StrainGE

Lucas van Dijk, Bruce Walker, Tim Straub, Colin Worby, Alexandra

Dec 02, 2021

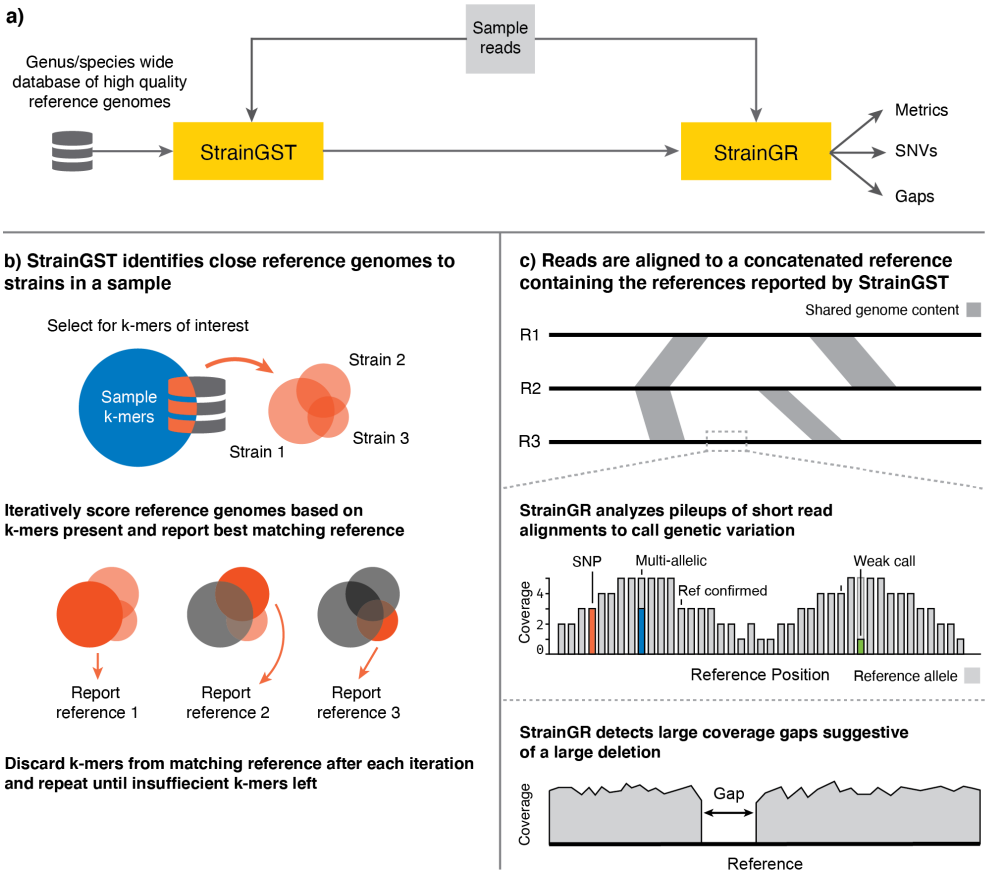
CONTENTS

CHAPTER

ONE

A TOOLKIT TO TRACK AND CHARACTERIZE LOW-ABUNDANCE STRAINS USING METAGENOMIC DATA

StrainGE is a set of tools to analyse conspecific strain diversity in bacterial populations. It consists of two main components: 1) Strain Genome Search tool (StrainGST), a tool to find close reference genomes to strain(s) present in a sample and 2) Strain Genome Recovery (StrainGR), a tool to perform strain-aware variant calling at low coverages, which in turn can be used to track strains across samples.



CHAPTER

TWO

INSTALLATION

StrainGE requires Python >= 3.7 and depends on the following packages:

- NumPy
- SciPy
- matplotlib
- scikit-bio
- pyvcf
- pysam
- h5py
- intervaltree

These packages will be automatically installed when installing through pip.

2.1 Install through *pip*

```
pip install strainge
```

2.2 Install manually from github

1. Clone the repository

   ```
   git clone https://github.com/broadinstitute/StrainGE
   ```

2. Install StrainGE

   ```
   cd StrainGE
   python setup.py install
   ```

CHAPTER

THREE

USAGE

3.1 StrainGST database creation

3.1.1 1. Download high quality reference genomes for your genus/species of interest

This tutorial assumes you have activated the *strainge* conda environment. The first step is to obtain high quality reference genomes for your genus or species of interest, any method suffices. We've found the tool ncbi-genome-download useful, and will use that tool for this step.

For example, to download all *Escherichia* genomes:

```
mkdir ref_genomes
ncbi-genome-download bacteria -l complete -g Escherichia,Shigella -H -F all \
    -o ref_genomes
```

The `-H` flag automatically organizes all downloaded files in a nice human -readable folder structure. Besides downloading references, this command downloads all associated metadata like gene annotations too, which is useful for downstram analyses.

Next, we organize all references in a single directory using a script available in the `bin/` directory of this repository: `prepare_strainge_db.py`. This script serves two main purposes: 1) it organizes all references in a single directory, 2) it optionally splits chromosomes and plasmids into separate files. When tracking strains we're usually more interested in tracking the chromosome, and we don't want StrainGST to report a strain as present because it shares a plasmid (although its algorithm should already prevent most of those cases.)

So download the prepare_strainge_db.py script to your analysis folder, and run it as follows:

```
mkdir strainge_db
python3 prepare_strainge_db.py ref_genomes/human_readable -s \
    -o strainge_db > strainge_db/references_meta.tsv
```

The `-s` flag enables splitting chromosomes and plasmids. The file `references_meta.tsv` contains metadata on each reference (for example its accession no.)

3.1.2 2. K-merize your reference sequences

Next, we k-merize each genome:

```
for f in strainge_db/*.fa.gz; do straingst kmerize -o $f.hdf5 $f; done;
```

The FASTA files with only chromosomes have a suffix of `*.chrom.fna.gz`. For each FASTA file, there is now an accompanying HDF5 file containing the k-mer data. With `-k` you can optionally specify a different k-mer size, which by default is 23.

3.1.3 3. Compare the k-mer sets and cluster similar references

The goal of StrainGST is to identify close reference genomes to strains present in a sample. These reference genomes are in turn used for variant calling and sample comparisons. Here lies a trade-off: the reference genome should be close enough for accurate variant calling, but sample comparisons are more easy to perform when the variant calling step is done using the same reference genome, so you don't want to be too specific. Furthermore, limiting the database size reduces computational time. The database of reference genomes should cover the diversity of the species of interest but not contain too many highly similar genomes. Therefore a clustering step is performed to reduce redundancy in the database.

We remove redundant reference genomes two ways:

1. Remove reference genomes that are a near perfect subset of another genome. An example of this is an *E. coli* strain used for synthetic biology applications that was basically a K-12 strain with many genes removed.
2. Cluster closely related genomes based on k-mer similarity and pick one representative.

To do this, we need to compute the pairwise similarities between k-mer sets, and a metric to identify whether a k-mer set is a subset of another. Both can be obtained using `straingst kmersim`.

```
straingst kmersim --all-vs-all -t 4 -S jaccard -S subset strainge_db/*.hdf5 >␣
↪similarities.tsv
```

This command produces as tab separated file, where each line contains a pair of k-mer sets with their accompanying similarity scores. With the `-S` flag we enable which scoring metrics to calculate, and in this case we enable the *Jaccard* similarity and the *subset* score. The output file contains for each pair of k-mer sets the requested scores, sorted by the first scoring metric (in our case the jaccard similarity). With the parameter `-t` you specify the number of processes to spawn, to allow for parallel computation of these pairwise similarities.

We can now cluster our references using the `straingst cluster` command.

```
straingst cluster -i similarities.tsv -d -C 0.99 -c 0.90 \
   --clusters-out clusters.tsv \
    strainge_db/*.hdf5 > references_to_keep.txt
```

The cluster command reads our previously created file `similarities.tsv` to determine which references to keep. The first step is to discard any genome where more than 99% of its kmers are present in another genome, as enabled by `-d` and `-C 0.99`. Afterwards, we cluster similar genomes based on the *Jaccard* similarity between k-mersets: if the Jaccard similarity between two k-mer sets is higher than 0.90 (`-c 0.90`), those two genomes will be clustered together. For each cluster we pick one representative genome: the genome with the smallest mean distance to the other cluster members. Each genome to keep is written to `references_to_keep.txt`. With the option `--clusters-out` we specify another file where we write the clustering results. Each line in this file specifies a cluster along with its entries, separated by a tab. The genomes in the first column represent the cluster representatives. This option is optional, but can be useful for debugging purposes.

3.1.4 4. Create pan-genome k-mer database

Using our list of references, we finally create a single database file which will contain all k-mers of the given references.

```
straingst createdb -f references_to_keep.txt -o pan-genome-db.hdf5
```

Now our database lives in the file `pan-genome-db.hdf5`, created from reference sequences read from the file given by `-f`.

It is also possible to give the list of k-mer sets to include in the database as positional arguments, like in the following example:

```
straingst createdb -o pan-genome-db.hdf5 ref1.hdf5 ref2.hdf5 ...
```

Combining the two methods described above works too.

3.2 Running StrainGST

Identify close reference genome(s) to strain(s) in a sample.

3.2.1 Prerequisites

1. A pre-built database for the genus or species of interest
2. A whole metagenomic sequencing (WMS) sample

3.2.2 Usage

1. K-merize the sample reads

StrainGST iteratively compares the k-mer profiles of references in the database to the k-mers in the sample to identify close reference genomes to strains in a sample.

Our first step is to kmerize the sample reads. For example, if you have a FASTQ file named `patient1.fastq` with all reads, then we generate its corresponding k-mer set as follows:

```
straingst kmerize -k 23 -o patient1.hdf5 patient1.fastq
```

Similar to the first step of the database creation section, this will generate a HDF5 file named `patient1.hdf5` with all k-mers and their corresponding counts. Make sure the value of `k` is the same as used in the database creation step.

You can specify multiple FASTQ files to the command above, which is useful if you have paired-end reads. Furthermore, it will also automatically decompress *gzipped* files. For example, if you have gzipped paired-end FASTQ files, then the following command also works:

```
straingst kmerize -k 23 -o patient1.hdf5 \
    patient1.1.fastq.gz patient1.2.fastq.gz
```

2. Run StrainGST

We can now run `straingst run` with our database HDF5 and our sample HDF5:

```
straingst run -o results.tsv pan-genome-db.hdf5 patient1.hdf5
```

This will output a *tab separated values* (tsv) file, containing statistics about the sample k-mer set and a list of identified reference strains with accompanying metrics.

3.2.3 Output file description

Example output

```
sample    totalkmers      distinct   pkmers  pkcov   pan%
UMB11_01  2277023860      380759656  50090   6.984   1.536
i         strain          gkmers     ikmers  skmers  cov    kcov   gcov   acct   even␣
↪  spec   rapct  wscore  score
0         Esch_coli_NGF1  49631      49622   50090   0.985  7.009  6.831  0.980  0.
↪987  1.000  1.507  0.940   0.940
```

Sample statistics

The first two lines contain statistics on the whole sample.

Columns:

- *sample*: Sample name, derived from the k-mer set filename
- *totalkmers*: total number of k-mers counted in the sample, including k-mers that occur multiple times
- *distinct*: total *unique* number of k-mers
- *pkmers*: total unique number of k-mers that are also present in the database
- *pkcov*: average "coverage" (multiplicity) of each unique k-mer in the sample that is also present in the database.
- *pan%*: total number of k-mers (including duplicates) that are present in both the sample and database divided by the total number of k-mers in the sample (*totalkmers*), i.e. an estimation of the relative abundance of the species/genus of interest in this sample.

Reference strain statistics

The next lines contain the close reference genomes identified by StrainGST.

Columns:

- *i*: Iteration number
- *strain*: Reference strain name
- *gkmers*: Total number of unique k-mers in the original reference genome (or its fingerprint).
- *ikmers*: Remaining unique k-mers in the genome after discarding k-mers excluded in an earlier iteration or because their average coverage was too high
- *skmers*: Remaining unique k-mers from the sample
- *cov*: Breadth of coverage of this reference, i.e. what fraction of k-mers in the reference is also present in the sample

- *kcov*: Average depth of coverage of k-mers both present in the reference and in the sample
- *gcov*: Average depth of coverage of *all* k-mers in the reference
- *acct*: What fraction of the sample k-mers can be explained by this reference?
- *even*: Evenness of coverage. A value close to 1 indicates that the coverage is evenly distributed along the genome, a value close to zero indicates that only a small part of the genome is well covered.
- *spec*: Obsolete
- *rapct*: Estimated strain relative abundance. The relative abundance of the first strain is biased upwards because it included "core" k-mers that get discarded in next iterations.
- *wscore*: Obsolete
- *score*: Score used to rank each reference in the database at each iteration. A high score represents high confidence in this prediction. Scores cannot be compared across iterations or across samples, and it is possible that a strain in a second iteration has a higher score than the strain in the first iteration.

3.2.4 Tips and Tricks

Easily parse StrainGST file in Python:

```
from strainge.io.utils import parse_straingst

results = ['sample1.tsv', 'sample2.tsv']

for sample in results:
    print('#', sample)
    with open(sample) as f:
        for strain in parse_straingst(f):
            print(strain)  # strain is a dict with above columns
```

With sample statistics:

```
from strainge.io.utils import parse_straingst

results = ['sample1.tsv', 'sample2.tsv']

for sample in results:
    print('#', sample)
    with open(sample) as f:
        straingst_iter = iter(parse_straingst(f, return_sample_stats=True))
        sample_stats = next(straingst_iter)
        print(sample_stats)

        for strain in straingst_iter:
            print(strain)  # strain is a dict with above columns
```

3.3 Running StrainGR

Characterize strains in a metagenomic sample.

3.3.1 Prerequisites

- StrainGST results on one or more samples
- Directory containing the reference genomes used to create the StrainGST database
- BWA-MEM
- Mummer4
- Optional: Picard (for MarkDuplicates)

3.3.2 Usage

1. Prepare a concatenated reference FASTA with `straingr prepare-ref`

Our strategy to deconvolve strains in a mixture sample is to create a FASTA containing a close reference genome for each strain present in a sample, and then aligning the sample reads to this concatenated FASTA file. StrainGR provides a tool `straingr prepare-ref` to automatically create and analyze a concatenated reference genome from a list of StrainGST result files.

By including multiple reference genomes into a single FASTA file, reads with an allele specific to a strain will be placed to the optimal location in the concatenated reference. On the other hand, the reference genomes included in the concatenated reference may share (conserved) parts of their genome because they are the same species, and an aligner will be unable to unambiguously place reads in those regions. This is a trade-off: include as many reference genomes as required to deconvolve strains in a sample, without combining too closely related reference genomes such that they share a vast chunk of their genomes. StrainGR will not call variants in shared regions.

The `prepare-ref` subcommand aids in building a concatenated reference from StrainGST result files. It determines which strains have been reported by StrainGST, and performs another clustering step on the reported strains to ensure the included reference strains are not too closely related. For example, sometimes it happens that a patient has a strain that's somewhat in the middle between two reference genomes sitting next to each other on the tree. Due to stochasticity in sequencing, StrainGST may report one reference genome in one sample, while reporting the other reference in another sample with the same strain, but taken at a different time point. Here the clustering step ensures that only one of these two closely related strains gets included in the concatenated reference.

After concatenating the selected references, `prepare-ref` runs `nucmer` from the MUMmer toolkit to estimate how "repetitive" the concatenated reference is, i.e. how much sequence do the genomes concatenated share, by computing maximal exact matches of at least a configurable size within the concatenated reference. By default the minimum exact match size is 250 bp, but its recommended to change this value to the average insert size of the sample read set to most accurately estimate the actual repetitiveness. These estimates are used to normalize strain abundances in a later step.

To create a concatenated reference, use `straingr prepare-ref` as follows:

```
straingr prepare-ref -s path/to/straingst/*.tsv \
   -p "path/to/refdir/{ref}.fa.gz" \
   -S path/to/straingst_db/similarities.tsv
   -o refs_concat.fasta
```

We give multiple StrainGST TSV result files to `prepare-ref` with the `-s` flag. Usually these are all StrainGST results file belonging to a single patient, or an other related set of samples. Next, we need to specify how `prepare-ref`

can find the actual FASTA files belonging to strains reported by StrainGST, this is done using the "path-template" switch `-p`: in this given path "{ref}" will be replaced with the actual strain name. Don't forgot to use quotes, because { and } are special characters in many shells. We specify the similarities.tsv file created at the StrainGST database construction step, to reuse the calculated k-mer similarities again for clustering. The resulting concatenated reference will be written to `refs_concat.fasta`.

2. Align reads to the reference

StrainGR is built to be used with `bwa mem`, as it uses the supplied information on alternative alignment locations encoded in the `XA` SAM tag to deal with shared regions introduced by concatenating reference genomes.

The following command aligns the reads with `bwa mem` and outputs a sorted BAM file:

```
bwa mem -I 300 -t 2 refs_concat.fasta sample1.1.fq.gz sample1.2.fq.gz \
    | samtools sort -@ 2 -O BAM -o sample1.bam -

# Also create BAM index
samtools index sample1.bam
```

We specify a fixed insert size to `bwa mem`, because if the species of interest in a metagenomic sample is at low abundance, there may be not enough reads per batch for `bwa mem` to infer the mean insert size, and reads in such a batch will be marked as improperly paired. Optionally you can run `picard MarkDuplicates` on your alignment file.

3. Analyze read alignments to call variants

To call any variants in your sample run the StrainGR variant caller:

```
straingr call refs_concat.fasta sample1.bam --hdf5-out sample1.hdf5 --summary sample1.
↪tsv --tracks all
```

All variant calling data will be stored in the given HDF5 file `sample1.hdf5`. A table with summary statistics like coverage, SNP rate, gaps and more is written to `sample1.tsv`. You can also specify to output this table to a TSV file with the `-s` switch in the above command. There are more options for data output, it can output VCF files, BED tracks and more, see the CLI reference documentation below.

You can recreate many of the additional data files from the HDF5 file using `straingr view`.

3.3.3 Output files

StrainGR summary

Example output

```
ref                                              name           length    coverage␣
↪uReads  abundance  median  callable  callablePct  confirmed  confirmedPct  snps␣
↪snpPct  multi  multiPct  lowmq    lowmqPct  high    highPct  gapCount  gapLength
Esch_coli_H3                                     NZ_CP010167.1  4630919   0.247     18␣
↪     0.000      0       281       0.006        275        97.865        6     2.135␣
↪  0      0.000     308810   6.668     21045   0.454    13        447553
Esch_coli_H3                                     NZ_CP010168.1  48243     0.025     0 ␣
↪     0.000      0       0         0.000        0          0.000         0     0.000␣
↪  0      0.000     263      0.545     0       0.000    0         0
```

(continues on next page)

(continued from previous page)

```
Esch_coli_NGF1                                  NZ_CP016007.1  5026105   3.549
↪85824   0.823      3       2506998   49.880       2506921    99.997        77    0.
↪003   70     0.003     1668501  33.197    3681    0.073    1         16868
Esch_coli_NGF1                                  NZ_CP016008.1  40158     6.942
↪2458    0.008      7       39096     97.355       39094      99.995        2     0.
↪005   2      0.005     982      2.445     12      0.030    0         0
Esch_coli_NGF1                                  NZ_CP016009.1  8556      0.000     0
↪      0.000      0       0         0.000         0          0.000         0     0.000
↪   0      0.000     0        0.000     0       0.000    1         8556
Esch_coli_clone_D_i14                           NC_017652.1    5038386   1.341
↪210     0.002      0       5022      0.100        5018       99.920        4     0.
↪080   0      0.000     1792289  35.573    196601  3.902    30        575694
Esch_coli_f974b26a-5e81-11e8-bf7f-3c4a9275d6c8  NZ_LR536430.1  4975029   0.224     49
↪      0.000      0       548       0.011         548        100.000       0     0.000
↪   0      0.000     298901   6.008     18373   0.369    16        767577
Esch_coli_1190                                  NZ_CP023386.1  4900891   0.260     24
↪      0.000      0       351       0.007         334        95.157        17    4.843
↪   0      0.000     342936   6.997     24117   0.492    25        848801
Esch_coli_1190                                  NZ_CP023387.1  86147     0.000     0
↪      0.000      0       0         0.000         0          0.000         0     0.000
↪   0      0.000     0        0.000     0       0.000    1         86147
TOTAL                                           -              24754434  1.148
↪88583   0.834      0       2552296   10.310       2552190    99.996        106   0.
↪004   72     0.003     4412682  17.826    263829  1.066    87        2751196
```

Column descriptions

For each scaffold in the concatenated reference StrainGR outputs several metrics.

- *ref*: original reference genome this scaffold belongs to
- *name*: Scaffold name
- *length*: Scaffold length
- *coverage*: Average depth of coverage along this scaffold. **includes multimapped reads, and multimapped reads are counted multiple times (for each alternative alignment location)**
- *uReads*: Number of reads uniquely aligned to this scaffold
- *abundance*: Estimated relative abundance of this scaffold. Calculated by dividing the number uniquely aligned reads to this scaffold by the total number of reads uniquely aligned, normalized by the estimated repetitiveness in the `prepare-ref` stage.
- *median*: median depth of coverage
- *callable* (*callablePct*): Number (percentage) of positions in this scaffold with *strong* evidence for an allele (i.e. two good reads supporting a single allele)
- *confirmed* (*confirmedPct*): Number (percentage) of positions where there's *strong* evidence for the reference allele (does not exclude positions with multiple alleles).
- *snps* (*snpPct*): Number (percentage) of positions with strong evidence for a **single** allele **different** than the reference. Our best estimate of ANI.
- *multi* (*multiPct*): Number (percentage) of positions with strong evidence for **multiple alleles** (whether it includes the reference or not).

- *lowmq* (*lowmqPct*): Number (percentage) of positions where the majority of reads are mapped with low mapping quality, i.e. representing shared or repetitive regions.
- *high* (*highPct*): Number (percentage) of positions with abnormally high coverage.
- *gapCount*: Number of gaps predicted
- *gapLength*: Number of positions in the genome marked as gap

3.4 Comparing strains across samples

3.4.1 Prerequisites

- StrainGR call data (HDF5 files) for the samples of interest

3.4.2 Comparing strains in different samples

Strains in different samples that match the same close reference genome can be compared in more detail (at the nucleotide level) using StrainGR.

To compare strains run `straingr compare`:

```
straingr compare sample1.hdf5 sample2.hdf5 \
   -o sample1.vs.sample2.summary.tsv -d sample1.vs.sample2.details.tsv
```

`straingr compare` takes in two HDF5 files as generated by `straingr call`, and the compares the base calls in each sample for each scaffold in the concatenated reference. If different concatenated references were used for each sample, only the scaffolds the two concatenated references have in common will be compared.

3.4.3 Output file description

Summary TSV

This file contains several metrics that summarizes the comparisons of each strain (scaffold).

Warning: this file currently contains a ton of metrics, several of which are slight variations on others. In the final version of StrainGE we will likely remove a few and only keep the most relevant ones.

Columns:

- *sample1, sample2*: Sample names (from filename)
- *ref*: The name of the original reference this scaffold belongs to
- *scaffold*: scaffold name
- *length*: length of the scaffold
- *common* (*commonPct*): Number (percentage) of positions of this scaffold that's callable in both samples
- *single* (*singlePct*): Number (percentage) of positions where both samples have a single strong call (i.e. no evidence for multiple alleles)
- *singleAgree* (*singleAgreePct*): Number (percentage) of positions where both sample have single strong call, and the base call is the same. *singleAgreePct* is the *ACNI* metric as described in the paper.

- *sharedAlleles* (*sharedAllelesPct*): Number (percentage) of positions where both samples share an allele. This allows for positions to have multiple alleles, and at least one allele should match.
- *variants* (*variantsPct*): Number (percentage) of positions where either sample has an allele other than the reference.
- *commonVariant* (*commonVariantPct*): Number (percentage) of variants where both samples share an allele
- *variantExact* (*variantExactPct*): Number (percentage) of variants that are exactly the same in both samples (including the same positions with multiple alleles).
- *AnotB* (*AnotBPct*): Number (percentage) of variants in Sample A but not in Sample B
- *BnotA* (*BnotAPct*): Number (percentage) of variants in Sample B but not in Sample A
- *gapJaccardSimilarity*: Jaccard similarity between samples of set of positions **not** marked as gap (i.e. analogous to gene content similarity).

CHAPTER

FOUR

CITATION

If you use StrainGE in your project, please consider citing our publication: TODO

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search