Gubbins v3 Manual

Nicholas Croucher

11th October, 2021

Analysing whole genome alignments with Gubbins

Introduction

Gubbins (Genealogies Unbiased By recomBinations In Nucleotide Sequences) is an algorithm that iteratively identifies loci containing elevated densities of base substitutions while concurrently constructing a phylogeny based on the putative point mutations outside of these regions. Simulations demonstrate the algorithm generates highly accurate reconstructions under realistic models of short-term diversification of sequences through both point mutation and recombination, and can be run on alignments of many hundreds of bacterial genome sequences. It is therefore not appropriate for looking at recombination across species-wide diversity - this can be done gene-by-gene using software such as fastGEAR. Instead, it works on **samples of limited diversity, sharing a recent common ancestor** - a strain or lineage.

The time taken for the algorithm to converge on a stable solution increases approximately quadratically with the number of samples; this increase can be ameliorated to some extent by using faster and/or simpler phylogenetic algorithms to generate trees within the analysis pipeline. The input should be a *whole genome sequence alignment*; there is no need to remove accessory genome loci, as the algorithm should cope with regions of missing data. Gubbins will not produce a sensible alignment on concantentations of core genes output but software such as Roary, because it requires information on the spatial distribution of polymorphisms.

Gubbins cannot distinguish elevated densities of polymorphisms arising through recombination from other potential causes. These may be assembly or alignment errors, mutational hotspots or regions of the genome with relaxed selection. Such false positives are more likely to arise on longer branches within a phylogeny; it is recommended that populations be subdivided into smaller groups of less diverse samples that can each be independently analysed with Gubbins. This can be achieved with software such as PopPUNK or fastBAPS. Further discussion of potential confounding factors in the analysis of such population genomic datasets can be found elsewhere.

Description of the algorithm

A brief overview of the algorithm is, within each iteration:

- A set of polymorphic sites assumed to have arisen through point mutation is extracted from the whole genome alignment
- A tree is generated from these sites
- A phylogenetic model is fitted to the tree and all polymorphic sites in the alignment
- The pattern of base substitutions resulting in the observed distribution of alleles across polymorphic sites is reconstructed

- A spatial scanning statistic is iteratively applied to the base substitutions reconstructed as occurring on each branch, to identify all regions with an elevated density of base substitutions
- These regions are assumed to have arisen through recombination, and base substitutions within these regions in the taxa descended from the branch are excluded from the set used to generate the tree in the next iteration

Iterations continue until the same tree is observed in multiple iterations, or the maximum number of iterations is reached. In terms of runtime, the first iteration requires a tree to be generated from all polymorphic sites, as none have yet been excluded as recombinant, and therefore this step is usually the slowest part of the analysis.

Installation and dependencies

Gubbins is a command line program designed to be run on Linux or Max OSX systems and requires Python version 3.8 or greater. Gubbins can also be run on Windows operating systems using the Powershell within Windows >=10, a Bio-Linux virtual machine. The recommended installation approach is to use conda:

```
conda config --add channels r
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
conda install gubbins
```

Alternative approaches are described on the Github page. Gubbins relies on multiple other phylogenetics software packages, including:

- RAxML
- IQTree
- RAxML-NG
- FastTree
- Rapidnj

These will automatically be installed within the conda environment. Please cite any of these methods you use as part of a Gubbins analysis - these are listed in a `.log` file output by Gubbins.

Input files and workflow

The required input file for Gubbins is a whole genome FASTA alignment. Each sequence should have a unique identifier, and special characters should be avoided. The sequences should only use the characters `ACGT` (DNA bases), `N` (unknown base) or `-` (alignment gap). If a starting tree is to be included, then this should be a Newick format.

The alignment is most easily generated through mapping sequences against a reference sequence. This can be generated using the Gubbins script `generate_ska_alignment.py`, which creates an alignment using SKA, which can be installed through `conda install -c bioconda ska`. For instance,

```
generate_ska_alignment.py --reference seq_X.fa --fasta fasta_files.list --fastq fastq_files.list --out
```

Where `fasta_files.list` is a two column tab-delimited file containing sequence names (in the first column) and FASTA sequence assembly file paths (in the second column); `fastq_files.list` contains the same for unassembled FASTQ-format read data.

The alignment can then be analysed with Gubbins:

```
run_gubbins.py --prefix gubbins_out out.aln
```

The output of this analysis can then be visualised using Phandango or RCandy. Further downstream analysis can use BactDating to generate a time-calibrated phylogeny, and SkyGrowth for reconstructing past population sizes. For an example of such a workflow, see D'Aeth *et al*.

Input options

Version 3 of Gubbins has an extended range of analysis options:

Input and output options

Gubbins can take a starting tree, to speed up the analysis - you may have generated one as part of an initial analysis, e.g. with PopPUNK. This must contain all the taxa, but superfluous taxa will be ignored (e.g. within a species-wide tree). The analysis can also be sped up by using multiple threads, if you have multiple processors available to you, using `--threads`. Almost all parts of the Gubbins algorithm are multithreaded.

```
--prefix PREFIX, -p PREFIX
                      Add a prefix to the final output filenames (default: None)
--starting-tree STARTING_TREE, -s STARTING_TREE
                      Starting tree (default: None)
--use-time-stamp, -u  Use a time stamp in file names (default: False)
--version             show program's version number and exit
--threads THREADS, -c THREADS
                      Number of threads to use for parallelisation (default: 1)
--verbose, -v         Turn on debugging (default: False)
--no-cleanup, -n      Do not cleanup intermediate files (default: False)
```

Data processing options

Gubbins can remove duplicate or low-quality sequences from samples. It can also run in a special mode (`--pairwise`) to identify recombinations distinguishing two sequences, without generating a tree.

```
--pairwise            Compare two sequences (without using a tree) (default: False)
--filter-percentage FILTER_PERCENTAGE, -f FILTER_PERCENTAGE
                      Filter out taxa with more than this percentage of gaps (default: 25.0)
--remove-identical-sequences, -d
                      Remove identical sequences (default: False)
```

Tree building options

Multiple phylogenetic packages can be used to run a Gubbins analysis. Typically, we would recommend a fast, simple tree builder is used for the first phylogeny (`--first-tree-builder` set to `star`,`rapidnj` or

`fasttree`), and a more accurate, slower maximum-likelihood tree builder is used for subsequent iterations (`--tree-builder` set to `raxml`, `raxmlng` or `iqtree`).

The robustness of the final tree can be assessed using bootstraps, transfer bootstraps or an Shimodaira–Hasegawa test (`--sh-test`) of node likelihoods.

```
--tree-builder {raxml,raxmlng,iqtree,fasttree,hybrid,rapidnj}, -t {raxml,raxmlng,iqtree,fasttree,hybri
                      Application to use for tree building (default: raxml)
--tree-args TREE_ARGS
                      Quoted string of further arguments passed to tree building algorithm (start stri
--first-tree-builder {raxml,raxmlng,iqtree,fasttree,rapidnj,star}
                      Application to use for building the first tree (default: None)
--first-tree-args FIRST_TREE_ARGS
                      Further arguments passed to first tree building algorithm (default: None)
--outgroup OUTGROUP, -o OUTGROUP
                      Outgroup name for rerooting. A list of comma separated names can be used if they
--bootstrap BOOTSTRAP, -# BOOTSTRAP
                      Number of bootstrap replicates to perform with final alignment (default: 0)
--transfer-bootstrap  Calculate bootstrap supporting transfer bootstrap expectation (default: False)
--sh-test             Perform an SH test of node likelihoods (default: False)
```

Nucleotide substitution model options

The available nucleotide substitution models are:

- **JC** - Jukes-Cantor - all model-fitting software
- **K2P** - Kimura 2-parameter - available for RAxML, RAxML-NG, IQtree and Rapidnj
- **HKY** - Hasegawa, Kishino and Yano - available for RAxML, RAxML-NG and IQtree
- **GTR** - General time reversible - available for RAxML-NG and IQtree
- **GTRGAMMA** - General time reversible with a Gamma model of between-site rate heterogeneity - available for RAxML, RAxML-NG and IQtree
- **GTRCAT** - General time reversible with a categorisation of between-site rate heterogeneity - available for RAxML

The model fitting software must be consistent with the selected model - by default, the fitting software will be the same as the tree builder, but this can be changed with `--model-fitter` and `--first-model-fitter`. To reduce run time, it may be most efficient to use a simple model (e.g. `--first-model JC`) for the first tree, which is likely to be inaccurate, and a more realistic model (e.g. `--model GTR`) for later trees.

```
--model-fitter {raxml,raxmlng,iqtree,fasttree,None}, -F {raxml,raxmlng,iqtree,fasttree,None}
                      Application to use for model fitting [if unspecified: same as tree builder if po
--model {JC,K2P,HKY,GTR,GTRGAMMA,GTRCAT}, -M {JC,K2P,HKY,GTR,GTRGAMMA,GTRCAT}
                      Nucleotide substitution model (not all available for all tree building algorithm
--model-args MODEL_ARGS
                      Quoted string of further arguments passed to model fitting algorithm (start stri
--custom-model CUSTOM_MODEL
                      String corresponding to a substitution model for the selected tree building algo
--first-model-fitter {raxml,raxmlng,iqtree,fasttree,None}
                      Application to use for model fitting in first iteration [if unspecified: same as
--first-model {JC,K2P,HKY,GTR,GTRGAMMA,GTRCAT}
```

```
                      Nucleotide substitution model used for first tree (default: None)
--first-model-args FIRST_MODEL_ARGS
                      Further arguments passed to model fitting algorithm used in firstiteration (if
--custom-first-model CUSTOM_FIRST_MODEL
                      String corresponding to a substitution model for the selected tree building alg
```

Ancestral sequence reconstruction options

Gubbins was originally designed to use a joint ancestral state reconstruction, which identifies the most likely pattern of base substitutions across the entire tree. Version 2 used a marginal ancestral state reconstruction, which reconstructed each branch independently, to maintain the package as being open source and easy to install. Version 3 now implements a multi-threaded version of pyjar to enable rapid joint ancestral state reconstruction by default, although marginal ancestral state reconstructions are still possible by specifying `--mar`.

```
--mar                 Use marginal, rather than joint, ancestral reconstruction (default: False)
--seq-recon {raxml,raxmlng,iqtree,None}
                      Algorithm to use for marginal reconstruction [if unspecified: same as tree buil
--seq-recon-args SEQ_RECON_ARGS
                      Further arguments passed to sequence reconstruction algorithm (start string wit
```

Recombination detection options

Recombination is detected using a spatial scanning statistic, which relies on a sliding window. The size of this window may need to be reduced if you apply Gubbins to very small genomes (e.g. viruses).

```
--min-snps MIN_SNPS, -m MIN_SNPS
                      Min SNPs to identify a recombination block (default: 3)
--min-window-size MIN_WINDOW_SIZE, -a MIN_WINDOW_SIZE
                      Minimum window size (default: 100)
--max-window-size MAX_WINDOW_SIZE, -b MAX_WINDOW_SIZE
                      Maximum window size (default: 10000)
```

Algorithm stop options

Given the scale of available dataset sizes, and the size of tree space, it is unlikely that any Gubbins analysis will ever converge based on identifying identical trees in subsequent iterations. In practice, there is little improvement to the tree after three iterations.

```
--iterations ITERATIONS, -i ITERATIONS
                      Maximum No. of iterations (default: 5)
--converge-method {weighted_robinson_foulds,robinson_foulds,recombination}, -z {weighted_robinson_foul
                      Criteria to use to know when to halt iterations (default: weighted_robinson_foul
```

Output files

A successful Gubbins run will generate files with the suffixes:

- `.recombination_predictions.embl` - Recombination predictions in EMBL file format.
- `.recombination_predictions.gff` - Recombination predictions in GFF format

- `.branch_base_reconstruction.embl` - Base substitution reconstruction in EMBL format
- `.summary_of_snp_distribution.vcf` - VCF file summarising the distribution of SNPs
- `.per_branch_statistics.csv` - per branch reporting of the base substitutions inside and outside recombination events
- `.filtered_polymorphic_sites.fasta` - FASTA format alignment of filtered polymorphic sites used to generate the phylogeny in the final iteration
- `.filtered_polymorphic_sites.phylip` - Phylip format alignment of filtered polymorphic sites used to generate the phylogeny in the final iteration
- `.final_tree.tree` - this file contains the final phylogeny in Newick format
- `.node_labelled.final_tree.tre` - final phylogenetic tree in Newick format but with internal node labels
- `.log` - log file specifying the software used at each step of the analysis, with accompanying citations

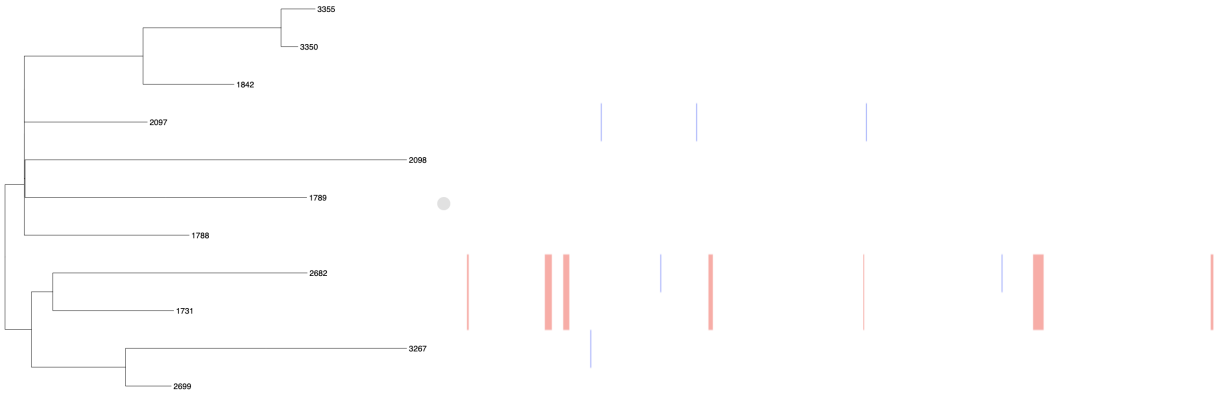
To generate a recombination-masked alignment (i.e., with sequences predicted to have been introduced by recombination removed, leaving just the clonal frame), the post-processing script `mask_gubbins_aln.py` can be used:

```
mask_gubbins_aln.py --aln out.aln --gff out.recombination_predictions.gff --out out.masked.aln
```

Examples

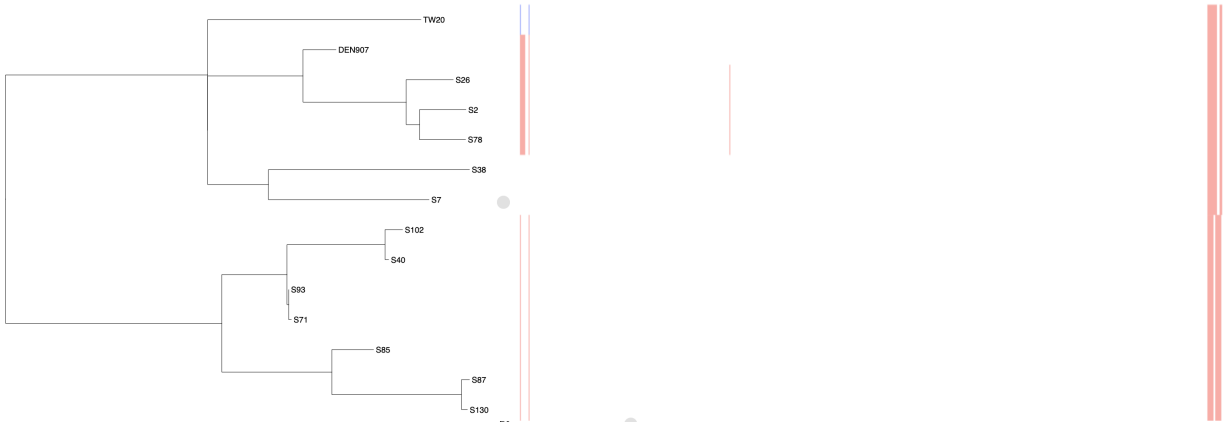
Two example alignments can be downloaded from http://sanger-pathogens.github.io/gubbins/:

- *Streptococcus pneumoniae* PMEN1, for which the expected output is:



This used the command `run_gubbins.py --prefix PMEN1 --first-tree-builder rapidnj --first-model JC --tree-builder raxmlng --model GTR PMEN1.aln` and took ~20s on a single CPU.

- *Staphylococcus aureus* ST239, for which the expected output is:



This used the command `run_gubbins.py --prefix ST239 --first-tree-builder rapidnj --first-model JC --tree-builder raxmlng --model GTR ST239.aln` and took ~30s on a single CPU.

Troubleshooting

Please log any issues you encounter on the GitHub site: https://github.com/sanger-pathogens/gubbins. Please be patient, as there is currently no specific funding support for Gubbins.

Citation

Please cite the Gubbins paper, plus those of any other phylogenetic methods used, as listed in the `.log` file. Thank you!