drep Documentation

Release 3.0.0

Matt Olm

Apr 09, 2017

Contents

1 de-replication of microbial genomes **1**

2 Contents **3**

- 2.1 Overview . . . 3
  - 2.1.1 Genome de-replication . . . 3
  - 2.1.2 Genome comparison . . . 4
- 2.2 Installation . . . 5
  - 2.2.1 Using pip . . . 5
  - 2.2.2 Dependencies . . . 6
  - 2.2.3 Testing . . . 6
  - 2.2.4 pyenv . . . 6
- 2.3 Quick Start . . . 6
  - 2.3.1 De-replication . . . 7
  - 2.3.2 Genome comparison . . . 7
- 2.4 Example Output . . . 7
  - 2.4.1 Figures . . . 8
    - 2.4.1.1 Primary_clustering_dendrogram . . . 8
    - 2.4.1.2 Secondary_clustering_dendrograms . . . 9
    - 2.4.1.3 Cluster_scoring . . . 11
    - 2.4.1.4 Other figures . . . 13
  - 2.4.2 Warnings . . . 13
  - 2.4.3 Other data . . . 13
- 2.5 Module Descriptions . . . 13
  - 2.5.1 Work Directory . . . 14
  - 2.5.2 Filter . . . 14
  - 2.5.3 Cluster . . . 15
  - 2.5.4 Choose . . . 16
  - 2.5.5 Analyze . . . 18
  - 2.5.6 Evaluate . . . 18
  - 2.5.7 Other . . . 19
- 2.6 Choosing parameters . . . 20
  - 2.6.1 What defines genomes as being “same”? . . . 20
    - 2.6.1.1 Secondary clustering algorithm . . . 20
    - 2.6.1.2 Minimum secondary ANI . . . 20
    - 2.6.1.3 Minimum aligned fraction . . . 21
  - 2.6.2 What is the minimum genome completeness allowed in analysis? . . . 21

2.6.3 The Rest . . . 24
2.7 Advanced Use . . . 24
2.7.1 Accessing Internal Information . . . 24
2.7.1.1 work directory file-tree . . . 24
2.7.1.2 Data Tables . . . 24
2.7.1.3 Clustering files . . . 25
2.7.1.4 Raw data . . . 25
2.7.2 Caching . . . 25
2.7.3 API . . . 26

CHAPTER 1

de-replication of microbial genomes

The publication is available at 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

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.

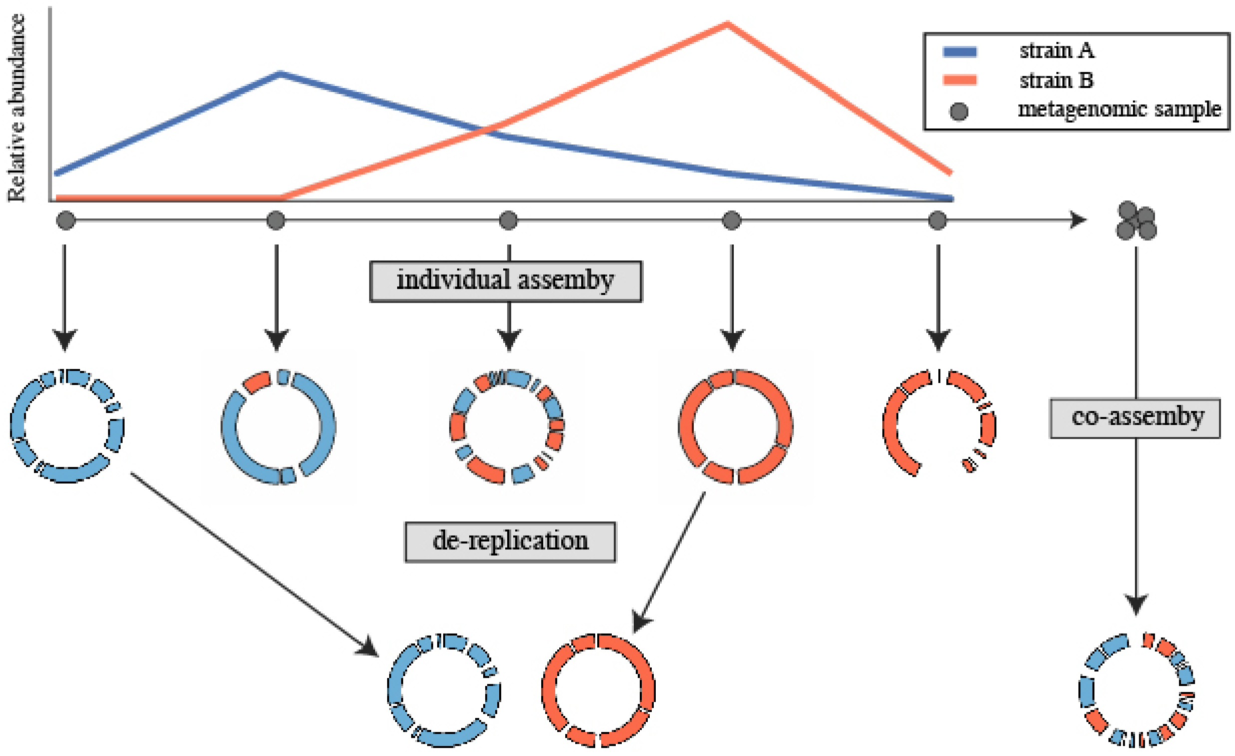
The publication is available at bioRxiv.

Source code is available on GitHub.

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:

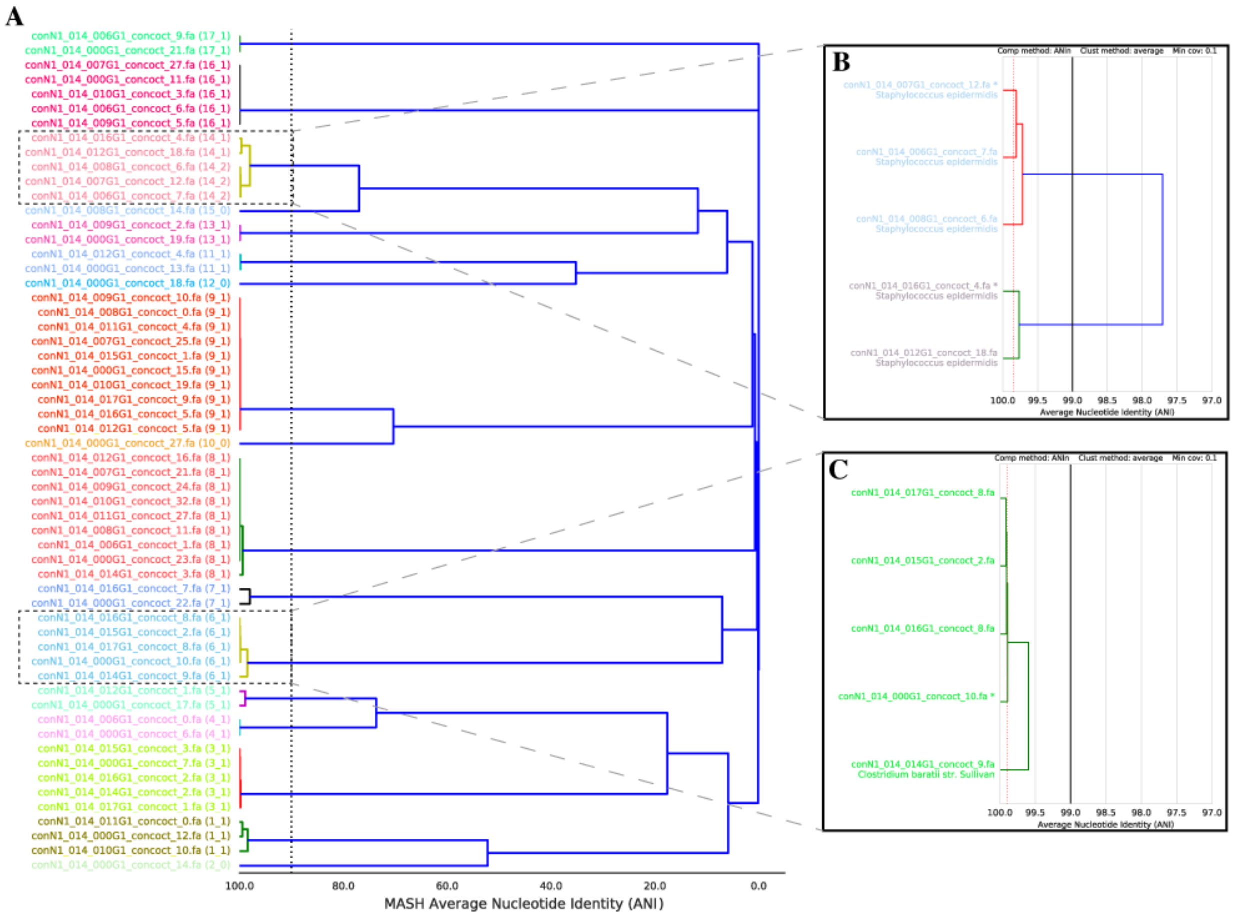
- Assemble each sample separately using your favorite assembler. You can also perform a co-assembly to catch low-abundance microbes
- Bin each assembly (and co-assembly) separately using your favorite binner
- Pull the bins from all assemblies together and run dRep on them
- Perform downstream analysis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

Genome comparison

Genome comparison is simply comparing 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 (gANI). 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. gANI 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 (gANI or ANIm) on sets of genomes that have at least 90% Mash ANI. This results it a great decrease in the number of (slow) secondary comparisons that need to be run while maintaining the sensitivity of gANI.



Installation

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

Dependencies

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

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 CalculateANI) - 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.

Testing

To make sure everything is installed correctly you can run the dRep test suite:

```
$ cd drep/tests

$ python test_suite.py
```

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)

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 v0.5.0 :::...

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

 Workflows:
  dereplicate_wf  -> Combine several of the operations below to de-replicate a␣
↪genome list
  compare_wf      -> 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
  adjust          -> Adjust genome clusters
  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
```

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_wf 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

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

Example Output

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

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

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

See also:

Overview for general information on the program

Quick Start for more information on dereplicate_wf

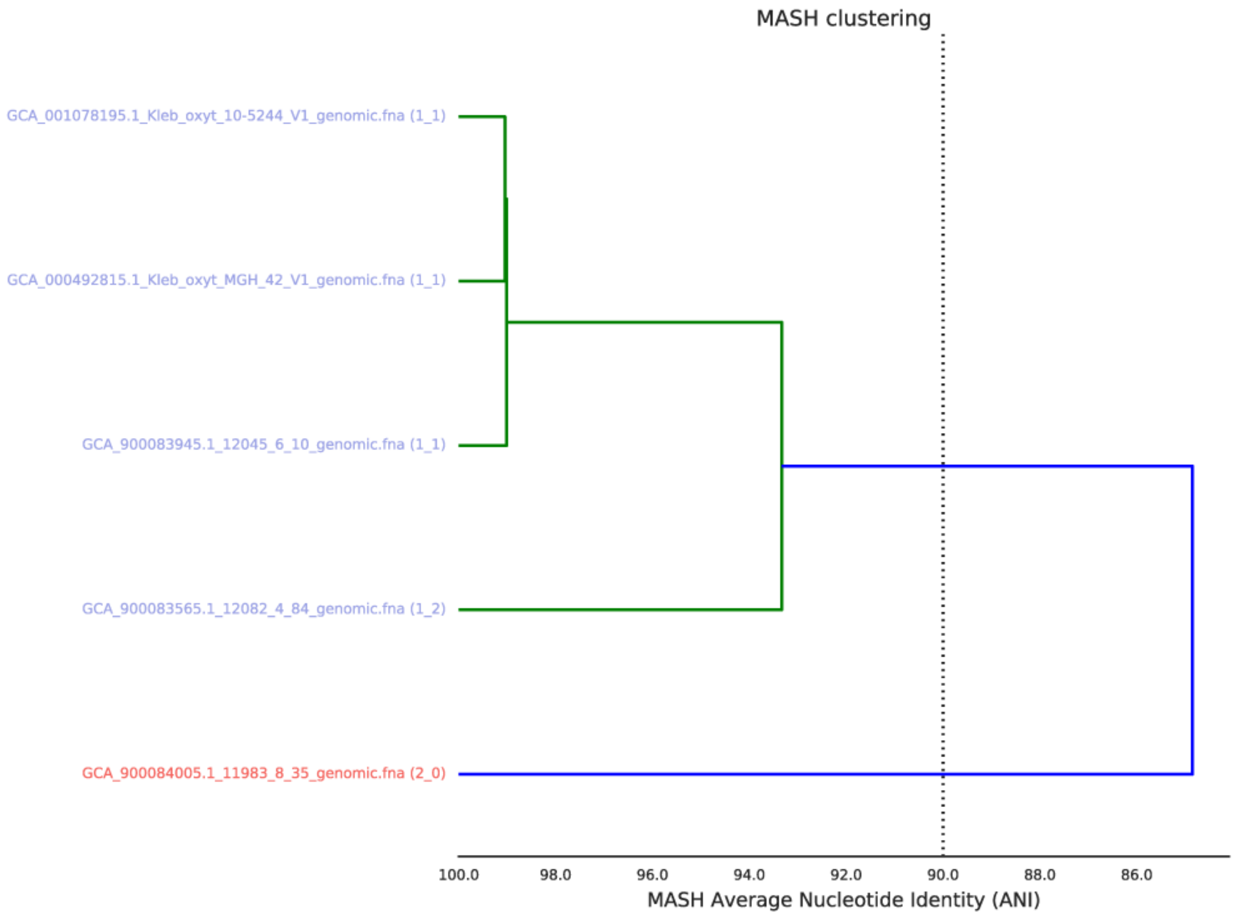
Module Descriptions for a more detailed description of what the modules do

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

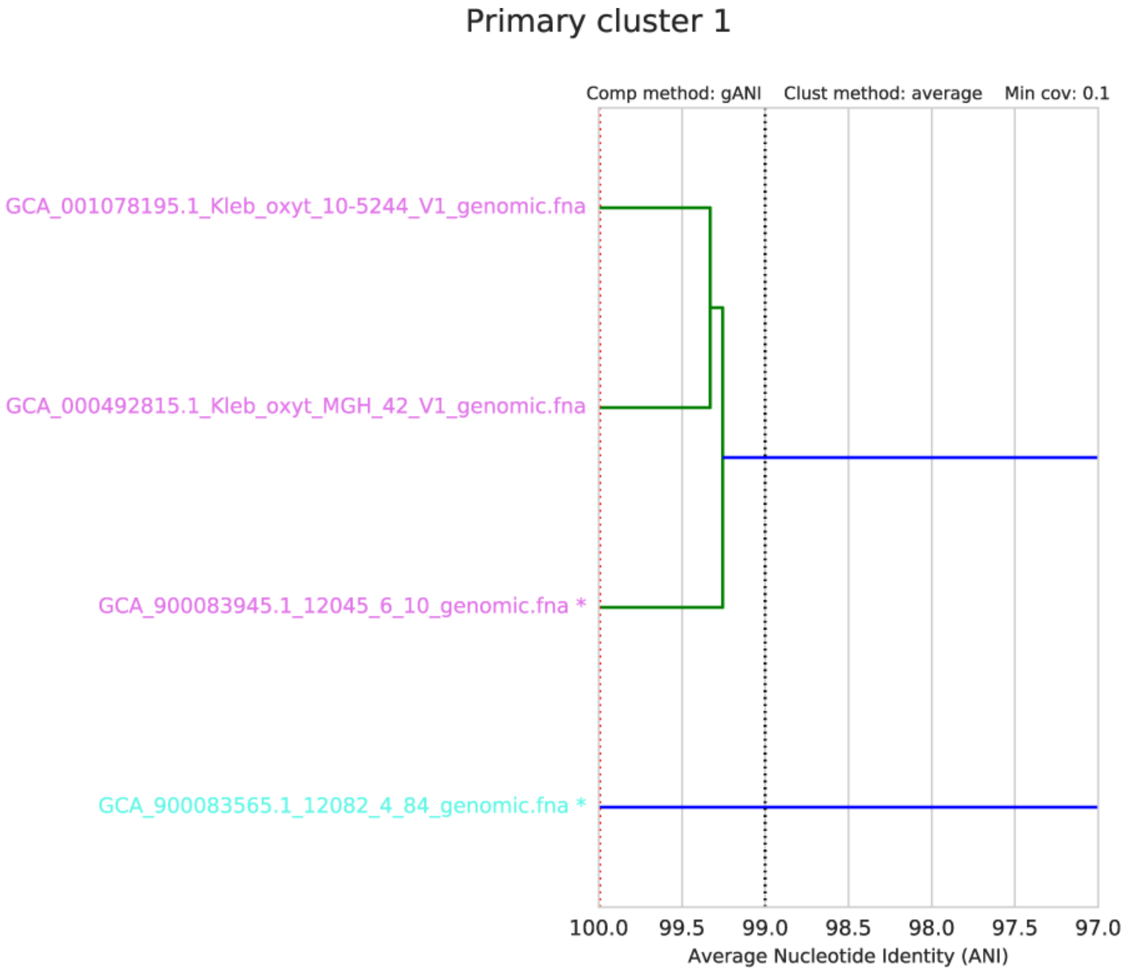


The primary clustering dendrogram summarizes the pair-wise Mash distance between all genomes in the genome list.

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

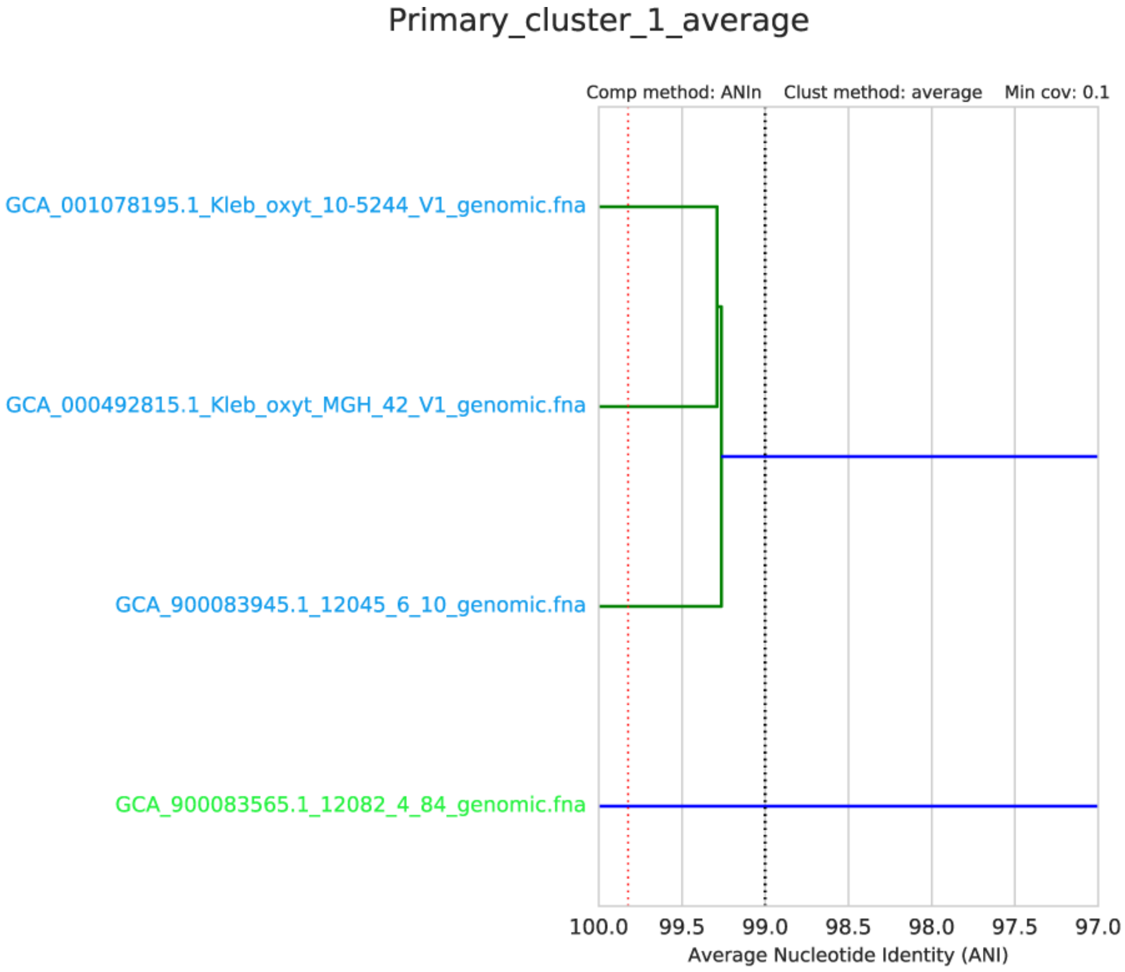


Each primary cluster with more than one member will have a page in the Secondary clustering dendrograms file. In this example, there is only one primary cluster with > 1 member.

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.

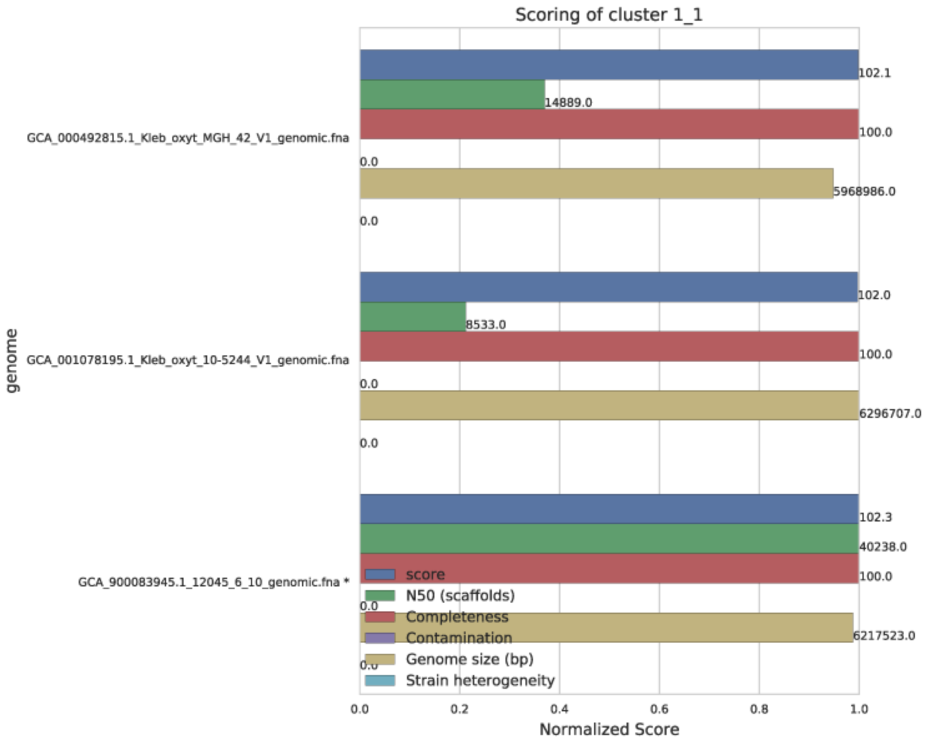
The red line shows the lowest ANI for a “self-vs-self” comparison of all genomes in the genome list. That is, when each genome in this primary cluster is compared to itself, the red line is the lowest ANI you get. This represents a “limit of detection” of sorts. gANI always results in 100% ANI when self-vs-self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

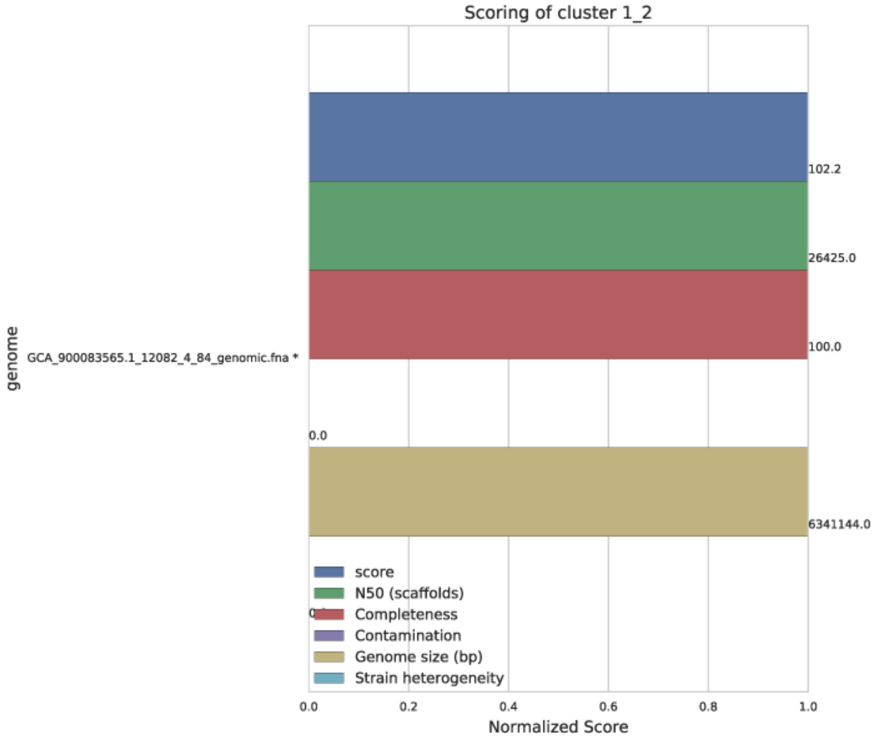


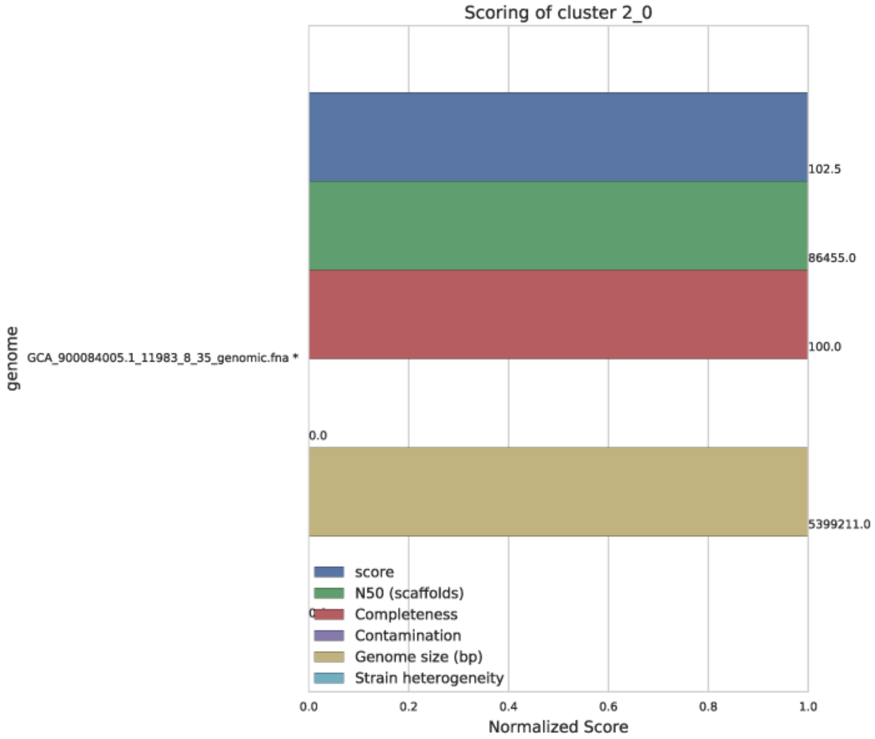
Note: The above figure was made with the command:

```
$ dRep analyze complete_only/ -c 1
```

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.

Using the `analyze` operation to test out alternative clusterings will be within this `figures` folder. See *Module Descriptions* for more information on that.

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.

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 hidden, see *Advanced Use*

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 comparison_wf 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:

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ dRep -h

                ...::: dRep v0.2.0 :::...

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

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

  Single opterations:
    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
    adjust          -> Adjust genome clusters
    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 realted to the above operations; test alternative␣
↪clustering
```

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)

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:** Due to a bug in checkM, 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:

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ 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)
```

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:

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ 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)
```

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:

$$score = A(completeness) + B(log_{10}(N50))\breve{}C(contamination)\breve{}D(strainheterogeneity) + E(log_{10}(genomesize))$$

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

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

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ dRep choose -h
usage: dRep choose [-p PROCESSORS] [-d] [-o] [-h] [-comW COMPLETENESS_WEIGHT]
                   [-conW CONTAMINATION_WEIGHT] [-N50W N50_WEIGHT]
                   [-sizeW SIZE_WEIGHT] [-strW STRAIN_HETEROGENEITY_WEIGHT]
                   [-c CLUSTER] [-t THRESHOLD] [-m {gANI,ANIn}]
                   [-mc MINIMUM_COVERAGE]
                   [-a {average,complete,weighted,single}]
                   [--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

SCORRING CHRITERIA
Based off of the formula: Completeness - Contamination + log(N50) + log(size):
  -comW COMPLETENESS_WEIGHT, --completeness_weight COMPLETENESS_WEIGHT
                        completeness weight (default: 1)
  -conW CONTAMINATION_WEIGHT, --contamination_weight CONTAMINATION_WEIGHT
                        contamination weight (default: 5)
  -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)
  -strW STRAIN_HETEROGENEITY_WEIGHT, --strain_heterogeneity_weight STRAIN_
↪HETEROGENEITY_WEIGHT
                        strain heterogeneity weight (default: 1)

RE-CLUSTER PRIMARY CLUSETERS:
  -c CLUSTER, --cluster CLUSTER
                        primary cluster to be adjusted (default: None)
  -t THRESHOLD, --threshold THRESHOLD
                        clustering threshold to apply (default: 0.99)
  -m {gANI,ANIn}, --clustering_method {gANI,ANIn}
                        Clustering method to apply (default: ANIn)
  -mc MINIMUM_COVERAGE, --minimum_coverage MINIMUM_COVERAGE
                        Minimum coverage for ANIn (default: 0.1)
  -a {average,complete,weighted,single}, --clusterAlg {average,complete,weighted,
↪single}
                        Algorithm used to cluster genomes (passed to
                        scipy.cluster.hierarchy.linkage) (default: average)

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

Analyze

Analyze is the module that makes all of the figures. It also has the option to visualize how a secondary cluster would look with different parameters (for example, using ANIm instead of gANI). To do that, use the `RE-CLUSTER PRIMARY CLUSTERS` arguments. To make plots, just use the -pl argument.

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

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ 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

RE-CLUSTER PRIMARY CLUSETERS:
  -c CLUSTER, --cluster CLUSTER
                        primary cluster to be adjusted (default: None)
  -t THRESHOLD, --threshold THRESHOLD
                        clustering threshold to apply (default: 0.99)
  -m {ANIn,gANI}, --clustering_method {ANIn,gANI}
                        Clustering method to apply (default: ANIn)
  -mc MINIMUM_COVERAGE, --minimum_coverage MINIMUM_COVERAGE
                        Minimum coverage for ANIn (default: 0.1)
  -a {complete,average,single,weighted}, --clusterAlg {complete,average,single,
 ↪weighted}
                        Algorithm used to cluster genomes (passed to
                        scipy.cluster.hierarchy.linkage) (default: average)

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)
```

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:

```
mattolm@Matts-MacBook-Pro:~/Programs/drep/docs$ 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)
```

Other

The other modules, **adjust** and **bonus**, are not part of the normal de-replication pipeline but can be very useful.

adjust allows the user to change the secondary clustering settings for a single primary cluster. This can be especially helpful when following up on a warning (generated using **evaluate**) to change the way the cluster is made.

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.

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

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.

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.

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)

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.

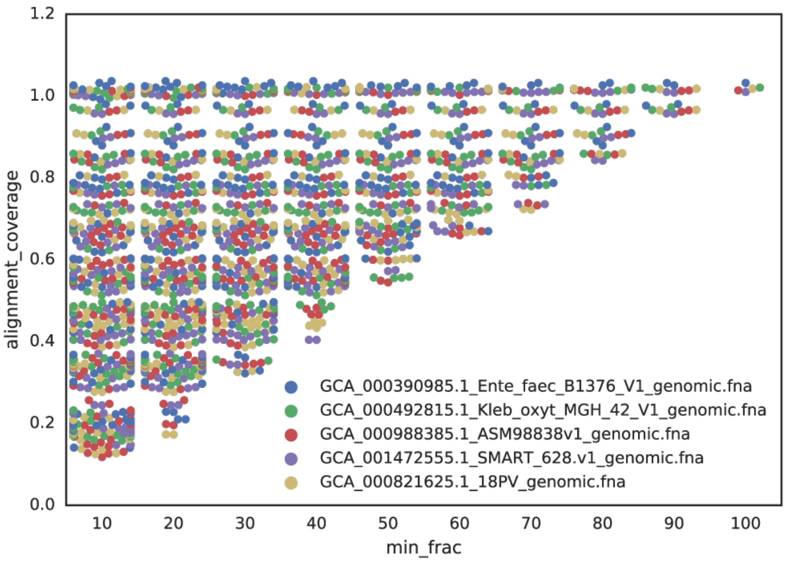


Fig. 2.1: **Figure A**: Five genomes are subset to fractions ranging from 10% - 100%, and fractions from the same genome are compared. The x-axis is the minimum genome completeness allowed. The looser this value is, the wider the range of aligned fractions.

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

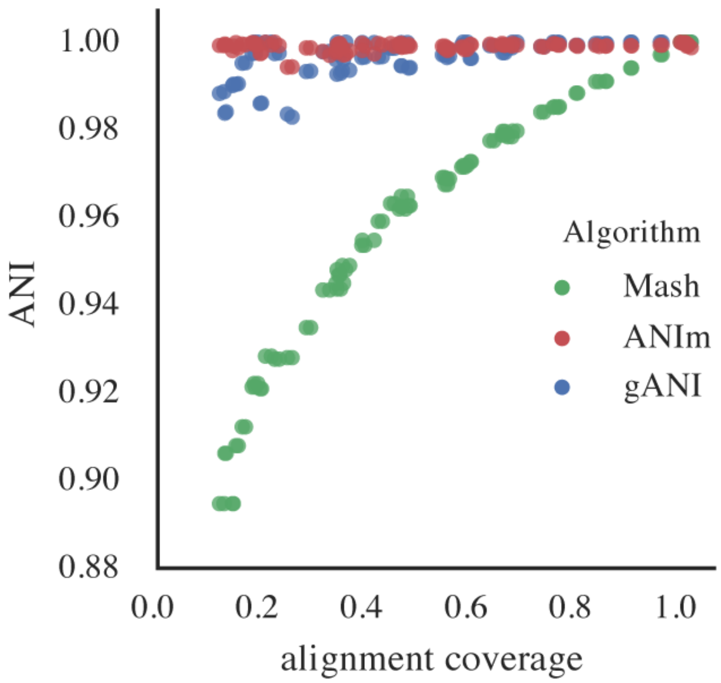


Fig. 2.2: **Figure B**: An identical *E. coli* genome is subset to fractions ranging from 10% - 100% and fractions are compared. When lower amounts of the genome align (due to incompleteness), Mash ANI is severely impacted

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

Still with me?

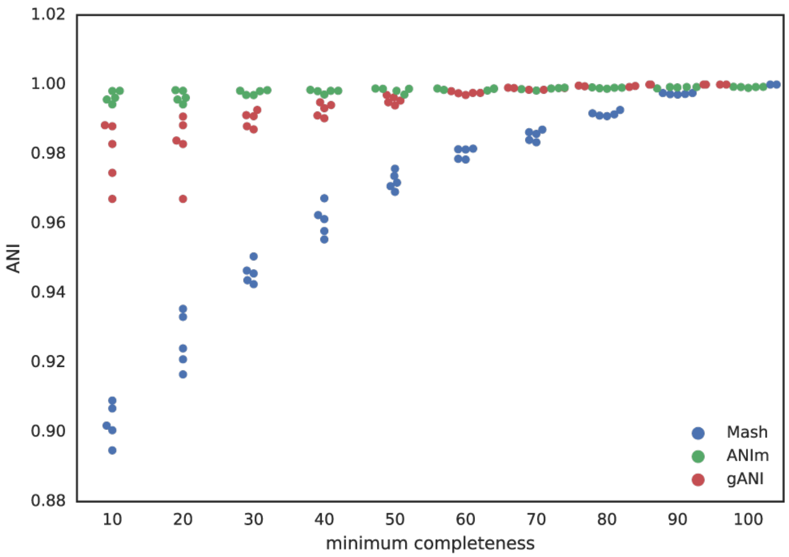
For example, say I set the minimum genome completeness to 50%. If I take an *E. coli* genome, subset it 50% 2 times, and compare those 2 subset genomes together, Mash will report an ANI of 96%. Therefore, the primary cluster threshold must be at least 96%, otherwise the two genomes could end up in different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

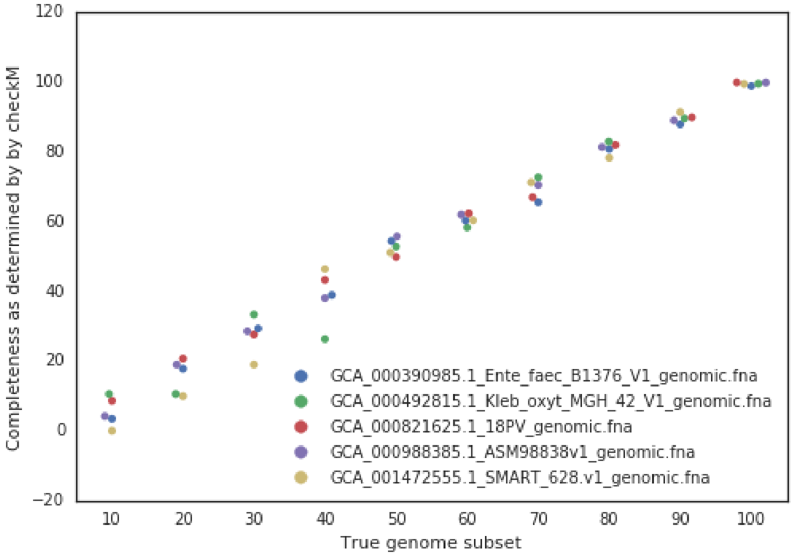
You don't want to set the primary cluster threshold super low, however, as this would result in more genomes being included in each primary cluster, and thus more secondary comparisons (which are slow), and thus a higher run-time.

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:


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

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

---

The Rest

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

Advanced Use

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.

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

API

This is no formal API for dRep, but of course you can read through the source-code and import any function that you'd like. The one class that is particularly helpful, however, is the WorkDirectory. By loading the results of your dRep run as a WorkDirectory object, you have direct access to all of the data-frames.

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