ggCaller Documentation

Release 1.3.0

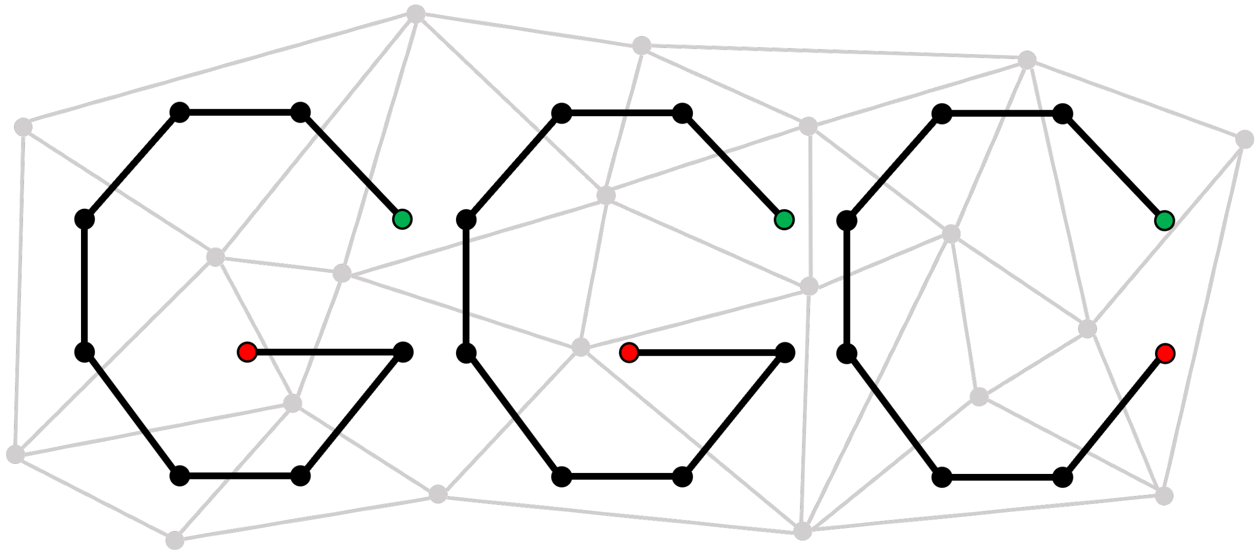
Sam Horsfield

Aug 07, 2023

CONTENTS:

ggCaller is a novel bacterial gene annotation and pangenome analysis tool, designed to enable fast, accurate analysis of large single-species genome datasets.

ggCaller traverses de Bruijn graphs (DBGs) built by Bifrost, using temporal convolutional networks from Balrog for gene filtering and Panaroo for pangenome analysis and quality control.



CHAPTER ONE

QUICKSTART

Important: We are aware of issues installing from conda version at the moment. We recommend installing via Docker at this time.

1.1 Installation

The easiest way to get up and running is using Docker. To get up and running, pull the latest image:

```
docker pull samhorsfield96/ggcaller:latest
```

1.2 Preparing the data

Place all of you samples to analyse in the same directory. Then navigate inside and run:

```
ls -d -1 $PWD/*.fasta > input.txt
```

If using Docker, instead navigate to the directory containing the fasta files and run the below command, to ensure file paths are relative (the docker version will not work with absolute paths):

```
ls -d -1 *.fasta > input.txt
```

1.3 Running ggCaller

Important: Assemblies with many Ns generate disjointed DBGs leading to underclustering. To ensure optimal performance, avoid using assemblies containing Ns.

To run ggCaller with just assemblies:

```
ggcaller --refs input.txt --out output_path
```

To run ggCaller with just reads:

```
ggcaller --reads input.txt --out output_path
```

If using Docker, run with the below command. You must ensure all paths are relative, including in `input.txt`:

```
docker run --rm -it -v $(pwd):/workdir samhorsfield96/ggcaller:latest ggcaller --refs␣
↪input.txt --out output_path
```

Important: We haven't extensively tested calling genes within read datasets yet. Exercise caution when interpreting results.

Results will be saved to the directory `ggCaller_output` by default. To change this, specify `--out <path>`.

CHAPTER TWO

INSTALLATION

ggCaller is available on Linux. If you are running Windows 10/11, Linux can be installed via the Windows Subsystem for Linux (WSL). If running via Docker, ensure you install WSL2.

We plan to get a MacOS version up and running in the future.

Important: ggCaller requires python3.9 to run (which on many default Linux installations is run using `python3` rather than `python`).

2.1 Installing with Docker (recommended)

First, install Docker for your OS. If running with WSL2, you should still download Docker Desktop for Windows.

To use the latest image, run:

```
docker pull samhorsfield96/ggcaller:latest
```

To run ggCaller from the Docker Hub image, run:

```
cd test && docker run --rm -it -v $(pwd):/workdir samhorsfield96/ggcaller:latest␣
↪ggcaller --refs pneumo_CL_group2.txt
```

You can also build the image yourself. First download and switch to the ggCaller repository:

```
git clone --recursive https://github.com/samhorsfield96/ggCaller && cd ggCaller
```

Finally, build with Docker. This should take between 5-10 minutes to fully install.:

```
docker build -t ggc_env:latest -f docker/Dockerfile .
```

To run ggCaller from a local Docker build, run:

```
cd test && docker run --rm -it -v $(pwd):/workdir ggc_env:latest ggcaller --refs pneumo_
↪CL_group2.txt
```

2.2 Installing with singularity

If you encounter permissions issues using Docker, you can download the singularity image from Zenodo

Once downloaded, set up the singularity container using:

```
singularity shell --writable <singulatiry image>.sif
```

Once loaded, add the conda bin directory to your path variable and run ggCaller as normal:

```
PATH=$PATH:/opt/conda/bin
ggcaller --refs input.txt --out output_path
```

2.3 Installing with conda

Important: We are aware of issues installing from conda at the moment. We recommend installing from docker or source at this time.

Installing with conda is the easiest way to get ggCaller up and running, and will install all dependencies.

If you do not have `conda` you can install it through miniconda and then add the necessary channels:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Then run:

```
conda install ggcaller
```

2.4 Installing from source

You can also clone the github code to run the latest version.

You will need to install the dependencies yourself (you can still use conda or pip for this purpose). See `environment_linux.yml` or `environment_macOS.yml`. In addition, a C++17 compiler (e.g. gcc >=7.3) is required.

We highly recommend using mamba over conda due to the large number of dependencies, making mamba significantly faster.

To install dependencies (creates `` `ggc_env` `` environment):

```
mamba env create -f environment_linux.yml
mamba activate ggc_env
```

Then clone the code and install:

```
git clone --recursive https://github.com/samhorsfield96/ggCaller && cd ggCaller
python setup.py install
```

2.5 Test installation

After any of the above steps, check correct setup by running `ggcaller --help`.

CHAPTER

THREE

USAGE

ggCaller has two main modes: *Gene-calling* and *Querying*.

3.1 Gene-calling

Gene-calling predicts and annotates genes within a pangenome de Bruijn Graph (DBG), before conducting orthologue clustering and pangenome analysis using Panaroo.

3.1.1 Predicting genes

To generate an input for ggCaller, create a directory containing of all the sequences you wish to analyses. We recommend placing all samples of the same type in a single directory; place read and assembly files in separate directories.

Important: Ensure you have write access to the directories where the FASTA/FASTQ files are saved, as ggCaller saves intermediate FMINDEX files in the same locations.

If not using Docker, generate the input file for ggCaller, navigate inside the directory containing the genomes, and run:

```
ls -d -1 $PWD/*.fasta > input.txt
```

If using Docker, you must navigate to the directory containing the fasta files and run:

```
ls -d -1 *.fasta > input.txt
```

This will generate a list of all the `.fasta` files in the directory. Change this extension as required.

Important: All of the below commands can be run with docker installations, however they must be run as: `docker run --rm -it -v $(pwd):/workdir samhorsfield96/ggcaller:latest ggcaller <commands>`. This command must be run within the same directory as the *.fasta* files and *input.txt*. All paths provided must be relative, as absolute paths will not work within the docker container.

DBG building with reads or assemblies is different, with k-mers that appear only once being removed from the graph. Therefore it is important to specify whether `input.txt` contains reads or assemblies.

Important: Assemblies with many Ns generate disjointed DBGs leading to underclustering. To ensure optimal performance, avoid using assemblies containing Ns.

To run ggCaller with just assemblies:

```
ggcaller --refs input.txt
```

To run ggCaller with just reads:

```
ggcaller --reads input.txt
```

To run ggCaller with reads and assemblies:

```
ggcaller --refs input1.txt --reads input2.txt
```

Important: We haven't extensively tested calling genes within read datasets yet. Exercise caution when interpreting results.

ggCaller can also be run on a pre-built Bifrost DBG and its associated colours file:

```
ggcaller --graph input.gfa --colours colours.color.bfg
```

This assumes all sequences used to build the graph are assemblies. If only some sequences are assemblies and the rest are reads, specify which files are references using `--refs`:

```
ggcaller --graph input.gfa --colours colours.color.bfg --refs input1.txt
```

If all sequences are reads, specify `--not-ref`:

```
ggcaller --graph input.gfa --colours colours.color.bfg --not-ref
```

Results from all commands above will be saved to a directory called `ggCaller_output` by default. To change this, specify `--out <path>`. Note that ggCaller will overwrite results if an already existing directory is specified.

By default, ggCaller will generate:

- Predicted genes (nucleotide and amino-acid) in FASTA format
- Gene presence/absence matrix in CSV and RTAB formats
- Pre/post Panaroo quality control gene graphs in GML format
- Structural variant presence/absence in RTAB format
- Summary graph: gene frequency, cluster size and rarefaction curve
- Roary-style gene frequency statistics
- A pangenome reference FASTA, containing all cluster centroids
- A gene presence/absence neighbour joining tree in NWK format

Additionally, ggCaller generates some intermediate files:

- Two Bifrost files, a GFA file and BFG_COLORS file, with the same file path as `input.txt`
- FMINDEX files for each of the sample FASTAs with the same file path the input files.

3.1.2 Annotating genes

ggCaller comes with two default databases for functional annotation of genes. - Bacterial and Viral databases from Uniprot, used by DIAMOND - HMM profiles from Prokka, used by HMMER3

Important: Ensure you are connected to the internet when first running ggCaller as these databases are downloaded automatically. Subsequent runs can be conducted offline.

There are three sensitivity levels for annotation:

- `fast`: only DIAMOND in fast mode
- `sensitive`: only DIAMOND in sensitive mode
- `ultrasensitive`: HMMER3 and DIAMOND in sensitive mode

For example, to run DIAMOND only in fast mode, run:

```
ggcaller --refs input.txt --annotation fast
```

By default these commands will annotate using DIAMOND with the `Bacteria` uniprot database. To change this to the `Viruses` database, run:

```
ggcaller --refs input.txt --annotation fast --diamonddb Viruses
```

Custom databases can also be specified for both DIAMOND using `--diamonddb` and HMMER3 using `--hmmdb`. DIAMOND databases must be amino-acid FASTA files. HMMER3 databases must be HMM-profile `.HAMAP` files built using `hmmbuild` which is part of the HMMER3 package.

To run with custom DIAMOND and HMMER3 databases:

```
ggcaller --refs input.txt --annotation ultrasensitive --diamonddb annotation.fasta --
↪hmmdb annotation.HAMAP
```

Annotation is not on by default. If annotation is specified, ggCaller will additionally generate:

- GFF files for each input genome in a separate directory `GFF`
- Annotations will be added to gene call FASTA files

3.1.3 Aligning genes

ggCaller also supports generation of within-cluster and core genome alignments using MAFFT.

There are two alignment algorithms implemented:

- `def` or default, which uses the standard MAFFT multiple sequence alignment algorithm. This is faster when aligning <=500 sequences in a cluster.
- `ref` or reference, which uses reference-guided alignment. This is faster when aligning >500 sequences in a cluster.

There are also two modes for alignment:

- `core` aligns genes only within core clusters, and generates a concatenated core genome alignment.
- `pan` aligns genes within all clusters (pangenome alignment), as well as generating a concatenated core genome alignment.

To generate a core genome alignment using default MAFFT, run:

```
ggcaller --refs input.txt --aligner def --alignment core
```

To generate a pangenome alignment using reference-guided MAFFT, run:

```
ggcaller --refs input.txt --aligner ref --alignment pan
```

To change the frequency of genes deemed to be core, use *–core-threshold* (default = 0.95, or 95% frequency). For example, only include genes found at 100% frequency:

```
ggcaller --refs input.txt --aligner def --alignment core --core-threshold 1.0
```

Alignment is off by default. If specified, ggCaller will additionally generate:

- Core genome alignment in FASTA format
- Core genome Neighbour-joining tree in NWK format
- Per-cluster alignment files in FASTA format in a separate directory `aligned_gene_sequences`
- Per-cluster VCF file generated by SNP-SITES in separate directory `VCF`

3.1.4 Quality control and clustering

ggCaller implements Panaroo to identify spurious clusters that are generated by assembly fragmentation and contamination.

Panaroo identifies spurious clusters as those with <2 edges in the gene graph. Spurious clusters are then removed based on their population frequency, determined by three settings:

- `strict`; remove spurious clusters with <5% frequency. Good for datasets >100 genomes where rare plasmids are not expected.
- `moderate`; remove spurious clusters with <1% frequency (default). Good for datasets <=100 genomes where rare plasmids are not expected.
- `sensitive`; do not remove clusters. Good for datasets where rare plasmids are expected.

For example, to run ggCaller in strict mode:

```
ggcaller --refs input.txt --clean-mode strict
```

More information can be found here.

If you use the full pipeline of ggCaller, also please cite Panaroo.

3.2 Querying

Querying maps a set of query DNA sequences to an annotated DBG, identifying genes that the query overlaps with.

3.2.1 Saving datastructures

Annotate a DBG as before, adding the `--save` flag. This will write the intermediate datastructures containing DBG coordinates of the predicted genes to a directory called `ggc_data`.

Important: We suggest using an annotation database, either the default ones provided or a custom one, as this will enable better functional analysis of your queries.

For example, run with sensitive annotation and save intermediate files:

```
ggcaller --refs input.txt --annotation sensitive --save
```

3.2.2 Querying the DBG

Queries sequences can either be in multi-FASTA format, or in a single file with each sequence on its own line.

Provide paths to the DBG `.gfa` and `.color.bfg` files, the `ggc_data` directory and query file:

```
ggcaller --query queries.fasta --graph inputs.gfa --colours inputs.color.bfg --data␣
↪ggCaller_output/ggc_data
```

By default, mapped queries >=80% matching k-mers to a given colour will be returned. This can be changed using `--query-id` flag.

To return queries with 100% match:

```
ggcaller --query queries.fasta --graph inputs.gfa --colours inputs.color.bfg --data␣
↪ggCaller_output/ggc_data --query-id 1.0
```

3.2.3 Interpreting results

Results will be output in `matched_queries.fasta` in the specified output directory. This is a multi-FASTA file describing all annotated genes that overlap with the query sequences.

An example format is below:

```
>Isolate10_9298 ggcID=10_9298 QUERY=Query_A;Query_B annotation=FUNCTION A;FUNCTION B;
ATGTTAAATAAAGTCAAAACTAAAGCCTTAATTAGTGTCGGAGCAGTGGCTGCAACTAGCTAG
```

The header contains:

- Sample name and gene number (`Isolate10_9298`)
- ggCaller identifier (`ggcID` field)
- Mapped query sequences or IDs (`QUERY` field) separated by semi-colons. These will be fasta IDs if `queries` file is a FASTA, otherwise DNA sequence.
- Annotation(s) (`annotation` field) separated by semi-colons

3.3 Parallelisation

ggCaller is fully parallelised using OpenMP and python multiprocessing. By default ggCaller runs single-threaded.

To specify the number of threads:

```
ggcaller --refs input.txt --threads 8
```

CHAPTER

FOUR

TUTORIAL

Here we'll walk through a typical run of ggCaller, including both *Gene-calling* and *Querying*.

Example results can be found here.

Important: Results will be consistent, but may not exactly match between your run and the example. This is due to the greedy clustering algorithm used by ggCaller, which can cause small differences in genes counts.

4.1 Installation and setup

Follow the guide in *Installation* for downloading and installing ggCaller.

4.2 Working Dataset

We'll use a dataset from Bentley et al. (2006). This dataset contains 91 sequences pneumococcal capsular polysaccharide synthetic (CPS) loci. These sequences are structurally diverse, but are only ~20,000 bp in length, so can be analysed quickly (~5-10 minutes) on a standard laptop or desktop.

Download the files from here and unzip:

```
tar xvf Bentley_et_al_2006_CPS_sequences.tar.bz2
```

We will also provide our own custom annotation database for DIAMOND. These will be the manually curated protein sequences from Bentley et al. Download from here and unzip:

```
tar xvf Bentley_et_al_2006_CPS_protein_sequences.tar.bz2
```

4.3 Gene-calling

First generate an input file for ggCaller. This must be a file containing paths (absolute recommended) to all sequences to be analysed. We recommend running the below command within the unzipped to generate this file:

```
cd Bentley_et_al_2006_CPS_sequences
ls -d -1 $PWD/*.fa > input.txt
cd ..
```

input.txt will now contain absolute paths to all .fa files in the directory Bentley_et_al_2006_CPS_sequences.

Now we will run ggCaller specifying the below settings:

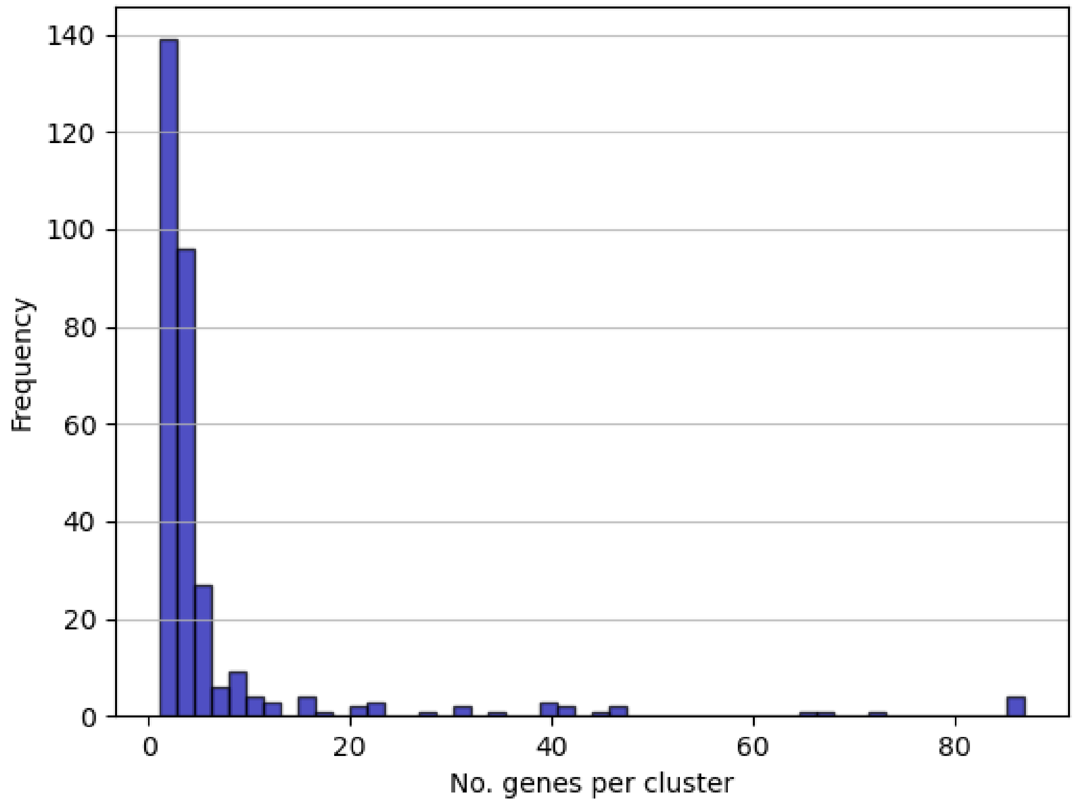
- Sensitive DIAMOND annotation using a custom database, and HMMER3 using the default database
- Pangenome-wide alignment using default MAFFT
- Saved intermediate datastructures, enabling sequence querying

To do this using 4 threads, run:

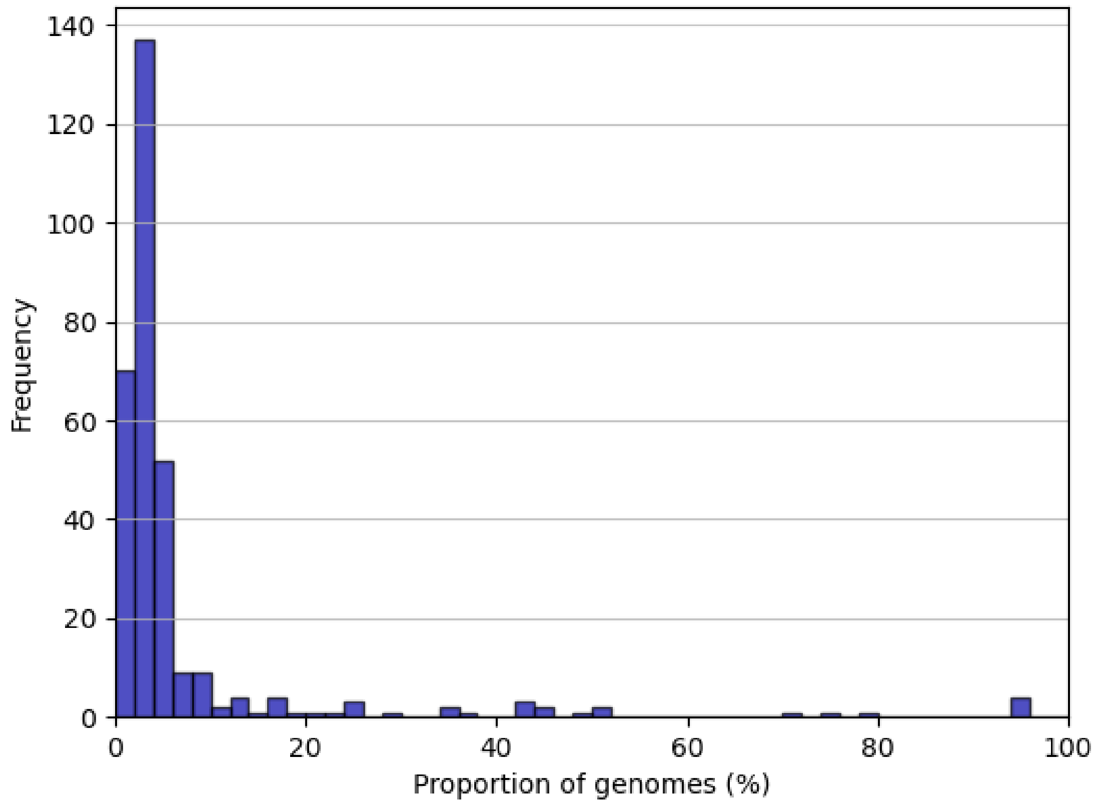
```
ggcaller --refs Bentley_et_al_2006_CPS_sequences/input.txt --annotation ultrasensitive --
↪diamonddb Bentley_et_al_2006_CPS_protein_sequences.faa --aligner def --alignment pan --
↪save --out ggc_Bentley_et_al_CPS --threads 4
```

You will find the following files in the output directory ggc_Bentley_et_al_CPS:

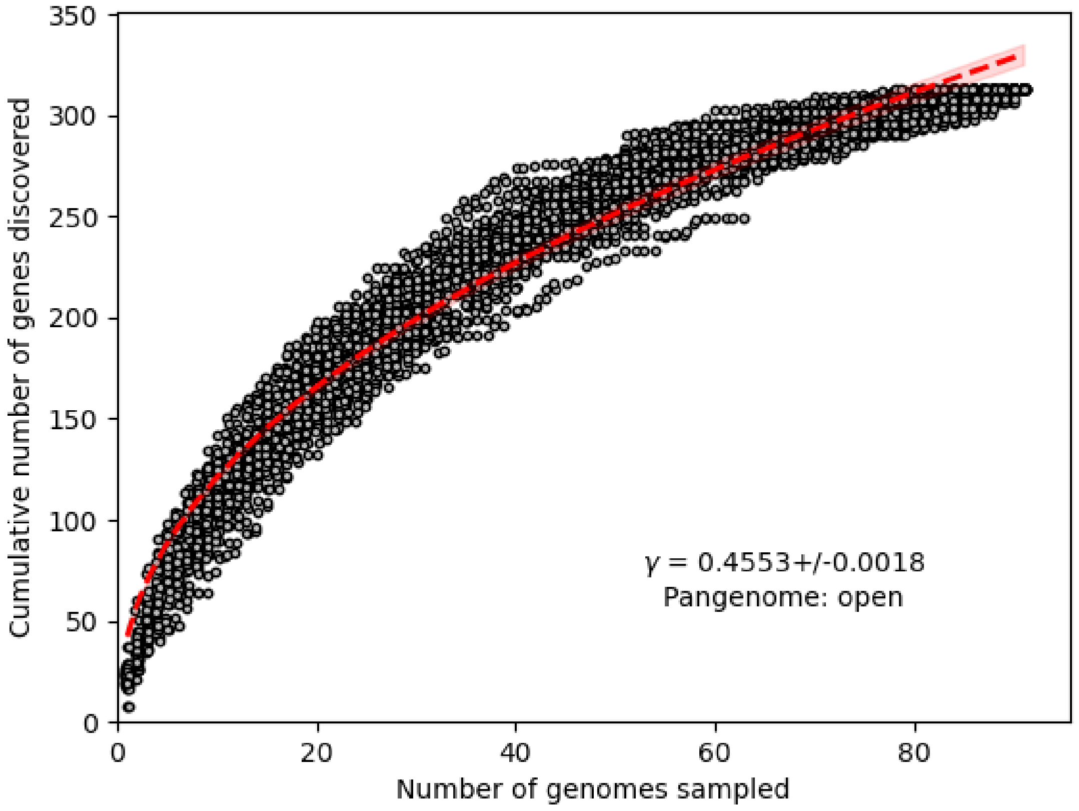
- cluster_size.png: a frequency distribution of clusters by the number of genes found within them



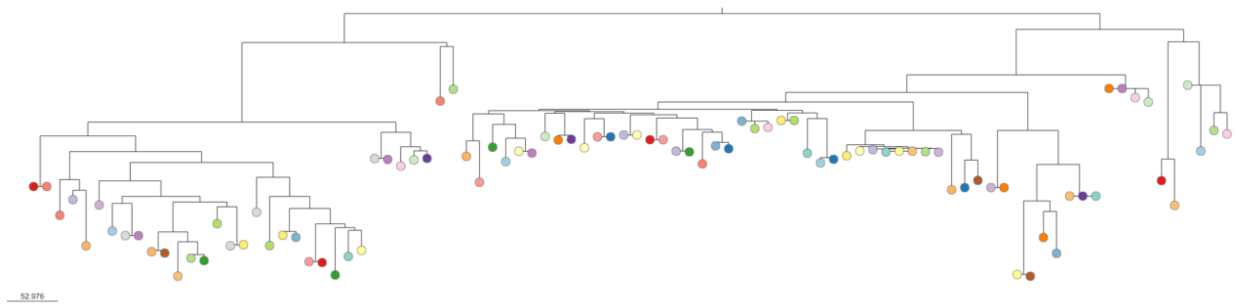
- gene_frequency.png: a frequency distribution of clusters by proportion of dataset



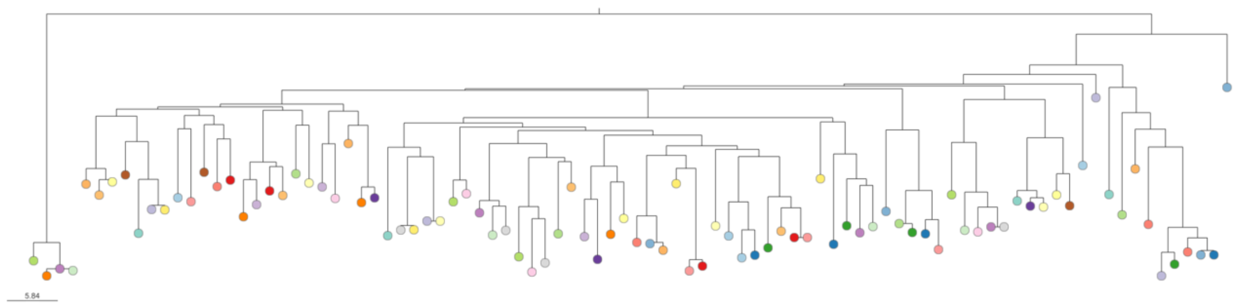
- `rarefaction_curve.png`: rarefaction curve, describes the number of new genes discovered with random addition of a single genome. Also includes power-law fit for determination of pangenome openness, based on Tettelin et al. (2005).



- `core_gene_alignment.aln`: concatenated core genome alignment
- `core_alignment_header.embl`: core genome alignment in EMBL format
- `core_tree_NJ.nwk`: Neighbour joining tree from core genome alignment generated by RapidNJ. This can be visualised in Microreact



- `pangenome_NJ.nwk`: Neighbour joining tree from gene presence/absence matrix generated by RapidNJ (can also be visualised in Microreact).



- `pan_genome_reference.fa`: contains centroids for each cluster in FASTA format
- `gene_calls.faa` and `gene_calls.ffn`: gene predictions with annotations in amino-acid and nucleotide FASTA formats
- `pre_filt_graph.gml` and `final_graph.gml`: gene graphs pre- and post-quality control with Panaroo
- `gene_presence_absence*`: gene presence absence files in three formats; Roary-CSV, CSV and Rtab
- `struct_presence_absence.Rtab`: structural variant presnce/absence matrix
- `summary_statistics.txt`: summary of gene frequencies based on Roary
- `VCF`: directory containing VCF files for each cluster generated by SNP-SITES
- `aligned_gene_sequences`: directory of alignment files for each cluster in FASTA format
- `GFF`: directory of GFF files for each sample in GFF3 format
- `ggc_data`: intermediate datastructures written to disk, required for querying.

4.4 Querying the graph

We can now query the graph. To do so, run:

```
ggcaller --query CPS_queries.fasta --graph Bentley_et_al_2006_CPS_sequences/input.gfa --
↪colours Bentley_et_al_2006_CPS_sequences/input.color.bfg --data ggc_Bentley_et_al_CPS/
↪ggc_data --out ggc_Bentley_et_al_CPS --threads 4
```

Results will be saved in `ggc_Bentley_et_al_CPS/matched_queries.fasta`.

Details on the output can be found in *Interpreting results*.

From `matched_queries.fasta`, we can see that all the genes queried were identified in the graph.

As we searched for specific gene variants, this search was too stringent to return orthologues in other genomes.

Important: We recommend searching for partial gene sequences, or lowering `--query-id` to return more distantly related sequences.

CHAPTER

FIVE

ADVANCED

For advanced users, ggCaller has a number of parameters for altering gene prediction, annotation and quality control.

5.1 Input/output

- `--kmer`: value of k used to build Bifrost DBG (Default and max value = 31).
- `--all-seq-in-graph`: Output gene graph GML file with all DNA and amino acid sequences. Off by default due to large file size.

5.2 Traversal and gene-calling cut-off settings

- `--max-path-length`: Maximum path length traversed during ORF finding (bp) (Default = 20000)
- `--min-orf-length`: Minimum ORF length to return (bp) (Default = 90)
- `--score-tolerance`: Probability threshold for shorter alternative start sites based on average stop codon frequency (Default = 0.2)
- `--max-ORF-overlap`: Maximum overlap allowed between two ORFs (bp) (Default = 60)
- `--min-path-score`: Minimum total BALROG score for a maximum tiling path of ORFs to be returned (Default = 100)
- `--min-orf-score`: Minimum individual Balrog score for an ORF to be returned (Default = 100)
- `--max-orf-orf-distance`: Maximum distance between two ORFs to be connected (bp) (Default = 10000)

5.3 Avoid/include algorithms

- `--no-filter`: Do not filter ORF calls using Balrog, will return all ORF calls (Default = False)
- `--no-write-idx`: Do not write FMIndexes to file (Default = False)
- `--no-write-graph`: Do not write Bifrost GFA and colours to file (Default = False)
- `--repeat`: Enable traversal of nodes multiple times, only applicable when DBG built from reads (Default = False)
- `--no-clustering`: Do not cluster ORFs (Default = False)
- `--no-refind`: Do not refind missed genes (Default = False)

5.4 Gene clustering options

- `--identity-cutoff`: Minimum identity at amino acid level between two ORFs for lowest-level clustering (Default = 0.98)
- `--len-diff-cutoff`: Minimum ratio of length between two ORFs for lowest-level clustering (Default = 0.98)
- `--family-threshold`: Gene family sequence identity threshold (default=0.7)
- `--merge-paralogs`: Don't split paralogs during Panaroo quality control (Default = False)

5.5 Annotation options

- `--evalue`: Maximum e-value to return for DIAMOND and HMMER searches during annotation (Default = 0.001)
- `--truncation-threshold`: Sequences in a cluster less than *centroid length * truncation-threshold* will be annotated as 'potential pseudogene' (Default = 0.8)

5.6 Gene-refinding options

- `--search-radius`: The distance (bp) surrounding the neighbour of an accessory gene in which to search for it (Default = 5000)
- `--refind-prop-match`: The proportion of an accessory gene's length that must be found in order to consider it a match (Default = 0.2)

5.7 Gene graph correction stringency options (determined by clean-mode)

- `--min-trailing-support`: Minimum cluster size to keep a gene called at the end of a contig.
- `--trailing-recursive`: Number of times to perform recursive trimming of low support nodes near the end of contigs
- `--edge-support-threshold`: Minimum support required to keep an edge that has been flagged as a possible mis-assembly
- `--length-outlier-support-proportion`: Proportion of genomes supporting a spurious long gene (>1.5x outside the IQR of cluster)
- `--min-edge-support-sv`: Minimum edge support required to call structural variants in the presence/absence sv file
- `--no-clean-edges`: Turn off edge filtering in the final output graph

5.8 Alignment options

- `--no-variants`: Do not call variants using SNP-sites after alignment (Default = False)
- `--ignore-pseduogenes`: Ignore ORFs annotated as 'potential pseudogenes' in alignments (Default = False)

5.9 Misc. options

- `--quiet`: Suppress additional output to console (Default = False)
- `--version`: Show program's version number and exit (Default = False)

CHAPTER

SIX

CITATIONS

If you use ggCaller, please cite our preprint:

Horsfield, S.T., Croucher, N.J., Lees, J.A. "Accurate and fast graph-based pangenome annotation and clustering with ggCaller" bioRxiv 2023.01.24.524926 (2023). doi: https://doi.org/10.1101/2023.01.24.524926

ggCaller relies on a number of other tools. In addition, please cite:

6.1 DBG building and querying

- **Bifrost:** Holley, G., Melsted, P. "Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs." Genome Biol 21(249) (2020). https://doi.org/10.1186/s13059-020-02135-8

6.2 FM-index generation and querying

- **Kseq:** seqtk: https://github.com/lh3/seqtk
- **SDSL v3:** Succinct Data Structure Library 3.0

6.3 Gene scoring and overlap penalisation

- **Balrog:** Sommer M.J., Salzberg S.L. "Balrog: A universal protein model for prokaryotic gene prediction." PLoS Comput Biol 17(2): e1008727 (2021). https://doi.org/10.1371/journal.pcbi.1008727
- **Eigen v3:** Guennebaud, G., Jacob, B. et al. "Eigen v3" (2010). http://eigen.tuxfamily.org
- **Boost graph library:** Siek, J., Lee, L.Q. & Lumsdaine, A. "Boost graph library" (2002) https://www.boost.org/doc/libs/1_79_0/libs/graph/doc/index.html

6.4 Pairwise gene comparisons

- **Edlib:** Šošić, M., Šikić, M. "Edlib: a C/C++ library for fast, exact sequence alignment using edit distance." Bioinformatics 33(9) (2017). https://doi.org/10.1093/bioinformatics/btw753

6.5 Gene annotation

- **DIAMOND:** Buchfink B., Reuter K., Drost H.G. "Sensitive protein alignments at tree-of-life scale using DIAMOND", Nature Methods 18:366–368 (2021). https://doi.org/10.1038/s41592-021-01101-x
- **HMMER3:** Eddy S.R. "A New Generation of Homology Search Tools Based on Probabilistic Inference." Genome Inform., 23:205-211 (2009).

6.6 Alignment and variant calling:

- **MAFFT:** Katoh, K., Misawa, K., Kuma, K. & Miyata, T. "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform." Nucleic Acids Research. 30 (14), 3059–3066 (2002). https://doi.org/10.1093/nar/gkf436
- **SNP-sites:** Page, A.J., Taylor, B., Delaney, A.J., Soares, J., Seemann, T., Keane, J.A. & Harris, S.R. "SNP-sites: rapid efficient extraction of SNPs from multi-FASTA alignments. Microbial genomics." 2 (4), e000056 (2016). https://doi.org/10.1099/mgen.0.000056
- **RapidNJ:** Simonsen, M., Pedersen, C. "Rapid computation of distance estimators from nucleotide and amino acid alignments" Proceedings of the ACM Symposium on Applied Computing (2011) https://doi.org/10.1145/1982185.1982208

6.7 Clustering and pangenome analysis

- **Panaroo:** Tonkin-Hill, G., MacAlasdair, N., Ruis, C. et al. "Producing polished prokaryotic pangenomes with the Panaroo pipeline." Genome Biol 21(180) (2020). https://doi.org/10.1186/s13059-020-02090-4

CHAPTER

SEVEN

WHY GGCALLER?

ggCaller uses population-frequency information at several stages of gene annotation and pangenome analysis. This has several benefits:

- Consistent identification of start and stop codons across orthologs, improving clustering accuracy.
- Reduced gene-annotation sensitivity to assembly fragmentation.
- Reduced runtime verses existing gene-annotation and pangenome analysis workflows.
- One-line command from fasta -> gene annotations, gene frequency matrices, clusters of orthologous genes (COGs), core genome/pangenome alignments, phylogenetic trees, small/structural variants and more!
- Annotated DBG-querying for functional PanGenome-Wide Association Studies (PGWAS), compatible with results from Pyseer.

CHAPTER

EIGHT

FOR THE IMPATIENT

See *Quickstart* to get ggCaller up and running quickly.

CHAPTER

NINE

EVERYONE ELSE

We recommend starting with *Installation* to ensure things are installed correctly, followed by *Usage* to get an overview of the commands, and finally *Tutorial* for a step-by-step walkthrough.

CHAPTER

TEN

CONTENTS