inStrain

Release 1.0.0

Aug 24, 2023

Contents

InStrain is a tool for analysis of co-occurring genome populations from metagenomes that allows highly accurate genome comparisons, analysis of coverage, microdiversity, and linkage, and sensitive SNP detection with gene localization and synonymous non-synonymous identification

Source code is available on GitHub.

Publication is available in Nature Biotechnology and on bioRxiv

See links to the left for *Installation* instructions

Bugs reports and feature requests can be submitted through GitHub.

InStrain was developed by Matt Olm and Alex Crits-Christoph in the Banfield Lab at the University of California, Berkeley.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Installation

InStrain is written in python. There are a number of ways that it can be installed.

pip

To install inStrain using the PyPi python repository, simply run

```
$ pip install instrain
```

That’s it!

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

bioconda

To inStrain inStrain from bioconda, run

```
$ conda install -c conda-forge -c bioconda -c defaults instrain
```

From source

To install inStrain from the source code, run

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

$ cd instrain

$ pip install .
```

Dependencies

inStrain requires a few other programs to run. Not all dependencies are needed for all operations. There are a number of python package dependencies, but those should install automatically when inStrain is installed using pip

Essential

- samtools This is needed for pysam

Optional

- coverM This is needed for the quick_profile operation
- Prodigal This is needed to profile on a gene by gene level

1.1.2 Pre-built genome database

An established set of public genomes can be downloaded for your inStrain analysis at the following link - https://doi.org/10.5281/zenodo.4441269. See Tutorial #2 in *Tutorial* for usage instructions.

1.1.3 Docker image

A Docker image with inStrain and dependencies already installed is available on Docker Hub at mattolm/instrain. This image also has a wrapper script in it to make it easier to use inStrain with AWS. See the docker folder of the GitHub page for use instructions.

1.2 Glossary & FAQ

1.2.1 Glossary of terms used in inStrain

Note: This glossary is meant to give a conceptual overview of the terms used in inStrain. See *Expected output* for explanations of specific output data.

ANI Average nucleotide identity. The average nucleotide distance between two genomes or .fasta files. If two genomes have a difference every 100 base-pairs, the ANI would be 99%

conANI Consensus ANI - average nucleotide identity values calculated based on consensus sequences. This is commonly reported as "ANI" in other programs. Each position on the genome is represented by the most common allele (also referred to as the consensus allele), and minor alleles are ignored.

popANI Population ANI - a new term to describe a unique type of ANI calculation performed by inStrain that considers both major and minor alleles. If two populations share any alleles at a loci, including minor alleles, it does not count as a difference when calculating popANI. It's easiest to describe with an example: consider a genomic position where the reference sequence is 'A' and 100 reads are mapped to the position. Of the 100 mapped reads, 60 have a 'C' and 40 have an 'A' at this position. In this example the reads share a minor allele with the reference genome at the position, but the consensus allele (most common allele) is different. Thus, this position **would** count as a difference in conANI calculations (because the consensus alleles are different) and **would not** count as a difference in popANI calculations (because the reference sequence is present as an allele in the reads). See *Important concepts* for examples.

Representative genomes (RGs) are genomes that are used to represent some taxa. For example you could have a series of representative genomes to represent each clade of E. coli (one genome for each clade), or you could have one representative genome for the entire species of E. coli (in that case it would be a Species Representative Genome (SRG)). The base unit of inStrain-based analysis is the representative genome, and they are usually generated using the program dRep

Species representative genome A Species Representative Genome (SRG) is a representative genome that is used to represent an entire single microbial species.

Genome database A collection of representative genomes that are mapped to simultaneously (competitive mapping).

nucleotide diversity A measurement of genetic diversity in a population (microdiversity). We measure nucleotide diversity using the method from Nei and Li 1979 (often referred to as 'pi' π in the population genetics world). InStrain calculates nucleotide diversity at every position along the genome, based on all reads, and averages values across genes / genomes. This metric is influenced by sequencing error, but within study error rates should be consistent and this effect is often minor compared to the extent of biological variation observed within samples. This metric is nice because it is not affected by coverage. The formula for calculating nucleotide diversity is the sum of the frequency of each base squared: 1 - [(frequency of A)^2 + (frequency of C)^2 + (frequency of G)^2 + (frequency of T)^2].

microdiversity We use the term microdiversity to refer to intraspecific genetic variation, i.e. the genetic variation between cells within a microbial species.

clonality The opposite of nucleotide diversity (1 - nucleotide diversity). A deprecated term used in older versions of the program.

SNV Single nucleotide variant. A single nucleotide change that is present in a faction of a population. Can also be described as a genomic loci with multiple alleles present. We identify and call SNVs using a simple model to distinguish them from errors, and more importantly in our experience, careful read mapping and filtering of paired reads to be assured that the variants (and the reads that contain them) are truly from the species being profiled, and not from another species in the metagenome (we call it 'mismapping' when this happens). Note that a SNV refers to genetic variation *within a read set*.

SNS Single nucleotide substitution. A single nucleotide change that has a fixed difference between two populations. If the reference genome has a 'A' at some position, but all of the reads have a 'C' at that position, that would be a SNS (if half of the reads have an 'A' and half of the reads have a 'C', that would be an SNV).

divergent site A position in the genome where either an SNV or SNS is present.

SNP Single Nucleotide Polymorphism. In our experience this term means different things to different people, so we have tried to avoid using it entirely (instead referring to SNSs, SNVs, and divert sites).

linkage A measure of how likely two divergent sites are to be inherited together. If two alleles are present on the same read, they are said to be "linked", meaning that they are found together on the same genome. Loci are said to be in "linkage disequilibrium" when the frequency of association of their different alleles is higher or lower than what would be expected if the loci were independent and associated randomly. In the context of microbial population genetics, linkage decay is often used as a way to detect recombination among members of a microbial population. InStrain uses the metrics r2 (r squared) and D' (D prime) to measure linkage.

coverage A measure of sequencing depth. We calculate coverage as the average number of reads mapping to a region. If half the bases in a scaffold have 5 reads on them, and the other half have 10 reads, the coverage of the scaffold will be 7.5

breadth A measure of how much of a region is covered by sequencing reads. Breadth is an important concept that is distinct from sequencing coverage, and gives you an approximation of how well the reference sequence you're using is represented by the reads. Calculated as the percentage of bases in a region that are covered by at least a single read. A breadth of 1 means that all bases in a region have at least one read covering them

expected breadth The breadth that would be expected if reads are evenly distributed along the genome, given a specific coverage value. Based on the function breadth = 1 - e ^{-0.883 * coverage}. This is useful to establish whether or not the scaffold is actually in the reads, or just a fraction of the scaffold. If your coverage is 10x, the expected breadth will be ~1. If your actual breadth is significantly lower then the expected breadth, this means that reads are mapping only to a specific region of your scaffold (transposon, prophage, etc.). See *Important concepts* for more info.

relative abundance The percentage of total reads that map a particular entity. If a metagenome has 1,000,000 reads and 1,000 reads to a particular genome, that genome is at 0.1% relative abundance

contig A contiguous sequence of DNA. Usually used as a reference sequence for mapping reads against. The terms contig and scaffold are used interchangeably by inStrain.

scaffold A sequence of DNA that may have a string of "N"s in it representing a gap of unknown length. The terms contig and scaffold are used interchangeably by inStrain.

iRep A measure of how fast a population was replicating at the time of DNA extraction. Based on comparing the sequencing coverage at the origin vs. terminus of replication, as described in Brown et. al., Nature Biotechnology 2016

mutation type Describes the impact of a nucleotide mutation on the amino acid sequence of the resulting protein. N = non-synonymous mutation (the encoded amino-acid changes due to the mutation). S = synonymous mutation (the encoded amino-acid does not change due to the mutation; should happen ~1/6 of the time by random chance due to codon redundancy). I = intergenic mutation. M = multi-allelic SNV with more than one change (rare).

dN/dS A measure of whether the set of mutations in a gene are biased towards synonymous (S) or non-synonymous (N) mutations. dN/dS is calculated based on mutations relative to the reference genome. dN/dS > 1 means the bias is towards N mutations, indicating the gene is under active selection to mutate. dN/dS < 1 means the bias is towards S mutations, indicated the gene is under stabilizing selection to not mutate. dN/dS = 1 means that N and S mutations are at the rate expected by mutating positions randomly, potentially indicating the gene is non-functional.

pN/pS Very similar to dN/dS, but calculated at positions with at least two alleles present rather than in relation to the reference genome.

fasta file A file containing a DNA sequence. Details on this file format can be found on wikipedia

bam file A file containing metagenomic reads mapped to a DNA sequence. Very similar to a *.sam* file. Details can be found online

scaffold-to-bin file A .text file with two columns separated by tabs, where the first column is the name of a scaffold and the second column is the name of the bin / genome the scaffold belongs to. Can be created using the script parse_stb.py that comes with the program `dRep` See *Expected output* for more info

genes file A file containing the nucleotide sequences of all genes to profile, as called by the program Prodigal. See *Expected output* for more info

mismapped read A read that is erroneously mapped to a genome. InStrain profiles a population by looking at the reads mapped to a genome. These reads are short, and sometimes reads that originated from one microbial population map to the representative genome of another (for example if they share homology). There are several techniques that can be used to reduce mismapping to the lowest extent possible.

multi-mapped read A read that maps equally well to multiple different locations in the .fasta file. Most mapping software will randomly select one position to place multi-mapped reads. There are several techniques that can be used to reduce multi-mapped reads to the lowest extent possible, including increasing the minimum MAPQ cutoff to >2 (which will eliminate them entirely).

inStrain profile An inStrain profile (aka IS_profile, IS, ISP) is created by running the `inStrain profile` command. It contains all of the program's internal workings, cached data, and is where the output is stored. Additional commands can then be run on an IS_profile, for example to analyze genes, compare profiles, etc., and there is lots of nice cached data stored in it that can be accessed using python.

null model The null model describes the probability that the number of true reads that support a variant base could be due to random mutation error, assuming Q30 score. The default false discovery rate with the null model is 1e-6 (one in a million).

mm The maximum number of mismatches a read-pair can have to be considered in the metric being considered. Behind the scenes, inStrain actually calculates pretty much all metrics for every read pair mismatch level. That is, only including read pairs with 0 mismatches to the reference sequences, only including read pairs with >= 1 mis-match to the reference sequences, all the way up to the number of mismatches associated with the "PID" parameter. Most of the time when it then generates user-facing output, it uses the highest mm possible and deletes the column label. If you'd like access to information on the mm-level, see the section titled "Dealing with mm"

mapQ score MapQ scores are a measure of how well a read aligns to a particular position. They are assigned to each read mapped by bowtie2, but the details of how they are generated are incredibly confusing (see the following link for more information). MapQ scores of 0 and 1 have a special meaning: if a read maps equally well to multiple different locations on a .fasta file, it always gets a MapQ score of 0 or 1.

1.2.2 FAQ (Frequently asked questions)

How does inStrain compare to other bioinformatics tools for strains analysis?

A major difference is inStrain's use of the popANI and conANI, which allow consideration of minor alleles when performing genomic comparisons. See *Important concepts* for more information.

What can inStrain do?

inStrain includes calculation of nucleotide diversity, calling SNPs (including non-synonymous and synonymous variants), reporting accurate coverage / breadth, and calculating linkage disequilibrium in the contexts of genomes, contigs, and individual genes.

inStrain also includes comparing the frequencies of fixed and segregating variants between sequenced populations with extremely high accuracy, out-performing other popular strain-resolved metagenomics programs.

The typical use-case is to generate a *.bam* file by mapping metagenomic reads to a bacterial genome that is present in the metagenomic sample, and using inStrain to characterize the microdiversity present.

Another common use-case is detailed strain comparisons that involve comparing the genetic diversity of two populations and calculating the extent to which they overlap. This allows for the calculation of population ANI values for extremely similar genomic populations (>99.999% average nucleotide identity).

See also:

Installation To get started using the program

Expected output To view example output

User Manual For information on how to prepare data for inStrain and run inStrain

Important concepts For detailed information on how to make sure inStrain is running correctly

How does inStrain work?

The reasoning behind inStrain is that every sequencing read is derived from a single DNA molecule (and thus a single cell) in the original population of a given microbial species. During assembly, the consensus of these reads are assembled into contigs and these contigs are binned into genomes - but by returning to assess the variation in the reads that assembled into the contigs, we can characterize the genetic diversity of the population that contributed to the contigs and genomes.

The basic steps:

1. Map reads to a *.fasta* file to create a *.bam* file
2. Stringently filter mapped reads and calculate coverage and breadth
3. Calculate nucleotide diversity and SNVs
4. Calculate SNV linkage
5. Optional: calculate gene statistics and SNV function
6. Optional: compare SNVs between samples.

What is unique about the way that inStrain compares strains?

Most strain-resolved pipelines compare the dominant allele at each position. If you have two closely related strains A and B in sample 1, with B being at higher abundance, and two closely related strains A and C in sample 2, with C being at higher abundance, most strain comparison pipelines will in actuality compare strain B and C. This is because they work on the principle of finding the dominant strain in each sample and then comparing the dominant strains. InStrain, on the other hand, is able to identify the fact that A is present in both samples. This is because it doesn't just compare the dominant alleles, but compares all alleles in the two populations. See module_descriptions and choosing_parameters for more information.

What is a population?

To characterize intra-population genetic diversity, it stands to reason that you first require an adequate definition of "population". InStrain relies mainly on population definitions that are largely technically limited, but also coincide conveniently with possibly biological real microbial population constraints (see Olm et. al. mSystems 2020 and Jain et. al. Nature Communications 2018). Often, we dereplicate genomes from an environment at average nucleotide identities (ANI) from 95% to 99%, depending on the heterogeneity expected within each sample - lower ANIs might be preferred with more complex samples. We then assign reads to each genome's population by stringently requiring that combined read pairs for SNP calling be properly mapped pairs with an similarity to the consensus of at least 95% by default, so that the cell that the read pair came from was at least 95% similar to the average consensus genotype at that position. Within an environment, inStrain makes it possible to adjust these parameters as needed and builds plots which can be used to estimate the best cutoffs for each project.

What are inStrain's computational requirements?

The two computational resources to consider when running inStrain are the number of processes given (`-p`) and the amount of RAM on the computer (usually not adjustable unless using cloud-based computing). Using inStrain v1.3.3, running inStrain on a .bam file of moderate size (1 Gbp of less) will generally take less than an hour with 6 cores, and use about 8Gb of RAM. InStrain is designed to handle large .bam files as well. Running a huge .bam file (30 Gbp) with 32 cores, for example, will take ~2 hours and use about 128Gb of RAM. The more processes you give inStrain the longer it will run, but also the more RAM it will use. See *Important concepts* for information on reducing compute requirements.

How can I infer the relative abundance of each strain cluster within the metagenomes?

At the moment you can only compare the relative abundance of the populations between samples. Say strain A, based on genome X, is in samples 1 and 2. You now know that genome X is the same strain in both samples, so you could compare the relative abundance of genome X in samples 1 and 2. But if multiple strains are present within genome X, there's no way to phase them out.

InStrain compare isn't really phasing out multiple strains in a sample, it's just seeing if there is micro-diversity overlap between samples. Conceptually inStrain operates on the idea of "strain clouds" more than distinct strains. InStrain isn't able to tell the number of strains that are shared between two samples either, just that there is population-level overlap for some particular genome. Doing haplotype phasing is something we've considered and may add in the future, but the feature won't be coming any time in the near future.

How can I determine the relative abundance of detected populations?

Relative abundance can be calculated a number of different ways, but the way I like to do it "percentage of reads". So if your sample has 100 reads, and 15 reads map to genome X, the relative abundance of genome X is 15%. Because inStrain does not know the number of reads per sample, it cannot calculate this metric for you. You have to calculate it yourself by dividing the total reads in the sample by the value filtered_read_pair_count reported in the inStrain *genome_wide* output.

What mapping software can be used to generate .bam files for inStrain?

Bowtie2 is a common one the works well, but any software that generates .bam files should work. Some mapping software modifies .fasta file headers during mapping (including the tool BBMap and SNAP). Include the flag `--use_full_fasta_header` when mapping with these programs to properly handle this.

1.3 Important concepts

There are a number of things to be aware of when performing metagenomic analysis with inStrain. This page will address the following key concepts:

1. An overview of inStrain and the data it generates. A brief introduction to microbial population genomics.

2. Representative genomes and their utility. InStrain runs on "representative genomes"; this section describes what they are and the benefit of using them.

3. Picking and evaluating representative genomes. Some things to think about when picking and mapping to representative genomes.

4. Establishing and evaluating genome databases. Some things to think about when dereplicating genomes to create a genome database.

5. Handling and reducing mis-mapping reads. Ways to ensure that sequencing reads align to the correct genomes.

6. Detecting organisms in metagenomic data. Determining whether an organism is "present" in a sample is more complicated than meets the eye.

7. Strain-level comparisons and popANI. A description of how inStrain performs detailed strain-level comparisons with the popANI metric.

8. Thresholds for determining "same" vs. "different" strains. How similar do strains need to be for them to be considered identical?

9. Importance of representative genomes when calculating popANI. Appropriate representative genomes are need for popANI to work correctly.

10. Using inStrain for gene-based functional analysis. Some ways to tie inStrain results to gene-based functional questions.

11. Reducing inStrain resource usage. Tips to reduce inStrain run-time and RAM usage.

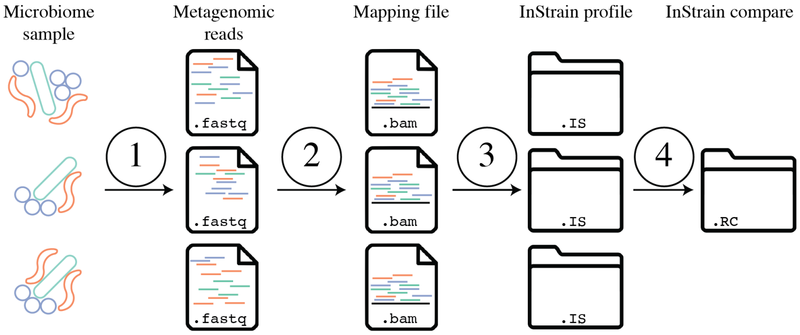
1.3.1 1. An overview of inStrain and the data it generates

InStrain is a program for microbial metagenomic analysis. When you sequence any microbial genome(s), you sequence a population of cells. This population may be a nearly clonal population grown up from an isolate in a culture flask, or a highly heterogeneous population in the real world, but there is always real biological genetic heterogeneity within that population. Every cell does not have the same genotype at every single position. **InStrain can determine organism presence / absence in a community, measure and interrogate the genetic heterogeneity in microbial population, and perform detailed comparisons between organisms in different samples.**

A community is a collection of taxa in a metagenome. After mapping your metagenomic reads to a set of representative genomes, inStrain can generate a number of metrics that help understand community composition. These include the percentage of reads that map to your representative genome database, the abundance of each microbe in the community, and a detailed picture of the organisms that are present or absent (measured using *breadth* of coverage, expected breadth of coverage, and coverage s.e.m).

A population is the collection of cells that make up an individual taxa in a community. After mapping your metagenomic reads to a set of representative genomes, inStrain can generate a number of metrics characterizing the **population-level diversity** of each detected organism. These metrics include *nucleotide diversity*, *SNSs* and *SNVs*, *linkage*, *pN/pS*, *iRep*, and others. Most metrics are calculated on the gene level, scaffold level, and genome level.

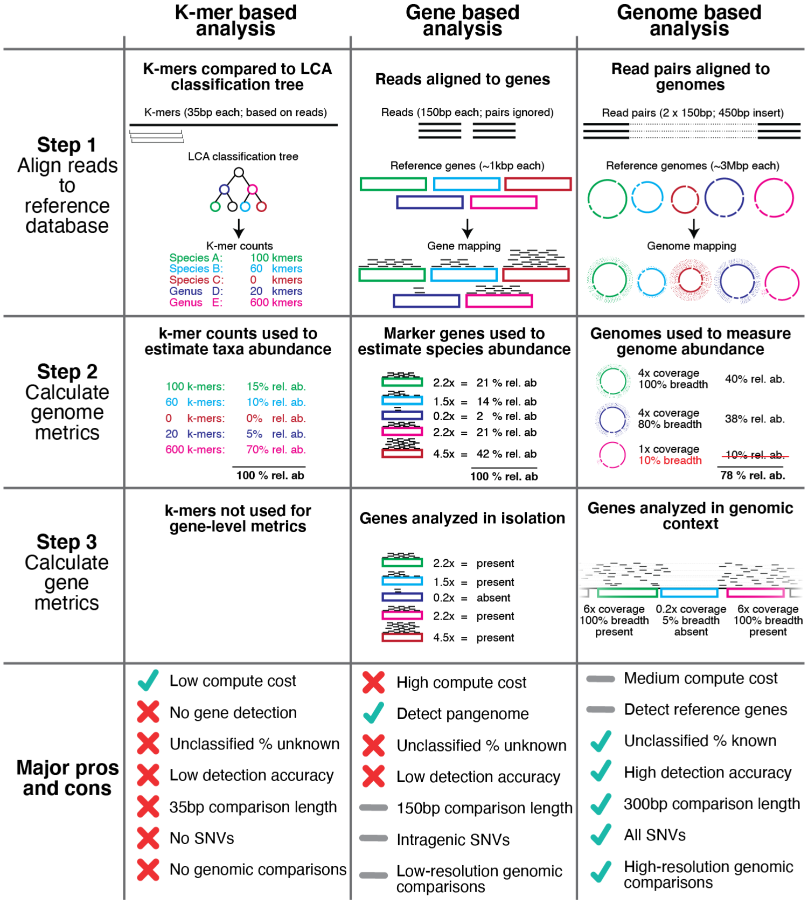
Strain-level comparisons between populations in different communities are notoriously different to perform with high accuracy. After profiling the communities of metagenomic samples, inStrain can compare the populations in the different communities in a highly-accurate manner by taking into account the population-level diversity. This analysis reports comparison metrics including the percentage of the genome covered in each sample, *popANI*, conNI, and the locations of all differences between strains.



The above figure provides a conceptual overview of the steps involved when running inStrain.

1.3.2 2. Representative genomes and their utility

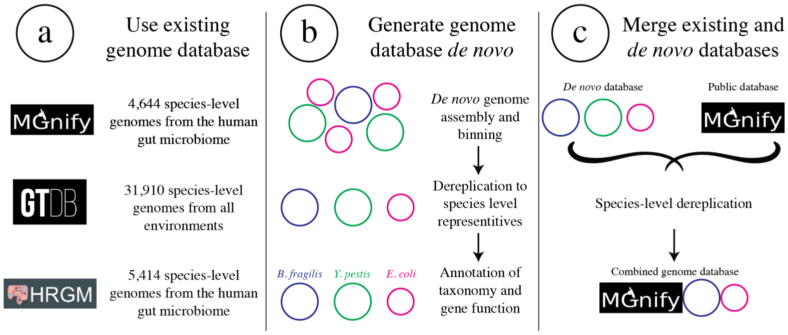
Representative genomes are genomes that are chosen to represent some group of taxa, and they are the base unit of inStrain-based metagenomic analyses. If one wanted to study the species-level composition of a community with inStrain they would use a set of *Species representative genomes* (SRGs), but Representative genomes can also be used at more specific taxonomic levels. They are similar to OTUs in 16S-based analysis. There are some things to be aware of when using Representative genomes, including ensuring that they truly represent the taxa they are meant to, but using them has several advantages over other common approaches.



	K-mer based analysis	Gene based analysis	Genome based analysis
Step 1 Align reads to reference database	**K-mers compared to LCA classification tree** K-mers (35bp each; based on reads) LCA classification tree K-mer counts Species A: 100 kmers Species B: 60 kmers Species C: 0 kmers Genus D: 20 kmers Genus E: 600 kmers	**Reads aligned to genes** Reads (150bp each; pairs ignored) Reference genes (~1kbp each) Gene mapping	**Read pairs aligned to genomes** Read pairs (2 x 150bp; 450bp insert) Reference genomes (~3Mbp each) Genome mapping
Step 2 Calculate genome metrics	**k-mer counts used to estimate taxa abundance** 100 k-mers: 15% rel. ab. 60 k-mers: 10% rel. ab. 0 k-mers: 0% rel. ab. 20 k-mers: 5% rel. ab. 600 k-mers: 70% rel. ab. 100 % rel. ab	**Marker genes used to estimate species abundance** 2.2x = 21 % rel. ab 1.5x = 14 % rel. ab 0.2x = 2 % rel. ab 2.2x = 21 % rel. ab 4.5x = 42 % rel. ab 100 % rel. ab	**Genomes used to measure genome abundance** 4x coverage 100% breadth 40% rel. ab. 4x coverage 80% breadth 38% rel. ab. 1x coverage 10% breadth ~~10% rel. ab.~~ 78 % rel. ab.
Step 3 Calculate gene metrics	**k-mers not used for gene-level metrics**	**Genes analyzed in isolation** 2.2x = present 1.5x = present 0.2x = absent 2.2x = present 4.5x = present	**Genes analyzed in genomic context** 6x coverage 100% breadth present 0.2x coverage 5% breadth absent 6x coverage 100% breadth present
Major pros and cons	✓ Low compute cost ✗ No gene detection ✗ Unclassified % unknown ✗ Low detection accuracy ✗ 35bp comparison length ✗ No SNVs ✗ No genomic comparisons	✗ High compute cost ✓ Detect pangenome ✗ Unclassified % unknown ✗ Low detection accuracy — 150bp comparison length — Intragenic SNVs — Low-resolution genomic comparisons	— Medium compute cost — Detect reference genes ✓ Unclassified % known ✓ High detection accuracy ✓ 300bp comparison length ✓ All SNVs ✓ High-resolution genomic comparisons

The above figure shows a visual representation of k-mer based metagenomic analysis, gene-based metagenomic analysis, and Representative genome based metagenomic analysis. Advantages include the ability to align full read pairs to target sequences, use the entire genome to determine presence and absence (significantly improving detection accuracy; see *Benchmarks* for proof), and perform high-resolution comparisons, among other things.

A collection of representative genomes is referred to as a *Genome database*. *Genome databases* can be downloaded from public repositories, generated via de novo sequence assembly and binning, or a combination of the two. It is important to ensure that each genome in the *Genome database* is distinct enough from other genomes in the database to avoid mapping confusion, and by mapping to all genomes in a *Genome database* simultaneously (competitively) one can significantly reduce the number of mis-mapped reads overall.


a
Use existing genome database
MGnify
4,644 species-level genomes from the human gut microbiome
GTDB
31,910 species-level genomes from all environments
HRGM
5,414 species-level genomes from the human gut microbiome
b
Generate genome database *de novo*
De novo genome assembly and binning
Dereplication to species level representitives
B. fragilis
Y. pestis
E. coli
Annotation of taxonomy and gene function
c
Merge existing and *de novo* databases
De novo database
Public database
MGnify
Species-level dereplication
Combined genome database
MGnify

The figure above provides a visual overview of options for generating *Genome databases* for use with inStrain. For technical details on how this is done, see *User Manual*. For a pre-generated Genome database for immediate download, see *Tutorial*.

1.3.3 3. Picking and evaluating representative genomes

Representative genomes are typically chosen by first clustering a set of genomes using some *ANI* threshold, and second picking a single genome to represent each cluster. Choosing *ANI* thresholds are discussed in the section below. A good Representative genome is high quality, contiguous, shares a high degree of gene content with the taxa it is meant to represent, and has a similar *ANI* to all genomes it's meant to represent. The program dRep is commonly used to pick representative genomes, and it uses a scoring system to score each genome and pick the genome with the highest score.

Running `inStrain profile` will generate a plethora of information about each Representative genome detected in your sample (see *Expected output*). This information can be used to determine how good of a fit each representative genome is to the true population that it is recruiting reads from. Helpful metrics are mean read ANI, reference conANI, reference popANI, and breadth vs. expected breath. If there are regions of the genome with much higher coverage than the rest, it is likely that that region is recruiting reads from another population (*mismapped read*). Looking at these wavy coverage patterns can be confusing, however. Here is a link for more information on this phenomenon.

One way of increasing the similarity between a Representative genome and the organisms in your sample is to assemble genomes from your sample directly. Something to keep in mind is that when multiple closely related genomes are present in a sample, the assembly algorithm can break and you can fail to recover genomes from either organism. A solution to this problem is to assemble and bin genomes from all metagenomic samples individually, and dereplicate the genome set at the end. For more information on this, see the publication "dRep: a tool for fast and accurate genomic comparisons that enables improved genome recovery from metagenomes through de-replication"

1.3.4 4. Establishing and evaluating genome databases

Genome databases are typically created by clustering a set of genomes using some *ANI* threshold using the program dRep. The dRep documentation describes some considerations to think about when choosing an ANI threshold. The most common thresholds are 95% *ANI*, which represents species-level clustering (Olm mSystems 2020), and 98%

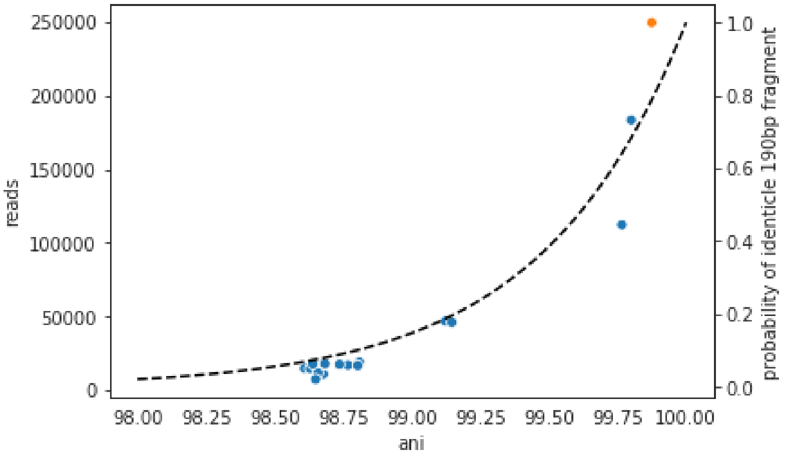
ANI, which is about the most stringent threshold recommended. **Using either of these thresholds is generally a safe bet**, and which threshold you choose depends on the level of resolution you'd like to perform your analysis at. These thresholds are ensure that genomes are distinct enough from each other, but not too distinct. Details on why this is important are below.

a) Ensure that genomes are distinct from one another.

Note: When genomes share stretches of identical sequence, read mapping software cannot reliably determine which genome a read should map to. The exact level of how distinct genomes need to be depends on the read length and the heterogeneity of differences across the genome, but **having a maximum of 98% ANI between all genomes in the genome database is a good rule of thumb.**

When mapping to a *Genome database*, if bowtie2 finds a read that maps equally well to multiple different positions in your *fasta file* it will randomly choose one of the two positions to place the read at. This is the best thing it could do, as you don't want reads "duplicated" and mapped to multiple positions, but it also means that you really don't want to have multiple positions in your .fasta file that are identical. The reason we go through the hassle of dereplication to generate a *Genome database* is to limit the number of positions in which the alignment algorithm cannot tell where the read should actually map to, and this is why we can't just map to all possible genomes.

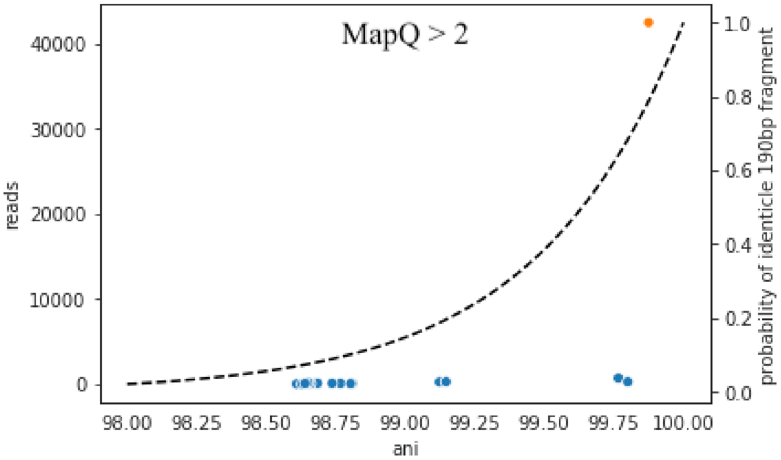
To determine how distinct genomes need to be to avoid having identical regions, we performed a simple experiment. We mapped to a randomly selected genome in isolation, and in the context of many other genomes in a *Genome database* dereplicated at 99.8% ANI. We then looked for reads that mapped to the genome of interest when mapping to that genome individually, but mapped elsewhere when mapping in the context of the entire *Genome database*. The results from this experiment are displayed below.



Each dot represents a genome in the full *Genome database*, the position on the x-axis indicates that genome's ANI to the genome of interest (orange dot), and the position on the y-axis indicates the number of reads that were "stolen" from the genome of interest (stolen reads are those that mapped to the genome of interest when mapped in isolation, but mapped to a different genome when mapped in the context of the entire *Genome database*). As you can see, the more closely related an alternate genome is to a genome of interest, the more likely it is to steal reads. This makes sense, because assuming that the genomes represented by blue dots are not actually present in the sample (likely true in this case), the only way these genomes have reads mapped to them is by having regions that are identical to the genome that is actually present in the sample. In fact, you can even calculate the probability of having an identical region as long as a pair of reads (190bp in this case; 2 x 95bp) based on the genome ANI using the formula:

$$\text{Probability of 190bp fragment} = (\text{genome ANI})^{190}$$

This simple formula was used to generate the black dotted line in the figure above. The line fits observed trend remarkably well, providing pretty compelling evidence that simple genome-ANI-based read stealing explains the phenomena. To be sure though, we can did final check based on *mapQ score*. Reads that map equally well to multiple different locations in a *fasta file* always get a MapQ score of 0-2. Thus, by filtering out reads with MapQ scores < 2, we can see reads that map uniquely to one genome only. Below we will re-generate the above figure while only including reads with *mapQ scores* above 2.



Just as we suspected, reads no longer map to alternate genomes at all. This provides near conclusive evidence that the organisms with these genomes are not truly in the sample, but are merely stealing reads from the genome of the organism that is there by having regions of identical DNA. For this reason it can be smart to set a minimum MapQ score of 2 to avoid mis-mapping, but at the same time, look at the difference in the number of reads mapping to the correct genome when the MapQ filter is used (compare the y-axis in the first and second figure)- 85% of the reads are filtered out. Using MapQ filters is a matter of debate depending on your specific use-case.

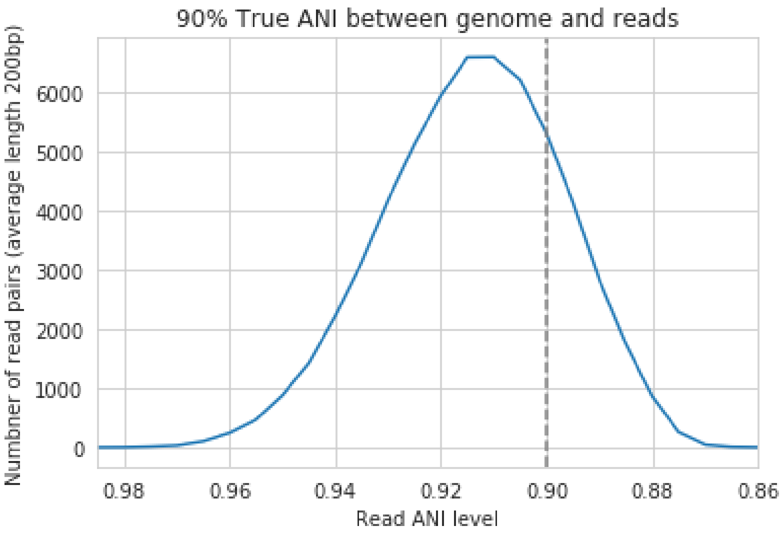
The data above can also be used to evaluate the most stringent threshold that can be used for dereplication. With 190bp reads (used in the figure above), we can see that read stealing approaches 0 at ~98% ANI. We can also plug this into the formula above to see that there is a ~2% change of genomes that are 98% ANI from each other sharing a 190bp identical stretch of DNA (`0.98 ^ 190 = 0.02`). This is how we arrived at our recommended minimum of 98% ANI. However it is important to note that longer reads change the formula and differences between genomes are not uniformly spread across the genome. This is a complicated question and 98% ANI is just a good rule of thumb.

A symptom of having a *Genome database* in which genomes are too similar to one another is detecting lots of closely related organisms at similar abundance levels in samples.

b) Ensure that genomes aren't too distinct from one another.

Note: When representative genomes are too distinct from the sample population they can have trouble with read mapping. The exact level of how similar genomes need to be depends on a number of factors, but **a having a minimum of 95% ANI between all genomes in the genome database (representing species-level dereplication) is a good rule of thumb.**

Genomes need to be similar enough to the population being mapped that they can properly recruit reads. If one were to generate a *Genome database* using an ANI threshold of 85% ANI, for example, implicit in that choice is the requirement that organisms which share 85% ANI to a representative genome will have their reads mapped to that genome. This begs the question- how similar do reads have to be to a genome for bowtie2 to map them? The answer is "it's complicated":



In the above example we generated synthetic reads that have a mean of 90% ANI to the reference genome. We then mapped these reads back to the reference genome and measured the ANI of mapped reads. Critically, the density of read ANI is not centered around 90% ANI, as it would be if all reads mapped equally well. The peak is instead centered at ~91% ANI, with a longer tail going left than right. This means that reads which have <92% ANI to the reference genome sometimes don't map at all. Sometimes they do map, however, as we see some read pairs mapping that have ~88% ANI. The reason for this pattern is because **bowtie2 doesn't have a stringent ANI cutoff, it just maps whatever read-pairs it can**. Where the SNPs are along the read, whether they're in the seed sequence that bowtie2 uses, and other random things probably determine whether a low-ANI read pair maps or not. Thus, while bowtie2 can map reads that are up to 86% ANI with the reference genome, 92% seems to be a reasonable minimum based on this graph.

However, this does not mean that a representative genome that has 92% ANI to an organism of interest will properly recruit all it's reads. ANI is calculated as a genome-wide average, and some regions will have more mutations than others. This is why the figure above has a wide distribution. Further, genomes that share 92% ANI have diverged from each other for a very long time, and likely have undergone changes in gene content as well. Recent studies have shown that organisms of the same species usually share >= 95% ANI, and that organisms of the same species share much more gene content than organisms from different species (Olm mSystems 2020). In sections below we also show that a buffer of ~3% ANI is needed to account for genomic difference heterogeneity, meaning that genomes dereplciated at 95% should be able to recruit reads at 92% ANI (the minimum for bowtie2). **Thus for a number of reasons 95% ANI is a good minimum ANI threshold for establishing genome databases.**

A symptom of having a *Genome database* in which genomes are too distinct from one another is genomes having low mean read ANI and *breadth*, and having an overall low percentage of reads mapping.

c) Ensure that all microbes in a sample have an appropriate representative genome.

Populations with appropriate representative genomes will be most accurately profiled, and populations that do not have a representative genome in the genome database will be invisible. **Using a combination of de novo assembly and integration with public databases can result in genome databases that are both accurate and comprehensive.** Instructions for how to do this are available in the *Tutorial* and *User Manual*. A great way to determine how complete your *Genome database* is is to calculate the percentage of reads that map to genomes in your database. The higher this percentage, the better (expect ~20-40% for soil, 60-80% for human microbiome, and 90%+ for simple, well defined communities).

1.3.5 5. Handling and reducing mis-mapping reads

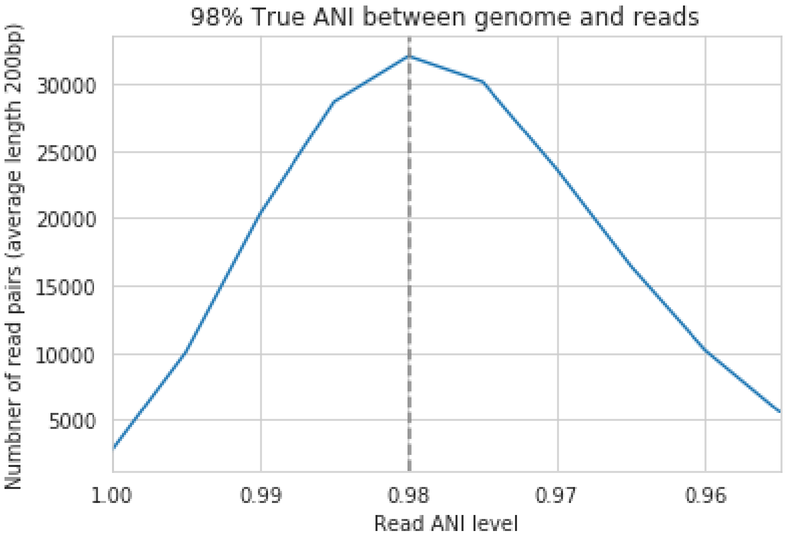
As discussed above, a major aspect of using and establishing *Genome databases* with inStrain is reducing the number of reads that map to the wrong genome. When metagenomic sequencing is performed on a community, reads are generated from each population in that community. The goal of read mapping is to assign each read to the genome representing the population from which the read originated. When a read maps to a genome that does not represent the population from which the read originated, it is a mis-mapped read. Read mis-mapping can happen when a read maps equally well to multiple genomes (and is then randomly assigned to one or the other) or when a read from a distantly-related population maps to an inappropriate genome. Read mis-mapping can be reduced using a number of different techniques as discussed below.

Reducing read mis-mapping with competitive mapping

Competitive mapping is when reads are mapped to multiple genomes simultaneously. When we establish and map to a *Genome database* we are performing competitive mapping. When bowtie2 maps reads, by default, it only maps reads to a single location. That means that if a read maps at 98% ANI to one genome, and 99% ANI to another genome, it will place the read at the position with 99% ANI. If the read only maps to one scaffold at 98% ANI, however, bowtie2 will place the read there. Thus, by including more reference genome sequences when performing the mapping, reads will end up mapping more accurately overall. Ensuring that you have the most comprehensive genome set possible is a great way to reduce read mis-mapping via competitive mapping.

Reducing read mis-mapping by adjusting min_read_ani

InStrain calculates the ANI between all read-pairs and the genomes they map to. The inStrain profile parameter `-l` / `--min_read_ani` dictates the minimum ANI a read pair can have; all pairs below this threshold are discarded. Adjusting this parameter can ensure that distantly related reads don't map, but setting this parameter to be too stringent will reduce the ability of a genome to recruit reads with genuine variation.



For the figure above synthetic read pairs were generated to be 98% ANI to a random E. coli genome, reads were mapped back to that genome, and the distribution of ANI values of mapped reads was plotted. Most read pairs have 98%, as expected, but there is a wide distribution of read ANI values. This is because differences between reads and genomes are not evenly spread along the genome, a fact that is even more true when you consider that real genomes likely have even more heterogeneity in where SNPs occur than this synthetic example. You really don't want reads to fail to map to heterogeneous areas of the genome, because those areas with more SNPs are potentially the most interesting. Based on the figure above and some other confusing tests that aren't included in this documentation, it seems that **the minimum read pair ANI should be 2-3% lower than the actual difference between the reads and the genome to account for genomic heterogeneity.** Thus a `--min_read_ani` of 92% should be used when reads are expected to map to genomes that are 95% ANI away, for example when using *Species representative genomes*.

Warning: The inStrain default is 95% minimum read pair ANI, which is ideal in the case that you've assembled your reference genome from the sample itself. If you plan on using inStrain to map reads to a *Genome database* of Species representative genome's, you should lower the minimum read-pair ANI to ~92% (note that using the '–database_mode'' flag automatically adjusts `--min_read_ani` to 0.92)

Reducing read mis-mapping by adjusting MapQ

mapQ score's are numbers that describe how well a read maps to a genome. InStrain is able to set a minimum read-pair mapQ score using the parameter '–min_mapq''. MapQ scores in general are confusing, without consistent rules on how they're calculated using different mapping programs, but the values 0-2 have special meaning. **If a read maps equally well to multiple positions it is given a mapQ score of 1 or 2.** Thus by setting `--min_mapq` to 2, you can remove all reads that map equally well to multiple positions (*multi-mapped read*). Remember that with competitive mapping a read that maps equally well to multiple positions will be randomly assigned to one, giving that read a 50% chance of being mis-mapped.

Whether or not you should set `--min_mapq` to 2 is a difficult decision. On one hand these reads have a high probability of being mis-mapped, which is not ideal, but on the other hand doing this mapQ filtering can result in filtering out lots of reads (see figures in the above section "Establishing and evaluating genome databases"). One way of thinking about this is by imagining two genomes A and B that are very distinct from one another but share an identical transposon. If the population represented by genome A and not genome B is present in a sample, without mapQ filtering you'll see genome A having a *breadth* of 100% and genome B having a *breadth* of ~1%. If genome A is at 100X coverage you'll see the coverage across most of the genome at 100x, and at the transposon it will be at 50x. Genome B will have 0x coverage across most of the genome, and the transposon will be at 50x coverage. The benefit of this scenario is that we are still able detect that genome A has the transposon; the downside is that it that genome B is erroneously detected has having a transposon present in the sample (however when using recommended threshold of 50% *breadth* to determine detection genome B will still correctly be identified as not being present in the sample). Performing mapQ filtering on the above situation will result in genome A having a breadth of 99%, 0x coverage at the transposon, and no reads mapping to genome B. The benefit of this scenario is that we properly detect that no reads are mapping to genome B; the downside is that we incorrectly think that genome A does not have a transposon in this sample.

Note: In conclusion, filtering reads by *mapQ score* is not ideal for a number of reasons. It is best to instead reduce the number of multi-mapped reads using the advice in the sections above to make it so `--min_mapq` filtering isn't necessary.

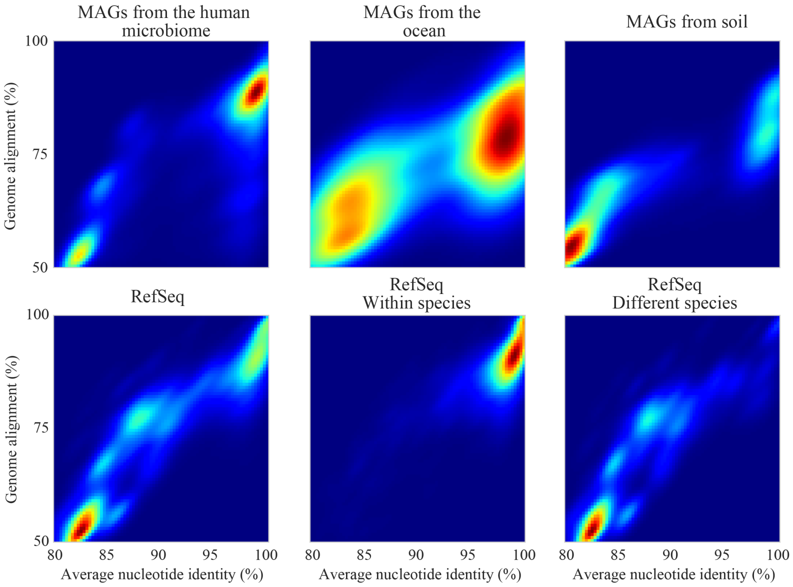
1.3.6 6. Detecting organisms in metagenomic data.

Note: Mis-mapping can fool abundance-based presence/absence thresholds. We recommend using a 50% *breadth* threshold to determine presence/absence instead.

A critical first step in metagenomic analysis is determining which Representative genomes are "present" or "absent" (and therefore the microbial populations they represent as well). This is actually more complex than meets the eye, mostly due to *multi-mapped reads* and *mismapped reads*. Details on these phenomena are discussed above, but the upshot is that **just because a genome has reads mapping to it does not mean that that genome is actually present in a sample.**

Many studies determine presence/absence based on metrics like *coverage* or relative abundance. This isn't great though, since there can easily be substantial numbers of mis-mapped reads. There are countless examples of a genome being detected at 100x *coverage* and 2% *relative abundance*, but when looking at the mapping it is discovered that all reads are mapped to a single prophage on the genome. The problem with these metrics is that they are genome-wide averages, so they cannot account for cases where substantial numbers of reads are map to a small region of the genome. Most would agree that detecting solely a prophage or transposon on a genome should not count as that genome being "present", so we need metrics beyond *coverage* and 2% *relative abundance* to determine presence / absence. See *Benchmarks* for more real-world examples of this phenomena.

A great metric for determining presence/absence is *breadth*, the percentage of a genome that's covered by at least one read. Using *breadth* to determine presence/absence allows the user to account for the problems above. Deciding on an appropriate breadth threshold requires the user to answer the question "How much of the genome do I need to have detected in a sample before I am confident that it's actually present"? The answer to this question depends on the particular study details and questions, but we can use data to help us decide on a rational breadth cutoff.



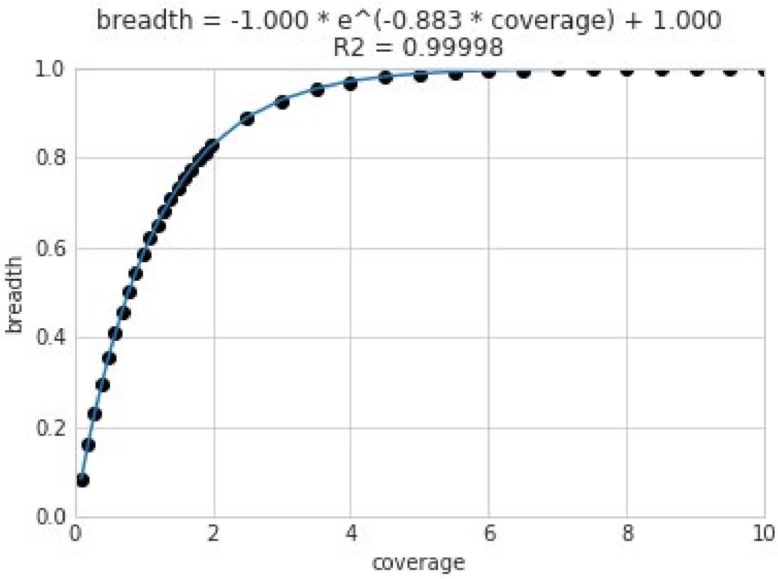
The figure above shows the expected genome overlap between genomes of various ANI values from different environments (adapted from "Consistent metagenome-derived metrics verify and define bacterial species boundaries"). As you can see, genomes that share >95% ANI tend to share ~75% of their genome content. Therefore, using a breadth detection cutoff of somewhere around 50-75% seems to be reasonable when using *Species representative genome* s. **In my experience using a 50% breadth cutoff does a great job of ensuring that genomes are actually present when you say they are, and leads to very few false positives.** It's exceedingly rare for mis-mapping to lead to >50% genome breadth. See *Benchmarks* for real-world examples of the 50% breadth threshold in action.

A caveat of using a breadth threshold is that it requires thousands of reads to map to a genome for it to be considered present. This makes it less ideal for samples with low sequencing depth. To determine the *coverage* needed to detect a genome at some *breadth*, we performed an experiment based on synthetic E. coli and C. albicans reads). By generating reads, subsetting them to a number of different total read numbers, and mapping them back to the genome, we generated the following figure

This figure allows us to visually see the relationship between coverage and breadth **when reads are mapped randomly across the genome**. To achieve a 50% breadth an organism needs to have just under 1x coverage. At over 6x coverage, all organisms should have ~100% breadth. This data also allowed us to fit a curve to calculate the following formula:

$$breadth = 1 - e^{-0.883*coverage}$$

Applying this formula allows inStrain to calculate and report *expected breadth* for a given coverage value. **Effective use of expected breadth can allow users to lower their breadth thresholds and still have confidence in determining presence/absence**. Imagine that you detect an organism at 10x coverage and 85% breadth. The *expected breadth* at 10x coverage is 100%, but you only have 85% breadth. This means that 15% of your genome is likely not in the reads set, and that your representative genome has genome content that is 15% different from the organism in your



sample. Now imagine that you detect an organism at 3x coverage with 85% breadth. The *expected breadth* and actual breadth are approximately the same now, meaning that reads and randomly aligning to all parts of the genome and you likely have a very dialed in representative genome. Now imagine you detect organism A with 10% breadth and 0.1x coverage, and organism B with 10% breadth and 10x coverage. Both organisms have the same breadth, but organism A is much more likely to be actually present in your sample. That's because while few reads overall are mapping, they're mapping all across the genome in a random way (you know this because breadth is about equal to expected breadth), which is indicative of a true low abundance population. Organism B, however, should be abundant enough for reads to map all over the genome (expected breadth is 100%), but reads are only mapping to 10% of it. This indicates that no matter how deeply you sequence you will not see the rest of organism B's genome, and the 10% of it that you are seeing is likely due to mis-mapping.

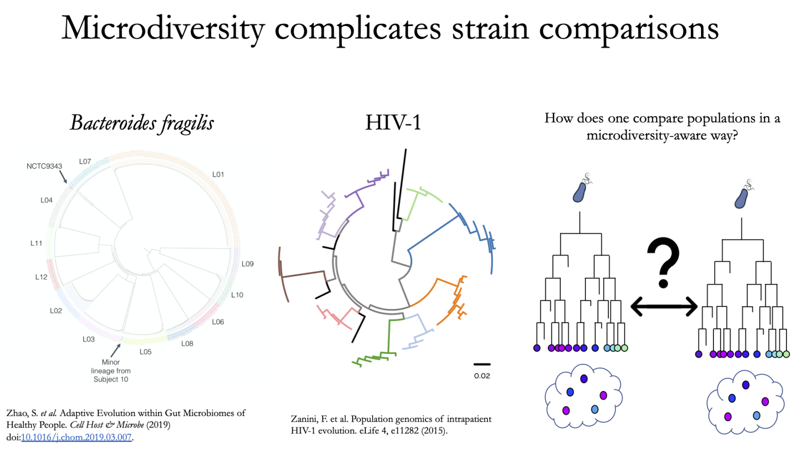
Note: Theoretical models have determined breadth to be: 1 - exp(-coverage) (Lander and Waterman (1988)), slightly different from the empirical derivation presented here and used in inStrain. More information on this subject can be found at this technical note from Illumina.

1.3.7 7. Strain-level comparisons and popANI.

InStrain is able to perform detailed, accurate, microdiversity-aware strain-level comparisons between organisms detected in multiple metagenomic samples. The is done using the command `inStrain compare` on multiple samples that have been profiled using the command `inStrain profile`, and technical details on how this is done is available in the *User Manual*.

To understand why "microdiversity-aware" genomic comparisons are important, consider the fact that all natural microbial populations have some level of genomic heterogeneity present within them.

The image above incorporates data from Zhao et. al. 2019 and Zanini et. al. 2015 (left and middle phylogenetic trees). In each case different colors represent different individuals, and each leaf represents an individual isolate. You can see from these data that although each individual has a distinct microbial population, there is substantial diversity within each individual as well (referred to as intraspecific genetic variation (within species), intrapatient genetic variation (within patient), or *microdiversity*). Knowledge of this fact leads to the question- **how does one accurately compare populations that have intraspecific genetic variation?** Some common approaches include comparing the "average" genome in each sample (the consensus genome) or comparing a number of individual isolates. See *Benchmarks* for



Microdiversity complicates strain comparisons

Bacteroides fragilis

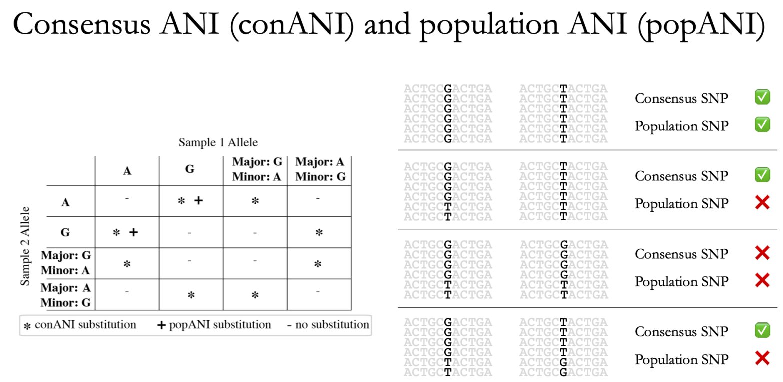
HIV-1

How does one compare populations in a microdiversity-aware way?

Zhao, S. *et al.* Adaptive Evolution within Gut Microbiomes of Healthy People. *Cell Host & Microbe* (2019) doi:10.1016/j.chom.2019.03.007.

Zanini, F. et al. Population genomics of intrapatient HIV-1 evolution. eLife 4, e11282 (2015).

some data on how well these approaches hold up.



Consensus ANI (conANI) and population ANI (popANI)

Sample 1 Allele

Sample 2 Allele	A	G	Major: G Minor: A	Major: A Minor: G
A	-	* +	*	-
G	* +	-	-	*
Major: G Minor: A	*	-	-	*
Major: A Minor: G	-	*	*	-

* conANI substitution   + popANI substitution   - no substitution

```
ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Consensus SNP   ✓
ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Population SNP  ✓
ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA

ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Consensus SNP   ✓
ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Population SNP  ✗
ACTGCTACTGA   ACTGCTACTGA
ACTGCTACTGA   ACTGCTACTGA

ACTGCGACTGA   ACTGCGACTGA
ACTGCGACTGA   ACTGCGACTGA   Consensus SNP   ✗
ACTGCGACTGA   ACTGCGACTGA
ACTGCGACTGA   ACTGCGACTGA   Population SNP  ✗
ACTGCTACTGA   ACTGCTACTGA
ACTGCTACTGA   ACTGCTACTGA

ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Consensus SNP   ✓
ACTGCGACTGA   ACTGCTACTGA
ACTGCGACTGA   ACTGCTACTGA   Population SNP  ✗
ACTGCTACTGA   ACTGCGACTGA
ACTGCTACTGA   ACTGCGACTGA
```

InStrain performs microdiversity-aware comparisons using the metric *popANI*, depicted above, which is also reported alongside the more common consensus-based ANI metric *conANI*. The calculation of *popANI* and *conANI* is not complicated once you understand it (really), but describing can be tricky, and the simplest way of describing it is with examples like those displayed above.

While not depicted in the above figure, the first step of calculating *conANI* and *popANI* is identifying all positions along the genome in which both samples have 5x coverage. This number is reported as the `compared_bases_count`, and it describes the number of base-pairs (bp) that are able to be compared. Next, inStrain goes through each one of these comparable base-pairs and determines if there is a conANI substitution at that position and/or if there is a popANI substitution at that position. The left half of the above figure describes the conditions that will lead to popANI and conANI substitutions. If both samples have the same major allele (e.g. the most common base at that position is the same in both samples), no substitutions will be called. If samples have different major alleles (e.g. the most common base in sample 1 is A, and the most common base in sample 2 is C), a conANI substitution will be called. If there are **no alleles that are shared between the two samples**, major or minor, a popANI substitution will be called. The calculations that determine whether or not a base is considered "present" as a minor allele in a sample (vs. it just being a sequencing error) are discussed in the *User Manual*.

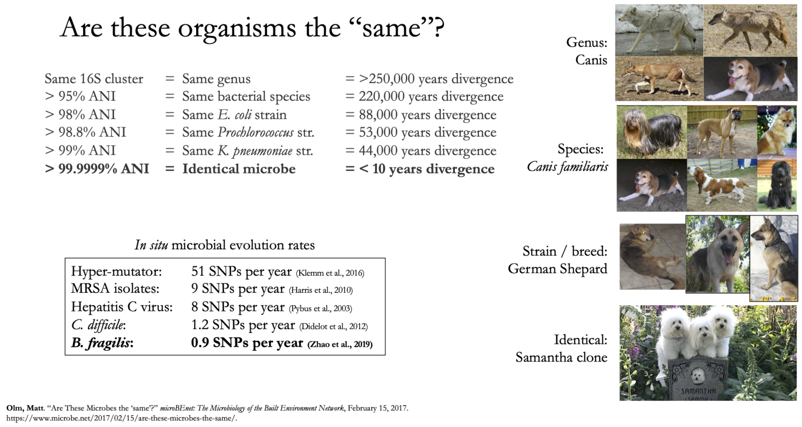
On the right side of the above figure we see several examples of this in action. In the top row there are no alleles that are the same in both samples, therefore the site will count as both a conANI SNP and a popANI SNP. In the second row the consensus allele is different in both samples (its G in the sample on the left and T in the sample on the right), so a conANI SNP will be called. However the samples DO share an allele (T is present in both samples), so this will NOT be considered a popANI substitution. In the third row both samples have the same consensus allele and share alleles, so no substitutions are called. In the last row the samples have different consensus alleles (G on the left and T on the right), so a conANI substitution will be called, but there is allele overlap between the samples (both samples have G and T) so a popANI substitution will NOT be called.

Once we have the `compared_bases_count`, number of conANI SNPs, and number of popANI SNPs, calculation of *conANI* and *popANI* is trivial.

$$popANI = (comparedbasescount - popANIsnps)/comparedbasescount$$
$$conANI = (comparedbasescount - conANIsnps)/comparedbasescount$$

Note: Notice that `compared_bases_count` is integral to conANI and popANI calculations. It essentially determines the "denominator" in the calculations, as it let's you know how bases were compared in the calculation. Attempting to calculate *conANI* and *popANI* using SNP-calling data from other programs will likely leave out this critical information. Remember- `compared_bases_count` is a measure of how many bases have at least 5x coverage in BOTH samples. Consideration of `compared_bases_count` is critical to ensure that *popANI* isn't high simply because one or both sample's doesn't have high enough coverage to detect SNPs

1.3.8 8. Thresholds for determining "same" vs. "different" strains.



Are these organisms the "same"?

Same 16S cluster	= Same genus	= >250,000 years divergence
> 95% ANI	= Same bacterial species	= 220,000 years divergence
> 98% ANI	= Same *E. coli* strain	= 88,000 years divergence
> 98.8% ANI	= Same *Prochlorococcus* str.	= 53,000 years divergence
> 99% ANI	= Same *K. pneumoniae* str.	= 44,000 years divergence
> 99.9999% ANI	**= Identical microbe**	**= < 10 years divergence**

In situ microbial evolution rates

Hyper-mutator:	51 SNPs per year (Klemm et al., 2016)
MRSA isolates:	9 SNPs per year (Harris et al., 2010)
Hepatitis C virus:	8 SNPs per year (Pybus et al., 2003)
C. difficile:	1.2 SNPs per year (Didelot et al., 2012)
***B. fragilis*:**	**0.9 SNPs per year (Zhao et al., 2019)**

Olm, Matt. "Are These Microbes the 'same'?" *microBEnet: The Microbiology of the Built Environment Network*, February 15, 2017. https://www.microbe.net/2017/02/15/are-these-microbes-the-same/.

Once inStrain performs it's strain-level comparisons, one must decide on some threshold to define microbes as being the "same" or "different" strains. The figure above illustrates some common ANI values used for defining various relationships between microbes (top left), some previously reported rates of in situ microbial evolution (bottom left), and estimates of divergence times for various *ANI* thresholds (top left). On the right is an analogy using canine taxonomy.

The figure above illustrates how loose *ANI* thresholds can be used to define relatively broad groups of organisms, for example the genus Canis or the species *Canis Familiaris*. Sub-species taxonomic levels, referred to as strains in the microbe world and breeds in the dog world, describe groups of organisms within particular species. Strain definitions in the microbial world are not consistent, but some example strain ANI thresholds are shown. **There is still generally some variation within strains, however.** This is exemplified by the fact that while dogs of the same breed are similar to one another, they're not **identical** to one another. Similarly, microbes of the same strain based on a 99% ANI definition can have diverged for roughly 44,000 years (based on the in situ mutation rate in bold in the bottom left). Clearly microbes that have diverged for tens of thousands of years are not **identical** to one another. **Thus if we want to know whether samples are linked by a recent microbial transmission event, we need an extremely stringent definition of "same" that is beyond the typical strain level**. Note that the dogs in the bottom right are clones that truly do represent identical dogs..

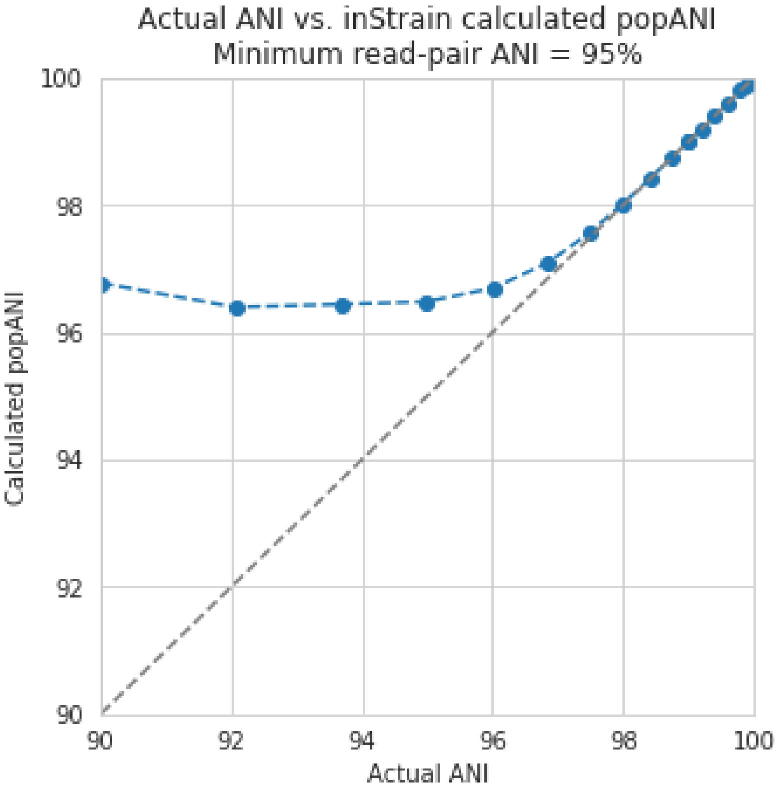
To identify microbes that are linked by a recent transmission event we want the most stringent ANI threshold possible. 99.9999% *ANI*, for example, represents less than 10 years of divergence time and could be a useful metric. Metagenomic sequencing is messy, however, and when working with this level of stringency we need to think about our limit of detection. The *Benchmarks* section contains data on the limit of detection for inStrain using defined microbial communities (see section "Benchmark with true microbial communities") **The conclusion is that 99.999% popANI is a good, highly stringent definition for identical strains that is within the limit of detection for metagenomic analysis.**. In addition to *popANI*, one must also consider the fraction of the genome that was at sufficient coverage in both samples being compared. This value (reported as `percent_genome_compared`) is more of a judgement call, but we recommend requiring a minimum of 25% or 50% `percent_genome_compared` in addition to the *popANI* threshold.

Note: In conclusion, organisms in different samples that are linked by a recent transmission event should have 99.999% popANI and 50% `percent_genome_compared`

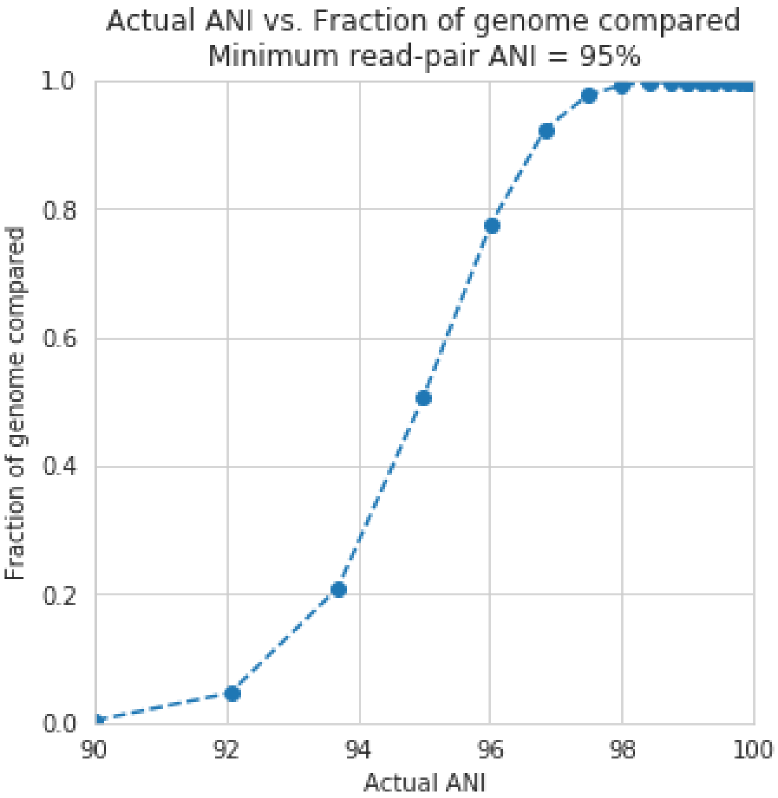
1.3.9 9. Importance of representative genomes when calculating popANI

InStrain strain-level comparisons are based on mappings to representative genomes. In order for this to work well, however **reads with variation must be able to map to the representative genomes within the ''–min_read_ani'' threshold.** Note that `inStrain compare` will use the `--min_read_ani` selected during the `inStrain profile` commands by default.

Below are a series of plots generated from synthetic data demonstrating this fact. In these plots a reference genome was downloaded from NCBI, mutated to a series of known ANI values, synthetic reads were generated from each of these mutated genomes, and synthetic reads were then mapped back to the original genome.



In the above plot the `--min_read_ani` is set to 95%. As you can see, when the true ANI value between the genomes is below 98%, popANI values reported by inStrain are not accurate. The reason that this happens is because reads with genuine variation are being filtered out by inStrain, leaving only the reads without variation, which artificially increases the reported *popANI* values. **In sections above we demonstrated that ‘‘–min_read_ani‘‘ should be ~3% looser than the population you'd like to recruit reads from; the same rule applies here.** If you'd like to compare organisms that have a popANI of 95%, your `--min_read_ani` needs to be 92%. Here we have a `--min_read_ani` of 95%, so we can detect accurate *popANI* values of 98% or above (as shown in the above figure). This phenomena is explored further in the following way.

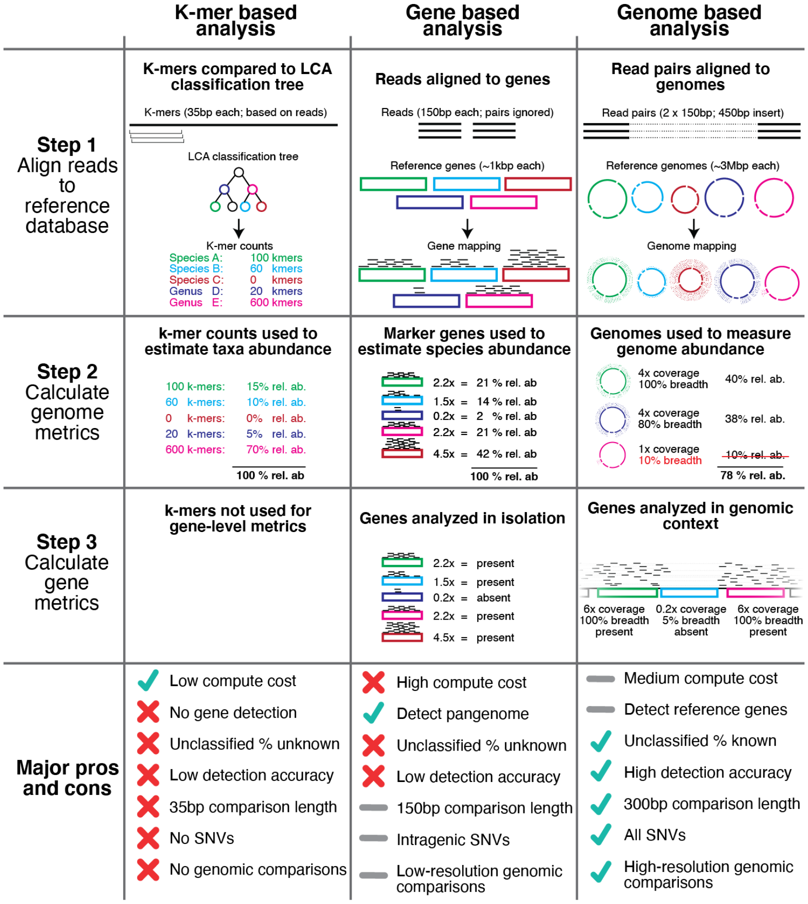


The above figure displays the `percent_genome_compared` for each of the comparisons in the first figure in this section. As expected, when comparing genomes of low ANI values with a read-pair ANI threshold of 95%, only a small amount of the genome is actually being compared. This genome fraction represents the spaces of the genome that happen to be the most similar, and thus the inStrain calculated ANI value is overestimated. **The conclusion here is that in order to get an accurate ANI value, you need to set your ‘‘–min_read_ani‘‘ at least 3% below the ANI value that you wish to detect.**

1.3.10 10. Using inStrain for gene-based functional analysis

The above figure shows a visual representation of k-mer based metagenomic analysis, gene-based metagenomic analysis, and Representative genome based metagenomic analysis. As you can see, among the advantages of genome-based metagenomic analysis is the ability to perform context-aware functional profiling.

InStrain does not have the ability to annotate genes. However, inStrain does have the ability to deeply profile all genes in a sample, including analysis of coverage, coverage variation, gene *pN/pS*, nucleotide diversity, individual *SNVs*, etc. This gene-level information can then be combined with gene annotations to perform robust functional analysis. Any database can be used for this type of analysis, including pFam for protein domain annotations, ABRicate for antibiotic resistance gene annotation, UniRef100 for general protein annotation, and dbCAN for CAZyme annotation.



	K-mer based analysis	Gene based analysis	Genome based analysis
Step 1 Align reads to reference database	**K-mers compared to LCA classification tree** K-mers (35bp each; based on reads) LCA classification tree K-mer counts Species A: 100 kmers Species B: 60 kmers Species C: 0 kmers Genus D: 20 kmers Genus E: 600 kmers	**Reads aligned to genes** Reads (150bp each; pairs ignored) Reference genes (~1kbp each) Gene mapping	**Read pairs aligned to genomes** Read pairs (2 x 150bp; 450bp insert) Reference genomes (~3Mbp each) Genome mapping
Step 2 Calculate genome metrics	**k-mer counts used to estimate taxa abundance** 100 k-mers: 15% rel. ab. 60 k-mers: 10% rel. ab. 0 k-mers: 0% rel. ab. 20 k-mers: 5% rel. ab. 600 k-mers: 70% rel. ab. 100 % rel. ab	**Marker genes used to estimate species abundance** 2.2x = 21 % rel. ab 1.5x = 14 % rel. ab 0.2x = 2 % rel. ab 2.2x = 21 % rel. ab 4.5x = 42 % rel. ab 100 % rel. ab	**Genomes used to measure genome abundance** 4x coverage 100% breadth 40% rel. ab. 4x coverage 80% breadth 38% rel. ab. 1x coverage 10% breadth ~~10% rel. ab.~~ 78 % rel. ab.
Step 3 Calculate gene metrics	**k-mers not used for gene-level metrics**	**Genes analyzed in isolation** 2.2x = present 1.5x = present 0.2x = absent 2.2x = present 4.5x = present	**Genes analyzed in genomic context** 6x coverage 100% breadth present 0.2x coverage 5% breadth absent 6x coverage 100% breadth present
Major pros and cons	✓ Low compute cost ✗ No gene detection ✗ Unclassified % unknown ✗ Low detection accuracy ✗ 35bp comparison length ✗ No SNVs ✗ No genomic comparisons	✗ High compute cost ✓ Detect pangenome ✗ Unclassified % unknown ✗ Low detection accuracy — 150bp comparison length — Intragenic SNVs — Low-resolution genomic comparisons	— Medium compute cost — Detect reference genes ✓ Unclassified % known ✓ High detection accuracy ✓ 300bp comparison length ✓ All SNVs ✓ High-resolution genomic comparisons

For examples of inStrain-based functional annotation in action, see Table 1 and Figure 6 of the inStrain publication and this GitHub repo focused on COVID-19 population genomics analysis

1.3.11 11. Reducing inStrain resource usage

Note: When mapping to a *Genome database* with more than a handful of genomes make sure to use the flag `--database_mode`

The two computational resources to consider when running inStrain are the number of processes given (`-p`) and the amount of RAM on the computer (usually not adjustable unless using cloud-based computing).

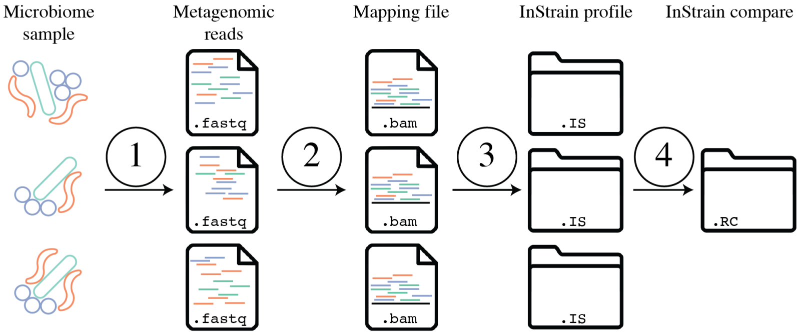
Using inStrain v1.3.3, running inStrain on a .bam file of moderate size (1 Gbp or less) will generally take less than an hour with 6 cores, and use about 8Gb of RAM. InStrain is designed to handle large .bam files as well. Running a huge .bam file (30 Gbp) with 32 cores, for example, will take ~2 hours and use about 128Gb of RAM. The more processes you give inStrain the faster it will run, but also the more RAM it will use.

In the log folder InStrain provides a lot of information on where it's spending it's time and where it's using it's RAM.

To reduce RAM usage, you can try the following things:

- Use the `--skip_mm` flag. This won't profile things on the *mm* level, and will treat every read pair as perfectly mapped. This is perfectly fine for most applications
- Make sure and use the `--database_mode` flag when mapping to *genome databases*. This will do a couple of things to try and reduce RAM usage
- Use less processes (`-p`). Using more processes will make inStrain run faster, but it will also use more RAM while doing so

1.4 Tutorial



The above figure provides a conceptual overview of the steps involved when running inStrain. Step 1 is generating sequencing reads, step 2 is mapping those sequencing reads to a *Genome database*, step 3 is profiling the mapping with `inStrain profile`, step 4 is comparing inStrian profiles using `inStrain compare`.

1.4.1 Quick Start

The two main operations of inStrain are `compare` and `profile`.

Profile

`InStrain profile` takes as input a *fasta file* and a *bam file* and runs a series of steps to characterize the *nucleotide diversity*, *SNSs* and *SNVs*, *linkage*, etc.. If one provides a *scaffold-to-bin file* it will calculate genome-level metrics, and if one provides a *genes file* it will calculate gene level metrics.

The most basic `inStrain profile` command has this form:

```
$ inStrain profile .bam_file .fasta_file -o IS_output_name
```

Compare

`InStrain compare` takes as input multiple `inStrain profile` objects (generated using the command above) and performs strain-level comparisons. Each `inStrain profile` object used by `InStrain compare` must be made from reads mapped to the same *fasta file*.

The most basic inStrain compare command looks like this:

```
$ inStrain compare -i IS_output_1 IS_output_2 IS_output_3
```

Other

There are a number of other operations that inStrain can perform as well, although these generally perform more niche tasks. Check the program help (`inStrain -h`) to see a full list of the available operations

```
$ inStrain -h

                ...::: inStrain v1.4.0 :::...

  Matt Olm and Alex Crits-Christoph. MIT License. Banfield Lab, UC Berkeley. 2019

  Choose one of the operations below for more detailed help. See https://instrain.
↪readthedocs.io for documentation.
  Example: inStrain profile -h

  Workflows:
    profile         -> Create an inStrain profile (microdiversity analysis) from a␣
↪mapping.
    compare         -> Compare multiple inStrain profiles (popANI, coverage_overlap,␣
↪etc.)

  Single operations:
    profile_genes   -> Calculate gene-level metrics on an inStrain profile␣
↪[DEPRECATED; PROVIDE GENES TO profile]
    genome_wide     -> Calculate genome-level metrics on an inStrain profile␣
↪[DEPRECATED; PROVIDE .stb FILES TO profile / compare]
    quick_profile   -> Quickly calculate coverage and breadth of a mapping using␣
↪coverM
    filter_reads    -> Commands related to filtering reads from .bam files
    plot            -> Make figures from the results of "profile" or "compare"
    other           -> Other miscellaneous operations
```

See also:

Installation To get started using the program

User Manual For descriptions of what the modules can do and information on how to prepare data for inStrain

Expected output To view example output and how to interpret it

1.4.2 Example inStrain commands

Running inStrain profile on a single genome

```
inStrain profile mappingfile.bam genomefile.fasta -o outputlocation.IS -p 6 -
↪g genesfile.fasta
```

Running inStrain profile on a large set of genomes

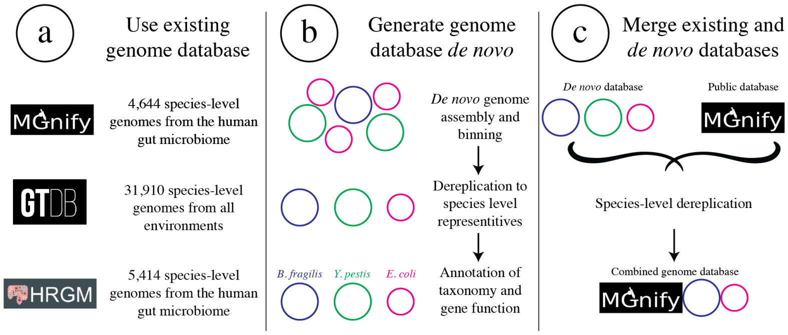
```
inStrain profile mappingfile.bam genomesfile.fasta -o outputlocation.IS -p 6␣
↪-g genesfile.fasta -s scaffoldtobin.stb --database_mode
```

Running inStrain compare on a large set of genomes

```
inStrain compare -i genomefile-vs-sample1.IS/ genomefile-vs-sample2.IS/ -o␣
↪genomefile.IS.COMPARE -p 6 -s scaffoldtobin.stb --database_mode
```

1.4.3 Tutorials

The following tutorials give step-by-step instructions on how to run inStrain a couple of different ways. The main difference between these tutorials is in how the *Genome database* used for inStrain analysis is generated. The user to inStrain decides on their own what genomes should be used for analysis, and there are a couple of broad options as depicted in the figure below.


a
Use existing genome database
MGnify
4,644 species-level genomes from the human gut microbiome
GTDB
31,910 species-level genomes from all environments
HRGM
5,414 species-level genomes from the human gut microbiome
b
Generate genome database de novo
De novo genome assembly and binning
Dereplication to species level representitives
Annotation of taxonomy and gene function
B. fragilis
Y. pestis
E. coli
c
Merge existing and de novo databases
De novo database
Public database
MGnify
Species-level dereplication
Combined genome database
MGnify

Tutorial #1 uses test data that comes packaged with the inStrain source code to go through the basic steps of the program. It also describes how you can run using genomes that you have generated on your own.

Tutorial #2 describes how to run inStrain using an existing, public genome database. This way of running inStrain avoids the need for metagenomic assembly and genome binning.

Tutorial #3 describes how to combine custom genomes with an existing genome database. This allows users to include both sample-specific representative genomes and an existing genome database, and allows for comprehensive, accurate analysis.

1.4.4 Tutorial #1) Running inStrain on provided test data

The following tutorial goes through an example run of inStrain. You can follow along with your own data, or use a small set of reads that are included in the inStrain source code for testing. They can be found in the folder `test/test_data/` of your install folder, or can be downloaded from the inStrain source code at this link on GitHub. The files that we'll use for this tutorial are the forward and reverse metagenomic reads (`N5_271_010G1.R1.fastq.gz` and `N5_271_010G1.R2.fastq.gz`) and a .fasta file to map to (`N5_271_010G1_scaffold_min1000.fa`). In case you're curious, these metagenomic reads come from a premature infant fecal sample.

Preparing input files

After downloading the genome file that you would like to profile (the *fasta file*) and at least one set of paired reads, the first thing to do is to map the reads to the .fasta file in order to generate a *bam file*.

When this mapping is performed it is important that you map to all genomes simultaneously (see *Important concepts* for why this is important). This involves combining all of the genomes that you'd like to map into a single .fasta file. In our case our .fasta file already has all of the genomes that we'd like to profile within it, but if you did want to profile a number of different genomes, you could combine them using a command like this

```
$  cat raw_data/S2_002_005G1_phage_Clostridioides_difficile.fasta raw_data/S2_018_
↪020G1_bacteria_Clostridioides_difficile.fasta > allGenomes_v1.fasta
```

When we do this we also need to generate a file to let inStrain know which scaffolds came from which genomes. We can do this by giving inStrain a list of the .fasta files that went into making the concatenated .fasta file, or we can make a *scaffold-to-bin file* file, which lists the genome assignment of each scaffold in a tab-delimited file. This is how to do the later method using the parse_stb.py script that comes with the program `dRep` (Installed with the command `pip install drep --upgrade`)

```
$ parse_stb.py --reverse -f raw_data/S2_002_005G1_phage_Clostridioides_difficile.
↪fasta  raw_data/S2_018_020G1_bacteria_Clostridioides_difficile.fasta  -o genomes.stb
```

Next we must map our reads to this *fasta file* to create *bam files*. In this tutorial we will use the mapping program Bowtie 2

```
$ mkdir bt2

$ bowtie2-build ~/Programs/inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa␣
↪bt2/N5_271_010G1_scaffold_min1000.fa

$ bowtie2 -p 6 -x bt2/N5_271_010G1_scaffold_min1000.fa -1 ~/Programs/inStrain/test/
↪test_data/N5_271_010G1.R1.fastq.gz -2 ~/Programs/inStrain/test/test_data/N5_271_
↪010G1.R2.fastq.gz > N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sam
```

At this point we have generated a .sam file, the precursor to .bam files. Lets make sure it's there and not empty

```
$ ls -lht

total 34944
-rw-r--r--  1 mattolm  staff    16M Jan 23 11:56 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.sam
drwxr-xr-x  8 mattolm  staff   256B Jan 23 11:54 bt2/
```

Perfect. At this point we could convert the .sam file to a sorted and indexed .bam file using samtools, but since inStrain can do that for us automatically we won't bother.

If we want inStrain to do gene-level profiling we need to give it a list of genes to profile. **Note - this is an optional step that is not required for inStrain to work in general, but without this you will not get gene-level profiles**

We will profile our genes using the program prodigal, which can be run using the following example command

```
$ prodigal -i ~/Programs/inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa -d␣
↪N5_271_010G1_scaffold_min1000.fa.genes.fna -a N5_271_010G1_scaffold_min1000.fa.
↪genes.faa
```

(continues on next page)

(continued from previous page)

```
```

Running inStrain profile

Now that we've gotten everything set up it's time to run inStrain. To see all of the options, run

```
$ inStrain profile -h
```

A long list of arguments and options will show up. For more details on what these do, see *User Manual*. The **only** arguments that are absolutely required, however, are a .sam or .bam mapping file, and the .fasta file that the mapping file is mapped to.

Note: In this case we're going to have inStrain profile the mapping, call genes, make the results genome wide, and plot the results all in one command. This is the recommended way to do things for the most computational efficiency. The other, not recommended way would be to run these all as separate steps (using the subcommands `inStrain profile`, `inStrain profile_genes`, `inStrain genome_wide`, and `inStrain plot`). See *User Manual* for more information.

Using all of the files we generated above, here is going to be our inStrain command

```
$ inStrain profile N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sam ~/Programs/
↪inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa -o N5_271_010G1_scaffold_
↪min1000.fa-vs-N5_271_010G1.IS -p 6 -g N5_271_010G1_scaffold_min1000.fa.genes.fna -s␣
↪~/Programs/inStrain/test/test_data/N5_271_010G1.maxbin2.stb
```

You should see the following as inStrain runs (should only take a few minutes)

```
You gave me a sam- I'm going to make it a .bam now
Converting N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sam to N5_271_010G1_
↪scaffold_min1000.fa-vs-N5_271_010G1.ba
m
samtools view -S -b N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sam > N5_271_
↪010G1_scaffold_min1000.fa-vs-N5_271_
010G1.bam
sorting N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.bam
samtools sort N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.bam -o N5_271_010G1_
↪scaffold_min1000.fa-vs-N5_271_010G1
.sorted.bam -@ 6
[bam_sort_core] merging from 0 files and 6 in-memory blocks...
Indexing N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam
samtools index N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam N5_271_
↪010G1_scaffold_min1000.fa-vs-N5_271_
010G1.sorted.bam.bai -@ 6
***************************************************
    ..:: inStrain profile Step 1. Filter reads ::..
***************************************************

Filtering reads: 100%|| 178/178 [00:02<00:00, 70.99it/s]
37.3% of reads were removed during filtering
1,727 read pairs remain (0.0004472 Gbp)
***************************************************
.:: inStrain profile Step 2. Profile scaffolds ::..
***************************************************
```

(continues on next page)

(continued from previous page)

```

Profiling splits: 100%|| 7/7 [00:05<00:00,  1.21it/s]
Merging splits and profiling genes: 100%|| 7/7 [00:08<00:00,  1.18s/it]
***************************************************
.:: inStrain profile Step 4. Make genome-wide ::..
***************************************************

Scaffold to bin was made using .stb file
85.66% of scaffolds have a genome
93.82% of scaffolds have a genome
99.30% of scaffolds have a genome
***************************************************
 .:: inStrain profile Step 5. Generate plots ::..
***************************************************
making plots 1, 2, 3, 4, 5, 6, 7, 8, 9
Plotting plot 1
Plotting plot 2
85.66% of scaffolds have a genome
Plotting plot 3
57.37% of scaffolds have a genome
Plotting plot 4
97.33% of scaffolds have a genome
Plotting plot 5
Plotting plot 6
Plotting plot 7
97.33% of scaffolds have a genome
Plotting plot 8
94.32% of scaffolds have a genome
Plotting plot 9
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

..:: inStrain profile finished ::..

Output tables........ N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/output/
Figures.............. N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/figures/
Logging.............. N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/log/

See documentation for output descriptions - https://instrain.readthedocs.io/en/latest/

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

The last but of the output shows you where the plots and figures have been made. Here's a list of the files that you should see

```
$ ls -lht N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/output/
total 91K
-rw-rw-r-- 1 mattolm infantgi  35K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_SNVs.tsv
-rw-rw-r-- 1 mattolm infantgi 1.2K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_genome_info.tsv
-rw-rw-r-- 1 mattolm infantgi  23K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_mapping_info.tsv
-rw-rw-r-- 1 mattolm infantgi  92K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_gene_info.tsv
-rw-rw-r-- 1 mattolm infantgi  15K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_linkage.tsv
```

(continues on next page)

(continued from previous page)

```
-rw-rw-r-- 1 mattolm infantgi  30K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_scaffold_info.tsv

$ ls -lht N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/figures/
total 3.5M
-rw-rw-r-- 1 mattolm infantgi 386K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_GeneHistogram_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 379K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_LinkageDecay_types_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 404K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_ScaffoldInspection_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 375K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_ReadFiltering_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 378K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_LinkageDecay_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 377K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_MajorAllele_frequency_plot.pdf
-rw-rw-r-- 1 mattolm infantgi 375K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_readANI_distribution.pdf
-rw-rw-r-- 1 mattolm infantgi 400K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_genomeWide_microdiveristy_metrics.pdf
-rw-rw-r-- 1 mattolm infantgi 376K Jan 15 10:10 N5_271_010G1_scaffold_min1000.fa-vs-
↪N5_271_010G1.IS_CoverageAndBreadth_vs_readMismatch.pdf
```

We have now successfully generated an inStrain profile! For help interpreting the output files, see *Expected output*

Running inStrain parse_annotations

`InStrain parse_annotations` creates output files that make it easier to perform functional gene analysis. One input file is the InStrain profile object, which we just created above, and the other input file is a table of gene annotations.

You can annotate your genes using whatever gene annotation database you like (depending on your specific project and questions). The section *Gene Annotation* in *User Manual* has instructions for a few databases. In this tutorial let's just annotate with KEGG Orthologies (KOs) and Carbohydrate-Active enZYmes (CAZymes).

To do the annotations we'll need the amino acid sequences of the genes (the file ending in .faa, created using the prodigal command above) even though the gene nucleotide sequences is what we provided to inStrain profile. Following the section *Gene Annotation* in *User Manual*, we'll then run the following commands

```
$ exec_annotation -p profiles -k ko_list --cpu 10 --tmp-dir ./tmp -o N5_271_010G1_
↪scaffold_min1000.fa.genes.faa.KO N5_271_010G1_scaffold_min1000.fa.genes.faa

$ hmmscan --domtblout N5_271_010G1_scaffold_min1000.fa.genes.faa_vs_dbCAN_v11.dm␣
↪dbCAN-HMMdb-V11.txt N5_271_010G1_scaffold_min1000.fa.genes.faa > /dev/null ; sh /
↪hmmscan-parser.sh N5_271_010G1_scaffold_min1000.fa.genes.faa_vs_dbCAN_v11.dm > N5_
↪271_010G1_scaffold_min1000.fa.genes.faa_vs_dbCAN_v11.dm.ps ; cat N5_271_010G1_
↪scaffold_min1000.fa.genes.faa_vs_dbCAN_v11.dm.ps | awk '$5<1e-15&&$10>0.35' > N5_
↪271_010G1_scaffold_min1000.fa.genes.faa_vs_dbCAN_v11.dm.ps.stringent
```

We'll now need to use python / R / Excel to parse and re-format the output of these (and any other) annotations. In the end they need be transformed into a single .csv file with the columns "gene" and "anno". See *User Manual* for more details on the specific formatting requirements. In our case the file, which I called *geneAnnotations_v1.csv*, should look like this:

Table 1: geneAnnotations_v1.csv

gene	anno
N5_271_010G1_scaffold_12_3	K06956
N5_271_010G1_scaffold_14_1	K09890
N5_271_010G1_scaffold_15_2	K07482
N5_271_010G1_scaffold_19_7	K09890
N5_271_010G1_scaffold_25_1	K20386
N5_271_010G1_scaffold_25_1	K15558
N5_271_010G1_scaffold_25_1	K19762
N5_271_010G1_scaffold_25_2	K06864
N5_271_010G1_scaffold_28_3	K07482

Now we can run *inStrain parse_annotations* with a command like the following

```
$ inStrain parse_annotations -i N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/ -
↪o genes_output_v1 -a geneAnnotations_v1.csv
```

Done! To see what output files you can expect, see *Expected output*

Running inStrain compare

`InStrain compare` compares genomes that have been profiled by multiple different metagenomic mappings. To compare genomes in the sample we just profiled above, we need to generate another *bam file* of reads from another sample to the **same** .fasta file. Provided in the inStrain test_data folder is exactly that- another different set of reads mapped to the same .fasta file (N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam). Let's run inStrain on this to make a new inStrain profile

```
$ inStrain profile test_data/N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.
↪bam N5_271_010G1_scaffold_min1000.fa -o N5_271_010G1_scaffold_min1000.fa-vs-N5_271_
↪010G2.IS -p 6 -g N5_271_010G1_scaffold_min1000.fa.genes.fna -s N5_271_010G1.maxbin2.
↪stb
```

To see the help section for `inStrain compare` run:

```
$ inStrain compare -h
```

As above, this will print out a whole list of parameters that can be turned depending on your specific use-case. *Important concepts* and *User Manual* provide some insight into what these parameters do and how to tune them. For the purposes of this tutorial we're going to use mostly default parameters, giving us the following command

```
$ inStrain compare -i N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.IS/ N5_271_
↪010G1_scaffold_min1000.fa-vs-N5_271_010G2.IS/ -s .N5_271_010G1.maxbin2.stb -p 6 -o␣
↪N5_271_010G1_scaffold_min1000.fa.IS.COMPARE
```

This command should produce the following output

```
Scaffold to bin was made using .stb file
***************************************************
    ..:: inStrain compare Step 1. Load data ::..
***************************************************

Loading Profiles into RAM: 100%|| 2/2 [00:00<00:00, 67.45it/s]
158 of 167 scaffolds are in at least 2 samples
```

(continues on next page)

(continued from previous page)

```
***************************************************
..:: inStrain compare Step 2. Run comparisons ::..
***************************************************

Running group 1 of 1
Comparing scaffolds: 100%|| 158/158 [00:04<00:00, 36.12it/s]
***************************************************
..:: inStrain compare Step 3. Auxiliary processing ::..
***************************************************

***************************************************
..:: inStrain compare Step 4. Store results ::..
***************************************************

making plots 10
Plotting plot 10
/home/mattolm/.pyenv/versions/3.6.10/lib/python3.6/site-packages/inStrain/
↪plottingUtilities.py:963: UserWarning: FixedFormatter should only be used together␣
↪with FixedLocator
  axes.set_xticklabels(labels)
/home/mattolm/.pyenv/versions/3.6.10/lib/python3.6/site-packages/inStrain/
↪plottingUtilities.py:963: UserWarning: FixedFormatter should only be used together␣
↪with FixedLocator
  axes.set_xticklabels(labels)
Done!
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

..:: inStrain compare finished ::..

Output tables........ N5_271_010G1_scaffold_min1000.fa.IS.COMPARE/output/
Figures.............. N5_271_010G1_scaffold_min1000.fa.IS.COMPARE/figures/
Logging.............. N5_271_010G1_scaffold_min1000.fa.IS.COMPARE/log/

See documentation for output descriptions - https://instrain.readthedocs.io/en/latest/

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

As before, the last part of the output shows you where the plots and figures have been made. Here's a list of the files that you should see

```
$ ls -lht N5_271_010G1_scaffold_min1000.fa.IS.COMPARE/output/
total 14K
-rw-rw-r-- 1 mattolm infantgi 28K Jan 15 10:33 N5_271_010G1_scaffold_min1000.fa.IS.
↪COMPARE_comparisonsTable.tsv
-rw-rw-r-- 1 mattolm infantgi 352 Jan 15 10:33 N5_271_010G1_scaffold_min1000.fa.IS.
↪COMPARE_strain_clusters.tsv
-rw-rw-r-- 1 mattolm infantgi 554 Jan 15 10:33 N5_271_010G1_scaffold_min1000.fa.IS.
↪COMPARE_genomeWide_compare.tsv

$ ls  -lht N5_271_010G1_scaffold_min1000.fa.IS.COMPARE/figures/
total 393K
-rw-rw-r-- 1 mattolm infantgi 376K Jan 15 10:33 N5_271_010G1_scaffold_min1000.fa.IS.
↪COMPARE_inStrainCompare_dendrograms.pdf
```

Success! As before, for help interpreting this output see *Expected output* .

1.4.5 Tutorial #2) Running inStrain using a public genome database

If you don't want to assemble and bin your metagenomic samples it is also possible to run inStrain using publicly available reference genomes. Here we will go through a tutorial on how to do this with the UHGG genome collection, a collection of all microbial species known to exist in the human gut. The steps in this tutorial could be repeated with any set of genomes though, including genomes assembled from non-industrialized human populations, as available at the following link - https://doi.org/10.5281/zenodo.7782709

Preparing a genome database

Note: The genome database created in this section is available for direct download at the following link - https://doi.org/10.5281/zenodo.4441269 . You can download those files directly and skip this section if you would like. **This genome set is based on UHGG version 1 and was created on Jan 14, 2021**.

Note: An alternative genome database that includes UHGG genomes AND genomes assembled from non-industrialized human populations is available for direct download at the following link - https://doi.org/10.5281/zenodo.7782709 . This genome set is described in the following publication - https://doi.org/10.1101/2022.03.30.486478

In order to create a genome database we need to download the genomes, create a *scaffold-to-bin file*, create a *genes file*, and merge all genomes into a single *fasta file* that we can make a bowtie2 mapping index out of. All genomes in a genome need to database need to be distinct from one another, but not too distinct. See section "Establishing and evaluating genome databases" in *Important concepts* for more info.

First we must download the UHGG genomes themselves. The FTP site is here, and metadata on genomes is genomes-all_metadata.tsv. Let's download this metadata file using curl:

```
$ curl http://ftp.ebi.ac.uk/pub/databases/metagenomics/mgnify_genomes/human-gut/v1.0/
↪genomes-all_metadata.tsv -o genomes-all_metadata.tsv
```

Now that we have this metadata file we need to download all species representative genomes. There are a number of ways to do this, but we're going to do it by parsing the metadata table in unix. Running the following command will 1) identify columns of species representatives, 2) parse the row to determine their FTP location, 3) create and run a `curl` command to download the genome:

```
$ cat genomes-all_metadata.tsv | awk -F "\t" '{if ($17 == $1) print "curl ftp://ftp.
↪ebi.ac.uk/pub/databases/metagenomics/mgnify_genomes/human-gut/v1.0/uhgg_catalogue/"␣
↪substr($18,0,13) "/" $18 "/genome/" $18 ".fna -o UHGG_reps/" $1 ".fna"}' | bash
```

The following command will let us check and make sure that we downloaded all 4644 genomes:

```
$ ls UHGG_reps/ | wc -l
  4644
```

Next we need to create a *scaffold-to-bin file*. This can easily be done using the script parse_stb.py that comes with the program `dRep`:

```
$ parse_stb.py --reverse -f UHGG_reps/* -o UHGG.stb
```

Next we'll profile the genes for each genome using Prodigal to create a *genes file*. This can be done on the concatenated genome file (created below) or on the individual genomes (as shown in this code chunk). The benefit of the later is that it allows Prodigal to be run in single genome mode, as opposed to metagenome mode, which can be more accurate:

```
$ mkdir UHGG_genes

$ cd UHGG_reps/

$ for genome in $(ls *.fna); do echo prodigal -i $genome -o ../UHGG_genes/$genome.
↪genes -a ../UHGG_genes/$genome.gene.faa -d ../UHGG_genes/$genome.gene.fna -m -p␣
↪single; done | parallel -j 6

$ cat UHGG_genes/*.gene.fna > UHGG_reps.genes.fna

$ cat UHGG_genes/*.gene.faa > UHGG_reps.genes.faa
```

Finally we need to concatenate all genomes together into a single *fasta file* and create a bowtie2 mapping index from it:

```
$ cat UHGG_reps/* > UHGG_reps.fasta

$ bowtie2-build UHGG_reps.fasta UHGG_reps.fasta.bt2 --large-index --threads 20
```

Mapping to the Genome Database

Here we will use the program Bowtie2 to align our reads to the reference database. If you downloaded the pre-made version of bowtie2 index, you'll need to extract it using the following command

```
$ tar -zxvf UHGG_reps_v1.bt2.tgz
```

This should yield a set of 5 files that end in `.bt2l`

Next we need to map our metagenomic reads to the database. For the purposes of this tutorial we'll use metagenomic reads that came from a premature infant fecal sample. The bowtie2 command to map these reads is

```
$ bowtie2 -p 10 -x /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.
↪fasta.bt2 -1 /groups/banfield/projects/human/data8/raw.d/NIH5/reads/HR/N5_216_039G1_
↪T0140F_S50_L002.HR.R1.fastq.gz -2 /groups/banfield/projects/human/data8/raw.d/NIH5/
↪reads/HR/N5_216_039G1_T0140F_S50_L002.HR.R2.fastq.gz > UHGG_reps.fasta-vs-N5_216_
↪039G1.sam

  7032881 reads; of these:
    7032881 (100.00%) were paired; of these:
      1690938 (24.04%) aligned concordantly 0 times
      1905098 (27.09%) aligned concordantly exactly 1 time
      3436845 (48.87%) aligned concordantly >1 times
      ----
      1690938 pairs aligned concordantly 0 times; of these:
        139804 (8.27%) aligned discordantly 1 time
      ----
      1551134 pairs aligned 0 times concordantly or discordantly; of these:
        3102268 mates make up the pairs; of these:
          1851642 (59.69%) aligned 0 times
          279669 (9.01%) aligned exactly 1 time
          970957 (31.30%) aligned >1 times
  86.84% overall alignment rate
```

This mapping took just over 10 minutes on my computer. Notice how the bowtie2 states that over 85% of reads align to the database- this is an important number to consider, as all reads that do not align to the database will be invisible to inStrain. For human microbiome samples 85% is pretty good, but performing de novo genome assembly and including sample-specific genomes would undoubtedly raise this number.

Running inStrain profile

Next we'll profile the .sam file created above with inStrain. To do this we'll need the *scaffold-to-bin file*, *genes file*, and *fasta file* for the database that we created in the first step. If you downloaded them you can decompress them with the commands

```
$ gzip -d UHGG_reps.fasta.gz

$ gzip -d UHGG_reps.genes.fna.gz
```

**When running inStrain on a big database like we have here it's critical to add the flag `--database mode`. This flag does some quick calculations to figure out which genomes are probably not present, and stops working on them right away. This leads to dramatic reductions in RAM usage and computational time.

The inStrain profile command we'll use now is

```
$ inStrain profile UHGG_reps.fasta-vs-N5_216_039G1.sam /groups/banfield/projects/
↪human/data8/ExternalData/UHGG/UHGG_reps.fasta -o UHGG_reps.fasta-vs-N5_216_039G1.IS␣
↪-p 10 -g /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.genes.
↪fna -s /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.stb --
↪database_mode
```

This took just over an hour to run on my computer. We have now successfully generated an inStrain profile! For help interpreting the output files, see *Expected output*. To link the genomes in the UHGG database with their taxonomy, use the file `genomes-nr_metadata.tsv` which we downloaded above and is part of the overall download as well. To subset to just the *species representative genomes* (SRGs) that make up this database, subset this table to only include rows where the column "Genome" is equal to the column "Species_rep".

Running inStrain compare

`InStrain compare` compare genomes that have been profiled by multiple different metagenomic mappings. To compare genomes in the sample we just profiled above, we need to generate another *bam file* of reads from another sample to the **same** .fasta file. For example, something like the command (based on reads from this fecal sample from the same premature infant):

```
bowtie2 -p 10 -x /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.
↪fasta.bt2 -1 /groups/banfield/projects/human/data8/raw.d/NIH5/reads/HR/N5_216_046G1_
↪T0140F_S50_L002.HR.R1.fastq.gz -2 /groups/banfield/projects/human/data8/raw.d/NIH5/
↪reads/HR/N5_216_046G1_T0140F_S50_L002.HR.R2.fastq.gz > UHGG_reps.fasta-vs-N5_216_
↪046G1.sam

$ inStrain profile UHGG_reps.fasta-vs-N5_216_046G1.sam /groups/banfield/projects/
↪human/data8/ExternalData/UHGG/UHGG_reps.fasta -o UHGG_reps.fasta-vs-N5_216_046G1.IS␣
↪-p 10 -g /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.genes.
↪fna -s /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.stb --
↪database_mode
```

Now that we have two inStrain profile objects based on reads mapped to the same .fasta file, we can compare all detected genomes using the following command:

```
inStrain compare -i UHGG_reps.fasta-vs-N5_216_039G1.IS/ UHGG_reps.fasta-vs-N5_216_
↪046G1.IS/ -s /groups/banfield/projects/human/data8/ExternalData/UHGG/UHGG_reps.stb -
↪p 6 -o UHGG_reps.fasta.IS.COMPARE --database_mode
```

(continues on next page)

(continued from previous page)

Success! As before, for help interpreting this output see *Expected output*.

1.4.6 Tutorial #3) Merging custom genomes with an existing genome database.

Using a combination of sample-specific genomes for accuracy and public genome databases for comprehensiveness can provide the best of both worlds. The steps are as follows:

1) Establish a set of data-set specific genomes through de novo genome assembly and binning. This could be done using a tool such as anvi'o, for example.

2) Download an entire database of individual genomes. See the top of Tutorial #2 for instructions on downloading UHGG.

3) Dereplicate both sets of genomes. The specific threshold you use for dereplication is important and some thoughts about choosing thresholds is available at *Important concepts*. A program that can be used for this purpose is dRep; just make sure you have dRep version 3 which is able to handle much larger genome sets than previous versions. An example command that could be used for this step is

   dRep dereplicate MergedGenomeSet -g FullListOfGenomes.txt –S_algorithm fastANI –multiround_primary_clustering –clusterAlg greedy -ms 10000 -pa 0.9 -sa 0.95 -nc 0.30 -cm larger -p 16

This command will result in a species-level dereplicated set of genomes that include both your custom genomes and the database genomes. More details on genome dereplication can be found here. To prioritize your custom genomes over the database genomes, use the flag *extra_weight_table* within dRep.

4) Create a genome database out of the genomes in the *dereplicated_genomes* folder produced in the step above. This can be done following the instructions at the top of Tutorial #2.

1.5 User Manual

1.5.1 Generating inStrain input

There are two main inputs to inStrain: a *fasta file* containing reference genome sequences, and a *bam file* containing reads mapped to these sequences. Additionally and optionally, by providing a genes *.fna* file inStrain can calculate gene-level metrics, and by providing a scaffold-to-bin file inStrain can calculate metrics on a genome level. Here we go over some considerations involved in generating these inputs.

Preparing the .fasta file

A *fasta file* contains the DNA sequences of the contigs that you map your reads to. Choosing what *fasta file* you will use (consensus / reference genomes) is important and will affect the interpretation of your *inStrain* results. Below we describe the three most common strategies.

Please note that the *fasta file* provided to inStrain must always be the same as, or a subset of, the *fasta file* used to create the *bam file* (i.e. the *fasta file* that reads were mapped to).

Using *de novo* assembled genomes (recommended)

This strategy involves assembling genomes from the metagenomic samples that you'd like to profile. This is the recommended workflow for running *inStrain*:

1. Assemble reads into contigs for each sample collected from the environment. Recommended software: IDBA_UD, MEGAHIT, metaSPADES.
2. Bin genomes out of each assembly using differential coverage binning. Recommended software: Bowtie2 (for mapping), MetaBAT, CONCOCT, DasTOOL (for binning).
3. Dereplicate the entire set of genomes that you would like to profile (all genomes from all environments) at 97-99% identity, and filter out low quality genomes. Recommended software: dRep, checkM.
4. Create a *scaffold-to-bin file* from the genome set. Recommended software: parse_stb.py
5. Create a bowtie2 index of the representative genomes from this dereplicated set and map reads to this set from each sample. Recommended software: Bowtie2
6. Profile the resulting mapping *.bam* files using inStrain to calculate genome-level microdiveristy metrics for each originally binned genome.

An important aspect of this workflow is to **map to many genomes at once**. Mapping to just one genome at a time is highly discouraged, because this encourages *mismapped reads* from other genomes to be recruited by this genome. By including many (dereplicated) genomes in your bowtie2 index, you will be able to far more accurately filter out *mismapped reads* and reduce false positive SNPs. See *Important concepts* for more info.

For instructions on merging your genomes with a public database, see Tutorial #3 of *Tutorial*.

Using a single genome .fasta file

If your .fasta file is a single genome, the main consideration is that it should be a good representative genome for some organism in your sample. Ideally, it was assembled directly from that sample, isolated from that sample, or you have some other evidence that this genome is highly representative of a species in that sample. Regardless, you should check your *inStrain plot* output and *scaffold_info.tsv* output file to be sure that your inStrain run had decent coverage and breadth of coverage of the genome that you use before attempting to interpret the results.

Remember, your .fasta file can be a subset of the .fasta file that was used to create the .bam file. You can create a .bam with all dereplicated genomes from your environment, but then just pass a .fasta file for only the genomes of particular interest. This approach is recommended as opposed to creating a *bam file* for just each genome, as it reduces *mismapped reads*

Using a metagenomic assembly

You can also pass inStrain an entire metagenomic assembly from a sample, including both binned and unbinned contigs. In this case, the output inStrain profile will include population information for each contig in the set. To break it down by microbial genome / species, you can include a *scaffold-to-bin file* to generate results by genome.

Preparing the .bam file

InStrain is designed primarily for paired-end Illumina read sequencing, though un-paired reads can also be used by adjusting the run-time parameters. We recommend using the program Bowtie2 to map your reads to your genome.

Bowtie2 default parameters are what we use for mapping, but it may be worth playing around with them to see how different settings perform on your data. It is important to note that the `-X` flag (capital X) is the expected insert length and is by default `500`. In many cases (e.g., 2x250 bp or simply datasets with longer inserts) it may be worthwhile to increase this value up to `-X 1000` for passing to Bowtie2. By default, if a read maps equally well to multiple genomes, Bowtie2 will pick one of the positions randomly and give the read a MAPQ score of 1. Thus, if you'd like to remove *multi-mapped reads*, you can set the minimum mapQ score to 2.

Other mapping software can also be used to generate .bam files for inStrain. However, some software (e.g. BBmap and SNAP) use the fasta file scaffold descriptions when generating the .bam files, which causes problems for inStrain. If using mapping software that does this, include the flag `--use_full_fasta_header` to let inStrain account for this.

Note: If the reads that you'd like to run with inStrain are not working, please post an issue on GitHub. We're happy to upgrade inStrain to work with new mapping software and/or reads from different technologies.

Preparing the genes file

You can run prodigal on your *fasta file* to generate an .fna file with the gene-level information. This .fna file can then be provided to inStrain profile to get gene-level characterizations.

Example:

```
$ prodigal -i assembly.fasta -d genes.fna -a genes.faa
```

Preparing a scaffold-to-bin file

After running `inStrain profile`, most results are presented on a scaffold-by-scaffold basis. There are a number of ways of telling *inStrain* which scaffold belongs to which genome, so that results can be analyzed on a genome-by-gene level as well.

1. Individual .fasta files. As recommended above, if you want to run *inStrain* on multiple genomes in the same sample, you should first concatenate all of the individual genomes into a single *.fasta* file and map to that. To view the results of the individual genomes used to create the concatenated .fasta file, you can pass a list of the individual .fasta files the `-s` argument.
2. Scaffold-to-bin file. This is a text file consists of two columns, with one column listing the scaffold name, and the second column listing the genome bin name. Columns should be separated by tabs. The script parse_stb.py can help you create a scaffold-to-bin file from a list of individual .fasta files, or to split a concatenated .fasta file into individual genomes. The script comes packaged with the program dRep, and can be installed with the command `pip install drep`.
3. Nothing. If all of your scaffolds belong to the same genome, by running `inStrain profile` without any *-s* options it will summarize the results of all scaffolds together as if they all belong to the same genome.

1.5.2 Description of inStrain modules and arguments

The functionality of inStrain is broken up into modules. To see a list of available modules, check the help:

```
$ inStrain -h

                ...::: inStrain v1.3.2 :::...

  Matt Olm and Alex Crits-Christoph. MIT License. Banfield Lab, UC Berkeley. 2019

  Choose one of the operations below for more detailed help. See https://instrain.
↪readthedocs.io for documentation.
  Example: inStrain profile -h

  Workflows:
```

(continues on next page)

(continued from previous page)

```
    profile         -> Create an inStrain profile (microdiversity analysis) from a␣
↪mapping.
    compare         -> Compare multiple inStrain profiles (popANI, coverage_overlap,␣
↪etc.)

  Single operations:
    profile_genes   -> Calculate gene-level metrics on an inStrain profile␣
↪[DEPRECATED; USE profile INSTEAD]
    genome_wide     -> Calculate genome-level metrics on an inStrain profile
    quick_profile   -> Quickly calculate coverage and breadth of a mapping using␣
↪coverM
    filter_reads    -> Commands related to filtering reads from .bam files
    plot            -> Make figures from the results of "profile" or "compare"
    other           -> Other miscellaneous operations
```

profile

Module description

The heart of inStrain. The input is a *fasta file* and a *bam file*, and the output is an *IS_profile*. The functionality of `inStrain profile` is broken into several steps.

First, all reads in the .bam file are filtered to only keep those that map with sufficient quality. All non-paired reads will be filtered out by default, and an additional set of filters are applied to each read pair (not the individual reads). Command line parameters can be adjusted to change the specifics, but in general:

- Pairs must be mapped in the proper orientation with an expected insert size. The minimum insert distance can be set with a command line parameter. The maximum insert distance is a multiple of the median insert distance. So if pairs have a median insert size of 500bp, by default all pairs with insert sizes over 1500bp will be excluded. For the max insert cutoff, the median_insert for all scaffolds is used.
- Pairs must have a minimum mapQ score. MapQ scores are confusing and how they're calculated varies based on the mapping algorithm being used, but are meant to represent both the number of mismatches in the mapping and how unique that mapping is. With bowtie2, if the read maps equally well to two positions on the genome (*multi-mapped read*), its mapQ score will be set to 2. The read in the pair with the higher mapQ is used for the pair.
- Pairs must be above some minimum nucleotide identity (ANI) value. For example if reads in a pair are 100bp each, and each read has a single mismatch, the ANI of that pair would be 0.99

Next, using only read pairs that pass filters, a number of *microdiversity* metrics are calculated on a scaffold-by-scaffold basis. This includes:

- Calculate the coverage at each position along the scaffold
- Calculate the *nucleotide diversity* at each position along the scaffold in which the coverage is greater than the min_cov argument.
- Identify *SNSs* and *SNVs*. The criteria for being reported as a *divergent site* are 1) More than min_cov number of bases at that position, 2) More than min_freq percentage of reads that are a variant base, 3) The number of reads with the variant base is more than the *null model* for that coverage.
- Calculate *linkage* between *divergent sites* on the same read pair. For each pair harboring a *divergent site*, calculate the linkage of that site with other *divergent sites* within that same pair. This is only done for pairs of *divergent sites* that are both on at least MIN_SNP reads

- Calculate scaffold-level properties. These include things like the overall coverage, breadth of coverage, average nucleotide identity (ANI) between the reads and the reference genome, and the expected breadth of coverage based on that true coverage.

Finally, this information is stored as an *IS_profile* object. This includes the locations of *divergent sites*, the number of read pairs that passed filters (and other information) for each scaffold, the linkage between SNV pairs, ect.

Module parameters

To see the command-line arguments for inStrain profile, check the help:

```
$ inStrain profile -h
usage: inStrain profile [-o OUTPUT] [--use_full_fasta_header] [-p PROCESSES]
                        [-d] [-h] [--version] [-l MIN_READ_ANI]
                        [--min_mapq MIN_MAPQ]
                        [--max_insert_relative MAX_INSERT_RELATIVE]
                        [--min_insert MIN_INSERT]
                        [--pairing_filter {paired_only,all_reads,non_discordant}]
                        [--priority_reads PRIORITY_READS]
                        [--detailed_mapping_info] [-c MIN_COV] [-f MIN_FREQ]
                        [-fdr FDR] [-g GENE_FILE] [-s [STB [STB ...]]]
                        [--mm_level] [--skip_mm_profiling] [--database_mode]
                        [--min_scaffold_reads MIN_SCAFFOLD_READS]
                        [--min_genome_coverage MIN_GENOME_COVERAGE]
                        [--min_snp MIN_SNP] [--store_everything]
                        [--scaffolds_to_profile SCAFFOLDS_TO_PROFILE]
                        [--rarefied_coverage RAREFIED_COVERAGE]
                        [--window_length WINDOW_LENGTH] [--skip_genome_wide]
                        [--skip_plot_generation]
                        bam fasta

REQUIRED:
  bam                   Sorted .bam file
  fasta                 Fasta file the bam is mapped to

I/O PARAMETERS:
  -o OUTPUT, --output OUTPUT
                        Output prefix (default: inStrain)
  --use_full_fasta_header
                        Instead of using the fasta ID (space in header before
                        space), use the full header. Needed for some mapping
                        tools (including bbMap) (default: False)

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
  --version             show program's version number and exit

READ FILTERING OPTIONS:
  -l MIN_READ_ANI, --min_read_ani MIN_READ_ANI
                        Minimum percent identity of read pairs to consensus to
                        use the reads. Must be >, not >= (default: 0.95)
  --min_mapq MIN_MAPQ   Minimum mapq score of EITHER read in a pair to use
                        that pair. Must be >, not >= (default: -1)
  --max_insert_relative MAX_INSERT_RELATIVE
```

(continues on next page)

(continued from previous page)

```
                        Multiplier to determine maximum insert size between
                        two reads - default is to use 3x median insert size.
                        Must be >, not >= (default: 3)
  --min_insert MIN_INSERT
                        Minimum insert size between two reads - default is 50
                        bp. If two reads are 50bp each and overlap completely,
                        their insert will be 50. Must be >, not >= (default:
                        50)
  --pairing_filter {paired_only,all_reads,non_discordant}
                        How should paired reads be handled?
                        paired_only = Only paired reads are retained
                        non_discordant = Keep all paired reads and singleton reads␣
↪that map to a single scaffold
                        all_reads = Keep all reads regardless of pairing status (NOT␣
↪RECOMMENDED; See documentation for deatils)
                         (default: paired_only)
  --priority_reads PRIORITY_READS
                        The location of a list of reads that should be
                        retained regardless of pairing status (for example
                        long reads or merged reads). This can be a .fastq file
                        or text file with list of read names (will assume file
                        is compressed if ends in .gz (default: None)

READ OUTPUT OPTIONS:
  --detailed_mapping_info
                        Make a detailed read report indicating deatils about
                        each individual mapped read (default: False)

VARIANT CALLING OPTIONS:
  -c MIN_COV, --min_cov MIN_COV
                        Minimum coverage to call an variant (default: 5)
  -f MIN_FREQ, --min_freq MIN_FREQ
                        Minimum SNP frequency to confirm a SNV (both this AND
                        the FDR snp count cutoff must be true to call a SNP).
                        (default: 0.05)
  -fdr FDR, --fdr FDR   SNP false discovery rate- based on simulation data
                        with a 0.1 percent error rate (Q30) (default: 1e-06)

GENE PROFILING OPTIONS:
  -g GENE_FILE, --gene_file GENE_FILE
                        Path to prodigal .fna genes file. If file ends in .gb
                        or .gbk, will treat as a genbank file (EXPERIMENTAL;
                        the name of the gene must be in the gene qualifier)
                        (default: None)

GENOME WIDE OPTIONS:
  -s [STB [STB ...]], --stb [STB [STB ...]]
                        Scaffold to bin. This can be a file with each line
                        listing a scaffold and a bin name, tab-seperated. This
                        can also be a space-seperated list of .fasta files,
                        with one genome per .fasta file. If nothing is
                        provided, all scaffolds will be treated as belonging
                        to the same genome (default: [])

READ ANI OPTIONS:
  --mm_level            Create output files on the mm level (see documentation
                        for info) (default: False)
```

(continues on next page)

(continued from previous page)

```
  --skip_mm_profiling   Dont perform analysis on an mm level; saves RAM and
                        time; impacts plots and raw_data (default: False)

PROFILE OPTIONS:
  --database_mode       Set a number of parameters to values appropriate for
                        mapping to a large fasta file. Will set:
                        --min_read_ani 0.92 --skip_mm_profiling
                        --min_genome_coverage 1 (default: False)
  --min_scaffold_reads MIN_SCAFFOLD_READS
                        Minimum number of reads mapping to a scaffold to
                        proceed with profiling it (default: 1)
  --min_genome_coverage MIN_GENOME_COVERAGE
                        Minimum number of reads mapping to a genome to proceed
                        with profiling it. MUST profile .stb if this is set
                        (default: 0)
  --min_snp MIN_SNP     Absolute minimum number of reads connecting two SNPs
                        to calculate LD between them. (default: 20)
  --store_everything    Store intermediate dictionaries in the pickle file;
                        will result in significantly more RAM and disk usage
                        (default: False)
  --scaffolds_to_profile SCAFFOLDS_TO_PROFILE
                        Path to a file containing a list of scaffolds to
                        profile- if provided will ONLY profile those scaffolds
                        (default: None)
  --rarefied_coverage RAREFIED_COVERAGE
                        When calculating nucleotide diversity, also calculate
                        a rarefied version with this much coverage (default:
                        50)
  --window_length WINDOW_LENGTH
                        Break scaffolds into windows of this length when
                        profiling (default: 10000)

OTHER  OPTIONS:
  --skip_genome_wide    Do not generate tables that consider groups of
                        scaffolds belonging to genomes (default: False)
  --skip_plot_generation
                        Do not make plots (default: False)
```

compare

Module description

Compare provides the ability to compare multiple *inStrain profiles* (created by running `inStrain profile`).

Note: *inStrain* can only compare inStrain profiles that have been mapped to the same .fasta file

`inStrain compare` does pairwise comparisons between each input *inStrain profile*. For each pair, a series of steps are undertaken.

1. All positions in which both IS_profile objects have at least *min_cov* coverage (5x by default) are identified. This information can be stored in the output by using the flag *–store_coverage_overlap*, but due to it's size, it's not stored by default
2. Each position identified in step 1 is compared to calculate both *conANI* and *popANI*. The way that it compares

positions is by testing whether the consensus base in sample 1 is detected at all in sample 2 and vice-versa. Detection of an allele in a sample is based on that allele being above the set *-min_freq* and *-fdr*. All detected differences between each pair of samples can be reported if the flag *–store_mismatch_locations* is set.

3. The coverage overlap and the average nucleotide identity for each scaffold is reported. For details on how this is done, see *Expected output*
4. **New in v1.6** Tables that list the coverage and base-frequencies of each SNV in all samples can be generated using the *–bams* parameter within inStrain compare. For each inStrain profile provided with the *-i* parameter, the corresponding bam file must be provided with the *–bams* parameter. The same read filtering parameters used in the original profile command will be used when running this analysis. See section *SNV POOLING OPTIONS:* in the help below for full information about this option, and see *Expected output* for the tables it creates.

Module parameters

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

```
$ inStrain compare -h
usage: inStrain compare -i [INPUT [INPUT ...]] [-o OUTPUT] [-p PROCESSES] [-d]
                    [-h] [--version] [-s [STB [STB ...]]] [-c MIN_COV]
                    [-f MIN_FREQ] [-fdr FDR] [--database_mode]
                    [--breadth BREADTH] [-sc SCAFFOLDS] [--genome GENOME]
                    [--store_coverage_overlap]
                    [--store_mismatch_locations]
                    [--include_self_comparisons] [--skip_plot_generation]
                    [--group_length GROUP_LENGTH] [--force_compress]
                    [-ani ANI_THRESHOLD] [-cov COVERAGE_TRESHOLD]
                    [--clusterAlg {centroid,weighted,ward,single,complete,average,
↪median}]
                    [-bams [BAMS [BAMS ...]]] [--skip_popANI]

REQUIRED:
  -i [INPUT [INPUT ...]], --input [INPUT [INPUT ...]]
                        A list of inStrain objects, all mapped to the same
                        .fasta file (default: None)
  -o OUTPUT, --output OUTPUT
                        Output prefix (default: instrainComparer)

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
  --version             show program's version number and exit

GENOME WIDE OPTIONS:
  -s [STB [STB ...]], --stb [STB [STB ...]]
                        Scaffold to bin. This can be a file with each line
                        listing a scaffold and a bin name, tab-seperated. This
                        can also be a space-seperated list of .fasta files,
                        with one genome per .fasta file. If nothing is
                        provided, all scaffolds will be treated as belonging
                        to the same genome (default: [])

VARIANT CALLING OPTIONS:
  -c MIN_COV, --min_cov MIN_COV
                        Minimum coverage to call an variant (default: 5)
```

(continues on next page)

(continued from previous page)

```
  -f MIN_FREQ, --min_freq MIN_FREQ
                        Minimum SNP frequency to confirm a SNV (both this AND
                        the FDR snp count cutoff must be true to call a SNP).
                        (default: 0.05)
  -fdr FDR, --fdr FDR   SNP false discovery rate- based on simulation data
                        with a 0.1 percent error rate (Q30) (default: 1e-06)

DATABASE MODE PARAMETERS:
  --database_mode       Using the parameters below, automatically determine
                        which genomes are present in each Profile and only
                        compare scaffolds from those genomes. All profiles
                        must have run Profile with the same .stb (default:
                        False)
  --breadth BREADTH     Minimum breadth_minCov required to count a genome
                        present (default: 0.5)

OTHER OPTIONS:
  -sc SCAFFOLDS, --scaffolds SCAFFOLDS
                        Location to a list of scaffolds to compare. You can
                        also make this a .fasta file and it will load the
                        scaffold names (default: None)
  --genome GENOME       Run scaffolds belonging to this single genome only.
                        Must provide an .stb file (default: None)
  --store_coverage_overlap
                        Also store coverage overlap on an mm level (default:
                        False)
  --store_mismatch_locations
                        Store the locations of SNPs (default: False)
  --include_self_comparisons
                        Also compare IS profiles against themself (default:
                        False)
  --skip_plot_generation
                        Dont create plots at the end of the run. (default:
                        False)
  --group_length GROUP_LENGTH
                        How many bp to compare simultaneously (higher will use
                        more RAM and run more quickly) (default: 10000000)
  --force_compress      Force compression of all output files (default: False)

GENOME CLUSTERING OPTIONS:
  -ani ANI_THRESHOLD, --ani_threshold ANI_THRESHOLD
                        popANI threshold to cluster genomes at. Must provide
                        .stb file to do so (default: 0.99999)
  -cov COVERAGE_TRESHOLD, --coverage_treshold COVERAGE_TRESHOLD
                        Minimum percent_genome_compared for a genome
                        comparison to count; if below the popANI will be set
                        to 0. (default: 0.1)
  --clusterAlg {centroid,weighted,ward,single,complete,average,median}
                        Algorithm used to cluster genomes (passed to
                        scipy.cluster.hierarchy.linkage) (default: average)

SNV POOLING OPTIONS:
  -bams [BAMS [BAMS ...]], --bams [BAMS [BAMS ...]]
                        Location of .bam files used during inStrain profile
                        commands; needed to pull low-frequency SNVs. MUST BE
                        IN SAME ORDER AS THE INPUT FILES (default: None)
  --skip_popANI         Only run SNV Pooling; skip other compare operations
```

(continues on next page)

(continued from previous page)

```
                        (default: False)
```

Other modules

The other modules are not commonly used, and mainly provide auxiliary functions or allow you to run certain steps of `profile` after the fact. It is recommended to provide a genes file and/or a scaffold-to-bin file during `inStrain profile` rather than using `profile_genes` or `genome_wide`, as it is more computationally efficient to do things that way.

parse_annotations

This is a new module (released in inStrain version 1.7) that provides a straightforward way to annotate genes for functional analysis with inStrain. It's inputs are 1 or more *inStrain profile* objects (a set of genes must of been provided when running profile) and a gene annotation table. The format of the gene annotation MUST be a .csv file that looks like the following:

Table 2: AnnotationTable.csv

gene	anno
AF21-42.Scaf7_79	K14374
AF21-42.Scaf7_79	K12633
AF21-42.Scaf7_79	K16034
AF21-42.Scaf7_79	K12705
AF21-42.Scaf7_79	K15467
AF21-42.Scaf7_79	K20592
AF21-42.Scaf7_79	K15958
AF21-42.Scaf7_79	K22590
AF21-42.Scaf7_80	K02835
AF21-42.Scaf7_81	K03431

The top line MUST read *gene,anno* exactly, and all other lines must have the format *gene*, comma (,), *annotation*. The *gene* column must match the names of the genes provided to inStrain profile, and the second column can be whatever you want it to be. It is fine if a gene as multiple annotations, just make that gene have multiple lines in this file.

For tips on running the annotations themselves, see the section *Gene Annotation* below. Also see the section *Running inStrain parse_annotations* in *Tutorial* for more info.

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

```
$ inStrain parse_annotations -h
usage: inStrain parse_annotations -i [INPUT [INPUT ...]] -a [ANNOTATIONS [ANNOTATIONS␣
↪...]] [-o OUTPUT] [-p PROCESSES] [-d] [-h] [--version] [-b MIN_GENOME_BREADTH]
                                  [-g MIN_GENE_BREADTH] [--store_rawdata]

REQUIRED:
  -i [INPUT [INPUT ...]], --input [INPUT [INPUT ...]]
                        A list of inStrain objects, all mapped to the same .fasta␣
↪file (default: None)
  -a [ANNOTATIONS [ANNOTATIONS ...]], --annotations [ANNOTATIONS [ANNOTATIONS ...]]
                        A table or set of tables with gene annotations.
                        Must be in specific format; see inStrain website for details␣
↪(default: None)
```

(continues on next page)

(continued from previous page)

```
  -o OUTPUT, --output OUTPUT
                        Output prefix (default: annotation_output)

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
  --version             show program's version number and exit

OTHER OPTIONS:
  -b MIN_GENOME_BREADTH, --min_genome_breadth MIN_GENOME_BREADTH
                        Only annotate genomes on genomes with at least this genome␣
↪breadth. Requires having genomes called. Set to 0 to include all genes. (default: 0.
↪5)
  -g MIN_GENE_BREADTH, --min_gene_breadth MIN_GENE_BREADTH
                        Only annotate genes with at least this breadth. Set to 0 to␣
↪include all genes. (default: 0.8)
  --store_rawdata       Store the raw data dictionary (default: False)
```

quick_profile

This is a quirky module that is not really related to any of the others. It is used to quickly profile a *bam file* to pull out scaffolds from genomes that are at a sufficient breadth. To use it you must provide a *.bam* file, the *.fasta* file that you mapped to to generate the *.bam* file, and a *scaffold to bin* file (see above section for details). On the backend this module is really just calling the program coverM

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

```
$ inStrain quick_profile -h
usage: inStrain quick_profile [-p PROCESSES] [-d] [-h] [--version]
                              [-s [STB [STB ...]]] [-o OUTPUT]
                              [--breadth_cutoff BREADTH_CUTOFF]
                              [--stringent_breadth_cutoff STRINGENT_BREADTH_CUTOFF]
                              bam fasta

REQUIRED:
  bam                   Sorted .bam file
  fasta                 Fasta file the bam is mapped to

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
  --version             show program's version number and exit

OTHER OPTIONS:
  -s [STB [STB ...]], --stb [STB [STB ...]]
                        Scaffold to bin. This can be a file with each line
                        listing a scaffold and a bin name, tab-seperated. This
                        can also be a space-seperated list of .fasta files,
                        with one genome per .fasta file. If nothing is
                        provided, all scaffolds will be treated as belonging
                        to the same genome (default: [])
```

(continues on next page)

(continued from previous page)

```
  -o OUTPUT, --output OUTPUT
                        Output prefix (default: QuickProfile)
  --breadth_cutoff BREADTH_CUTOFF
                        Minimum genome breadth to pull scaffolds (default:
                        0.5)
  --stringent_breadth_cutoff STRINGENT_BREADTH_CUTOFF
                        Minimum breadth to let scaffold into coverm raw
                        results (done with greater than; NOT greater than or
                        equal to) (default: 0.0)
```

plot

This module produces plots based on the results of *inStrain profile* and *inStrain compare*. In both cases, before plots can be made, *inStrain genome_wide* must be run on the output folder first. In order to make plots 8 and 9, *inStrain profile_genes* must be run first as well.

The recommended way of running this module is with the default *-pl a*. It will just try and make all of the plots that it can, and will tell you about any plots that it fails to make.

See *Expected output* for an example of the plots it can make.

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

```
$ inStrain plot -h
usage: inStrain plot -i IS [-pl [PLOTS [PLOTS ...]]] [-p PROCESSES] [-d] [-h]

REQUIRED:
  -i IS, --IS IS        an inStrain profile object (default: None)
  -pl [PLOTS [PLOTS ...]], --plots [PLOTS [PLOTS ...]]
                        Plots. Input 'all' or 'a' to plot all
                        1) Coverage and breadth vs. read mismatches
                        2) Genome-wide microdiversity metrics
                        3) Read-level ANI distribution
                        4) Major allele frequencies
                        5) Linkage decay
                        6) Read filtering plots
                        7) Scaffold inspection plot (large)
                        8) Linkage with SNP type (GENES REQUIRED)
                        9) Gene histograms (GENES REQUIRED)
                        10) Compare dendrograms (RUN ON COMPARE; NOT PROFILE)
                         (default: a)

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
```

other

This module holds odds and ends functionalities. As of version 1.4, all it can do is convert old *IS_profile* objects (>v0.3.0) to newer versions (v0.8.0) and create runtime summaries of complete inStrain runs. As the code base around *inStrain* matures, we expect more functionalities to be included here.

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

```
$  inStrain other -h
usage: inStrain other [-p PROCESSES] [-d] [-h] [--version] [--old_IS OLD_IS]
                      [--run_statistics RUN_STATISTICS]

SYSTEM PARAMETERS:
  -p PROCESSES, --processes PROCESSES
                        Number of processes to use (default: 6)
  -d, --debug           Make extra debugging output (default: False)
  -h, --help            show this help message and exit
  --version             show program's version number and exit

OTHER OPTIONS:
  --old_IS OLD_IS       Convert an old inStrain version object to the newer
                        version. (default: None)
  --run_statistics RUN_STATISTICS
                        Generate runtime reports for an inStrain run.
                        (default: None)
```

1.5.3 Other related operations

The goal of this section is to describe how to perform other operations that are commonly part of an inStrain-based workflow.

Gene Annotation

Below are some potential ways of annotating genes for follow-up inStrain analysis. The input to all operations is an amino acid fasta file (*.faa*), which should match the *.fna* file you passed to inStrain (see user_manual#preparing-the-genes-file for an example command)

If you have some other annotation you like to use, please add to to this list by submitting a pull request on GitHub! (https://github.com/MrOlm/inStrain/blob/master/docs/user_manual.rst)

KEGG Orthologies (KOs)

KOs can be annotated using KofamScan / KofamKOALA (https://www.genome.jp/tools/kofamkoala/)

```
# Download the database and executables
wget https://www.genome.jp/ftp/tools/kofam_scan/kofam_scan-1.3.0.tar.gz
wget https://www.genome.jp/ftp/db/kofam/ko_list.gz
wget https://www.genome.jp/ftp/db/kofam/profiles.tar.gz

# Unzip and untar
gzip -d ko_list.gz
tar xf profiles.tar.gz
tar xf kofam_scan-1.3.0.tar.gz

# Run kofamscan
exec_annotation -p profiles -k ko_list --cpu 10 --tmp-dir ./tmp -o genes.faa.
↪kofamscan genes.faa
```

The following python code parses the resulting table

```
import pandas as pd
from collections import defaultdict

def parse_kofamscan(floc):
    """
    v1.1: 5/1/2023
    - update "KO_definition" to be 5

    v1.0: 1/6/2023

    Parse kofamscan results. Only save results where KO score > threshold

    Returns:
        Adb: DataFrame with KOfam results
    """
    table = defaultdict(list)

    with open(floc, 'r') as o:

        # This blockis for RAM efficiency
        while True:
            line = o.readline()
            if not line:
                break

            line = line.strip()
            if line[0] == '#':
                continue

            lw = line.split()
            if lw[0] == '*':
                del lw[0]

            if lw[2] == '-':
                lw[2] = 0

            try:
                if float(lw[3]) >= float(lw[2]):
                    g = lw[0]
                    k = lw[1]

                    table['gene'].append(g)
                    table['KO'].append(k)
                    table['thrshld'].append(float(lw[2]))
                    table['score'].append(float(lw[3]))
                    table['e_value'].append(float(lw[4]))
                    table['KO_definition'].append(' '.join(lw[5:]))
            except:
                print(line)
                assert False
    o.close()

    Adb = pd.DataFrame(table)
    return Adb

floc = "genes.faa.kofamscan genes.faa"
Adb = parse_kofamscan(floc)
```

Where Adb is a pandas DataFrame that looks like:

Table 3: Adb

gene	KO	thrshld	score	e_value	KO_definition
AP010889.1_1	K02313	130.13	593.2	1.200000e-178	chromosomal replication initiator protein
AP010889.1_2	K02338	52.73	345.9	1.300000e-103	DNA polymerase III subunit beta [EC:2.7.7.7]
AP010889.1_2	K22359	0.00	12.5	3.000000e-02	alkene monooxygenase gamma subunit [EC:1.14.13…
AP010889.1_3	K03629	115.43	397.5	1.600000e-119	DNA replication and repair protein RecF
AP010889.1_5	K02470	946.10	986.6	1.500000e-297	DNA gyrase subunit B [EC:5.6.2.2]

Carbohydrate-Active enZYmes (CAZymes)

CAZymes can be profiled using the HMMs provided by dbCAN, which are based on CAZyDB (http://www.cazy.org/)

```
# Download the HMMs and executables
wget https://bcb.unl.edu/dbCAN2/download/Databases/V11/dbCAN-HMMdb-V11.txt
wget https://bcb.unl.edu/dbCAN2/download/Databases/V11/hmmscan-parser.sh

# Prepare HMMs
hmmpress dbCAN-HMMdb-V11.txt

# Run (based on readme here - https://bcb.unl.edu/dbCAN2/download/Databases/V11/
↪readme.txt)
hmmscan --domtblout genes.faa_vs_dbCAN_v11.dm dbCAN-HMMdb-V11.txt genes.faa > /dev/
↪null ; sh /hmmscan-parser.sh genes.faa_vs_dbCAN_v11.dm > genes.faa_vs_dbCAN_v11.dm.
↪ps ; cat genes.faa_vs_dbCAN_v11.dm.ps | awk '$5<1e-15&&$10>0.35' > genes.faa_vs_
↪dbCAN_v11.dm.ps.stringent
```

The following python code parses the resulting table

```
import pandas as pd
from collections import defaultdict

def parse_dbcan(floc):
    """
    v1.0 - 1/6/2023

    Parse dbcan2 results

    Returns:
        Cdb: DataFrame with dbCAN2 results
    """

    h = ['Family_HMM', 'HMM_length', 'gene', 'Query_length', 'E-value', 'HMM_start',
↪'HMM_end', 'Query_start', 'Query_end', 'Coverage']
    Zdb = pd.read_csv(floc, sep='\t', names=h)

    # Parse names
    def get_type(f):
```

(continues on next page)

(continued from previous page)

```
        for start in ['PL', 'AA', 'GH', 'CBM', 'GT', 'CE']:
            if f.startswith(start):
                return start
        if f in ['dockerin', 'SLH', 'cohesin']:
            return 'cellulosome'
        print(f)
        assert False

    def get_family(f):
        for start in ['PL', 'AA', 'GH', 'CBM', 'GT', 'CE']:
            if f.startswith(start):
                if f == 'CBM35inCE17':
                    return 35
                try:
                    return int(f.replace(start, '').split('_')[0])
                except:
                    print(f)
                    assert False
        if f in ['dockerin', 'SLH', 'cohesin']:
            return f
        print(f)
        assert False

    def get_subfamily(f):
        if f.startswith('GT2_'):
            if f == 'GT2_Glycos_transf_2':
                return 0
            else:
                return f.split('_')[-1]
        if '_' in f:
            try:
                return int(f.split('_')[1])
            except:
                print(f)
                assert False
        else:
            return 0

    t2n = {'GH':'glycoside hydrolases',
          'PL':'polysaccharide lyases',
          'GT':'glycosyltransferases',
          'CBM':'non-catalytic carbohydrate-binding modules',
          'AA':'auxiliary activities',
          'CE':'carbohydrate esterases',
          'cellulosome':'cellulosome'}

    ZIdb = Zdb[['Family_HMM']].drop_duplicates()
    ZIdb['raw_family'] = [x.split('.')[0] for x in ZIdb['Family_HMM']]
    ZIdb['class'] = [get_type(f) for f in ZIdb['raw_family']]
    ZIdb['class_name'] = ZIdb['class'].map(t2n)
    ZIdb['family'] = [get_family(f) for f in ZIdb['raw_family']]
    ZIdb['subfamily'] = [get_subfamily(f) for f in ZIdb['raw_family']]

    ZIdb['CAZyme'] = [f"{c}{f}_{s}" for c, f, s in zip(ZIdb['class'], ZIdb['family'],␣
↪ZIdb['subfamily'])]

    ZSdb = pd.merge(Zdb, ZIdb[['Family_HMM', 'class', 'family',
```

(continues on next page)

(continued from previous page)

```
        'subfamily', 'CAZyme']], on='Family_HMM', how='left')

    # Reorder
    ZSdb = ZSdb[[
        'gene',
        'CAZyme',
        'class',
        'family',
        'subfamily',
        'Family_HMM',
        'HMM_length',
        'Query_length',
        'E-value',
        'HMM_start',
        'HMM_end',
        'Query_start',
        'Query_end',
        'Coverage',
        ]]

    return ZSdb

floc = "/LAB_DATA/CURRENT/CURRENT_Metagenomics_PROJECTS/2022_Misame/gene_annotation/
↪dbCAN/DeltaI_NewBifido.faa_vs_dbCAN_v11.dm.ps"
Cdb = parse_dbcan(floc)
```

Where Cdb is a pandas DataFrame that looks like:

Table 4: Cdb

gene	CAZyme	class	family	subfamily	Family_HMM	HMM_length	Query_length	E-value	HMM_start	HMM_end	Query_start	Query_end	Coverage
AP010888.1_1503	GH51_0	GH	51	0	GH51.hmm	630	516	1.700000e-137	84	542	9	515	0.726984
AP010888.1_1107	GH13_18	GH	13	18	GH13_18.hmm	343	509	4.100000e-114	2	343	35	379	0.994169
AP010888.1_1113	GH13_30	GH	13	30	GH13_30.hmm	365	605	3.100000e-163	1	365	33	403	0.997260
AP010888.1_1175	GH77_0	GH	77	0	GH77.hmm	494	746	4.700000e-134	2	482	204	728	0.971660
AP010888.1_1320	GH31_0	GH	31	0	GH31.hmm	427	846	1.300000e-129	1	427	198	627	0.997658

Pfams (protein families)

Pfams can be profiled using the HMMs provided by Pfam

```
# Download the HMMs and executables
wget https://ftp.ebi.ac.uk/pub/databases/Pfam/current_release/Pfam-A.hmm.gz

# Install cath-resolve-hits on your system
# https://github.com/UCLOrengoGroup/cath-tools/releases/tag/v0.16.10

```

(continues on next page)

(continued from previous page)

```
# Prepare HMMs
gzip -d Pfam-A.hmm.gz

# Run with gathering cutoff
hmmsearch --cut_ga --cpu 6 --domtblout genes.faa_vs_Pfam.hmm --acc Pfam-A.hmm genes.
↪faa

# Resolve domain overlaps using cath-resolve-hits
cath-resolve-hits.ubuntu14.04 --input-format hmmer_domtblout genes.faa_vs_Pfam.hmm --
↪hits-text-to-file genes.faa_vs_Pfam.hmm.filtered.txt --quiet
```

The following python code parses the resulting table

```
def parse_Pfam(floc, aloc):
    """
    v1.0 - 4/20/2023

    Parse Pfam

    Returns:
        Hdb: DataFrame with Pfam results
    """

    PFdb = pd.read_csv(floc, header=1, delim_whitespace=True, names=['query-id',
↪'match-id', 'score', 'boundaries', 'resolved', 'cond-evalue', 'indp-evalue', 'junk
↪'], index_col=None)
    del PFdb['junk']

    # Compress
    Ofdb = PFdb.rename(columns={'query-id':'gene', 'match-id':'pFam'})
    Ofdb = Ofdb.sort_values('score').drop_duplicates(subset=['gene', 'pFam'], keep=
↪'last').sort_values('gene')
    #Ofdb = Ofdb[['gene', 'pFam']]
    Ofdb['pFam'] = Ofdb['pFam'].astype('category')

    # Add more annotation data
    PFDdb = pd.read_csv(aloc)

    TDdb = pd.merge(Ofdb, PFDdb[['NAME', 'DESC', 'ACC']].rename(columns={'NAME':'pFam
↪'}), on='pFam')
    TDdb = TDdb.drop_duplicates()

    return TDdb

def make_Pfam_info(LOCATION):
    table = defaultdict(list)
    a2c = {}
    grab_next = False
    with open(LOCATION) as f:
        for line in f.readlines():
            line = line.strip()
            if grab_next:
                assert line.split()[0] == 'ACC', line
                a2c[acc] = ' '.join(line.split()[1:])
                grab_next = False

            if line[:4] == 'NAME':
```

(continues on next page)

(continued from previous page)

```
                acc = line.split()[1]
                grab_next = True
    Ddb = pd.DataFrame(list(a2c.items()))
    Ddb.rename(columns={0:'NAME', 1:'ACC'}, inplace=True)

    a2c = {}
    grab_next = False
    with open(LOCATION) as f:
        for line in f.readlines():
            line = line.strip()
            if grab_next:
                if line.split()[0] == 'NC':
                    a2c[acc] = float(line.split()[1])
                    grab_next = False

            if line[:3] == 'ACC':
                acc = line.split()[1]
                grab_next = True
    Ndb = pd.DataFrame(list(a2c.items()))
    Ndb.rename(columns={0:'ACC', 1:'NC'}, inplace=True)

    a2c = {}
    grab_next = False
    with open(LOCATION) as f:
        for line in f.readlines():
            line = line.strip()
            if grab_next:
                assert line.split()[0] == 'DESC', line
                a2c[acc] = ' '.join(line.split()[1:])
                grab_next = False

            if line[:3] == 'ACC':
                acc = line.split()[1]
                grab_next = True
    DEdb = pd.DataFrame(list(a2c.items()))
    DEdb.rename(columns={0:'ACC', 1:'DESC'}, inplace=True)

    PFdb = pd.merge(Ddb, Ndb).merge(DEdb)

PIdb = make_Pfam_info("Pfam-A.hmm")
PIdb.to_csv('Pfam-A.info.csv', index=False)

floc = 'genes.faa_vs_Pfam.filtered.txt'
aloc = 'Pfam-A.info.csv'

Hdb = parse_Pfam(floc, aloc)
Hdb
```

Where Hdb is a pandas DataFrame that looks like:

Table 5: Hdb

gene	pFam	score	boundaries	resolved	cond-evalue	indp-evalue	DESC	ACC
COASSEMBLY_8000__NODE_10003_length_3966_cov_2.	ParA	293.8	40-278	40-278	9.100000e-88	5.500000e-85	NUBPL iron-transfer P-loop NTPase	PF10609.9
COASSEMBLY_8000__NODE_1115_length_48820_cov_22.	ParA	298.1	33-277	33-277	4.500000e-89	2.700000e-86	NUBPL iron-transfer P-loop NTPase	PF10609.9
COASSEMBLY_8000__NODE_1595_length_34257_cov_43.	ParA	313.2	41-286	41-286	1.100000e-93	6.400000e-91	NUBPL iron-transfer P-loop NTPase	PF10609.9
COASSEMBLY_8000__NODE_187_length_143579_cov_8.	ParA	300.3	92-342	92-342	9.600000e-90	5.800000e-87	NUBPL iron-transfer P-loop NTPase	PF10609.9
COASSEMBLY_8000__NODE_2172_length_24655_cov_4.	ParA	305.3	25-267	25-267	2.800000e-91	1.700000e-88	NUBPL iron-transfer P-loop NTPase	PF10609.9

Antibiotic Resistance Genes

There are many, many different ways of identifying antibiotic resistance genes. The method below is based on identifying homologs to know antibiotic resistance genes using the CARD database (https://card.mcmaster.ca/download)

```
# Download and unzip database
wget https://card.mcmaster.ca/download/0/broadstreet-v3.2.5.tar.bz2
tar -xvjf broadstreet-v3.2.5.tar.bz2

# Make a diamond database out of it
diamond makedb --in protein_fasta_protein_homolog_model.fasta -d protein_fasta_
↪protein_homolog_model.dmd --threads 6

# Run
diamond blastp -q genes.faa -d protein_fasta_protein_homolog_model.dmd -f 6 -e 0.0001␣
↪-k 1 -p 6 -o genes.faa_vs_CARD.dm
```

The following python code parses the resulting table

```
import pandas as pd
from collections import defaultdict

def parse_card(floc, jloc=None):
    """
    v1.0 - 1/6/2023

    Parse CARD

    Returns:
        Rdb: DataFrame with CARD results
    """
    h = ['gene', 'target', 'percentID', 'alignment_length', 'mm', 'gaps',
        'querry_start', 'querry_end', 'target_start', 'target_end', 'e-value', 'bit_
↪score',
```

(continues on next page)

(continued from previous page)

```
        'extra']
    db = pd.read_csv(floc, sep='\t', names=h)
    del db['extra']

    db['protein_seq_accession'] =  [t.split('|')[1] for t in db['target']]
    db['ARO'] =  [t.split('|')[2].split(':')[-1] for t in db['target']]
    db['CARD_short_name'] =  [t.split('|')[3].split(':')[-1] for t in db['target']]

    # Reorder
    header = ['gene', 'CARD_short_name', 'ARO', 'target']
    db = db[header + [x for x in list(db.columns) if x not in header]]

    if jloc is None:
        return db

    # Parse more
    import json
    j = json.load(open(jloc))

    aro2name = {}
    aro2categories = {}

    for n, m2t in j.items():
        if type(m2t) != type({}):
            continue

        if 'ARO_description' in m2t:
            aro2name[m2t['ARO_accession']] = m2t['ARO_description']

        if 'ARO_category' in m2t:
            cats = []
            for cat, c2t in m2t['ARO_category'].items():
                if 'category_aro_accession' in c2t:
                    cats.append(c2t['category_aro_accession'])
            aro2categories[m2t['ARO_accession']] = '|'.join(cats)


    db['ARO_description'] = db['ARO'].map(aro2name)
    db['ARO_category_accessions'] = db['ARO'].map(aro2categories)

    header = ['gene', 'CARD_short_name', 'ARO', 'ARO_description', 'ARO_category_
↪accessions', 'target']
    db = db[header + [x for x in list(db.columns) if x not in header]]

    return db

floc = "/genes.faa_vs_CARD.dm"
Rdb = parse_card(floc, jloc = "card.json")
```

Where Rdb is a pandas DataFrame that looks like:

Table 6: Rdb

gene	CARD_ID	short_name	ARO	ARO_description	category	target	percentID	alignment_length	mm	gaps	querry_start	querry_end	target_start	target_end	e-value	bit_score	protein_seq_accession
AP010889.1_[illegible]	[illegible]	macB	3000535	MacB is an ATP-binding cassette (ABC) transpor…	001000[illegible]	gb|AAV85982.1|ARO:3000535|macB	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	227	3	223	7.17000e-33	123.0	AAV85982.1
AP010889.1_[illegible]	[illegible]	lin	3004651	Listeria monocytogenes EGD-e lin gene for linc…	300022[illegible]	gb|AEO25219.1|ARO:3004651|lin	[illegible]	[illegible]	[illegible]	[illegible]	170	560	80	452	1.56000e-07	50.2	AEO25219.1
AP010889.1_[illegible]	[illegible]	patA	3000024	PatA is an ABC transporter of Streptococcus pn…	001000[illegible]	gb|AAK76137.1|ARO:3000024|patA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	344	560	1.56000e-14	70.9	AAK76137.1
AP010889.1_[illegible]	[illegible]	bcrA	3002987	bcrA is an ABC transporter found in Bacillus l…	001000[illegible]	gb|AAA99504.1|ARO:3002987|bcrA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	212	7.43000e-26	99.8	AAA99504.1
AP010889.1_[illegible]	[illegible]	abaF	3004573	Expression of abaF in E. coli resulted	001000[illegible]	gb|ABO11759.2|ARO:3004573|abaF	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	425	7.75000e-72	230.0	ABO11759.2

Human milk oligosaccharide (HMO) Utilization genes

This is pretty niche, but it’s something I (Matt Olm) am interested in. So here is how it can be done!

```
# Download the Supplemental Table S4 from here: https://data.mendeley.com/datasets/
↪gc4d9h4x67/2

wget https://data.mendeley.com/public-files/datasets/gc4d9h4x67/files/565528fe-585a-
↪4f71-bb84-9f76625a872b/file_downloaded -O humann2_HMO_annotation.csv

# Download the reference genome from here: https://www.ncbi.nlm.nih.gov/nuccore/
↪CP001095.1

Click “Send to:" -> “Coding Sequences” -> Format: “FASTA Protein” -> Rename to
↪"Bifidobacterium_longum_subsp_infantis_ATCC_15697.NCBI.faa"

# Pull the HMO genes using pullseq

pullseq -i Bifidobacterium_longum_subsp_infantis_ATCC_15697.NCBI.faa -n HMO_list >␣
↪Blon_HMO_genes.faa

# Search against them

diamond makedb --in Blon_HMO_genes.faa -d /LAB_DATA/DATABASES/HMO_ID/Blon_HMO_genes.
↪faa.dmd

diamond blastp -q genes.faa -d Blon_HMO_genes.faa.dmd.dmnd  -f 6 -e 0.0001 -k 1 -p 6 -
↪o genes.faa_vs_HMO.b6
```

The following python code parses the resulting table

```python
def parse_HMOs(floc, iloc):
    """
    v1.0 - 1/6/2023

    Parse HMOs

    Returns:
        Hdb: DataFrame with HMO results
    """

    Hdb = pd.read_csv(iloc, sep=';')
    Hdb['target'] = [x.replace('_cds_', '_prot_').replace('lcl.', 'lcl|').strip() for␣
↪x in Hdb['HMOgenes']]
    Hdb = Hdb[['target', 'Blon', 'Cluster']]

    h = ['gene', 'target', 'percentID', 'alignment_length', 'mm', 'gaps',
        'querry_start', 'querry_end', 'target_start', 'target_end', 'e-value', 'bit_
↪score',
        'extra']
    db = pd.read_csv(floc, sep='\t', names=h)
    del db['extra']

    # Filter a bit
    db = db[(db['percentID'] >= 50)]

    # Merge
```

(continues on next page)

(continued from previous page)

```
    Hdb = pd.merge(db, Hdb, how='left')

    # Re-order
    header = ['gene', 'Blon', 'Cluster', 'target']
    Hdb = Hdb[header + [x for x in list(Hdb.columns) if x not in header]]

    return Hdb

floc = '/LAB_DATA/CURRENT/CURRENT_Metagenomics_PROJECTS/2022_Misame/gene_annotation/
↪HMO/DeltaI_NewBifido.faa_vs_HMO.b6'
iloc = '/LAB_DATA/DATABASES/HMO_ID/humann2_HMO_annotation.csv'

Hdb = parse_HMOs(floc, iloc)
```

Where Hdb is a pandas DataFrame that looks like:

Table 7: Hdb

gene	Blon	Cluster	target	percentID	alignment_length	mm	gaps	querry_start	querry_end	target_start	target_end	e-value	bit_score
AP010889.1_103	Blon_0104	Urease	lcl|CP001095.1_prot_ACJ51233.1_197	99.8	433	1	0	1	433	1	433	1.570000e-318	852.0
AP010889.1_104	Blon_0105	Urease	lcl|CP001095.1_prot_ACJ51234.1_198	100.0	294	0	0	1	294	1	294	1.660000e-195	530.0
AP010889.1_105	Blon_0106	Urease	lcl|CP001095.1_prot_ACJ51235.1_199	100.0	371	0	0	1	371	1	371	1.930000e-267	718.0
AP010889.1_106	Blon_0107	Urease	lcl|CP001095.1_prot_ACJ51236.1_200	100.0	257	0	0	49	305	14	270	3.000000e-186	506.0
AP010889.1_107	Blon_0108	Urease	lcl|CP001095.1_prot_ACJ51237.1_201	100.0	235	0	0	1	235	2	236	2.300000e-166	451.0

1.6 Expected output

InStrain produces a variety of output in the IS folder depending on which operations are run. Generally, output that is meant for human eyes to be easily interpretable is located in the `output` folder.

1.6.1 inStrain profile

A typical run of inStrain will yield the following files in the output folder:

scaffold_info.tsv

This gives basic information about the scaffolds in your sample at the highest allowed level of read identity.

Table 8: scaffold_info.tsv

scaffold	length	coverage	breadth	nucl_diversity	coverage_median	coverage_std	coverage_SEM	breadth_minCov	breadth_expected	nucl_diversity_median	nucl_diversity_rarefied	nucl_diversity_rarefied_median	breadth_rarefied	conANI_reference	popANI_reference	SNS_count	SNV_count	divergent_site_count	consensus_divergent_sites	population_divergent_sites
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0	1.0	1.0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0	1.0	1.0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0	0.0	0.0	0	00	0	0	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									

scaffold The name of the *scaffold* in the input .fasta file

length Full length of the *scaffold* in the input .fasta file

coverage The average depth of coverage on the scaffold. If half the bases in a scaffold have 5 reads on them, and the other half have 10 reads, the coverage of the scaffold will be 7.5

breadth The percentage of bases in the scaffold that are covered by at least a single read. A breadth of 1 means that all bases in the scaffold have at least one read covering them

nucl_diversity The mean *nucleotide diversity* of all bases in the scaffold that have a nucleotide diversity value calculated. So if only 1 base on the scaffold meets the minimum coverage to calculate nucleotide diversity, the nucl_diversity of the scaffold will be the nucleotide diversity of that base. Will be blank if no positions have a base over the minimum coverage.

coverage_median The median depth of coverage value of all bases in the scaffold, included bases with 0 coverage

coverage_std The standard deviation of all coverage values

coverage_SEM The standard error of the mean of all coverage values (calculated using scipy.stats.sem)

breadth_minCov The percentage of bases in the scaffold that have at least min_cov coverage (e.g. the percentage of bases that have a nucl_diversity value and meet the minimum sequencing depth to call SNVs)

breadth_expected *expected breadth*; this tells you the breadth that you should expect if reads are evenly distributed along the genome, given the reported coverage value. Based on the function breadth = -1.000 * e^(0.883 * coverage) + 1.000. This is useful to establish whether or not the scaffold is actually in the reads, or just a fraction of the scaffold. If your coverage is 10x, the expected breadth will be ~1. If your actual breadth is significantly lower then the expected breadth, this means that reads are mapping only to a specific region of your scaffold (transposon, prophage, etc.)

nucl_diversity_median The median *nucleotide diversity* value of all bases in the scaffold that have a *nucleotide diversity* value calculated

nucl_diversity_rarefied The average *nucleotide diversity* among positions that have at least `--rarefied_coverage` (50x by default). These values are also calculated by randomly subsetting the reads at that position to `--rarefied_coverage` reads

nucl_diversity_rarefied_median The median rarefied *nucleotide diversity* (similar to that described above)

breadth_rarefied The percentage of bases in a scaffold that have at least `--rarefied_coverage`

conANI_reference The *conANI* between the reads and the reference genome

popANI_reference The *popANI* between the reads and the reference genome

SNS_count The total number of *SNSs* called on this scaffold

SNV_count The total number of *SNVs* called on this scaffold

divergent_site_count The total number of *divergent sites* called on this scaffold

consensus_divergent_sites The total number of *divergent sites* in which the reads have a different consensus allele than the reference genome. These count as "differences" in the conANI_reference calculation, and `breadth_minCov` * `length` counts as the denominator.

population_divergent_sites The total number of *divergent sites* in which the reads do not have the reference genome base as any allele at all (major or minor). These count as "differences" in the popANI_reference calculation, and `breadth_minCov` * `length` counts as the denominator.

mapping_info.tsv

This provides an overview of the number of reads that map to each scaffold, and some basic metrics about their quality. The header line (starting with #; not shown in the table below) describes the parameters that were used to filter the reads

Table 9: mapping_info.tsv

scaffold	pass_pairing_filter	filtered_pairs	median_insert	mean_PID	pass_min_insert	unfiltered_reads	unfiltered_pairs	pass_min_read_ani	filtered_priority_reads	filtered_singletons	mean_insert_distance	[illegible]	[illegible]	[illegible]	unfiltered_priority_reads	pass_max_insert	unfiltered_singletons	mean_pair_length
all_sc[illegible]	[illegible]	[illegible]	318.7[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	255.52
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	346.0	0	95	373.7[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	274.7106481481
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	461.0	0	359	387.9[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	285.5033738191
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	253.0	0	169	349.0[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	243.3103448275
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	205.0	0	486	327.8[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	251.2624584717
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	153.0	0	76	372.3[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	269.2300469483
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	116.0	0	81	376.4[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	246.8059701492
N5_[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	130.0	0	36	372.45	140.0	4.864[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	261.3071428571429

scaffold The name of the *scaffold* in the input .fasta file. For the top row this will read `all_scaffolds`, and it has the sum of all rows.

pass_pairing_filter The number of individual reads that pass the selecting pairing filter (only paired reads will pass this filter by default)

filtered_pairs The number of pairs of reads that pass all cutoffs

median_insert Among all pairs of reads mapping to this scaffold, the median insert distance.

mean_PID Among all pairs of reads mapping to this scaffold, the average percentage ID of both reads in the pair to the reference .fasta file

pass_min_insert The number of pairs of reads mapping to this scaffold that pass the minimum insert size cutoff

unfiltered_reads The raw number of reads that map to this scaffold

unfiltered_pairs The raw number of pairs of reads that map to this scaffold. Only paired reads are used by inStrain

pass_min_read_ani The number of pairs of reads mapping to this scaffold that pass the min_read_ani cutoff

filtered_priority_reads The number of priority reads that pass the rest of the filters (will only be non-0 if a priority reads input file is provided)

unfiltered_singletons The number of reads detected in which only one read of the pair is mapped

mean_insert_distance Among all pairs of reads mapping to this scaffold, the mean insert distance. Note that the insert size is measured from the start of the first read to the end of the second read (2 perfectly overlapping 50bp reads will have an insert size of 50bp)

pass_min_mapq The number of pairs of reads mapping to this scaffold that pass the minimum mapQ score cutoff

mean_mistmaches Among all pairs of reads mapping to this scaffold, the mean number of mismatches

mean_mapq_score Among all pairs of reads mapping to this scaffold, the average mapQ score

unfiltered_priority_reads The number of reads that pass the pairing filter because they were part of the `priority_reads` input file (will only be non-0 if a priority reads input file is provided).

pass_max_insert The number of pairs of reads mapping to this scaffold that pass the maximum insert size cutoff- that is, their insert size is less than 3x the median insert size of all_scaffolds. Note that the insert size is measured from the start of the first read to the end of the second read (2 perfectly overlapping 50bp reads will have an insert size of 50bp)

filtered_singletons The number of reads detected in which only one read of the pair is mapped AND which make it through to be considered. This will only be non-0 if the filtering settings allows non-paired reads

mean_pair_length Among all pairs of reads mapping to this scaffold, the average length of both reads in the pair summed together

> **Warning:** Adjusting the pairing filter will result in odd values for the "filtered_pairs" column; this column reports the number of pairs AND singletons that pass the filters. To calculate the true number of filtered pairs, use the formula filtered_pairs - filtered_singletons

SNVs.tsv

This describes the *SNVs* and *SNSs* that are detected in this mapping. While we should refer to these mutations as *divergent sites*, sometimes SNV is used to refer to both SNVs and SNSs

> **Warning:** inStrain reports 0-based values for "position". The first base in a scaffold will be position "0", second based position "1", etc.

Table 10: SNVs.tsv

scaffold	position	position_coverage	allele_count	ref_base	con_base	var_base	ref_freq	con_freq	var_freq	A	C	T	G	gene	mutation	mutation_type	cryptic	class
N5_271_010G1_scaffold_120	174	5	2	C	C	A	0.6	0.6	0.4	2	3	0	0			I	False	SNV
N5_271_010G1_scaffold_120	195	6	1	T	C	A	0.0	1.0	0.0	0	6	0	0			I	False	SNS
N5_271_010G1_scaffold_120	1401	8	2	A	A	C	0.75	0.75	0.25	6	2	0	0	N5_271_010G1_scaffold_120_1	N:V163G	N	False	SNV
N5_271_010G1_scaffold_120	1406	9	2	G	G	T	0.7777777777777778	0.7777777777777778	0.2222222222222222	0	0	2	7	N5_271_010G1_scaffold_120_1	N:S178Y	N	False	SNV
N5_271_010G1_scaffold_120	1481	6	2	C	T	C	0.3333333333333333	0.6666666666666666	0.3333333333333333	0	2	4	0	N5_271_010G1_scaffold_120_1	N:D238N	N	False	con_SNV
N5_271_010G1_scaffold_120	1484	6	2	G	A	G	0.3333333333333333	0.6666666666666666	0.3333333333333333	4	0	0	2	N5_271_010G1_scaffold_120_1	N:P236S	N	False	con_SNV
N5_271_010G1_scaffold_120	1488	5	1	T	C	T	0.2	0.8	0.2	0	4	1	0	N5_271_010G1_scaffold_120_1	S:240	S	False	SNS
N5_271_010G1_scaffold_120	1801	5	1	T	A	T	0.2	0.8	0.2	4	0	1	0	N5_271_010G1_scaffold_120_1	N:N563Y	N	False	SNS
N5_271_010G1_scaffold_120	1897	7	2	G	G	T	0.7142857142857143	0.7142857142857143	0.2857142857142857	0	0	2	5			I	False	SNV

See the module_descriptions for what constitutes a SNP (what makes it into this table)

scaffold The scaffold that the SNV is on

position The genomic position of the SNV

position_coverage The number of reads detected at this position

allele_count The number of bases that are detected above background levels (according to the *null model*. An allele_count of 0 means no bases are supported by the reads, an allele_count of 1 means that only 1 base is supported by the reads, an allele_count of 2 means two bases are supported by the reads, etc.

ref_base The reference base in the .fasta file at that position

con_base The consensus base (the base that is supported by the most reads)

var_base Variant base; the base with the second most reads

ref_freq The fraction of reads supporting the ref_base

con_freq The fraction of reds supporting the con_base

var_freq The fraction of reads supporting the var_base

A, C, T, and G The number of mapped reads encoding each of the bases

gene If a gene file was included, this column will be present listing if the SNV is in the coding sequence of a gene

mutation Short-hand code for the amino acid switch. If synonymous, this will be S: + the position. If nonsynonymous, this will be N: + the old amino acid + the position + the new amino acid. **NOTE** - the position of the amino acid is always calculated from left to right on the genome file, whether or not it's the forward or reverse strand. Codons are calculated correctly (considering strandedness), this only impacts the reported "position" in this column. I know this is weird behavior and it will change in future inStrain versions.

mutation_type What type of mutation this is. N = nonsynonymous, S = synonymous, I = intergenic, M = there are multiple genes with this base so you cant tell

cryptic If an SNV is cryptic, it means that it is detected when using a lower read mismatch threshold, but becomes undetected when you move to a higher read mismatch level. See "dealing with mm" in the advanced_use section for more details on what this means.

class The classification of this divergent site. The options are *SNS* (meaning allele_count is 1 and con_base does not equal ref_base), *SNV* (meaning allele_count is > 1 and con_base *does* equal ref_base), con_SNV (meaning allele_count is > 1, con_base does not equal ref_base, and ref_base *is* present in the reads; these count as differences in conANI calculations), pop_SNV (meaning allele_count is > 1, con_base does not equal ref_base, and ref_base *is not* present in the reads; these count as differences in popANI and conANI calculations), DivergentSite (meaning allele count is 0), and AmbiguousReference (meaning the ref_base is not A, C, T, or G)

linkage.tsv

This describes the *linkage* between pairs of SNPs in the mapping that are found on the same read pair at least min_snp times.

> **Warning:** inStrain reports 0-based values for "position". The first base in a scaffold will be position "0", second based position "1", etc.

Table 11: linkage.tsv

scaffold	position_A	position_B	distance	r2	d_prime	r2_normalized	d_prime_normalized	allele_A	allele_a	allele_B	allele_b	countab	countAb	countaB	countAB	total
N5_271_010G1_scaffold_93	58	59	1	0.021739130434782608	1.0	0.031141868512110725	1.0	C	T	G	A	0	3	4	20	27
N5_271_010G1_scaffold_93	58	70	12	0.012820512820512851	1.0			C	T	T	A	0	2	4	22	28
N5_271_010G1_scaffold_93	58	80	22	0.016722408026755852	1.0	0.005847953216374271	1.0	G	A	G	A	0	2	5	21	28
N5_271_010G1_scaffold_93	58	84	26	0.7652173913043479	1.0	0.6296296296296297	1.0	C	T	G	C	4	0	1	22	27
N5_271_010G1_scaffold_93	58	101	43	0.009070294784580067	1.0			C	T	C	A	0	2	2	19	23
N5_271_010G1_scaffold_93	58	126	68	0.017543859649122806	1.0	0.002770083102493075	1.0	C	T	A	T	0	2	3	16	21
N5_271_010G1_scaffold_93	58	133	75	0.008333333333333352	1.0			C	T	G	T	0	1	3	17	21
N5_271_010G1_scaffold_93	59	70	11	0.010869565217391304	1.0	0.027777777777777779	1.0	G	A	T	A	0	2	3	21	26
N5_271_010G1_scaffold_93	59	80	21	0.6410256410256397	1.0	1.0	1.0	G	A	G	A	2	0	1	25	28

Linkage is used primarily to determine if organisms are undergoing horizontal gene transfer or not. It's calculated for pairs of SNPs that can be connected by at least `min_snp` reads. It's based on the assumption that each SNP has two alleles (for example, a A and b B). This all gets a bit confusing and has a large amount of literature around each of these terms, but I'll do my best to briefly explain what's going on

scaffold The scaffold that both SNPs are on

position_A The position of the first SNP on this scaffold

position_B The position of the second SNP on this scaffold

distance The distance between the two SNPs

r2 This is the r-squared linkage metric. See below for how it's calculated

d_prime This is the d-prime linkage metric. See below for how it's calculated

r2_normalized, d_prime_normalized These are calculated by rarefying to `min_snp` number of read pairs. See below for how it's calculated

allele_A One of the two bases at position_A

allele_a The other of the two bases at position_A

allele_B One of the bases at position_B

allele_b The other of the two bases at position_B

countab The number of read-pairs that have allele_a and allele_b

countAb The number of read-pairs that have allele_A and allele_b

countaB The number of read-pairs that have allele_a and allele_B

countAB The number of read-pairs that have allele_A and allele_B

total The total number of read-pairs that have have information for both position_A and position_B

Python code for the calculation of these metrics:

```
freq_AB = float(countAB) / total
freq_Ab = float(countAb) / total
freq_aB = float(countaB) / total
freq_ab = float(countab) / total

freq_A = freq_AB + freq_Ab
freq_a = freq_ab + freq_aB
```

(continues on next page)

(continued from previous page)

```
freq_B = freq_AB + freq_aB
freq_b = freq_ab + freq_Ab

linkD = freq_AB - freq_A * freq_B

if freq_a == 0 or freq_A == 0 or freq_B == 0 or freq_b == 0:
    r2 = np.nan
else:
    r2 = linkD*linkD / (freq_A * freq_a * freq_B * freq_b)

linkd = freq_ab - freq_a * freq_b

# calc D-prime
d_prime = np.nan
if (linkd < 0):
    denom = max([(-freq_A*freq_B),(-freq_a*freq_b)])
    d_prime = linkd / denom

elif (linkD > 0):
    denom = min([(freq_A*freq_b), (freq_a*freq_B)])
    d_prime = linkd / denom

################
# calc rarefied

rareify = np.random.choice(['AB','Ab','aB','ab'], replace=True, p=[freq_AB,freq_Ab,
↪freq_aB,freq_ab], size=min_snp)
freq_AB = float(collections.Counter(rareify)['AB']) / min_snp
freq_Ab = float(collections.Counter(rareify)['Ab']) / min_snp
freq_aB = float(collections.Counter(rareify)['aB']) / min_snp
freq_ab = float(collections.Counter(rareify)['ab']) / min_snp

freq_A = freq_AB + freq_Ab
freq_a = freq_ab + freq_aB
freq_B = freq_AB + freq_aB
freq_b = freq_ab + freq_Ab

linkd_norm = freq_ab - freq_a * freq_b

if freq_a == 0 or freq_A == 0 or freq_B == 0 or freq_b == 0:
    r2_normalized = np.nan
else:
    r2_normalized = linkd_norm*linkd_norm / (freq_A * freq_a * freq_B * freq_b)


# calc D-prime
d_prime_normalized = np.nan
if (linkd_norm < 0):
    denom = max([(-freq_A*freq_B),(-freq_a*freq_b)])
    d_prime_normalized = linkd_norm / denom

elif (linkd_norm > 0):
    denom = min([(freq_A*freq_b), (freq_a*freq_B)])
    d_prime_normalized = linkd_norm / denom

rt_dict = {}
for att in ['r2', 'd_prime', 'r2_normalized', 'd_prime_normalized', 'total', 'countAB
↪', \
```

(continues on next page)

(continued from previous page)

```
            'countAb', 'countaB', 'countab', 'allele_A', 'allele_a', \
            'allele_B', 'allele_b']:
    rt_dict[att] = eval(att)
```

gene_info.tsv

This describes some basic information about the genes being profiled

> **Warning:** inStrain reports 0-based values for "position", including the "start" and "stop" in this table. The first base in a scaffold will be position "0", second based position "1", etc.

Table 12: gene_info.tsv

scaffold	gene	gene_length	coverage	breadth	breadth_minCov	nucl_diversity	start	end	direction	partial	dNdS_substitutions	pNpS_variants	SNV_count	SNV_S_count	SNV_N_count	SNS_count	SNS_S_count	SNS_N_count	divergent_site_count
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	283	-1	False			0.0	0.0	0.0	0.0	0.0	0.0	0.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		0.0	0.0	0.0	0.0	0.0	0.0	0.0	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		0.0	0.0	0.0	0.0	0.0	0.0	0.0	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	335	-1	False			0.0	0.0	0.0	0.0	0.0	0.0	0.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		[illegible]	[illegible]	[illegible]	[illegible]	0.0	0.0	0.0	9.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		[illegible]	[illegible]	[illegible]	[illegible]	0.0	0.0	0.0	4.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False			5.0	0.0	5.0	0.0	0.0	0.0	5.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		0.0	2.0	2.0	0.0	0.0	0.0	0.0	2.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	False		[illegible]	[illegible]	[illegible]	[illegible]	0.0	0.0	0.0	4.0

scaffold Scaffold that the gene is on

gene Name of the gene being profiled

gene_length Length of the gene in nucleotides

breadth The number of bases in the gene that have at least 1x coverage

breadth_minCov The number of bases in the gene that have at least min_cov coverage

nucl_diversity The mean *nucleotide diversity* of all bases in the gene that have a nucleotide diversity value calculated. So if only 1 base on the scaffold meets the minimum coverage to calculate nucleotide diversity, the nucl_diversity of the scaffold will be the nucleotide diversity of that base. Will be blank if no positions have a base over the minimum coverage.

start Start of the gene (position on scaffold; 0-indexed)

end End of the gene (position on scaffold; 0-indexed)

direction Direction of the gene (based on prodigal call). If -1, means the gene is not coded in the direction expressed by the .fasta file

partial If True this is a partial gene; based on not having *partial=00* in the record description provided by Prodigal

dNdS_substitutions The *dN/dS* of *SNSs* detected in this gene. Will be blank if 0 N and/or 0 S substitutions are detected

pNpS_variants The *pN/pS* of *SNVs* detected in this gene. Will be blank if 0 N and/or 0 S SNVs are detected

SNV_count Total number of *SNVs* detected in this gene

SNV_S_count Number of synonymous *SNVs* detected in this gene

SNV_N_count Number of non-synonymous *SNVs* detected in this gene

SNS_count Total number of *SNSs* detected in this gens

SNS_S_count Number of synonymous *SNSs* detected in this gens

SNS_N_count Number of non-synonymous *SNSs* detected in this gens

divergent_site_count Number of *divergent sites* detected in this gens

genome_info.tsv

Describes many of the above metrics on a genome-by-genome level, rather than a scaffold-by-scaffold level.

Table 13: genome_info.tsv

genome	cov-er-age	breadth	[illegible]	[illegible]	[illegible]	de-tected_scaffolds	[illegible]	[illegible]	cov-er-age_SEM	[illegible]	[illegible]	[illegible]	co-nANI_reference	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	con-sen-sus_divergent_sites	pop-u-la-tion_divergent_sites	SNS_count	SNV_count	fil-tered_read_pair_count
nt	reads_unfiltered_pairs	[illegible]	[illegible]	di-ver-gent_site_count	filtered_reads																				
fobin.fasta	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]										
3844	0.99	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5 926 599	0.92	[illegible]	[illegible]	[illegible]	157436										
max_bin	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	457											
8497	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	7368	9309	0.978	2331602	4248924											
528	11373																								

genome The name of the genome being profiled. If all scaffolds were a single genome, this will read "all_scaffolds"

coverage Average *coverage depth* of all scaffolds of this genome

breadth The *breadth* of all scaffolds of this genome

nucl_diversity The average *nucleotide diversity* of all scaffolds of this genome

length The full length of this genome across all scaffolds

true_scaffolds The number of scaffolds present in this genome based off of the scaffold-to-bin file

detected_scaffolds The number of scaffolds with at least a single read-pair mapping to them

coverage_median The median *coverage* among all bases in the genome

coverage_std The standard deviation of all coverage values

coverage_SEM The standard error of the mean of all coverage values (calculated using scipy.stats.sem)

breadth_minCov The percentage of bases in the scaffold that have at least min_cov coverage (e.g. the percentage of bases that have a nucl_diversity value and meet the minimum sequencing depth to call SNVs)

breadth_expected This tells you the breadth that you should expect if reads are evenly distributed along the genome, given the reported coverage value. Based on the function breadth = -1.000 * e^(0.883 * coverage) + 1.000. This is useful to establish whether or not the scaffold is actually in the reads, or just a fraction of the scaffold. If your coverage is 10x, the expected breadth will be ~1. If your actual breadth is significantly lower then the expected breadth, this means that reads are mapping only to a specific region of your scaffold (transposon, prophage, etc.)

nucl_diversity_rarefied The average *nucleotide diversity* among positions that have at least `--rarefied_coverage` (50x by default). These values are also calculated by randomly subsetting the reads at that position to `--rarefied_coverage` reads

conANI_reference The *conANI* between the reads and the reference genome

popANI_reference The *popANI* between the reads and the reference genome

iRep The *iRep* value for this genome (if it could be successfully calculated)

iRep_GC_corrected A True / False value of whether the iRep value was corrected for GC bias

linked_SNV_count The number of *divergent sites* that could be *linked* in this genome

SNV_distance_mean Average distance between linked *divergent sites*

r2_mean Average r2 between linked SNVs (see explanation of linkage.tsv above for more info)

d_prime_mean Average d prime between linked SNVs (see explanation of linkage.tsv above for more info)

consensus_divergent_sites The total number of *divergent sites* in which the reads have a different consensus allele than the reference genome. These count as "differences" in the conANI_reference calculation, and `breadth_minCov` * `length` counts as the denominator.

population_divergent_sites The total number of *divergent sites* in which the reads do not have the reference genome base as any allele at all (major or minor). These count as "differences" in the popANI_reference calculation, and `breadth_minCov` * `length` counts as the denominator.

SNS_count The total number of *SNSs* called on this genome

SNV_count The total number of *SNVs* called on this genome

filtered_read_pair_count The total number of read pairs that pass filtering and map to this genome

reads_unfiltered_pairs The total number of pairs, filtered or unfiltered, that map to this genome

reads_mean_PID The average ANI of mapped read pairs to the reference genome for this genome

reads_unfiltered_reads The total number of reads, filtered or unfiltered, that map to this genome

divergent_site_count The total number of *divergent sites* called on this genome

1.6.2 inStrain parse_annotations

A typical run of inStrain *parse_gene_annotations* will yield the following files in the output folder. For more information, see *User Manual*

LongFormData.csv

All of the annotation information a very long table

Table 14: LongFormData.csv

sample	anno	genomes	genes	bases
2bag10_1.bam	K03737	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa'}	1	666
2bag10_1.bam	K06973	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa'}	1	1068
2bag10_1.bam	K04066	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	2	195761
2bag10_1.bam	K15558	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	96	1074879
2bag10_1.bam	K19762	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	97	1092005
2bag10_1.bam	3000025	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	2	16896
2bag10_1.bam	K18888	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	3	504008
2bag10_1.bam	K20386	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa', 'Bifidobacterium_longum_subsp_infantis_ATCC_15697.fna'}	98	1100781
2bag10_1.bam	K07979	{'REFINED_METABAT215_TOP10_CONTIGS_1500_ASSEMBLY_K77_MERGED__Hadza_MoBio_hadza-E-H_A_23_1707.16.fa'}	1	742

sample The sample this row refers to (based on the name of the .bam file used to create the inStrain profile)

anno The annotation this row refers to (based on the input annotation table)

genomes The specific genomes that have this particular annotation. Represented as a python set

genes The total number of genes detected with this annotation in this sample

bases The total number of base-pairs mapped to all genes with this annotation in this sample

SampleAnnotationTotals.csv

Totals for each sample. Used to generate the *_fraction* tables enumerated below.

Table 15: SampleAnnotationTotals.csv

sample	detected_genes	detected_genomes	bases_mapped_to_genes	detected_annotations	detected_genes_with_anno
2bag10_1.bam	2625	2	222405987	3302	1677
2bag10_2.bam	20909	10	2418511040	32225	15513

sample The sample this row refers to (based on the name of the .bam file used to create the inStrain profile)

detected_genes The total number of genes detected in this sample after passing the set filters

detected_genomes The total number of genomes detected in this sample after passing the set filters

bases_mapped_to_genes The total number of bases mapped to detected genes. See *ParsedGeneAnno_bases.csv* below for more info

detected_annotations The total number of annotations detected; this can be higher than *detected_genes_with_anno* if some genes have multiple annotations

detected_genes_with_anno The total number of genes detected with at least one annotation

ParsedGeneAnno_*.csv

There are a total of 6 tables like this generated in the output folder, each looking like the following:

Table 16: ParsedGeneAnno_bases.csv

sample	3000005	3000024	3000025	3000026	3000027	3000074	3000118	3000165	3000166
2bag10_1.bam	131097	1286827	168916	1656	0	0	0	0	0
2bag10_2.bam	104013	5016854	955645	2552	633275	1034042	95617	409295	541951

In each case the column *sample* is the sample the row refers to (based on the name of the .bam file used to create the inStrain profile), and all other columns are annotations from the input annotation_table provides. The number values differ depending on the individual output table being analyzed. Below you can find descriptions on what the numbers refer to:

ParsedGeneAnno_bases.csv The total number of base pairs mapped to all genes with this annotation. The number of base pairs mapped for each gene with this annotation is calculated as the gene length * the coverage of the gene, and the number reported is the sum of this value of all genes

ParsedGeneAnno_bases_fraction.csv The values in *ParsedGeneAnno_bases.csv* divided by the total number of bases mapped to all detected genes (the value *bases_mapped_to_genes* reported in *SampleAnnotationTotals.csv*)

ParsedGeneAnno_genes.csv The total number of detected genes with this annotation

ParsedGeneAnno_genes_fraction.csv The values in *ParsedGeneAnno_genes.csv* divided by the total number of genes detected (the value *detected_genes* reported in *SampleAnnotationTotals.csv*)

ParsedGeneAnno_genomes.csv The total number of genomes with at least one detected gene with this annotation

ParsedGeneAnno_genomes_fraction.csv The values in *ParsedGeneAnno_genomes.csv* divided by the total number of genomes detected (the value *detected_genomes* reported in *SampleAnnotationTotals.csv*)

1.6.3 inStrain compare

A typical run of inStrain will yield the following files in the output folder:

comparisonsTable.tsv

Summarizes the differences between two inStrain profiles on a scaffold by scaffold level

Table 17: comparisonsTable.tsv

scaffold	name1	name2	coverage_overlap	compared_bases_count	percent_genome_compared	length	consensus_SNPs	population_SNPs	popANI	conANI
N5_271_010G1_scaffold_98	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	61	0.05290546400693842	1153	0	0	1.0	1.0
N5_271_010G1_scaffold_133	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	78	0.0741444866920152	1052	0	0	1.0	1.0
N5_271_010G1_scaffold_144	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	172	0.16715257531584066	1029	0	0	1.0	1.0
N5_271_010G1_scaffold_158	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	36	0.035749751737835164	1007	0	0	1.0	1.0
N5_271_010G1_scaffold_57	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	24	0.0183206106870229	1310	0	0	1.0	1.0
N5_271_010G1_scaffold_139	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	24	0.023121387283236997	1038	0	0	1.0	1.0
N5_271_010G1_scaffold_92	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	336	0.286934244235696	1171	0	0	1.0	1.0
N5_271_010G1_scaffold_97	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	22	0.01901469317199654	1157	0	0	1.0	1.0
N5_271_010G1_scaffold_100	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	21	0.018292682926829267	1148	0	0	1.0	1.0

scaffold The scaffold being compared

name1 The name of the first *inStrain profile* being compared

name2 The name of the second *inStrain profile* being compared

coverage_overlap The percentage of bases that are either covered or not covered in both of the profiles (covered = the base is present at at least min_snp coverage). The formula is length(coveredInBoth) / length(coveredInEither). If both scaffolds have 0 coverage, this will be 0.

compared_bases_count The number of considered bases; that is, the number of bases with at least min_snp coverage in both profiles. Formula is length([x for x in overlap if x == True]).

percent_genome_compared The percentage of bases in the scaffolds that are covered by both. The formula is length([x for x in overlap if x == True])/length(overlap). When ANI is np.nan, this must be 0. If both scaffolds have 0 coverage, this will be 0.

length The total length of the scaffold

consensus_SNPs The number of locations along the genome where both samples have the base at >= 5x coverage, and the consensus allele in each sample is different. Used to calculate *conANI*

population_SNPs The number of locations along the genome where both samples have the base at >= 5x coverage, and no alleles are shared between either sample. Used to calculate *popANI*

popANI The average nucleotide identity among compared bases between the two scaffolds, based on population_SNPs. Calculated using the formula popANI = (compared_bases_count - population_SNPs) / compared_bases_count

conNI The average nucleotide identity among compared bases between the two scaffolds, based on consensus_SNPs. Calculated using the formula conANI = (compared_bases_count - consensus_SNPs) / compared_bases_count

pairwise_SNP_locations.tsv

Warning: inStrain reports 0-based values for "position". The first base in a scaffold will be position "0", second based position "1", etc.

Lists the locations of all differences between profiles. Because it's a big file, this will only be created is you include the flag `--store_mismatch_locations` in your inStrain compare command.

Table 18: pairwise_SNP_locations.tsv

scaffold	position	name1	name2	consensus_SNP	population_SNP	con_base_1	ref_base_1	var_base_1	position_coverage_1	A_1	C_1	T_1	G_1	con_base_2	ref_base_2	var_base_2	position_coverage_2	A_2	C_2	T_2	G_2
N5_271_010G1_scaffold_9	823	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	G	G	A	10.0	3.0	0.0	0.0	7.0	A	G	G	6.0	3.0	0.0	0.0	3.0
N5_271_010G1_scaffold_1	906	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	T	T	C	6.0	0.0	2.0	4.0	0.0	C	T	T	7.0	0.0	4.0	3.0	0.0
N5_271_010G1_scaffold_2	436	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	C	T	T	6.0	0.0	3.0	3.0	0.0	T	T	C	7.0	0.0	3.0	4.0	0.0
N5_271_010G1_scaffold_4	794	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	A	A	T	6.0	4.0	0.0	2.0	0.0	T	A	A	9.0	4.0	0.0	5.0	0.0
N5_271_010G1_scaffold_2	1608	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	G	G	A	8.0	2.0	0.0	0.0	6.0	A	G	G	6.0	5.0	0.0	0.0	1.0
N5_271_010G1_scaffold_2	600	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	A	G	G	6.0	4.0	0.0	0.0	2.0								
N5_271_010G1_scaffold_8	497	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	A	G	G	5.0	3.0	0.0	0.0	2.0								
N5_271_010G1_scaffold_5	1108	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	A	A	G	5.0	3.0	0.0	0.0	2.0	G	A	A	15.0	6.0	0.0	0.0	9.0
N5_271_010G1_scaffold_4	710	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	True	False	A	C	C	6.0	4.0	2.0	0.0	0.0	C	C	A	6.0	2.0	4.0	0.0	0.0

scaffold The scaffold on which the difference is located

position The position where the difference is located (0-based)

name1 The name of the first *inStrain profile* being compared

name2 The name of the second *inStrain profile* being compared

consensus_SNP A True / False column listing whether or not this difference counts towards *conANI* calculations

population_SNP A True / False column listing whether or not this difference counts towards *popANI* calculations

con_base_1 The consensus base of the profile listed in `name1` at this position

ref_base_1 The reference base of the profile listed in `name1` at this position (will be the same as `ref_base_2`)

var_base_1 The variant base of the profile listed in `name1` at this position

position_coverage_1 The number of reads mapping to this position in `name1`

A_1, C_1, T_1, G_1 The number of mapped reads with each nucleotide in `name1`

con_base_2, ref_base_2, . . . The above columns are also listed for the `name2` sample

genomeWide_compare.tsv

A genome-level summary of the differences detected by inStrain compare. Generated by running `inStrain genome_wide` on the results of `inStrain compare`, or by providing an stb file to the original inStrain compare command.

Table 19: genomeWide_compare.tsv

genome	name1	name2	coverage_overlap	compared_bases_count	consensus_SNPs	population_SNPs	popANI	conANI	percent_compared
all_scaffolds	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	1.0	[illegible]	0	0	1.0	1.0	0.36055490915600 11
all_scaffolds	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
all_scaffolds	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	1.0	[illegible]	0	0	1.0	1.0	0.5214821444553835

genome The genome being compared

name1 The name of the first *inStrain profile* being compared

name2 The name of the second *inStrain profile* being compared

coverage_overlap The percentage of bases that are either covered or not covered in both of the profiles (covered = the base is present at at least min_snp coverage). The formula is length(coveredInBoth) / length(coveredInEither). If both scaffolds have 0 coverage, this will be 0.

compared_bases_count The number of considered bases; that is, the number of bases with at least min_snp coverage in both profiles. Formula is length([x for x in overlap if x == True]).

percent_genome_compared The percentage of bases in the scaffolds that are covered by both. The formula is length([x for x in overlap if x == True])/length(overlap). When ANI is np.nan, this must be 0. If both scaffolds have 0 coverage, this will be 0.

length The total length of the genome

consensus_SNPs The number of locations along the genome where both samples have the base at >= 5x coverage, and the consensus allele in each sample is different. Used to calculate *conANI*

population_SNPs The number of locations along the genome where both samples have the base at >= 5x coverage, and no alleles are shared between either sample. Used to calculate *popANI*

popANI The average nucleotide identity among compared bases between the two scaffolds, based on population_SNPs. Calculated using the formula popANI = (compared_bases_count - population_SNPs) / compared_bases_count

conNI The average nucleotide identity among compared bases between the two scaffolds, based on consensus_SNPs. Calculated using the formula conANI = (compared_bases_count - consensus_SNPs) / compared_bases_count

strain_clusters.tsv

The result of clustering the pairwise comparison data provided in `genomeWide_compare.tsv` to generate strain-level clusters. Performed using hierarchical clustering in the same manner as the program `dRep`; see the dRep documentation for some info on the oddities of hierarchical clustering

Table 20: genomeWide_compare.tsv

cluster	sample	genome
1_1	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	fobin.fasta
1_1	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	fobin.fasta
2_1	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	maxbin2.maxbin.001.fasta
2_2	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	maxbin2.maxbin.001.fasta

cluster The strain identity of this genome in this sample. Each "strain" assigned by the hierarchical clustering algorithm will have a unique cluster. In the example table above strains of the genome `fobin.fasta` are the same in both samples (they have the same "cluster" identities), but strains of the genome `maxbin2.maxbin.001.fasta` are different in the two samples (they have different "cluster" identities).

sample The sample that the genome was detected in.

genome The genome that the cluster is referring to.

pooled_SNV_info.tsv, pooled_SNV_data.tsv, and pooled_SNV_data_keys.tsv

The tables *pooled_SNV_info.tsv*, *pooled_SNV_data.tsv*, and *pooled_SNV_data_keys.tsv* can be generated by inStrain compare by providing .bam files to the inStrain compare command. See *User Manual* for more information.

Table 21: pooled_SNV_info.tsv

scaffold	position	depth	A	C	T	G	ref_base	con_base	var_base	sample_detections	sample_5x_detections	DivergentSite_count	SNS_count	SNV_count	con_SNV_count	pop_SNV_count	sample_con_bases
N5_271_010G1_scaffold_114	3	10	2	0	7	1	T	T	A	2	1	0	0	1	0	0	['T']
N5_271_010G1_scaffold_114	20	13	0	11	2	0	C	C	T	2	2	0	0	1	0	0	['C']
N5_271_010G1_scaffold_114	24	35	29	0	2	4	A	A	G	2	2	0	0	2	0	0	['A']
N5_271_010G1_scaffold_114	25	38	2	36	0	0	C	C	A	2	2	0	0	1	0	0	['C']
N5_271_010G1_scaffold_114	55	71	66	5	0	0	A	A	C	2	2	0	0	1	0	0	['A']
N5_271_010G1_scaffold_114	57	67	2	0	0	65	G	G	A	2	2	0	0	1	0	0	['G']
N5_271_010G1_scaffold_114	75	95	4	90	0	1	C	C	A	2	2	0	0	1	0	0	['C']
N5_271_010G1_scaffold_114	76	95	0	90	2	3	C	C	G	2	2	0	0	2	0	0	['C']
N5_271_010G1_scaffold_114	79	98	0	3	0	95	G	G	C	2	2	0	0	1	0	0	['G']

This table has information about each SNV, summarized across all samples

scaffold The scaffold being analyzed

position The position in the scaffold where the SNV is located (0-based)

depth The total number of reads mapping to this scaffold across samples

A The number of reads with *A* at this position in this scaffold across samples

C The number of reads with *C* at this position in this scaffold across samples

T The number of reads with *T* at this position in this scaffold across samples

G The number of reads with *G* at this position in this scaffold across samples

ref_base The reference base at this position in this scaffold across samples

con_base The consensus base (most common) at this position in this scaffold across samples

var_base The variant base (second most common) at this position in this scaffold across samples

sample_detections The number of samples in which this position at this scaffold has at least one read mapping to it

sample_5x_detections The number of samples in which this position at this scaffold has at least 5 reads mapping to it

DivergentSite_count The number of samples with a *divergent sites* detected at this position

SNS_count The number of samples with a *SNSs* detected at this position

SNV_count The number of samples with a *SNVs* detected at this position

con_SNV_count The number of samples with consenus SNPs (*conANI*) detected at this position

pop_SNV_count The number of samples with population SNPs (*popANI*) detected at this position

sample_con_bases The number of different consensus bases at this position across all analyzed samples

Table 22: pooled_SNV_data.tsv

sample	scaffold	position	A	C	T	G
0	0	3	2	0	5	1
0	0	20	0	21	2	0
0	0	24	21	0	0	4
0	0	25	2	26	0	0
0	0	55	38	5	0	0
0	0	57	2	0	0	38
0	0	75	3	55	0	0
0	0	76	0	56	0	3
0	0	79	0	1	0	57

This table has information about each SNV in each sample. Because the table can be huge, names of scaffolds and samples are listed as "keys" to be translated using the also-provided *pooled_SNV_data_keys.tsv* table

sample The key for the sample being analyzed (as detailed in the *pooled_SNV_data_keys.tsv* table below)

scaffold The key for the scaffold being analyzed (as detailed in the *pooled_SNV_data_keys.tsv* table below)

position The position in the scaffold where the SNV is located (0-based)

A,C,T,G The number of reads with this base in this sample in this scaffold at this position

Table 23: pooled_SNV_data_keys.tsv

key	sample	scaffold
0	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	N5_271_010G1_scaffold_114
1	N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G2.sorted.bam	N5_271_010G1_scaffold_63
2		N5_271_010G1_scaffold_89
3		N5_271_010G1_scaffold_33
4		N5_271_010G1_scaffold_95
5		N5_271_010G1_scaffold_11
6		N5_271_010G1_scaffold_74
7		N5_271_010G1_scaffold_71
8		N5_271_010G1_scaffold_96

This table has "keys" needed to translate the *pooled_SNV_data.tsv* table

key The key in question. This is the number presented in the *sample* or *scaffold* column in the *pooled_SNV_data.tsv* table above

sample The name of the sample with this key. For example: for the row with a *0* as the *key*, sample *0* in *pooled_SNV_data.tsv* refers to the sample listed here

scaffold The name of the scaffold with this key. For example: for the row with a *0* as the *key*, scaffold *0* in *pooled_SNV_data.tsv* refers to the sample listed here

1.6.4 inStrain plot

This is what the results of inStrain plot look like.

1) Coverage and breadth vs. read mismatches

Breadth of coverage (blue line), coverage depth (red line), and expected breadth of coverage given the depth of coverage (dotted blue line) versus the minimum ANI of mapped reads. Coverage depth continues to increase while breadth of plateaus, suggesting that all regions of the reference genome are not present in the reads being mapped.

2) Genome-wide microdiversity metrics

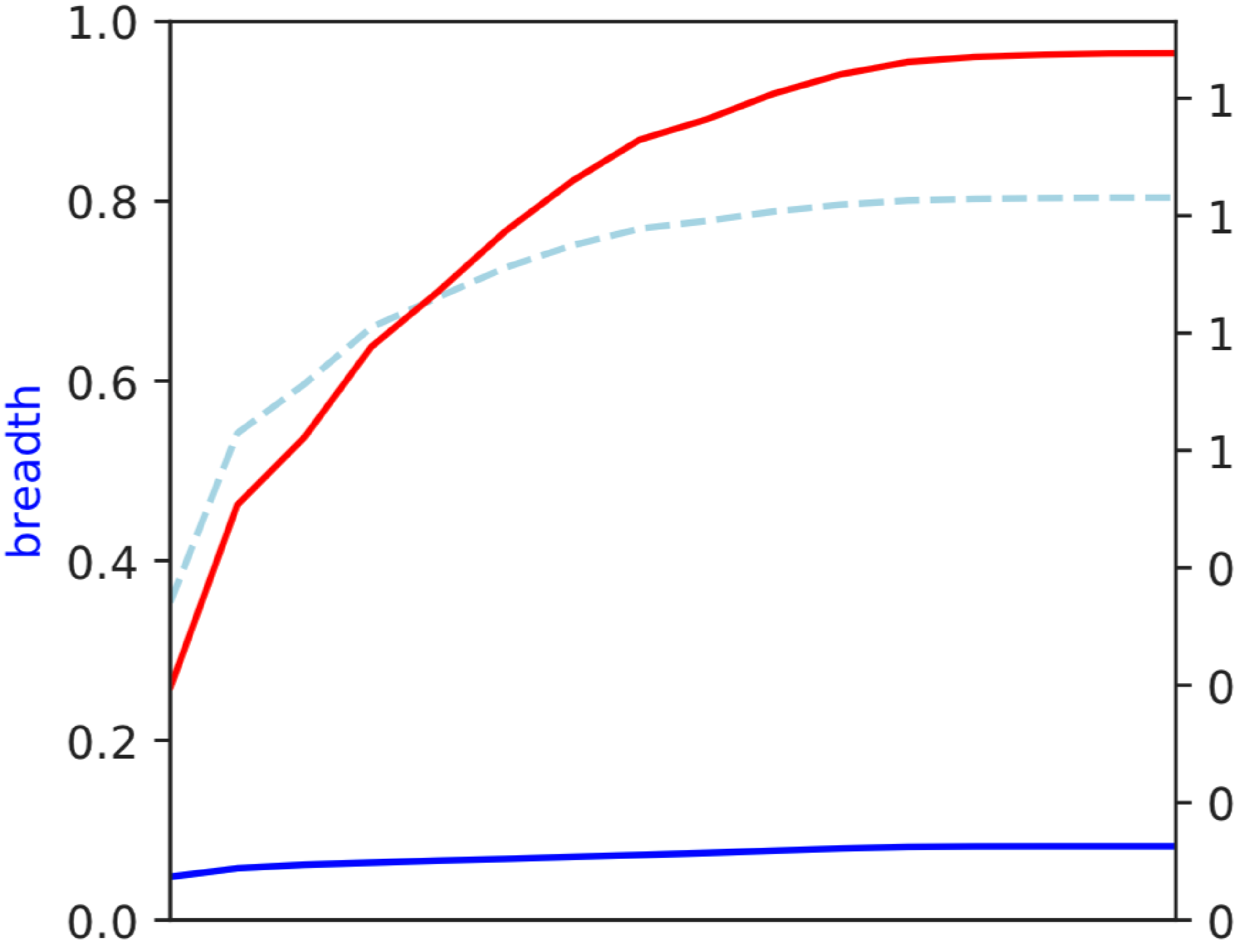
SNV density, coverage, and nucleotide diversity. Spikes in nucleotide diversity and SNV density do not correspond with increased coverage, indicating that the signals are not due to read mis-mapping. Positions with nucleotide diversity and no SNV-density are those where diversity exists but is not high enough to call a SNV

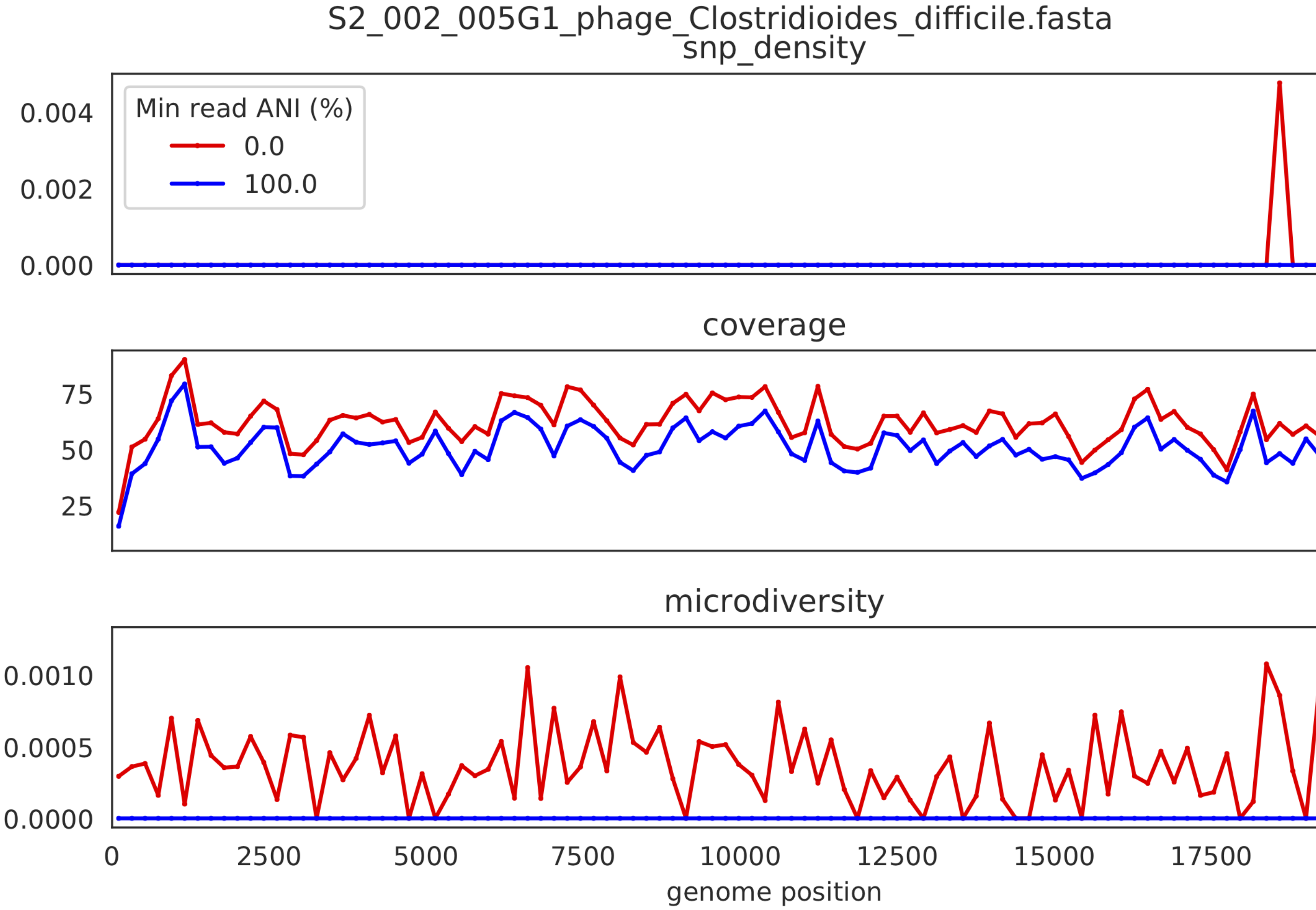
3) Read-level ANI distribution

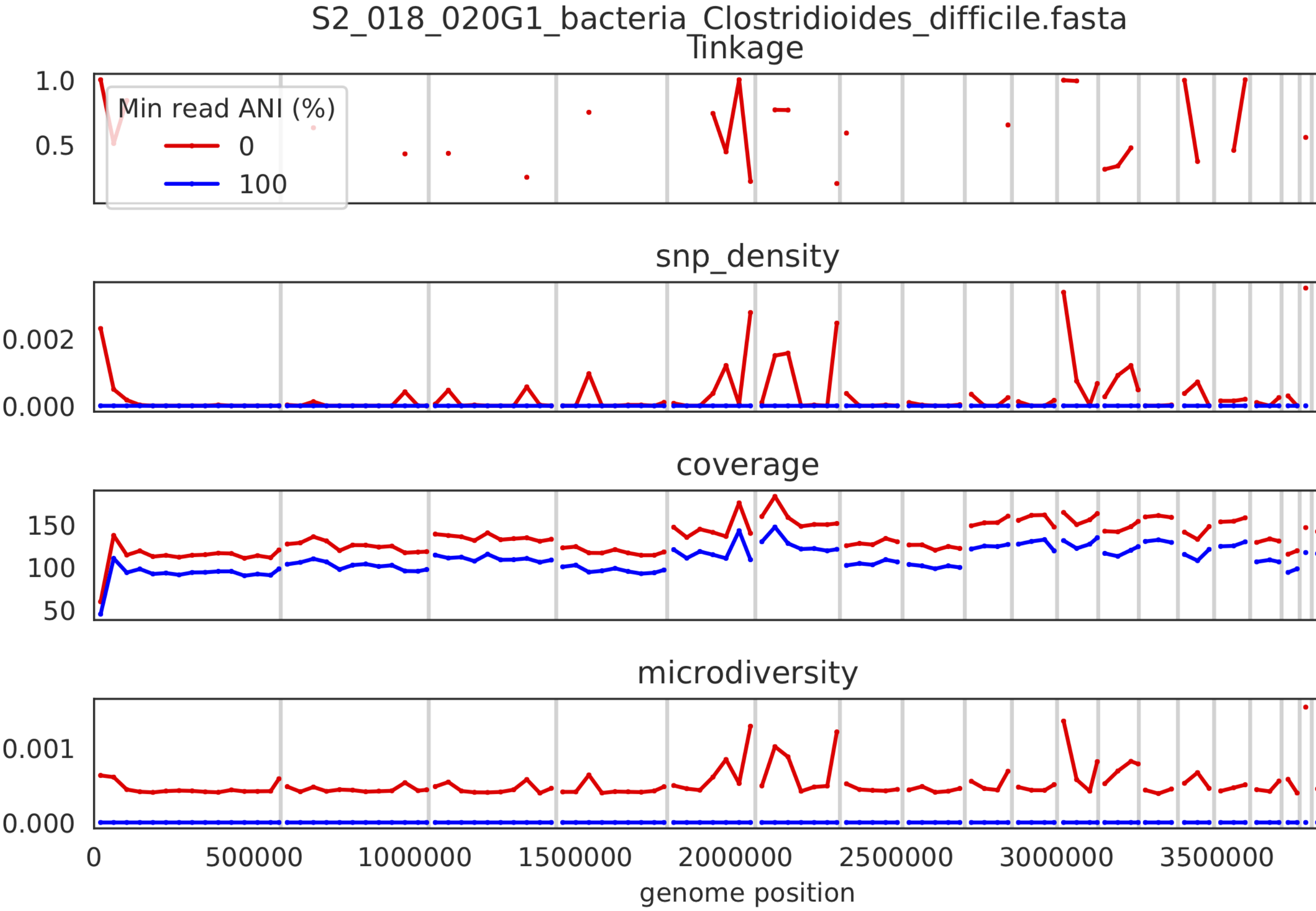
Distribution of read pair ANI levels when mapped to a reference genome; this plot suggests that the reference genome is >1% different than the mapped reads

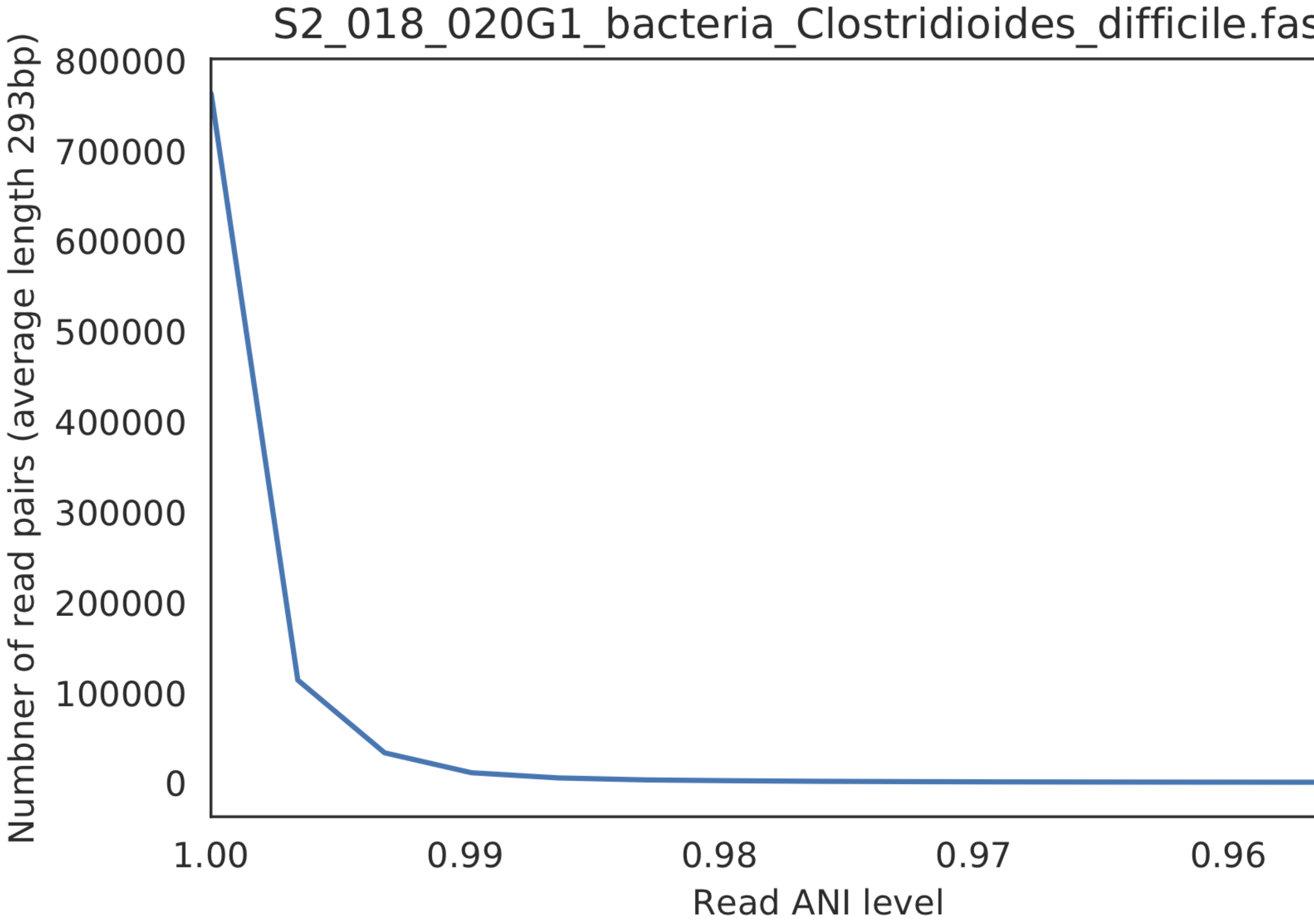
4) Major allele frequencies

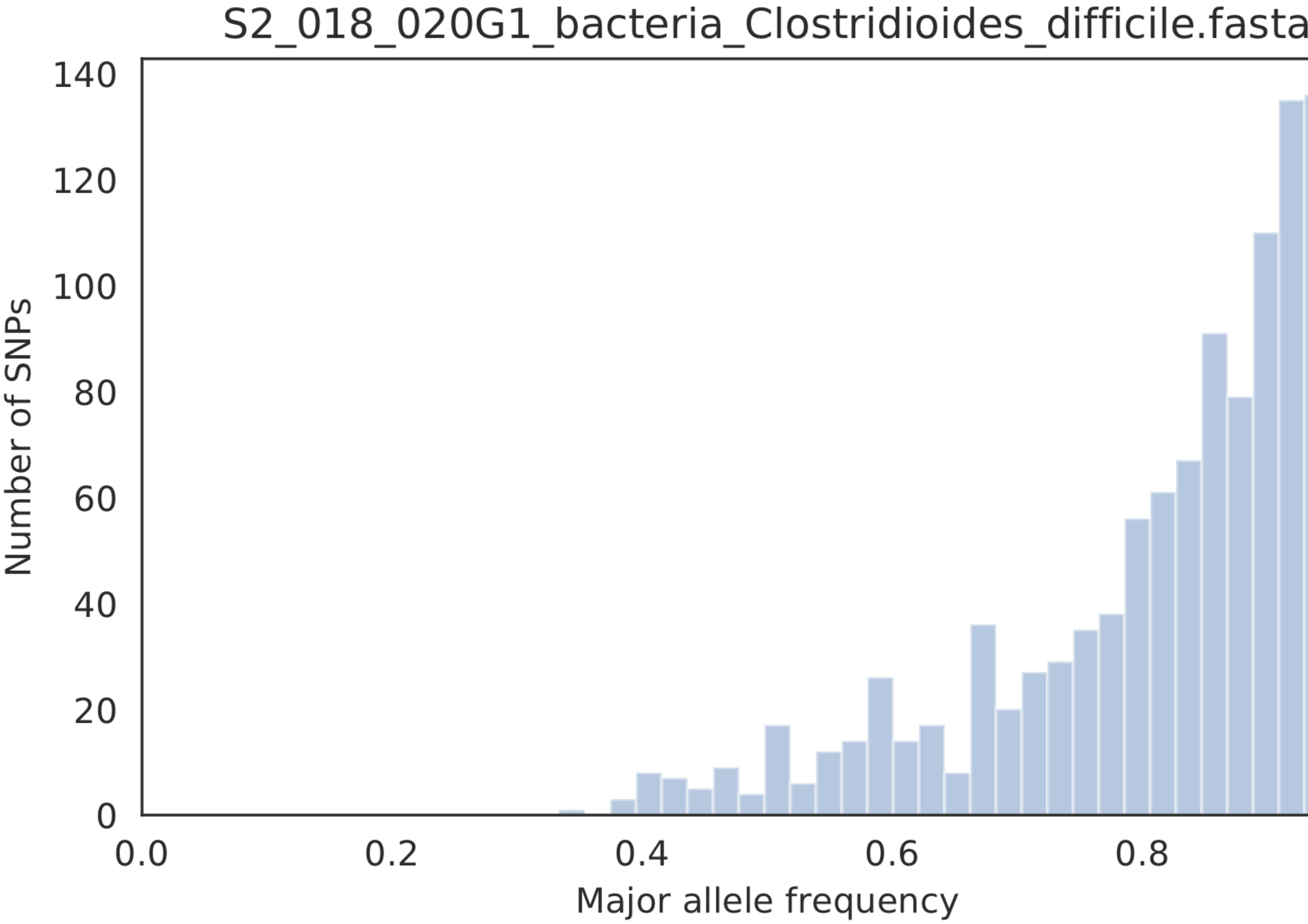
Distribution of the major allele frequencies of bi-allelic SNVs (the Site Frequency Spectrum). Alleles with major frequencies below 50% are the result of multiallelic sites. The lack of distinct puncta suggest that more than a few distinct strains are present.


breadth
1.0
0.8
0.6
0.4
0.2
0.0
1
1
1
1
0
0
0
0

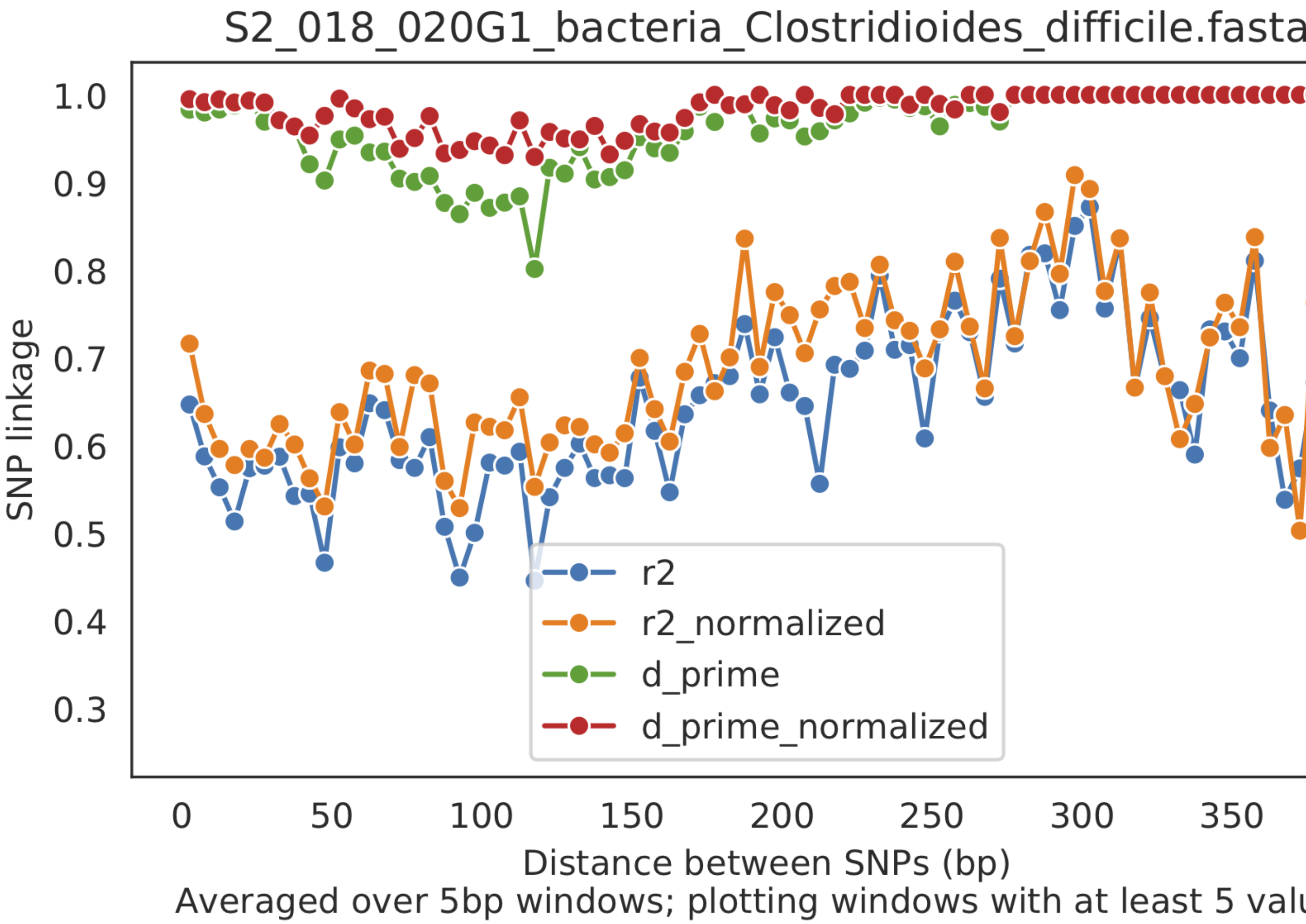

S2_002_005G1_phage_Clostridioides_difficile.fasta
snp_density
Min read ANI (%)
0.0
100.0
0.004
0.002
0.000
coverage
75
50
25
microdiversity
0.0010
0.0005
0.0000
0
2500
5000
7500
10000
12500
15000
17500
genome position


S2_018_020G1_bacteria_Clostridioides_difficile.fasta
linkage
1.0
0.5
Min read ANI (%)
0
100
snp_density
0.002
0.000
coverage
150
100
50
microdiversity
0.001
0.000
0
500000
1000000
1500000
2000000
2500000
3000000
3500000
genome position


S2_018_020G1_bacteria_Clostridioides_difficile.fas
Numbner of read pairs (average length 293bp)
800000
700000
600000
500000
400000
300000
200000
100000
0
1.00
0.99
0.98
0.97
0.96
Read ANI level


S2_018_020G1_bacteria_Clostridioides_difficile.fasta
Number of SNPs
140
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
Major allele frequency

5) Linkage decay



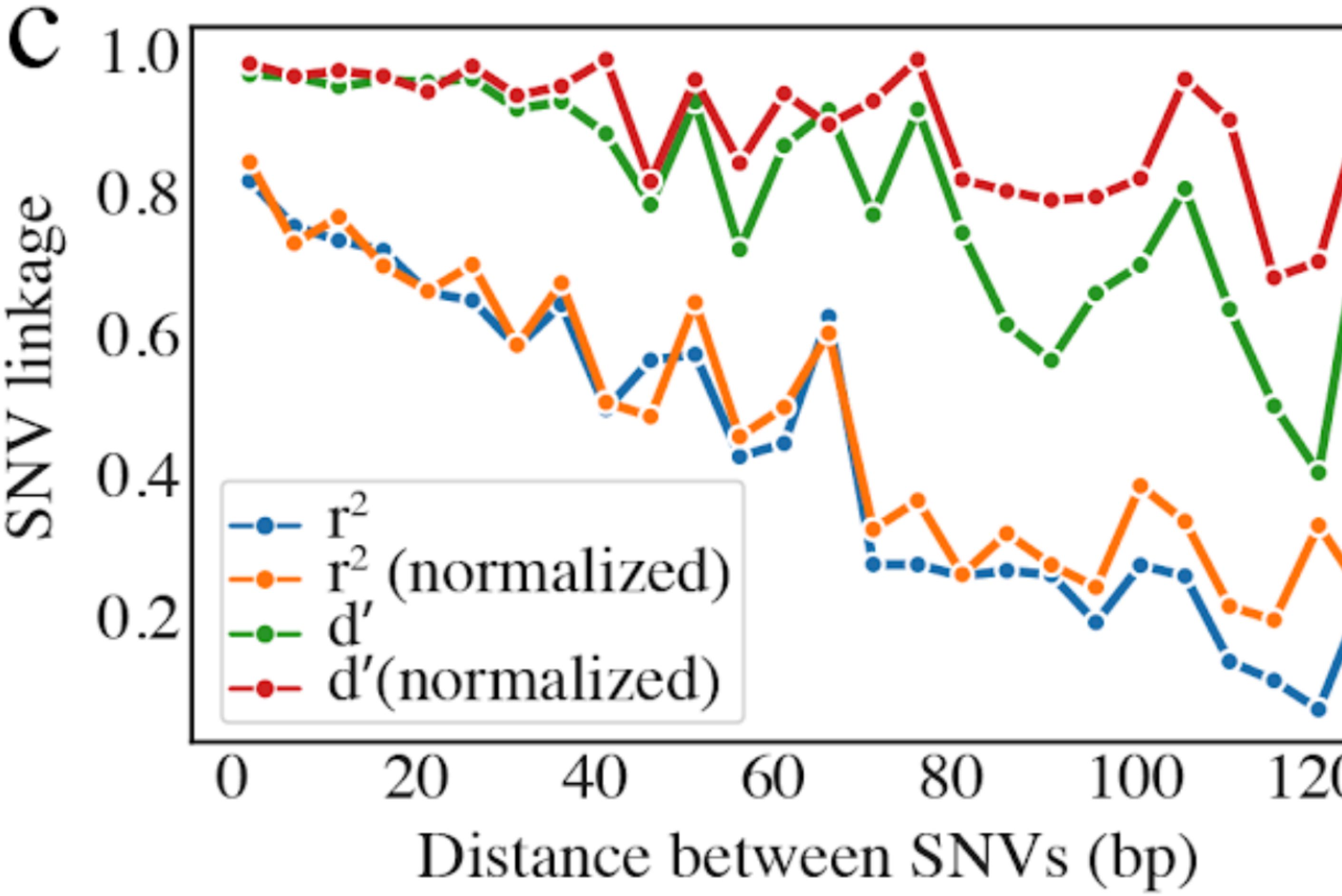
Metrics of SNV linkage vs. distance between SNVs; linkage decay (shown in one plot and not the other) is a common signal of recombination.

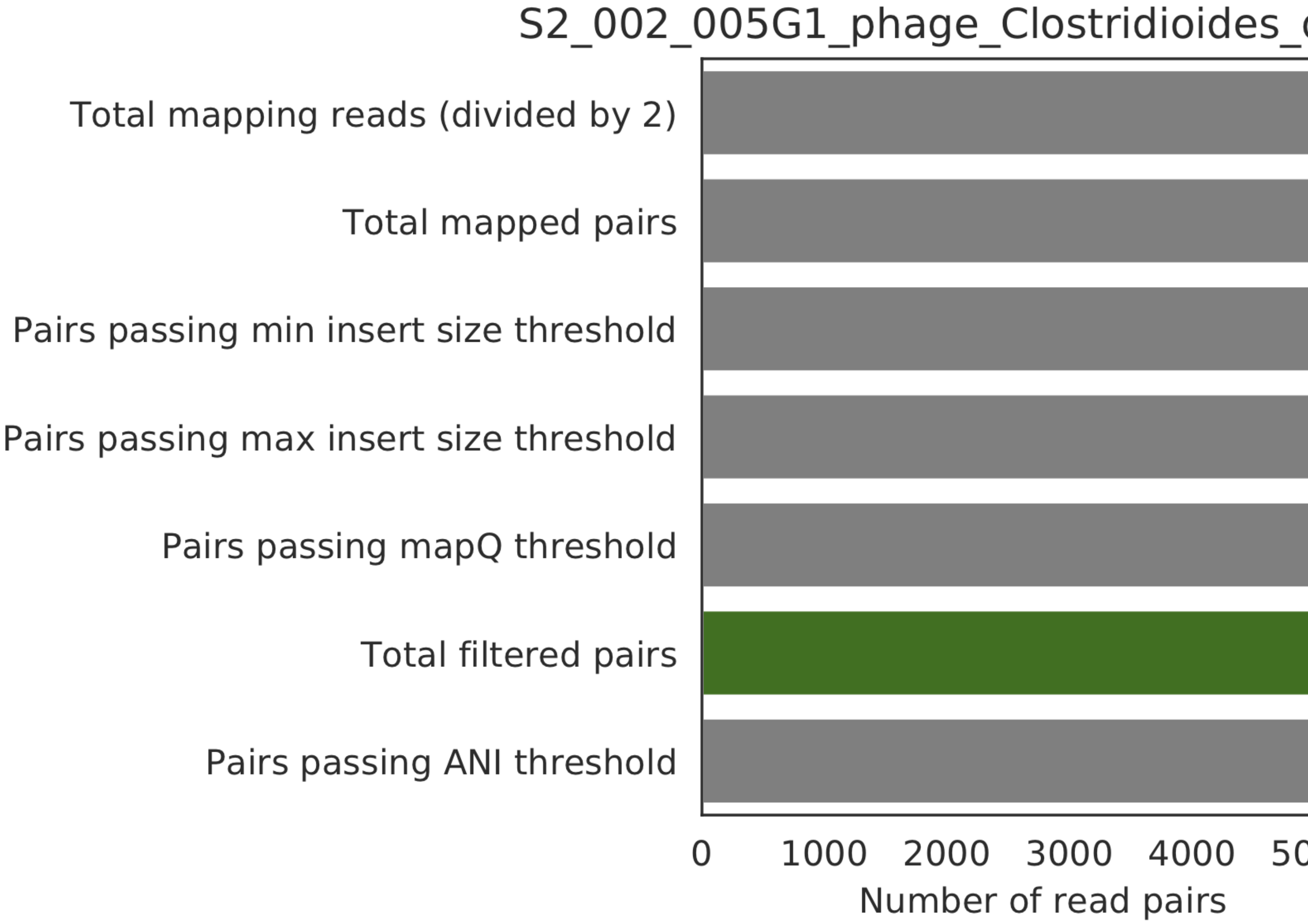
6) Read filtering plots

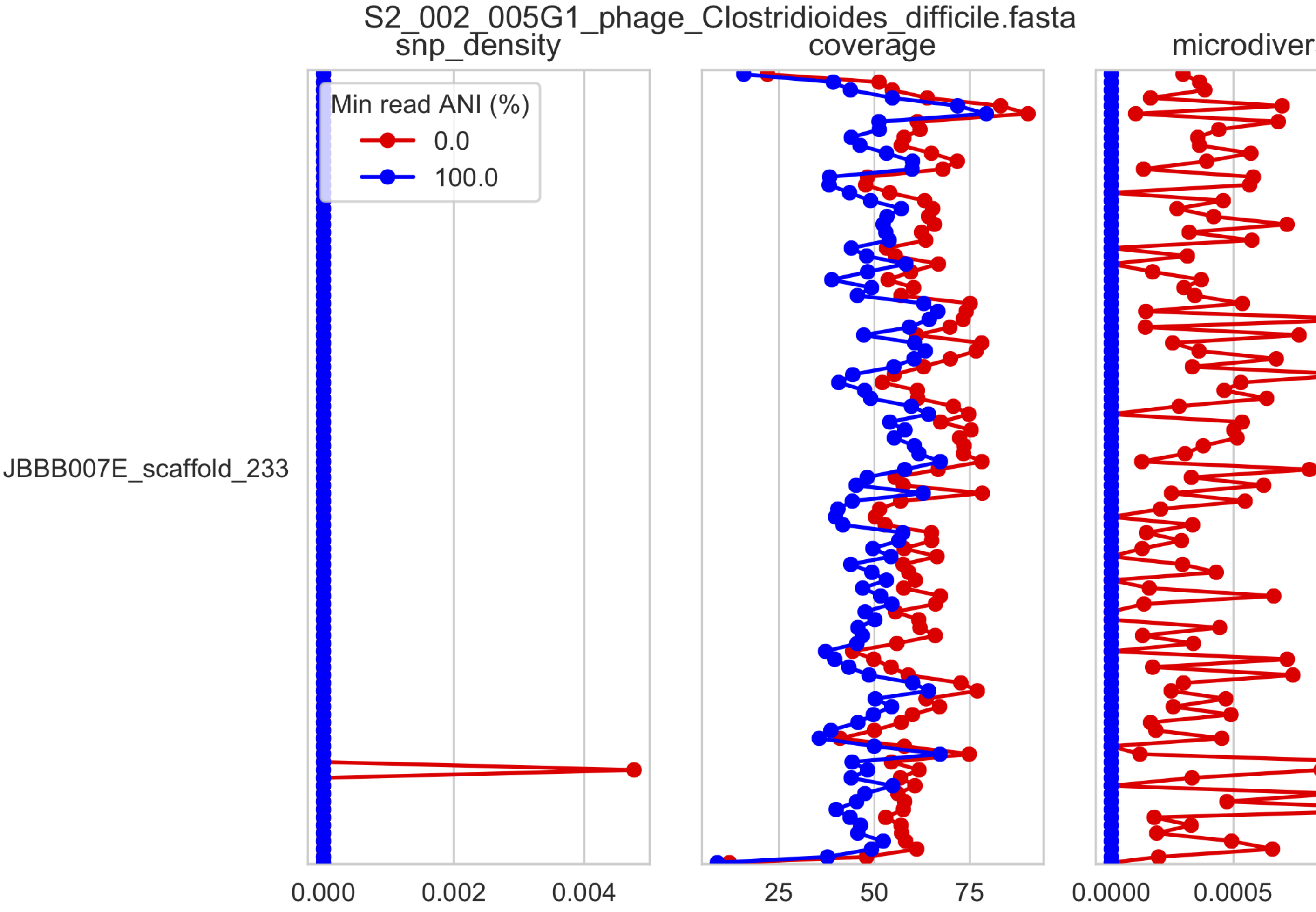
Bar plots showing how many reads got filtered out during filtering. All percentages are based on the number of paired reads; for an idea of how many reads were filtered out for being non-paired, compare the top bar and the second to top bar.

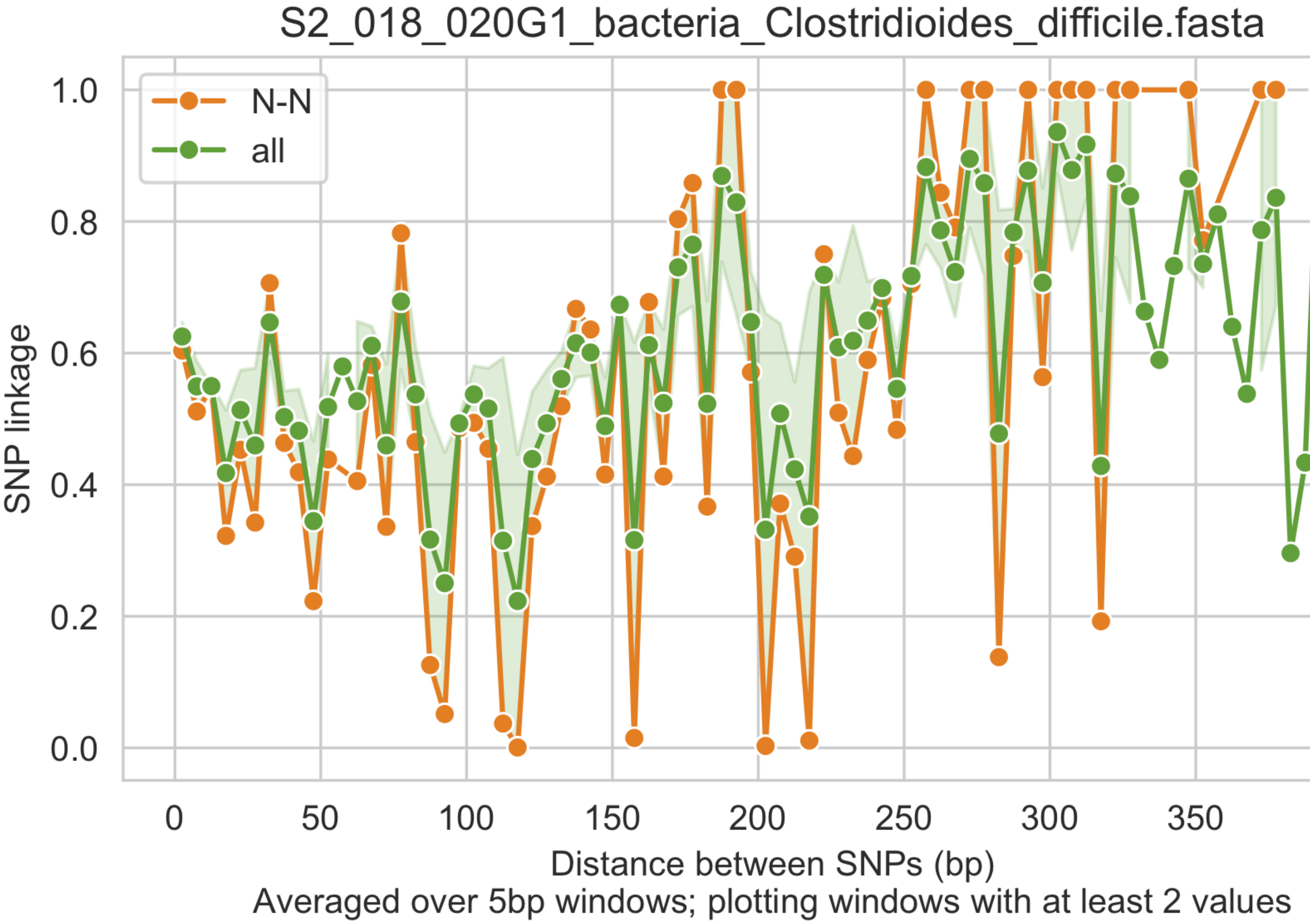
7) Scaffold inspection plot (large)

This is an elongated version of the genome-wide microdiversity metrics that is long enough for you to read scaffold names on the y-axis


C
SNV linkage
1.0
0.8
0.6
0.4
0.2
r^2
r^2 (normalized)
d′
d′(normalized)
0
20
40
60
80
100
120
Distance between SNVs (bp)


S2_002_005G1_phage_Clostridioides_c
Total mapping reads (divided by 2)
Total mapped pairs
Pairs passing min insert size threshold
Pairs passing max insert size threshold
Pairs passing mapQ threshold
Total filtered pairs
Pairs passing ANI threshold
0
1000
2000
3000
4000
50
Number of read pairs

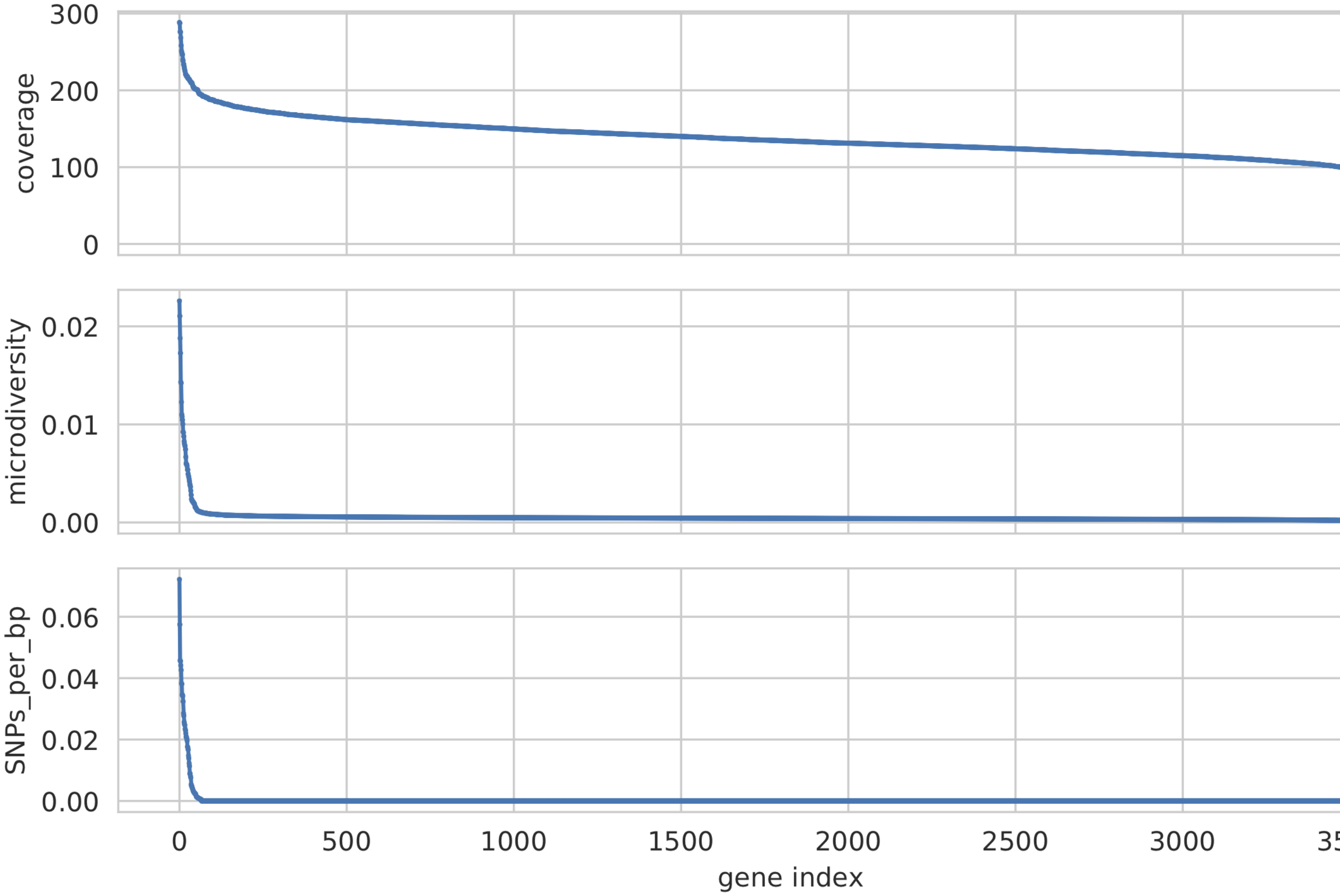

S2_002_005G1_phage_Clostridioides_difficile.fasta
snp_density
coverage
microdivers
Min read ANI (%)
0.0
100.0
JBBB007E_scaffold_233
0.000
0.002
0.004
25
50
75
0.0000
0.0005


S2_018_020G1_bacteria_Clostridioides_difficile.fasta
N-N
all
SNP linkage
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350
Distance between SNPs (bp)
Averaged over 5bp windows; plotting windows with at least 2 values

8) Linkage with SNP type (GENES REQUIRED)

Linkage plot for pairs of non-synonymous SNPs and all pairs of SNPs

9) Gene histograms (GENES REQUIRED)



Histogram of values for all genes profiled

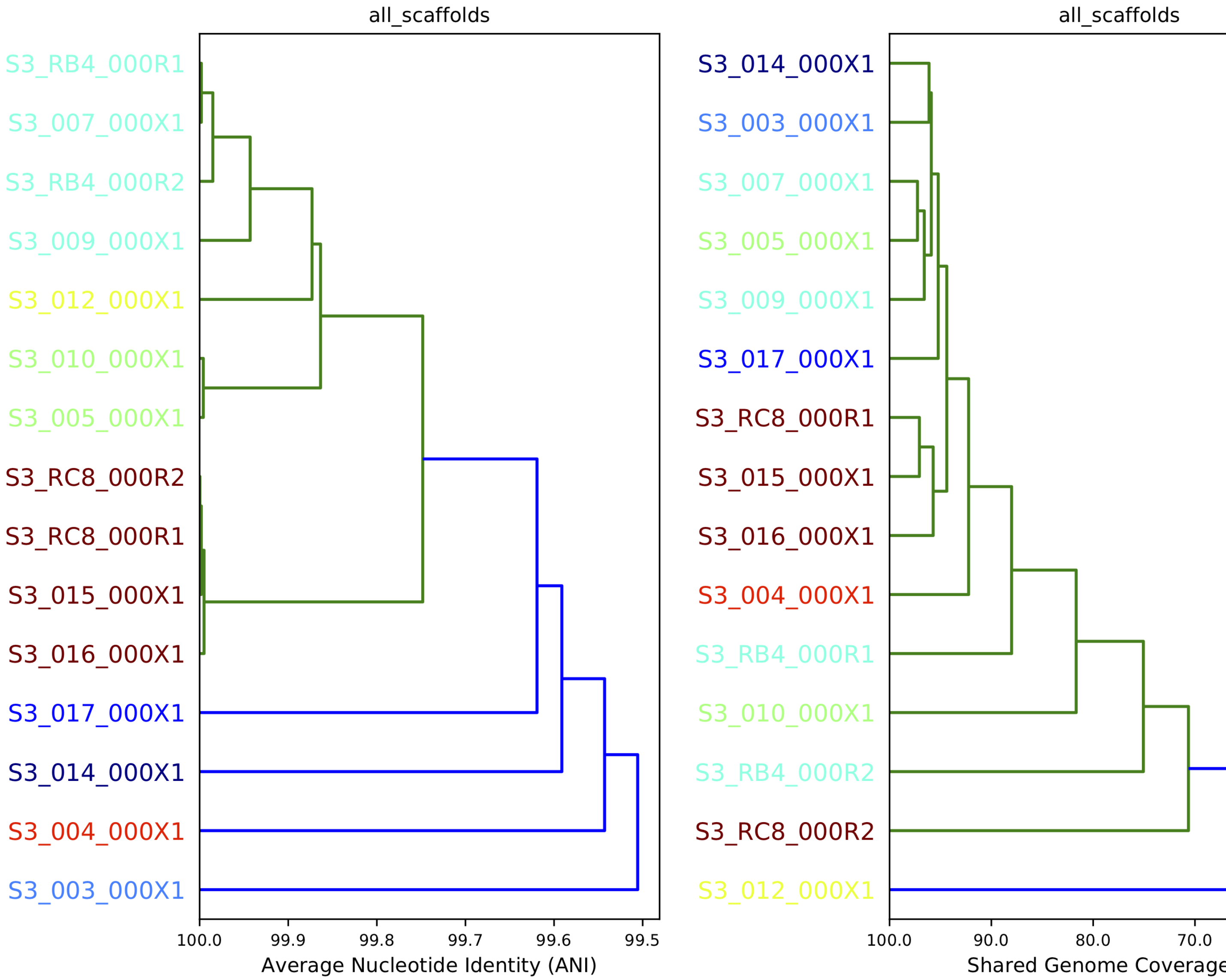
10) Compare dendrograms (RUN ON COMPARE; NOT PROFILE)

A dendrogram comparing all samples based on popANI and based on shared_bases.

1.7 Raw data access and API

1.7.1 API for accessing raw data

inStrain stores much more data than is shown in the output folder. It is kept in the `raw_data` folder, and is mostly stored in compressed formats. This data can be easily accessed using python, as described below.


all_scaffolds
S3_RB4_000R1
S3_007_000X1
S3_RB4_000R2
S3_009_000X1
S3_012_000X1
S3_010_000X1
S3_005_000X1
S3_RC8_000R2
S3_RC8_000R1
S3_015_000X1
S3_016_000X1
S3_017_000X1
S3_014_000X1
S3_004_000X1
S3_003_000X1
100.0
99.9
99.8
99.7
99.6
99.5
Average Nucleotide Identity (ANI)
all_scaffolds
S3_014_000X1
S3_003_000X1
S3_007_000X1
S3_005_000X1
S3_009_000X1
S3_017_000X1
S3_RC8_000R1
S3_015_000X1
S3_016_000X1
S3_004_000X1
S3_RB4_000R1
S3_010_000X1
S3_RB4_000R2
S3_RC8_000R2
S3_012_000X1
100.0
90.0
80.0
70.0
Shared Genome Coverage

To access the data, you first make an SNVprofile object of the inStrain output profile, and then you access data from that object. For example, the following code accessed the raw SNP table

```
import inStrain
import inStain.SNVprofile

IS = inStain.SNVprofile.SNVprofile(``/home/mattolm/inStrainOutputTest/``)
raw_snps = IS.get('raw_snp_table')
```

You can use the example above (`IS.get()`) to access any of the raw data described in the following section. There are also another special things that are accessed in other ways, as described in the section “Accessing other data”

Basics of raw_data

A typical run of inStrain will yield a folder titled “raw_data”, with lots of individual files in it. The specifics of what files are in there depend on how inStrain was run, and whether or not additional commands were run as well (like profile_genes).

There will always be a file titled “attributes.tsv”. This describes some basic information about each item in the raw data. Here’s an example

Table 24: attributes.tsv

name	value	type	description
location	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test	value	Location of SNVprofile object
version	1.3.3	value	Version of inStrain
Rdic	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/Rdic.json	dictionary	Read pair -> mismatches
mapping_info	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/mapping_info.csv.gz	pandas	Report on reads
fasta_loc	/Users/mattolm/Programs/inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa	value	Location of .fasta file used during profile
scaffold2length	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/scaffold2length.json	dictionary	Dictionary of scaffold 2 length
object_type	profile	value	Type of SNVprofile (profile or compare)
bam_loc	/Users/mattolm/Programs/inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa-vs-N5_271_010G1.sorted.bam	value	Location of .bam file
scaffold_list	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/scaffold_list.txt	list	1d list of scaffolds that were profiled
raw_linkage_table	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/raw_linkage_table.csv.gz	pandas	Raw table of linkage information
raw_snp_table	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/raw_snp_table.csv.gz	pandas	Contains raw SNP information on a mm level
cumulative_scaffold_table	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/cumulative_scaffold_table.csv.gz	pandas	Cumulative coverage on mm level. Formerly scaffoldTable.csv
cumulative_snv_table	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/cumulative_snv_table.csv.gz	pandas	Cumulative SNP on mm level. Formerly snpLocations.pickle
scaffold_2_mm_2_read_2_snvs	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/scaffold_2_mm_2_read_2_snvs.pickle	pickle	crazy nonsense needed for linkage
genes_coverage	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/genes_coverage.csv.gz	pandas	Coverage of individual genes
genes_clonality	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/genes_clonality.csv.gz	pandas	Clonality of individual genes
genes_SNP_count	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/genes_SNP_count.csv.gz	pandas	SNP density and counts of individual genes
SNP_mutation_types	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/SNP_mutation_types.csv.gz	pandas	The mutation types of SNPs
covT	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/covT.hd5	special	Scaffold -> mm -> position based coverage
clonT	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/clonT.hd5	special	Scaffold -> mm -> position based clonality
clonTR	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/clonTR.hd5	special	Scaffold -> mm -> rarefied position based clonality
genes_fileloc	/Users/mattolm/Programs/inStrain/test/test_data/N5_271_010G1_scaffold_min1000.fa.genes.fna	value	Location of genes file that was used to call genes
genes_table	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/genes_table.csv.gz	pandas	Location of genes in the associated genes_file
scaffold2bin	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/scaffold2bin.json	dictionary	Dictionary of scaffold 2 bin
bin2length	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/bin2length.json	dictionary	Dictionary of bin 2 total length
genome_level_info	/Users/mattolm/Programs/inStrain/test/test_backend/testdir/test/raw_data/genome_level_info.csv.gz	pandas	Table of genome level information

This is what the columns correspond to:

name The name of the data. This is the name that you put into `IS.get()` to have inStrain retrieve the data for you. See the section "Accessing raw data" for an example.

value This lists the path to where the data is located within the raw_data folder. If the type of data is a value, than this just lists the value

type This describes how the data is stored. Value = the data is whatever is listed under value; list = a python list; numpy = a numpy array; dictionary = a python dictionary; pandas = a pandas dataframe; pickle = a piece of data that's stored as a python pickle object; special = a piece of data that is stored in a special way that inStrain knows how to de-compress

description A one-sentence description of what's in the data.

Warning: Many of these pieces of raw data have the column "mm" in them, which means that things are calculated at every possible read mismatch level. This is often not what you want. See the section "Dealing with mm" for more information.

Accessing other data

In addition to the raw_data described above, there are a couple of other things that inStrain can make for you. You access these from methods that run on the IS object itself, instead of using the `get` method. For example:

```
import inStrain
import inStain.SNVprofile

IS = inStain.SNVprofile.SNVprofile(``/home/mattolm/inStrainOutputTest/``)
coverage_table = IS.get_raw_coverage_table()
```

The following methods work like that:

get_nonredundant_scaffold_table() Get a scaffold table with just one line per scaffold, not multiple mms

get_nonredundant_linkage_table() Get a linkage table with just one line per scaffold, not multiple mms

get_nonredundant_snv_table() Get a SNP table with just one line per scaffold, not multiple mms

get_clonality_table() Get a raw clonality table, listing the clonality of each position. Pass *nonredundant=False* to keep multiple mms

Dealing with "mm"

Behind the scenes, inStrain actually calculates pretty much all metrics for every read pair mismatch level. That is, only including read pairs with 0 mis-match to the reference sequences, only including read pairs with >= 1 mis-match to the reference sequences, all the way up to the number of mismatches associated with the "PID" parameter.

For most of the output that inStrain makes in the output folder, it removes the "mm" column and just gives the results for the maximum number of mismatches. However, it's often helpful to explore other mismatches levels, to see how parameters vary with more or less stringent mappings. Much of the data stored in "read_data" is on the mismatch level. Here's an example of what the looks like (this is the cumulative_scaffold_table):

```
,scaffold,length,breadth,coverage,coverage_median,coverage_std,bases_w_0_coverage,
↪mean_clonality,median_clonality,unmaskedBreadth,SNPs,breadth_expected,ANI,mm
0,N5_271_010G1_scaffold_102,1144,0.9353146853146853,5.106643356643357,5,2.
↪932067325774674,74,1.0,1.0,0.6145104895104895,0,0.9889923642060382,1.0,0
```

(continues on next page)

(continued from previous page)

```
1,N5_271_010G1_scaffold_102,1144,0.9353146853146853,6.421328671328672,6,4.
↪005996333777764,74,0.9992001028104149,1.0,0.6748251748251748,0,0.9965522492489882,1.
↪0,1
2,N5_271_010G1_scaffold_102,1144,0.9423076923076923,7.3627622377622375,7,4.
↪2747074564903285,66,0.9993874800638958,1.0,0.7928321678321678,0,0.998498542620078,1.
↪0,2
3,N5_271_010G1_scaffold_102,1144,0.9423076923076923,7.859265734265734,8,4.
↪748789115369562,66,0.9992251555869703,1.0,0.7928321678321678,0,0.9990314705263914,1.
↪0,3
4,N5_271_010G1_scaffold_102,1144,0.9423076923076923,8.017482517482517,8,4.
↪952541407151938,66,0.9992251555869703,1.0,0.7928321678321678,0,0.9991577528529144,1.
↪0,4
5,N5_271_010G1_scaffold_102,1144,0.9458041958041958,8.271853146853147,8,4.
↪9911156795536105,62,0.9992512780077317,1.0,0.8024475524475524,0,0.9993271891539499,
↪1.0,7
```

As you can see, the same scaffold is shown multiple times, and the last column is mm. At the row with mm = 0, you can see what the stats are when only considering reads that perfectly map to the reference sequence. As the mm goes higher, so do stats like coverage and breadth, as you now allow reads with more mismatches to count in the generation of these stats. In order to convert this files to what is provided in the output folder, the following code is run:

```
import inStrain
import inStain.SNVprofile

IS = inStain.SNVprofile.SNVprofile(``/home/mattolm/inStrainOutputTest/``)
scdb = IS.get('cumulative_scaffold_table')
ScaffDb = scdb.sort_values('mm')\
            .drop_duplicates(subset=['scaffold'], keep='last')\
            .sort_index().drop(columns=['mm'])
```

The last line looks complicated, but it's very simple what is going on. First, you sort the database by mm, with the lowest mms at the top. Next, for each scaffold, you only keep the row with the lowest mm. That's done using the `drop_duplicates(subset=['scaffold'], keep='last')` command. Finally, you re-sort the DataFrame to the original order, and remove the mm column. In the above example, this would mean that the only row that would survive would be where mm = 7, because that's the bottom row for that scaffold.

You can of course subset to any level of mismatch by modifying the above code slightly. For example, to generate this table only using reads with <=5 mismatches, you could use the following code:

```
import inStrain
import inStain.SNVprofile

IS = inStain.SNVprofile.SNVprofile(``/home/mattolm/inStrainOutputTest/``)
scdb = IS.get('cumulative_scaffold_table')
scdb = scdb[scdb['mm'] <= 5]
ScaffDb = scdb.sort_values('mm')\
            .drop_duplicates(subset=['scaffold'], keep='last')\
            .sort_index().drop(columns=['mm'])
```

Warning: You usually do not want to subset these DataFrames using something like `scdb = scdb[scdb['mm'] == 5]`. That's because if there are no reads that have 5 mismatches, as in the case above, you'll end up with an empty DataFrame. By using the drop_duplicates technique described above you avoid this problem, because in the cases where you don't have 5 mismatches, you just get the next-highest mm level (which is usually what you want)

A note for programmers

If you'd like to edit inStrain to add functionality for your data, don't hesitate to reach out to the authors of this program for help. Additionally, please consider submitting a pull request on GitHub so that others can use your changes as well.

1.8 Benchmarks

This page contains data from tests performed to evaluate the accuracy inStrain. In most cases similar tests were performed to compare inStrain's accuracy to other leading tools. Most of these benchmarks are adapted from the inStrain publication where you can find more details on how they were performed.

1.8.1 Strain-level comparisons

This section contains a series of benchmarks evaluating the ability of inStrain to perform detailed strain-level comparisons. In all cases inStrain is benchmarked against three leading tools:

MIDAS - an integrated pipeline to estimate bacterial species abundance and strain-level genomic variation. Strain-level comparisons are based on consensus alleles called on whole genomes. The script strain_tracking.py was used for benchmarks.

StrainPhlAn - a tool for strain-level resolution of species across large sample sets, based on consensus single nucleotide polymorphisms (SNPs) called on species marker genes. Based on the MetaPhlAn2 db_v20 database.

dRep - a genome alignment program. dRep does not purport to have accuracy over 99.9% *ANI* and was just used for comparison purposes.

Benchmark with synthetic data

A straightforward in silico test. A randomly selected E. coli genome was downloaded and mutated to various chosen ANI levels using SNP Mutator. The original genome was compared to the mutated genome, and we looked for the difference between the actual *ANI* and the calculated ANI (the ANI reported by each program).

All four methods performed well on this test, although dRep, inStrain, and MIDAS had lower errors in the ANI calculation than StrainPhlAn overall (0.00001%, 0.002%, 0.006% and 0.03%, respectively; average discrepancy between the true and calculated ANI). This is likely because dRep, inStrain, and MIDAS compare positions from across the entire genome (99.99998%, 99.7%, and 85.8% of the genome, respectively) and StrainPhlAn does not (0.3% of the genome).

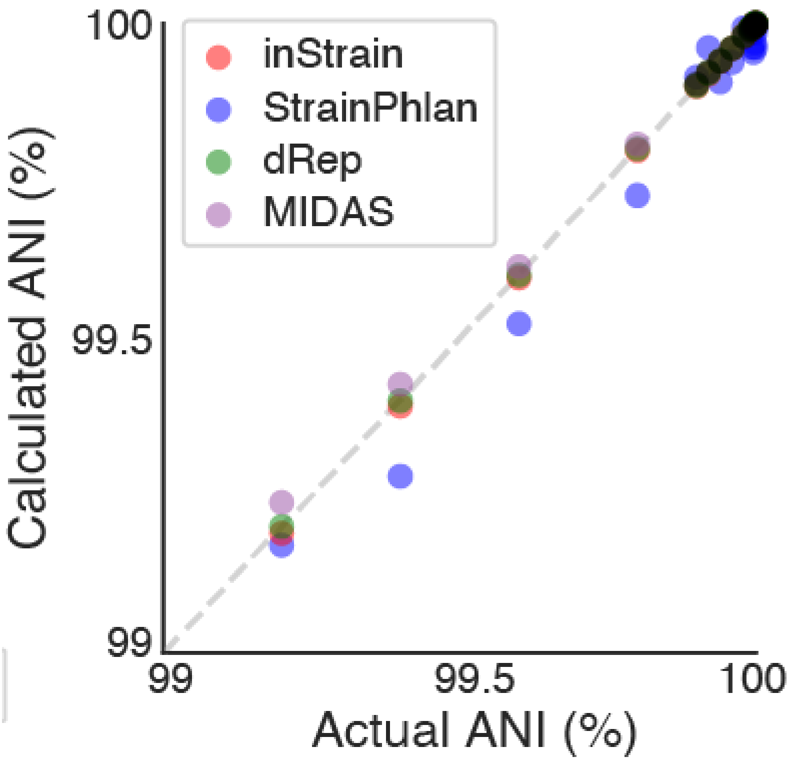
Methods for synthetic data benchmark:

For dRep, mutated genomes were compared to the reference genome using dRep on default settings. For inStrain, MIDAS, andStrainPhlAn, Illumina reads were simulated for all genomes at 20x coverage using pIRS.

For inStrain, synthetic reads were mapped back to the reference genome using Bowtie 2, profiled using "inStrain profile" under default settings, and compared using "inStrain compare" under default settings.

For StrainPhlAn, synthetic reads profiled with Metaphlan2, resulting marker genes were aligned using StrainPhlan, and the ANI of resulting nucleotide alignments was calculated using the class "Bio.Phylo.TreeConstruction.DistanceCalculator('identity')" from the BioPython python package.

For MIDAS, synthetic reads were provided to the program directly using the "run_midas.py species" command, and compared using the "run_midas.py snps" command. The ANI of the resulting comparisons was calculated as "[mean(sample1_bases, sample2_bases) - count_either] / mean(sample1_bases, sample2_bases)".



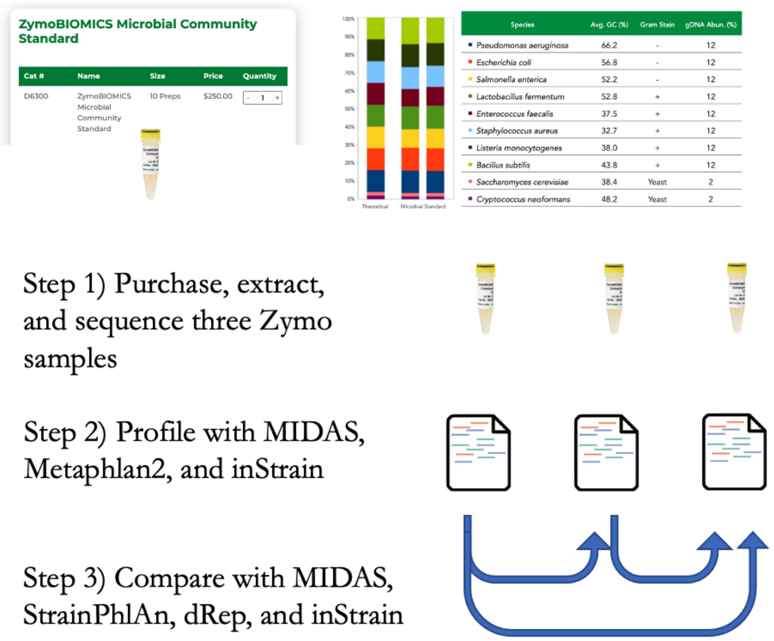
Benchmark with defined microbial communities

This test (schematic above) involved comparing real metagenomes derived from defined bacterial communities. The ZymoBIOMICS Microbial Community Standard, which contains cells from eight bacterial species at defined abundances, was divided into three aliquots and subjected to DNA extraction, library preparation, and metagenomic sequencing. **The same community of 8 bacterial species was sequenced 3 times, so each program should report 100% ANI for all species comparisons.** Deviations from this ideal either represent errors in sequence alignment, the presence of microdiversity consistent with maintenance of cultures in the laboratory, or inability of programs to handle errors and biases introduced during routine DNA extraction, library preparation, and sequencing with Illumina).

MIDAS, dRep, StrainPhlAn, and inStrain reported average ANI values of 99.97%, 99.98%, 99.990% and 99.999998%, respectively, with inStrain reporting average *popANI* values of 100% for 23 of the 24 comparisons and 99.99996% for one comparison. The difference in performance likely arises because the Zymo cultures contain non-fixed nucleotide variants that inStrain uses to confirm population overlap but that confuse the consensus sequences reported by dRep, StrainPhlAn, and MIDAS (*conANI*). We also used this data to establish a threshold for the detection of “same” versus “different” strains. The thresholds for MIDAS, dRep, StrainPhlAn, and inStrain, calculated based on the comparison with the lowest average ANI across all 24 sequence comparisons, are shown in the table below.

Program	Minimum reported ANI	Years divergence
MIDAS	99.92%	3771
dRep	99.94%	2528
StrainPhlAn	99.97%	1307
InStrain	99.99996%	2.2

Years divergence was calculated from “minimum reported ANI” using the previously reported rate of 0.9 *SNSs* accumulated per genome per year in the gut microbiome of healthy human adults (Zhao 2019) . **This benchmark demonstrates that inStrain can be used for detection of identical microbial strains with a stringency that is substantially higher than other tools**. Stringent thresholds are useful for strain tracking, as strains that have diverged



ZymoBIOMICS Microbial Community Standard

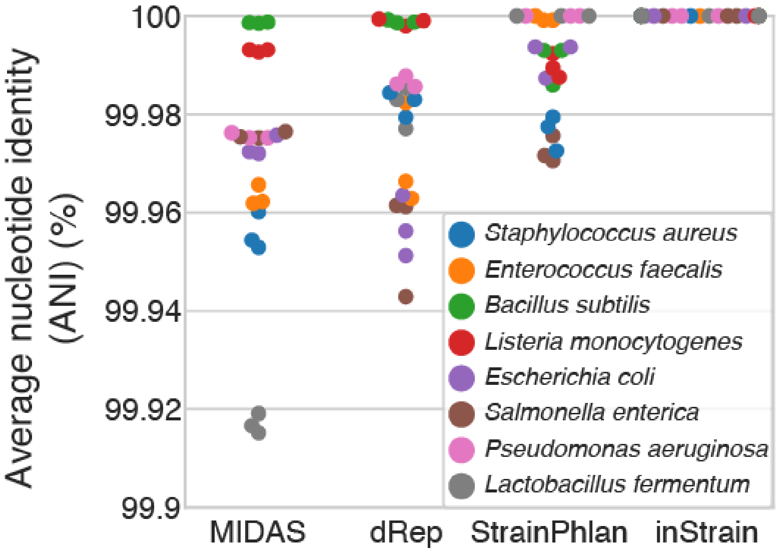
Cat #	Name	Size	Price	Quantity
D6300	ZymoBIOMICS Microbial Community Standard	10 Preps	$250.00	- 1 +

Species	Avg. GC (%)	Gram Stain	gDNA Abun. (%)
■ *Pseudomonas aeruginosa*	66.2	-	12
■ *Escherichia coli*	56.8	-	12
■ *Salmonella enterica*	52.2	-	12
■ *Lactobacillus fermentum*	52.8	+	12
■ *Enterococcus faecalis*	37.5	+	12
■ *Staphylococcus aureus*	32.7	+	12
■ *Listeria monocytogenes*	38.0	+	12
■ *Bacillus subtilis*	43.8	+	12
■ *Saccharomyces cerevisiae*	38.4	Yeast	2
■ *Cryptococcus neoformans*	48.2	Yeast	2

Step 1) Purchase, extract, and sequence three Zymo samples

Step 2) Profile with MIDAS, Metaphlan2, and inStrain

Step 3) Compare with MIDAS, StrainPhlAn, dRep, and inStrain



for hundreds to thousands of years are clearly not linked by a recent transmission event.

We also performed an additional benchmark with this data on inStrain only. InStrain relies on representative genomes to calculate *popANI*, so we wanted to know whether using non-ideal reference genomes would impact it's accuracy. By mapping reads to all 4,644 representative genomes in the Unified Human Gastrointestinal Genome (UHGG) collection we identified the 8 relevant representative genomes. These genomes had between 93.9% - 99.6% ANI to the organisms present in the Zymo samples. InStrain comparisons based on these genomes were still highly accurate (average 99.9998% ANI, lowest 99.9995% ANI, limit of detection 32.2 years), highlighting that inStrain can be used with reference genomes from databases when sample-specific reference genomes cannot be assembled.

Methods for defined microbial community benchmark:

Reads from Zymo samples are available under BioProject PRJNA648136

For dRep, reads from each sample were assembled independently using IDBA_UD, binned into genomes based off of alignment to the provided reference genomes using nucmer, and compared using dRep on default settings.

For StrainPhlAn, reads from Zymo samples profiled with Metaphlan2, resulting marker genes were aligned using StrainPhlan, and the ANI of resulting nucleotide alignments was calculated as described in the synthetic benchmark above.

For MIDAS, reads from Zymo samples were provided to MIDAS directly and the ANI of sample comparisons was calculated as described in the synthetic benchmark above.

For inStrain, reads from Zymo samples were aligned to the provided reference genomes using Bowtie 2, profiled using "inStrain profile" under default settings, and compared using "inStrain compare" under default settings. *popANI* values were used for inStrain.

Eukaryotic genomes were excluded from this analysis, and **raw values are available in Supplemental Table S1 of the inStrain manuscript**. To evaluate inStrain when using genomes from public databases, all reference genomes from the UHGG collection were downloaded and concatenated into a single .fasta file. Reads from the Zymo sample were mapped against this database and processed with inStrain as described above. The ability of each method to detect genomes was performed using all Zymo reads concatenated together.

Benchmark with true microbial communities

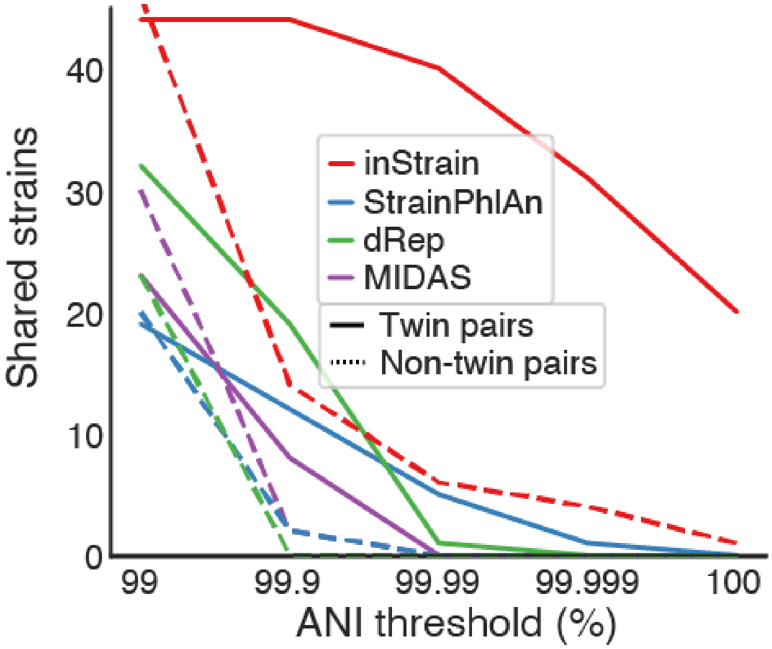
This test evaluated the stringency with which each tool can detect shared strains in genuine microbial communities. Tests like this are hard to perform because it is difficult to know the ground truth. We can never *really* know whether two true genuine communities *actually* share strains. For this test we leveraged the fact that new-born siblings share far more strains than unrelated newborns.. In this test, **we compared the ability of the programs to detect strains shared by twin premature infants (presumably True Positives) vs. their detection of strains shared by unrelated infants (presumably False Positives).**

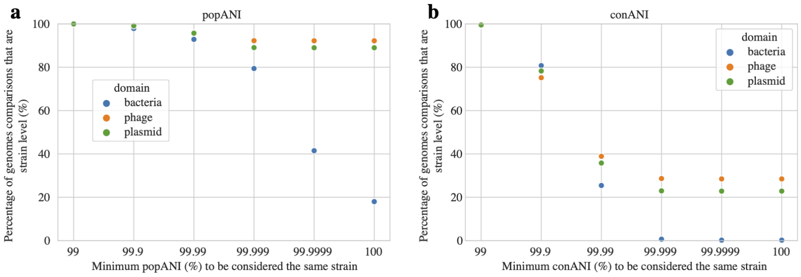
All methods identified significantly more strain sharing among twin pairs than pairs of unrelated infants, as expected, and inStrain remained sensitive at substantially higher ANI thresholds than the other tools. The reduced ability of StrainPhlAn and MIDAS to identify shared strains is likely based on their reliance on consensus-based ANI (*conANI*) measurements. We know that microbiomes can contain multiple coexisting strains, and when two or more strains of a species are in a sample at similar abundance levels it can lead to pileups of reads from multiple strains and chimeric sequences. The *popANI* metric is designed to account for this complexity.

It is also worth discussing Supplemental Figure S5 from the inStrain manuscript here.

This figure was generated from genomic comparisons between genomes present in the same infant over time (longitudinal data). In cases where the same genome was detected in multiple time-points over the time-series sampling of an infant, the percentage of comparisons between genomes that exceed various popANI (a) and conANI (b) thresholds is plotted. This figure shows that the use of popANI allows greater stringency than conANI.

Note: Based on the data presented in the above tests, a threshold of 99.999% popANI was chosen as the threshold to


Shared strains
0
10
20
30
40
99
99.9
99.99
99.999
100
ANI threshold (%)
inStrain
StrainPhlAn
dRep
MIDAS
Twin pairs
Non-twin pairs


a
popANI
Percentage of genomes comparisons that are strain level (%)
0
20
40
60
80
100
99
99.9
99.99
99.999
99.9999
100
Minimum popANI (%) to be considered the same strain
domain
bacteria
phage
plasmid
b
conANI
Percentage of genomes comparisons that are strain level (%)
0
20
40
60
80
100
99
99.9
99.99
99.999
99.9999
100
Minimum conANI (%) to be considered the same strain
domain
bacteria
phage
plasmid

define bacterial, bacteriophage, and plasmid strains for the work presented in the inStrain manuscript. This is likely a good threshold for a variety of communities.

Methods for true microbial community benchmark:

Twin-based comparisons were performed on three randomly chosen sets of twins that were sequenced during a previous study (Olm 2019). Reads can be found under Bioproject PRJNA294605

For StrainPhlAn, all reads sequenced from each infant were concatenated and profiled using Metaphlan2, compared using StrainPhlAn, and the ANI of resulting nucleotide alignments was calculated as described for the synthetic benchmark.

For MIDAS, all reads sequenced from each infant were concatenated and profiled with MIDAS, and the ANI of species profiled in multiple infants was calculated as described for the synthetic benchmark.

For dRep, all de-replicated bacterial genomes assembled and binned from each infant (available from (Olm 2019)) were compared in a pairwise manner using dRep under default settings.

For inStain, strain-sharing from these six infants was determined using the methods described below.

ANI values from all compared genomes and the number of genomes shared at a number of ANI thresholds are available for all methods in Supplemental Table S1 of the inStrain publication.

1.8.2 Species-level community profiling

This section contains tests evaluating the ability of inStrain and other tools to accurately profile microbial communities. Here inStrain is benchmarked against two other tools:

MIDAS - an integrated pipeline to estimate bacterial species abundance and strain-level genomic variation.

MetaPhlAn 2 - a computational tool for profiling the composition of microbial communities from metagenomic shotgun sequencing data. MetaPhlAn 2 uses unique clade-specific marker genes.

Benchmark with defined microbial communities

This test evaluated the ability of programs to identify the microbial species present in metagenomes of defined bacterial communities. For this test we purchased, extracted DNA from, and sequenced a ZymoBIOMICS Microbial Community Standard. The reads used for this test are available here. This community contains 8 defined bacterial species, and we simply evaluated the ability of each program to identify those and only those 8 bacterial species. Results in the table below.

Program	True species detected	False species detected	Accuracy
MIDAS	8	15	35%
MetaPhlAn 2	8	11	42%
InStrain	8	0	100%

All programs successfully identified the 8 bacteria present in the metagenome, but MIDAS and StrainPhlAn detected an additional 15 and 11 bacterial species as well. The raw tables produced by each tool are available at the bottom of this section. Looking at these tables, you'll notice that many of these False positive species detected are related to species that are actually present in the community. For example, MetaPhlAn2 reported the detection of *Bacillus cereus thuringiensis* (False Positive) as well as the detection of Bacillus subtilis (True Positive). Similarly, MIDAS reported the detection of Escherichia fergusonii (related to True Positive Escherichia coli) and Bacillus anthracis (related to True Positive Bacillus subtilis).

Importantly inStrain detected many of these same False Positives as well. However inStrain also provides a set of other metrics that properly filter out erroneous detections. Taking a look at the information reported by inStrain (at the very bottom of this page) shows that many genomes besides the 8 True Positives were detected. When using the recommended genome breadth cutoff of 50%, only the 8 True Positive genomes remain (see section "Detecting organisms in metagenomic data" in *Important concepts* for more info). You'll notice that no such info is reported with MIDAS or MetaPhlAn 2. While relative abundance could conceivably be used to filter out erroneous taxa with these tools, doing so would majorly limit their ability to detect genuine low-abundance taxa.

It's also worth noting that if one is just interested in measuring community presence / absence, as in this test, any program that accurately reports *breadth* should give similar results to inStrain when mapped against the UHGG genome set. One such program is coverM, a fast program for calculating genome coverage and breadth that can be run on its own or through inStrain using the command `inStrain quick_profile`.

Methods for defined microbial community profiling experiment:

For inStrain, all reference genomes from the UHGG collection were downloaded and concatenated into a single .fasta file, reads from the Zymo sample were mapped against this database, and inStrain profile was run on default settings.

Note: The UHGG genome database used for this section is available for download in the *Tutorial* section.

For MIDAS, the command `run_midas.py species` was used along with the default database. In cases where the same species was detected multiple times as part of multiple genomes, the species was only counted once.

For MetaPhlAn 2, the command `metaphlan2.py` was used along with the MetaPhlAn2 db_v20 database.

Eukaryotic genomes were excluded from this analysis.

Raw data for defined microbial community profiling experiment:

MetaPhlAn 2:

Table 25: metaphlan2/S3_CON_017Z2_profile.txt

species	abundance	Metaphlan2_species
Lactobacillus fermentum	23.1133	k__Bacteria|p__Firmicutes|c__Bacilli|o__Lactobacillales|f__Lactobacillaceae|g__Lactobacillus|s__Lactob
Escherichia coli	20.0587	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__
Salmonella enterica	18.44954	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__
Pseudomonas aeruginosa	14.42109	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Pseudomonadales|f__Pseudomonadaceae|g_
Enterococcus faecalis	12.21137	k__Bacteria|p__Firmicutes|c__Bacilli|o__Lactobacillales|f__Enterococcaceae|g__Enterococcus|s__Entero
Staphylococcus aureus	6.36267	k__Bacteria|p__Firmicutes|c__Bacilli|o__Bacillales|f__Staphylococcaceae|g__Staphylococcus|s__Staphy
Bacillus subtilis	2.44228	k__Bacteria|p__Firmicutes|c__Bacilli|o__Bacillales|f__Bacillaceae|g__Bacillus|s__Bacillus_subtilis
Listeria monocytogenes	1.8644	k__Bacteria|p__Firmicutes|c__Bacilli|o__Bacillales|f__Listeriaceae|g__Listeria|s__Listeria_monocytogen
Salmonella unclassified	0.67363	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__
Saccharomyces cerevisiae	0.20426	k__Eukaryota|p__Ascomycota|c__Saccharomycetes|o__Saccharomycetales|f__Saccharomycetaceae|g__S
Cryptococcus neoformans	0.05417	k__Eukaryota|p__Basidiomycota|c__Tremellomycetes|o__Tremellales|f__Tremellaceae|g__Filobasidiella
Listeria unclassified	0.02341	k__Bacteria|p__Firmicutes|c__Bacilli|o__Bacillales|f__Listeriaceae|g__Listeria|s__Listeria_unclassified
Klebsiella oxytoca	0.0165	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__
Naumovozyma unclassified	0.01337	k__Eukaryota|p__Ascomycota|c__Saccharomycetes|o__Saccharomycetales|f__Saccharomycetaceae|g__N
Klebsiella unclassified	0.01307	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__
Bacillus cereus thuringiensis	0.00809	k__Bacteria|p__Firmicutes|c__Bacilli|o__Bacillales|f__Bacillaceae|g__Bacillus|s__Bacillus_cereus_thuri
Clostridium perfringens	0.00554	k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Clostridiaceae|g__Clostridium|s__Clostridiu
Eremothecium unclassified	0.00319	k__Eukaryota|p__Ascomycota|c__Saccharomycetes|o__Saccharomycetales|f__Saccharomycetaceae|g__E
Veillonella parvula	0.0015	k__Bacteria|p__Firmicutes|c__Negativicutes|o__Selenomonadales|f__Veillonellaceae|g__Veillonella|s__V
Clostridium butyricum	0.00054	k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Clostridiaceae|g__Clostridium|s__Clostridiu
Enterobacter cloacae	0.00051	k__Bacteria|p__Proteobacteria|c__Gammaproteobacteria|o__Enterobacteriales|f__Enterobacteriaceae|g__

MIDAS:

Table 26: S3_CON_017Z2_MIDAS/species/species_profile.txt

species_id	count_reads	coverage	relative_abundance	species
Lactobacillus_fermentum_54035	22305	322.661072	0.202032	Lactobacillus fermentum
Salmonella_enterica_58156	18045	296.117276	0.185412	Salmonella enterica
Escherichia_coli_58110	19262	286.702733	0.179517	Escherichia coli

Continued on next page

Table 26 – continued from previous page

Pseudomonas_aeruginosa_57148	14214	214.266462	0.134162	Pseudomonas aeruginosa
Enterococcus_faecalis_56297	12382	183.37939	0.114822	Enterococcus faecalis
Staphylococcus_aureus_56630	6146	89.116402	0.0558	Staphylococcus aureus
Bacillus_subtilis_57806	3029	44.275375	0.027723	Bacillus subtilis
Salmonella_enterica_58266	3027	41.774295	0.026157	Salmonella enterica
Listeria_monocytogenes_53478	2250	33.367947	0.020893	Listeria monocytogenes
Escherichia_fergusonii_56914	2361	33.034998	0.020685	Escherichia fergusonii
Pseudomonas_aeruginosa_55861	927	12.402473	0.007766	Pseudomonas aeruginosa
Salmonella_enterica_53987	791	10.982231	0.006876	Salmonella enterica
Escherichia_coli_57907	713	9.860496	0.006174	Escherichia coli
Escherichia_albertii_56276	457	6.543769	0.004097	Escherichia albertii
Citrobacter_youngae_61659	455	6.248948	0.003913	Citrobacter youngae
Salmonella_bongori_55351	314	4.187424	0.002622	Salmonella bongori
Staphylococcus_aureus_37016	62	0.907418	0.000568	Staphylococcus aureus
Klebsiella_oxytoca_54123	29	0.418764	0.000262	Klebsiella oxytoca
Bacillus_sp_58480	17	0.233451	0.000146	Bacillus sp
Clostridium_perfringens_56840	12	0.182686	0.000114	Clostridium perfringens
Listeria_monocytogenes_56337	11	0.162597	0.000102	Listeria monocytogenes
Bacillus_subtilis_55718	9	0.127828	0.00008	Bacillus subtilis
Bacillus_anthracis_57688	2	0.031576	0.00002	Bacillus anthracis
Bacillus_cereus_58113	1	0.014684	0.000009	Bacillus cereus
Enterococcus_faecium_56947	1	0.014791	0.000009	Enterococcus faecium
Klebsiella_pneumoniae_54788	1	0.014852	0.000009	Klebsiella pneumoniae
Veillonella_parvula_57794	1	0.014925	0.000009	Veillonella parvula
Haemophilus_haemolyticus_58350	1	0.01351	0.000008	Haemophilus haemolyticus
Veillonella_parvula_58184	1	0.012646	0.000008	Veillonella parvula
Enterobacter_sp_59441	1	0.003478	0.000002	Enterobacter sp
Pseudomonas_sp_59807	1	0.003203	0.000002	Pseudomonas sp

InStrain:

Table 27: S3_CON_017Z2.genomeInfo.csv

gen	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Salmonella enterica	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Pseudomonas aeruginosa	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Escherichia coli	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Lactobacillus fermentum	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Enterococcus faecalis	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Staphylococcus aureus	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Escherichia fergusonii	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Listeria monocytogenes	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Bacillus subtilis	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Escherichia sp000208585	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GU	Escherichia al-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

1.9 Acknowledgements

1.9.1 People

InStrain was developed by Matt Olm and Alex Crits-Christoph in the Banfield Lab at the University of California, Berkeley.

Special thanks to all those who have provided feedback on GitHub and otherwise, especially early adopters Keith Bouma-Gregson, Ben Siranosian, Yue “Clare” Lou, Naïma Madi, and Antônio Pedro Camargo.

Many of the ideas in *Important concepts* were honed over many years during countless discussions with members of the Banfield Lab, the Sonnenburg Lab, and others. Special thanks to Christopher Brown, Keith Bouma-Gregson, Yue “Clare” Lou, Spencer Diamond, Alex Thomas, Patrick West, Alex Jaffe, Bryan Merrill, Matt Carter, and Dylan Dahan.

1.9.2 Software

InStrain relies on several previously published programs and python modules to run - see here and here for a full list of dependencies. Of special importance are samtools (the basis for parsing .bam files) and coverM (the heart of `quick_profile`).

While not a direct dependency, the open-source program anvi’o was used as significant inspiration for several implementation details, especially related to multiprocessing efficiency and memory management.

1.9.3 Citation

The manuscript describing inStrain is available in Nature Biotechnology and on bioRxiv and can be cited as follows:

```
Olm, M.R., Crits-Christoph, A., Bouma-Gregson, K., Firek, B.A., Morowitz, M.J.,␣
↪Banfield, J.F., 2021. inStrain profiles population microdiversity from metagenomic␣
↪data and sensitively detects shared microbial strains. Nature Biotechnology. https:/
↪/doi.org/10.1038/s41587-020-00797-0
```

Index