piece to consider is that when running dRep for real, the user doesn't actually know how incomplete their genomes are. They have to rely on metrics like single copy gene inventories to tell them. This is the reason phage and plasmids are not currently supported by dRep- there is no way of knowing how complete they are, and thus no way of filtering out the bins that are too incomplete. In general though, checkM is pretty good at accessing genome completeness:

Note: Some general guidelines for picking genome completeness thresholds:

- Going below 50% completeness is not recommended. The resulting genomes will be very crappy anyways, and even the secondary algorithms break-down at this point.
- Lowering the secondary ANI should result in a consummate lowering in MASH ANI. This is because you want Mash to group non-similar *and* incomplete genomes.


1.02
1.00
0.98
0.96
0.94
0.92
0.90
0.88
ANI
10
20
30
40
50
60
70
80
90
100
minimum completeness
Mash
ANIm
gANI


120
100
80
60
40
20
0
−20
Completeness as determined by by checkM
10
20
30
40
50
60
70
80
90
100
True genome subset
GCA_000390985.1_Ente_faec_B1376_V1_genomic.fna
GCA_000492815.1_Kleb_oxyt_MGH_42_V1_genomic.fna
GCA_000821625.1_18PV_genomic.fna
GCA_000988385.1_ASM98838v1_genomic.fna
GCA_001472555.1_SMART_628.v1_genomic.fna

- To make sure clusters are not being split unnecessarily, you can run the warnings at the end. See *Module Descriptions* for info

2.5.3 The Rest

The most important and confusing parameters are described above. For information on the other parameters, see *Module Descriptions*

2.6 Module Descriptions

The functionality of dRep is broken up into modules. The user can run the modules separately, or together in workflows. For example, you could run:

```
$ dRep filter example_workD -g path/to/genomes*.fasta

$ dRep cluster example_workD

$ dRep analyze example_workD -pl a
```

OR:

```
$ dRep compare example_workD -g path/to/genomes*.fasta
```

There are two ways of doing the same thing. To see a list of available modules, check the help:

```
$ dRep -h

               ...::: dRep v2.0.0 :::...

 Choose one of the operations below for more detailed help.
 Example: dRep dereplicate -h

 Workflows:
   dereplicate  -> Combine several of the operations below to de-replicate a genome␣
↪list
   compare      -> Simply compare a list of genomes

 Single operations:
   filter          -> Filter a genome list based on size, completeness, and/or␣
↪contamination
   cluster         -> Compare and cluster a genome list based on MASH and ANIn/gANI
   choose          -> Choose the best genome from each genome cluster
   evaluate        -> Evaluate genome de-replication
   bonus           -> Other random operations (currently just determine taxonomy)
   analyze         -> Make figures related to the above operations; test alternative␣
↪clustering
```

2.6.1 Work Directory

The work directory is where all of the program's internal workings, log files, cached data, and output is stored. When running dRep modules multiple times on the same dataset, **it is essential** that you use the same work directory so the program can find the results of previous runs.

See also:

Example Output for help finding where the output from your run is located in the work directory

Advanced Use for access to the raw internal data (which can be very useful)

2.6.2 Compare and Dereplicate

These are higher-level operations that call the modules below in succession.

Compare runs the modules:

- cluster
- bonus
- evaluate
- analyze

Dereplicate runs the modules:

- filter
- cluster
- choose
- bonus
- evaluate
- analyze

2.6.3 Filter

Filter is used filter the genome set (for why this is necessary, see *Choosing parameters*). This is done using checkM. All genomes which don't pass the length threshold are filtered first to avoid running checkM unnecessarily. All genomes which don't pass checkM thresholds are filtered before comparisons are run to avoid running comparisons unnecessarily.

Warning: All genomes must have at least one ORF called or else checkM will stall, so a length minimum of at least 10,000bp is recommended.

To see the command-line options, check the help:

```
$ dRep filter -h
usage: dRep filter [-p PROCESSORS] [-d] [-o] [-h] [-l LENGTH]
                   [-comp COMPLETENESS] [-con CONTAMINATION] [-str STRAIN_HTR]
                   [--skipCheckM] [-g [GENOMES [GENOMES ...]]] [--Chdb CHDB]
                   [--checkM_method {lineage_wf,taxonomy_wf}]
                   work_directory

positional arguments:
  work_directory        Directory where data and output
                        *** USE THE SAME WORK DIRECTORY FOR ALL DREP OPERATIONS ***

SYSTEM PARAMETERS:
  -p PROCESSORS, --processors PROCESSORS
```

```
                        threads (default: 6)
  -d, --dry             dry run- dont do anything (default: False)
  -o, --overwrite       overwrite existing data in work folder (default:
                        False)
  -h, --help            show this help message and exit

FILTERING OPTIONS:
  -l LENGTH, --length LENGTH
                        Minimum genome length (default: 500000)
  -comp COMPLETENESS, --completeness COMPLETENESS
                        Minumum genome completeness (default: 75)
  -con CONTAMINATION, --contamination CONTAMINATION
                        Maximum genome contamination (default: 25)
  -str STRAIN_HTR, --strain_htr STRAIN_HTR
                        Maximum strain heterogeneity (default: 25)
  --skipCheckM          Don't run checkM- will ignore con and comp settings
                        (default: False)

I/O PARAMETERS:
  -g [GENOMES [GENOMES ...]], --genomes [GENOMES [GENOMES ...]]
                        genomes to filter in .fasta format. Not necessary if
                        Bdb or Wdb already exist (default: None)
  --Chdb CHDB           checkM run already completed. Must be in --tab_table
                        format. (default: None)
  --checkM_method {lineage_wf,taxonomy_wf}
                        Either lineage_wf (more accurate) or taxonomy_wf
                        (faster) (default: lineage_wf)
```

2.6.4 Cluster

Cluster is the module that does the actual primary and secondary comparisons. Choosing parameters here can get a bit complicated- see *Choosing parameters* for information.

To see the command-line options, check the help:

```
$ dRep cluster -h
usage: dRep cluster [-p PROCESSORS] [-d] [-o] [-h] [-ms MASH_SKETCH]
                    [-pa P_ANI] [--S_algorithm {ANIn,gANI}] [-sa S_ANI]
                    [-nc COV_THRESH] [-n_PRESET {normal,tight}]
                    [--clusterAlg CLUSTERALG] [--SkipMash] [--SkipSecondary]
                    [-g [GENOMES [GENOMES ...]]]
                    work_directory

positional arguments:
  work_directory        Directory where data and output
                        *** USE THE SAME WORK DIRECTORY FOR ALL DREP OPERATIONS ***

SYSTEM PARAMETERS:
  -p PROCESSORS, --processors PROCESSORS
                        threads (default: 6)
  -d, --dry             dry run- dont do anything (default: False)
  -o, --overwrite       overwrite existing data in work folder (default:
                        False)
  -h, --help            show this help message and exit

CLUSTERING PARAMETERS:
  -ms MASH_SKETCH, --MASH_sketch MASH_SKETCH
```

```
                        MASH sketch size (default: 1000)
  -pa P_ANI, --P_ani P_ANI
                        ANI threshold to form primary (MASH) clusters
                        (default: 0.9)
  --S_algorithm {ANIn,gANI}
                        Algorithm for secondary clustering comaprisons
                        (default: ANIn)
  -sa S_ANI, --S_ani S_ANI
                        ANI threshold to form secondary clusters (default:
                        0.99)
  -nc COV_THRESH, --cov_thresh COV_THRESH
                        Minmum level of overlap between genomes when doing
                        secondary comparisons (default: 0.1)
  -n_PRESET {normal,tight}
                        Presents to pass to nucmer
                        tight   = only align highly conserved regions
                        normal  = default ANIn parameters (default: normal)
  --clusterAlg CLUSTERALG
                        Algorithm used to cluster genomes (passed to
                        scipy.cluster.hierarchy.linkage (default: average)
  --SkipMash            Skip MASH clustering, just do secondary clustering on
                        all genomes (default: False)
  --SkipSecondary       Skip secondary clustering, just perform MASH
                        clustering (default: False)

I/O PARAMETERS:
  -g [GENOMES [GENOMES ...]], --genomes [GENOMES [GENOMES ...]]
                        genomes to cluster in .fasta format. Not necessary if
                        already loaded sequences with the "filter" operation
                        (default: None)
```

2.6.5 Choose

Choose is the module that picks the best genome from each secondary cluster identified in **Cluster**. It does this based off of the formula:

$$score = A(completeness)^{\smile}B(contamination) + C(Contamination * (strain_{h}eterogeneity/100)) + D(log(N50)) + E(log(siz$$

Where A-E are command-line arguments, and the genome with the highest score is the "best". By default, A-E are 1,5,1,0.5,0 respectively.

To see the command-line options, check the help:

```
$ dRep choose -h
usage: dRep choose [-p PROCESSORS] [-d] [-o] [-h] [-comW COMPLETENESS_WEIGHT]
                   [-conW CONTAMINATION_WEIGHT]
                   [-strW STRAIN_HETEROGENEITY_WEIGHT] [-N50W N50_WEIGHT]
                   [-sizeW SIZE_WEIGHT]
                   [--checkM_method {taxonomy_wf,lineage_wf}]
                   work_directory

positional arguments:
  work_directory        Directory where data and output
                        *** USE THE SAME WORK DIRECTORY FOR ALL DREP OPERATIONS ***

SYSTEM PARAMETERS:
```

```
  -p PROCESSORS, --processors PROCESSORS
                        threads (default: 6)
  -d, --dry             dry run- dont do anything (default: False)
  -o, --overwrite       overwrite existing data in work folder (default:
                        False)
  -h, --help            show this help message and exit

SCORING CRITERIA
Based off of the formula:
A*Completeness - B*Contamination + C*(Contamination * (strain_heterogeneity/100)) +␣
↪D*log(N50) + E*log(size)

A = completeness_weight; B = contamination_weight; C = strain_heterogeneity_weight; D␣
↪= N50_weight; E = size_weight:
  -comW COMPLETENESS_WEIGHT, --completeness_weight COMPLETENESS_WEIGHT
                        completeness weight (default: 1)
  -conW CONTAMINATION_WEIGHT, --contamination_weight CONTAMINATION_WEIGHT
                        contamination weight (default: 5)
  -strW STRAIN_HETEROGENEITY_WEIGHT, --strain_heterogeneity_weight STRAIN_
↪HETEROGENEITY_WEIGHT
                        strain heterogeneity weight (default: 1)
  -N50W N50_WEIGHT, --N50_weight N50_WEIGHT
                        weight of log(genome N50) (default: 0.5)
  -sizeW SIZE_WEIGHT, --size_weight SIZE_WEIGHT
                        weight of log(genome size) (default: 0)

OTHER:
  --checkM_method {taxonomy_wf,lineage_wf}
                        Either lineage_wf (more accurate) or taxonomy_wf
                        (faster) (default: lineage_wf)
```

2.6.6 Analyze

Analyze is the module that makes all of the figures.

To see the command-line options, check the help:

```
$ dRep analyze -h
usage: dRep analyze [-p PROCESSORS] [-d] [-o] [-h] [-c CLUSTER] [-t THRESHOLD]
                    [-m {ANIn,gANI}] [-mc MINIMUM_COVERAGE]
                    [-a {complete,average,single,weighted}]
                    [-pl [PLOTS [PLOTS ...]]]
                    work_directory

positional arguments:
  work_directory        Directory where data and output
                        *** USE THE SAME WORK DIRECTORY FOR ALL DREP OPERATIONS ***

SYSTEM PARAMETERS:
  -p PROCESSORS, --processors PROCESSORS
                        threads (default: 6)
  -d, --dry             dry run- dont do anything (default: False)
  -o, --overwrite       overwrite existing data in work folder (default:
                        False)
  -h, --help            show this help message and exit

PLOTTING:
```

```
  -pl [PLOTS [PLOTS ...]], --plots [PLOTS [PLOTS ...]]
                        Plots. Input 'all' or 'a' to plot all
                        1) Primary clustering dendrogram
                        2) Secondary clustering dendrograms
                        3) Secondary clusters heatmaps
                        4) Comparison scatterplots
                        5) Cluster scorring plot
                        6) Winning genomes
                         (default: None)
```

2.6.7 Evaluate

Evaluate performs a series of checks to alert the user to potential problems with de-replication. It has two things that it can look for:

de-replicated genome similarity- this is comparing all of the de-replicated genomes to each other and making sure they're not too similar. This is to try and catch cases where similar genomes were split into different primary clusters, and thus failed to be de-replicated. *Depending on the number of de-replicated genomes, this can take a while*

secondary clusters that were almost different- this alerts you to cases where genomes are on the edge between being considered "same" or "different", depending on the clustering parameters you used. *This module reads the parameters you used during clustering from the work directory, so you don't need to specify them again.*

To see the command-line options, check the help:

```
$ dRep evaluate -h
usage: dRep evaluate [-p PROCESSORS] [-d] [-o] [-h] [--warn_dist WARN_DIST]
                     [--warn_sim WARN_SIM] [--warn_aln WARN_ALN]
                     [-e [EVALUATE [EVALUATE ...]]]
                     work_directory

positional arguments:
  work_directory        Directory where data and output
                        *** USE THE SAME WORK DIRECTORY FOR ALL DREP OPERATIONS ***

SYSTEM PARAMETERS:
  -p PROCESSORS, --processors PROCESSORS
                        threads (default: 6)
  -d, --dry             dry run- dont do anything (default: False)
  -o, --overwrite       overwrite existing data in work folder (default:
                        False)
  -h, --help            show this help message and exit

WARNINGS:
  --warn_dist WARN_DIST
                        How far from the threshold to throw cluster warnings
                        (default: 0.25)
  --warn_sim WARN_SIM   Similarity threshold for warnings between dereplicated
                        genomes (default: 0.98)
  --warn_aln WARN_ALN   Minimum aligned fraction for warnings between
                        dereplicated genomes (ANIn) (default: 0.25)

EVALUATIONS:
  -e [EVALUATE [EVALUATE ...]], --evaluate [EVALUATE [EVALUATE ...]]
                        Things to evaluate Input 'all' or 'a' to evaluate all
                        1) Evaluate de-replicated genome similarity
                        2) Throw warnings for clusters that were almost different
```

```
                        3) Generate a database of information on winning genomes
                         (default: None)
```

2.6.8 Bonus

Bonus consists of operations that don't really fit in with the functions of dRep, but can be helpful. Currently the only thing it can do is determine taxonomy of your bins. This is done using centrifuge, similar to how anvi'o does it. If you choose to use this option, the taxonomy of genome will be shown with the filename in most figures.

2.7 Advanced Use

2.7.1 Accessing Internal Information

All of internal information is stored in the work directory

work directory file-tree

```
workDirectory
./data
...../checkM/
...../Clustering_files/
...../gANI_files/
...../MASH_files/
...../ANIn_files/
...../prodigal/
./data_tables
...../Bdb.csv  # Sequence locations and filenames
...../Cdb.csv  # Genomes and cluster designations
...../Chdb.csv # CheckM results for Bdb
...../Mdb.csv  # Raw results of MASH comparisons
...../Ndb.csv  # Raw results of ANIn comparisons
...../Sdb.csv  # Scoring information
...../Wdb.csv  # Winning genomes
...../Widb.csv # Winning genomes' checkM information
./dereplicated_genomes
./figures
./log
...../cluster_arguments.json
...../logger.log
...../warnings.txt
```

Data Tables

Within the `data_tables` folder is where organized data lives. It's very easy to access this information, as it's all stored in .csv files.

Note: If you code in Python, I cannot recommend pandas enough for data-frame reading and manipulation. It's how all data is manipulated behind the scenes in dRep. See the API section below for easy access to these dataframes

---

Bdb Genome input locations, filenames, and lengths

Cdb Primary cluster, Secondary cluster, and information on clustering method for each genome

Chdb CheckM results for all genomes

Mdb Pair-wise Mash comparison results

Ndb Secondary comparison results

Tdb Taxonomy (as determined by centrifuge)

Sdb The score of each genome

Wdb The cluster and score of de-replicated genomes

Widb Useful checkM information on de-replicated genomes

Clustering files

These pickle files store information on both primary and secondary clusters. Loading the first value gives you the linkage, loading the second value gives you the db that was used to make the linkage, loading the third value give you a dictionary of the arguments that were used to make the linkage.

For example:

```
f = open(pickle, 'rb')
linkage = pickle.load(f)
db = pickle.load(f)
arguments = pickle.load(f)
```

Raw data

Refer to the above file structure to find the rest of the raw data. The data is kept from all program runs, so you can find the raw ANIm/gANI files, Mash sketches, prodigal gene predictions, centrifuge raw output, ect.

2.7.2 Using external genome quality information

If you already have your own genome quality information and would not like dRep to run checkM to generate it again, you can provide it using the *genomeInformation* flag.

The genomeInformation file must be in .csv format and have the columns "genome", "completeness", and "contamination". Columns "completeness" and "contamination" should be 0-100, and "genome" is the filename of the genome.

For example:

```
genome,completeness,contamination
Enterococcus_casseliflavus_EC20.fasta,98.28,0.0
Enterococcus_faecalis_T2.fna,98.28,0.0
Enterococcus_faecalis_TX0104.fa,96.55,0.0
Enterococcus_faecalis_YI6-1.fna,98.28,0.0
Escherichia_coli_Sakai.fna,100.0,0.0
```

2.7.3 Caching

The reason that dRep stores all of the raw information that it does is so that if future operations require the same file, it can just load the file that's already there instead of making it again. This goes for prodigal gene predictions, checkM, centrifuge, all ANI algorithms, ect. The data-frame that uses the information **will** be remade, but the information itself will not.

The reason I mention this is because if you would like to run another dRep operation that's similar to one you've already run, you can use the results of the last run to speed up the second run.

For example, say you've already run the dereplicate_wf using gANI and want to run the same thing but with ANIm to compare. You can make a copy of the gANI work directory, and then run the dereplicate_wf on the copy specifying the new secondary algorithm. It will have to run all of the ANIm comparisons, but will **not** re-run checkM, prodigal, centrifuge, ect., as the data will already be cached in the work directory.

Warning: Be warned, this is somewhat buggy and can easily get out of hand. While it does save time, sometimes it's just best to re-run the whole thing again with a clean start

2.7.4 API

See *dRep API* for the API to dRep.

For example:

```
from drep.WorkDirectory import WorkDirectory

wd = WorkDirectory('path/to/workdirectory')
Mdb = wd.get_db('Mdb')
Cdb = wd.get_db('Cdb')
...
```

This will work for all datatables

2.8 dRep API

This allows you to call the internal methods of dRep using your own python program

2.8.1 drep.d_filter

d_filter - a subset of drep

Filter genomes based on genome length or quality. Also can run prodigal and checkM

drep.d_filter.**calc_fasta_length**(*fasta_loc*)
Calculate the length of the .fasta file and retun length

Parameters **fasta_loc** – location of .fasta file

Returns total length of all .fasta files

Return type int

drep.d_filter.**calc_genome_info**(*genomes: list*)
Calculate the length and N50 of a list of genome locations

Parameters **genomes** – list of locations of genomes

Returns pandas dataframe with ["location", "length", "N50", "genome"]

Return type DataFrame

drep.d_filter.**calc_n50**(*loc*)
Calculate the N50 of a .fasta file

Parameters **fasta_loc** – location of .fasta file.

Returns N50 of .fasta file.

Return type int

drep.d_filter.**chdb_to_genomeInfo**(*chdb*)
Convert the output of checkM (chdb) into genomeInfo

Parameters **chdb** – dataframe of checkM

Returns genomeInfo

Return type DataFrame

drep.d_filter.**d_filter_wrapper**(*wd*, ***kwargs*)
Controller for the dRep filter operation

Parameters

- **wd** (WorkDirectory) – The current workDirectory
- ****kwargs** – Command line arguments

Keyword Arguments

- **genomes** – genomes to filter in .fasta format
- **genomeInfo** – location of .csv file with the columns: ["genome"(basename of .fasta file of that genome), "completeness"(0-100 value for completeness of the genome), "contamination"(0-100 value of the contamination of the genome)]
- **processors** – Threads to use with checkM / prodigal
- **overwrite** – Overwrite existing data in the work folder
- **debug** – If True, make extra output when running external scripts
- **length** – minimum genome length when filtering
- **completeness** – minimum genome completeness when filtering
- **contamination** – maximum genome contamination when filtering
- **noQualityFiltering** – Don't run checkM or do any quality-based filtering (not recommended)
- **checkM_method** – Either lineage_wf (more accurate) or taxonomy_wf (faster)

Returns stores Bdb.csv, Chdb.csv, and GenomeInfo.csv in the work directory

Return type Nothing

drep.d_filter.**filter_bdb**(*bdb*, *Gdb*, ***kwargs*)
Filter bdb based on Gdb

Parameters

- **bdb** – DataFrame with ["genome"]

- **Gdb** – DataFrame with ["genome", "completeness", "contamination"]

Keyword Arguments

- **min_comp** – Minimum genome completeness (%)
- **max_con** – Maximum genome contamination (%)

Returns bdb filtered based on completeness and contamination

Return type DataFrame

drep.d_filter.**run_checkM**(*genome_folder*, *checkm_outf*, ***kwargs*)
Run checkM

WARNING- this will result in wrong genome lenth and genome N50 estimate, due to it being run on prodigal output

Parameters

- **genome_folder** – location of folder to run checkM on - should be full of files ending in .faa (result of prodigal)
- **checkm_outf** – location of folder to store checkM output

Keyword Arguments

- **processors** – number of threads
- **checkm_method** – either lineage_wf or taxonomy_wf
- **debug** – log all of the commands
- **wd** – if you want to log commands, you also need the wd

drep.d_filter.**run_prodigal**(*genome_list*, *out_dir*, ***kwargs*)
Run prodigal on a set of genomes, store the output in the out_dir

Parameters

- **genome_list** – list of genomes to run prodigal on
- **out_dir** – output directory to store prodigal output

Keyword Arguments

- **processors** – number of processors to multithread with
- **exe_loc** – location of prodigal excutible (will try and find with shutil if not provided)
- **debug** – log all of the commands
- **wd** – if you want to log commands, you also need the wd

drep.d_filter.**validate_chdb**(*Chdb*, *bdb*)
Make sure all genomes in bdb are in Chdb

Parameters

- **Chdb** – dataframe of checkM information
- **bdb** – dataframe with ['genome']

2.8.2 drep.d_cluster

d_cluster - a subset of dRep

Clusters a list of genomes with both primary and secondary clustering

drep.d_cluster.**add_avani**(*db*)
add a column titled 'av_ani' to the passed in dataframe

dataframe must have rows reference, querey, and ani

Parameters **db** – dataframe

drep.d_cluster.**all_vs_all_MASH**(*Bdb*, *data_folder*, ***kwargs*)
Run MASH pairwise within all samples in Bdb

Parameters

- **Bdb** – dataframe with genome, location
- **data_folder** – location to store output files

Keyword Arguments

- **MASH_sketch** – size of mash sketches
- **dry** – dont actually run anything
- **processors** – number of processors to multithread with
- **mash_exe** – location of mash excutible (will try and find with shutil if not provided)
- **groupSize** – max number of mash sketches to hold in each folder
- **debug** – if True, log all of the commands
- **wd** – if you want to log commands, you also need the wd

drep.d_cluster.**cluster_genomes**(*genome_list*, *data_folder*, ***kwargs*)
Clusters a set of genomes using the dRep primary and secondary clustering method

Takes a number of command line arguments and returns a couple pandas dataframes. Done in a number of steps

Parameters

- **genomes** – list of genomes to be clustered
- **data_folder** – location where MASH and ANIn data will be stored

Keyword Arguments

- **processors** – Threads to use with checkM / prodigal
- **overwrite** – Overwrite existing data in the work folder
- **debug** – If True, make extra output when running external scripts
- **MASH_sketch** – MASH sketch size
- **S_algorithm** – Algorithm for secondary clustering comaprisons {ANImf,gANI,ANIn}
- **n_PRESET** – Presets to pass to nucmer {normal, tight}
- **P_ANI** – ANI threshold to form primary (MASH) clusters (default: 0.9)
- **S_ANI** – ANI threshold to form secondary clusters (default: 0.99)
- **SkipMash** – Skip MASH clustering, just do secondary clustering on all genomes
- **SkipSecondary** – Skip secondary clustering, just perform MASH clustering

- **COV_THRESH** – Minmum level of overlap between genomes when doing secondary comparisons (default: 0.1)
- **coverage_method** – Method to calculate coverage of an alignment {total,larger}
- **CLUSTERALG** – Algorithm used to cluster genomes (passed to scipy.cluster.hierarchy.linkage (default: average)
- **n_c** – nucmer argument c
- **n_maxgap** – nucmer argument maxgap
- **n_noextend** – nucmer argument noextend
- **n_method** – nucmer argument method
- **n_preset** – preset nucmer arrangements: {tight,normal}
- **wd** – workDirectory (needed to store clustering results and run prodigal)

Returns [Mdb(db of primary clustering), Ndb(db of secondary clustering, Cdb(clustering information))]

Return type list

drep.d_cluster.**cluster_hierarchical**(*db*, *linkage_method='single'*, *linkage_cutoff=0.1*)

Perform hierarchical clustering on a symmetrical distiance matrix

Parameters

- **db** – result of db.pivot usually
- **linkage_method** – passed to scipy.cluster.hierarchy.fcluster
- **linkage_cutoff** – distance to draw the clustering line (default = .1)

Returns [Cdb, linkage]

Return type list

drep.d_cluster.**cluster_mash_database**(*db*, ***kwargs*)

From a Mash database, cluster and return Cdb

Parameters **db** – Mdb (all_vs_all Mash results)

Keyword Arguments

- **clusterAlg** – how to cluster database (default = single)
- **P_ani** – threshold to cluster at (default = 0.9)

Returns [Cdb, [linkage, linkage_db, arguments]]

Return type list

drep.d_cluster.**compare_genomes**(*bdb*, *algorithm*, *data_folder*, ***kwargs*)

Compare a list of genomes using the algorithm specified

This method takes in bdb (a table with the columns location and genome), runs pair-wise comparisons between all genomes in the sample, and returns a table with at least the columns 'reference', 'querry', 'ani','coverage', depending on what algorithm is called

Parameters

- **bdb** – DataFrame with ['genome', 'location'] (drep.d_filter.load_genomes)
- **algorithm** – options are ANImf, ANIn, gANI
- **data_folder** – location to store output files

Keyword Arguments

- **wd** – either this or prod_folder needed for gANI
- **prod_folder** – either this or wd needed for gANI

Returns Ndb (['reference', 'querry', 'ani','coverage'])

Return type DataFrame

drep.d_cluster.**d_cluster_wrapper**(*workDirectory*, ***kwargs*)

Controller for the dRep cluster operation

Validates arguments, calls cluster_genomes, then stores the output

Parameters

- **wd** (WorkDirectory) – The current workDirectory
- ****kwargs** – Command line arguments

Keyword Arguments

- **processors** – Threads to use with checkM / prodigal
- **overwrite** – Overwrite existing data in the work folder
- **debug** – If True, make extra output when running external scripts
- **MASH_sketch** – MASH sketch size
- **S_algorithm** – Algorithm for secondary clustering comaprisons {ANImf,gANI,ANIn}
- **n_PRESET** – Presets to pass to nucmer {normal, tight}
- **P_ANI** – ANI threshold to form primary (MASH) clusters (default: 0.9)
- **S_ANI** – ANI threshold to form secondary clusters (default: 0.99)
- **SkipMash** – Skip MASH clustering, just do secondary clustering on all genomes
- **SkipSecondary** – Skip secondary clustering, just perform MASH clustering
- **COV_THRESH** – Minmum level of overlap between genomes when doing secondary comparisons (default: 0.1)
- **coverage_method** – Method to calculate coverage of an alignment {total,larger}
- **CLUSTERALG** – Algorithm used to cluster genomes (passed to scipy.cluster.hierarchy.linkage (default: average)
- **genomes** – genomes to cluster in .fasta format. Not necessary if already loaded sequences with the "filter" operation

Returns Stores Cdb, Mdb, Ndb, Bdb

drep.d_cluster.**estimate_time**(*comps*, *alg*)

Estimate time, in minutes, based on comparison algorithm and number of comparisons

Parameters

- **comps** – number of genomes comparisons to perform
- **alg** – algorthm used

Returns time to perfom comparison (in minutes)

Return type float

drep.d_cluster.**gen_animf_cmd**(*prefix*, *ref*, *querry*, ***kwargs*)
return animf command. It will be a single string, with the format: "nucmer cmd ; filter cmd"

Parameters

- **prefix** – desired file output name
- **ref** – location of reference genome
- **querry** – location of querry genome

Keyword Arguments **all** – passed on to gen_nucmer_cmd

Returns format is "nucmer cmd ; filter cmd"

Return type string

drep.d_cluster.**gen_filter_cmd**(*delta*, *out*)
return delta-filter command

Parameters

- **delta** – desired .delta file to filter
- **out** – desired output file

Returns cmd to run

Return type list

drep.d_cluster.**gen_gANI_cmd**(*file*, *g1*, *g2*, *exe*)
Generate a command to run gANI (ANIcalculator)

Will return [] if g1 == g2

Parameters

- **file** – output file name
- **g1** – location of the genes of genome1
- **g2** – location of the genes of genome2
- **exe** – location of gANI executible

Returns command to run

Return type list

drep.d_cluster.**gen_nucmer_cmd**(*prefix*, *ref*, *querry*, *c='65'*, *noextend=False*, *maxgap='90'*, *method='mum'*)
Generate command to run with nucmer

Parameters

- **prefix** – desired output name (will have .delta appended)
- **ref** – location of reference genome
- **querry** – location of querry genomes

Keyword Arguments

- **c** – c value
- **noextend** – either True or False
- **maxgap** – maxgap
- **method** – detault is 'mum'

Returns command to run number

Return type list

drep.d_cluster.**genome_hierarchical_clustering**(*Ndb*, ***kwargs*)
Cluster ANI database

Parameters **Ndb** – result of secondary clustering

Keyword Arguments

- **clusterAlg** – how to cluster the database (default = single)
- **S_ani** – thershold to cluster at (default = .99)
- **cov_thresh** – minumum coverage to be included in clustering (default = .5)
- **cluster** – name of the cluster
- **comp_method** – comparison algorithm used

Returns [Cdb, {cluster:[linkage, linkage_db, arguments]}]

Return type list

drep.d_cluster.**iteratre_clusters**(*Bdb*, *Cdb*, *id='primary_cluster'*)
An iterator: Given Bdb and Cdb, yeild smaller Bdb's in the same cluster

Parameters

- **Bdb** – [genome, location]
- **Cdb** – [genome, id]
- **id** – what to iterate on (default = 'primary_cluster')

Returns [d(subset of b), cluster(name of cluster)]

Return type list

drep.d_cluster.**load_genomes**(*genome_list*)
Takes a list of genome locations, returns a pandas dataframe with the locations and the basename of the genomes

Parameters **genome_list** – list of genome locations

Returns pandas dataframe with columns ['genome', 'location']

Return type DataFrame

drep.d_cluster.**make_linkage_Ndb**(*Ndb*, ***kwargs*)
Filter the Ndb in accordance with the kwargs. Average reciprical ANI values

Parameters **Ndb** – result of secondary clutsering

Keyword Arguments **cov_thresh** – minimum coverage threshold (default = 0.5)

Returns pivoted DataFrame ready for clustering

Return type DataFrame

drep.d_cluster.**parse_delta**(*filename*)
Parse a .delta file from nucmer

Parameters **filename** – location of .delta file

Returns [alignment_length, similarity_errors]

Return type list

drep.d_cluster.**parse_gani_file**(*file*)
Parse gANI file, return dictionary of results

Parameters **file** – location of gANI file

Returns results in the gANI file

Return type dict

drep.d_cluster.**process_deltafiles**(*deltafiles*, *org_lengths*, *logger=None*, ***kwargs*)
Parse a list of delta files into a pandas dataframe

Files NEED to be named in the format genome1_vs_genome2.delta, or genome1_vs_genome2.filtered.delta

Parameters

- **deltafiles** – list of .delta files
- **org_lengths** – dictionary of genome to genome length

Keyword Arguments

- **coverage_method** – default is larger
- **logger** – if not None, will log 0 errors

Returns information about the alignments

Return type DataFrame

drep.d_cluster.**process_gani_files**(*files*)
From a list of gANI output files, return a parsed DataFrame

Parameters **list** – files

Returns Ndb

Return type DataFrame

drep.d_cluster.**run_pairwise_ANImf**(*genome_list*, *ANIn_folder*, ***kwargs*)
Given a list of genomes and an output folder, compare all genomes using ANImf

Parameters

- **genome_list** – list of locations of genome files
- **ANIn_folder** – folder to store the output of comparison

Keyword Arguments

- **processors** – threads to use
- **debug** – if true save extra output
- **wd** – needed if debug is True

drep.d_cluster.**run_pairwise_ANIn**(*genome_list*, *ANIn_folder*, ***kwargs*)
Given a list of genomes and an output folder, compare all genomes using ANImf

Parameters

- **genome_list** – list of locations of genome files
- **ANIn_folder** – folder to store the output of comparison

Keyword Arguments

- **processors** – threads to use
- **debug** – if true save extra output

- **wd** – needed if debug is True

drep.d_cluster.**run_pairwise_gANI**(*bdb*, *gANI_folder*, *prod_folder*, ***kwargs*)
Run pairwise gANI on a list of Genomes

Parameters

- **bdb** – DataFrame with ['genome', 'location']
- **gANI_folder** – folder to store gANI output
- **prod_folder** – folder containing prodigal output from genomes (will run if needed)

Keyword Arguments

- **debug** – log all of the commands
- **wd** – if you want to log commands, you also need the wd
- **processors** – threads to use

Returns Ndb for gANI

Return type DataFrame

2.8.3 drep.d_choose

d_choose - a subset of drep

Choose best genome from each cluster

drep.d_choose.**choose_winners**(*Cdb*, *Gdb*, ***kwargs*)
Make a scoring database and pick the winner of each cluster

Parameters

- **Cdb** – clustering database
- **Gdb** – genome information database

Keyword Arguments **wrapper** (*See*) –

Returns [Sdb (scoring database), Wdb (winner database)]

Return type List

drep.d_choose.**d_choose_wrapper**(*wd*, ***kwargs*)
Controller for the dRep choose operation

Based off of the formula: A*Completeness - B*Contamination + C*(Contamination * (strain_heterogeneity/100)) + D*log(N50) + E*log(size)

A = completeness_weight; B = contamination_weight; C = strain_heterogeneity_weight; D = N50_weight; E = size_weight

Parameters

- **wd** (WorkDirectory) – The current workDirectory
- ****kwargs** – Command line arguments

Keyword Arguments

- **genomeInfo** – .csv genomeInfo file
- **noQualityFiltering** – Don't run checkM or do any quality-based filtering (not recommended)

- **checkM_method** – Either lineage_wf (more accurate) or taxonomy_wf (faster)
- **completeness_weight** – see formula
- **contamination_weight** – see formula
- **strain_heterogeneity_weight** – see formula
- **N50_weight** – see formula
- **size_weight** – see formula

Returns Makes Sdb (scoreDb) in the workDirectory

drep.d_choose.**pick_winners**(*Sdb*, *Cdb*)

Based on clustering and scores, pick the best genome from every cluster

Parameters

- **Sdb** – score of every genome
- **Cdb** – clustering

Returns Wdb (winner database)

Return type DataFrame

drep.d_choose.**score_genomes**(*genomes*, *Gdb*, ***kwargs*)

Calculate the scores for a list of genomes

Parameters

- **genomes** – list of genomes
- **Gdb** – genome information database

Keyword Arguments **wrapper** (*See*) –

Returns Sdb (scoring database)

Return type DataFrame

drep.d_choose.**score_row**(*row*, ***kwargs*)

Perform the scoring of a row based on kwargs

Parameters **row** – row of genome information

Keyword Arguments

- **noQualityFiltering** – Don't run checkM or do any quality-based filtering (not recommended)
- **completeness_weight** – see formula
- **contamination_weight** – see formula
- **strain_heterogeneity_weight** – see formula
- **N50_weight** – see formula
- **size_weight** – see formula

Returns score

Return type float

2.8.4 drep.d_analyze

d_analyze - a subset of drep

Make plots based on de-replication

drep.d_analyze.**calc_dist**(*x1*, *y1*, *x2*, *y2*)
Return distance from two points

Args: self explainatory

Returns distance

Return type int

drep.d_analyze.**cluster_test_wrapper**(*wd*, ***kwargs*)
DEPRICATED

drep.d_analyze.**d_analyze_wrapper**(*wd*, ***kwargs*)
Controller for the dRep analyze operation

Parameters

- **wd** – The current workDirectory
- ****kwargs** – Command line arguments

Keyword Arguments **plots** – List of plots to make [list of ints, 1-6]

Returns Makes some plots

drep.d_analyze.**fancy_dendrogram**(*linkage*, *names*, *name2color=False*, *threshold=False*, *self_thresh=False*)
Make a fancy dendrogram

drep.d_analyze.**gen_color_dictionary**(*names*, *name2cluster*)
Make the dictionary name2color

Parameters

- **names** – key in the returned dictionary
- **name2cluster** – a dictionary of name to it's cluster

Returns name -> color

Return type dict

drep.d_analyze.**gen_color_list**(*names*, *name2cluster*)
Make a list of colors the same length as names, based on their cluster

drep.d_analyze.**get_highest_self**(*db*, *genomes*, *min=0.0001*)
Return the highest ANI value resulting from comparing a genome to itself

drep.d_analyze.**mash_dendrogram_from_wd**(*wd*, *plot_dir=False*)
From the wd and kwargs, call plot_MASH_dendrogram

Parameters

- **wd** – WorkDirectory
- **plot_dir** (*optional*) – Location to store figure

Returns Shows plot, makes a plot in the plot_dir

drep.d_analyze.**normalize**(*df*)
Normalize all columns in df to 0-1 except 'genome' or 'location'

Parameters **df** – DataFrame

Returns Nomralized

Return type DataFrame

drep.d_analyze.**plot_ANIn_vs_ANIn_cov**(*Ndb*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_ANIn_vs_len**(*Mdb*, *Ndb*, *exclude_zero_MASH=True*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_MASH_dendrogram**(*Mdb*, *Cdb*, *linkage*, *threshold=False*, *plot_dir=False*)
Make a dendrogram of the primary clustering

Parameters

- **Mdb** – DataFrame of Mash comparison results
- **Cdb** – DataFrame of Clustering results
- **linkage** – Result of scipy.cluster.hierarchy.linkage
- **threshold** (*optional*) – Line to plot on x-axis
- **plot_dir** (*optional*) – Location to store plot

Returns Makes and shows plot

drep.d_analyze.**plot_MASH_vs_ANIn_ani**(*Mdb*, *Ndb*, *exclude_zero_MASH=True*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_MASH_vs_ANIn_cov**(*Mdb*, *Ndb*, *exclude_zero_MASH=True*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_MASH_vs_len**(*Mdb*, *Ndb*, *exclude_zero_MASH=True*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_MASH_vs_secondary_ani**(*Mdb*, *Ndb*, *Cdb*, *exclude_zero_MASH=True*)
Makes plot and retuns plt.cgf()

All parameters are obvious

drep.d_analyze.**plot_binscoring_from_wd**(*wd*, *plot_dir*, ***kwargs*)
From the wd and kwargs, call plot_winner_scoring_complex

Parameters

- **wd** – WorkDirectory
- **plot_dir** (*optional*) – Location to store figure

Returns Shows plot, makes a plot in the plot_dir

drep.d_analyze.**plot_clustertest**(*linkage*, *names*, *wd*, ***kwargs*)
DEPREICATED

names can be gotten like: db = db.pivot("reference","querry","ani") names = list(db.columns)

drep.d_analyze.**plot_scatterplots**(*Mdb*, *Ndb*, *Cdb*, *plot_dir=False*)
Make scatterplots comparing genome comparison algorithms

- plot_MASH_vs_ANIn_ani(Mdb, Ndb) - Plot MASH_ani vs. ANIn_ani (including correlation)
- plot_MASH_vs_ANIn_cov(Mdb, Ndb) - Plot MASH_ani vs. ANIn_cov (including correlation)
- plot_ANIn_vs_ANIn_cov(Mdb, Ndb) - Plot ANIn vs. ANIn_cov (including correlation)
- plot_MASH_vs_len(Mdb, Ndb) - Plot MASH_ani vs. length_difference (including correlation)
- plot_ANIn_vs_len(Ndb) - Plot ANIn vs. length_difference (including correlation)

Parameters

- **Mdb** – DataFrame of Mash comparison results
- **Ndb** – DataFrame of secondary clustering results
- **Cdb** – DataFrame of Clustering results
- **plot_dir** (*optional*) – Location to store plot

Returns Makes and shows plot

drep.d_analyze.**plot_scatterplots_from_wd**(*wd*, *plot_dir*, ***kwargs*)
From the wd and kwargs, call plot_scatterplots

Parameters

- **wd** – WorkDirectory
- **plot_dir** (*optional*) – Location to store figure

Returns Shows plot, makes a plot in the plot_dir

drep.d_analyze.**plot_secondary_dendrograms_from_wd**(*wd*, *plot_dir*, ***kwargs*)
From the wd and kwargs, make the secondary dendrograms

Parameters

- **wd** – WorkDirectory
- **plot_dir** (*optional*) – Location to store figure

Returns Makes plot

drep.d_analyze.**plot_secondary_mds_from_wd**(*wd*, *plot_dir*, ***kwargs*)
Make a .pdf of MDS of each cluster

Parameters

- **wd** – WorkDirectory
- **plot_dir** (*optional*) – Location to store figure

Returns Makes plot

drep.d_analyze.**plot_winner_scoring_complex**(*Wdb*, *Sdb*, *Cdb*, *Gdb*, *plot_dir=False*, ***kwargs*)
Make a plot showing the genome scoring for all genomes

Parameters

- **Wdb** – DataFrame of winning dereplicated genomes
- **Sdb** – Scores of all genomes
- **Cdb** – DataFrame of Clustering results

- **Gdb** – DataFrame of genome scoring information
- **plot_dir** (*optional*) – Location to store plot

Returns makes plot

drep.d_analyze.**plot_winners**(*Wdb*, *Gdb*, *Wndb*, *Wmdb*, *Widb*, *plot_dir=False*, ***kwargs*)
Make a bunch of plots about the de-replicated genomes

THIS REALLY NEEDS IMPROVED UPON

drep.d_analyze.**plot_winners_from_wd**(*wd*, *plot_dir*, ***kwargs*)
From the wd and kwargs, call plot_winners

Parameters

- **wd** – WorkDirectory
- **plot_dir** – Location to store figure

Returns Shows plot, makes a plot in the plot_dir

2.8.5 drep.WorkDirectory

This module provides access to the workDirectory

The directory layout:

```
workDirectory
./data
...../MASH_files/
...../ANIn_files/
...../gANI_files/
...../Clustering_files/
...../checkM/
........./genomes/
........./checkM_outdir/
...../prodigal/
./figures
./data_tables
...../Bdb.csv  # Sequence locations and filenames
...../Mdb.csv  # Raw results of MASH comparisons
...../Ndb.csv  # Raw results of ANIn comparisons
...../Cdb.csv  # Genomes and cluster designations
...../Chdb.csv # CheckM results for Bdb
...../Sdb.csv  # Scoring information
...../Wdb.csv  # Winning genomes
./dereplicated_genomes
./log
...../logger.log
...../cluster_arguments.json
```

class drep.WorkDirectory.**WorkDirectory**(*location*)
Bases: object

Object to interact with the workDirectory

Parameters **location** (*str*) – location to make the workDirectory

firstLevels = ['data', 'figures', 'data_tables', 'dereplicated_genomes', 'log']

get_cluster(*name*)
Get the cluster passed in

Parameters **name** – name of the cluster

Returns cluster

get_db(*name*, *return_none=True*)
Get database from self.data_tables

Parameters

- **name** – name of dataframe
- **return_none** – if True will return None if database not found; otherwise assert False

get_dir(*dir*)
Get the location of one of the named directory types

Parameters **dir** – Name of directory to find

Returns Location of requested directory

Return type string

get_loc(*what*)
Get the location of Things

Parameters **what** – string of what to get the location of

Returns location of what

Return type string

get_primary_linkage()
Get the primary linkage cluster

hasDb(*db*)
If db is in the data_tables, return True

import_arguments(*loc*)
Given the location of the log directory, load it

import_clusters(*loc*)
Given the location of the cluster files, load them

import_data_tables(*loc*)
Given the location of the datatables, load them

load_cached()
The wrapper to load everything it has into attributes

make_fileStructure()
Make the top level file structure

store_db(*db*, *name*, *overwrite=False*)
Store a dataframe in the workDirectory

Will make a physical copy in the datatables folder

Parameters

- **db** – pandas dataframe to store
- **name** – name to store it under (will add .csv automatically)
- **overwrite** – if True, overwrite if DataFrame with same name already exists

store_special(*name*, *thing*)
Store special items in the work directory

Parameters

- **name** – what to store
- **thing** – actual thing to store

Python Module Index

d

Index

N

P

R

S

V

W