FlaGs User Manual
July 2019

https://github.com/GCA-VH-lab/FlaGs/

Chayan Kumar Saha
chayan.kumar@umu.se

Gemma C. Atkinson
Gemma.atkinson@umu.se

Contents:

1. Description

FlaGs (for Flanking Genes) analyses genomic context around genes encoding a protein of interest using a Python 3 script combined with *Jackhmmer* and *ETE-toolkit* (optional). This tool finds the flanking genes of the genes of interest, cluster them based on the homology and represent them visually with distinguished identifiers (colour and number).

At initiation the tool verifies the input file which is either a list of NCBI protein accessions (-p option), or a tab delimited list of both protein accessions and corresponding genome assembly identifers (-a). The latter option is to inform FlaGs which specific genome it should be searching with the protein accession; since NCBI protein accessions are non-redundant, identical protein sequences can be present in multiple genomes with the same accession number. If FlaGs is given protein accessions alone using the –p option, it takes each accession and finds the corresponding list of assembly identifiers (eg. *GCF_000001635.23 or GCA_000001635.5)*. Then it can either report all results for all identifiers (-r option), or for one representative identifier (default). The –r option is not recommended as identical proteins can sometimes be found in up to thousands of genomes from different species of the same strain.

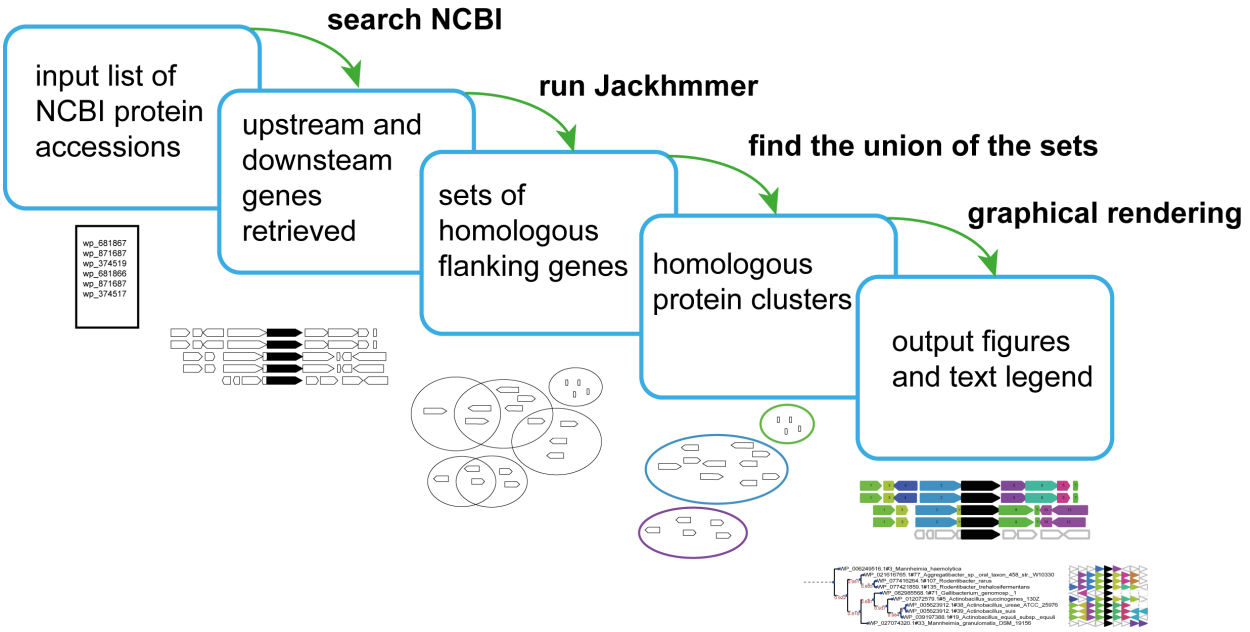

Figure 1. The FlaGs workflow. The user inputs a list of protein accession numbers – optionally with GCF assembly identifiers – and can specify the number of adjacent flanking genes to consider and the sensitivity of the Jackhmmer search though changing the E value cut-off and number of iterations. The output always includes a to-scale figure of flanking genes, a description of the flanking gene identities as a legend, and optionally, a phylogenetic tree annotated with colour- and number- coded pennant flags. Unconserved proteins are uncoloured.

After verification of the input query list, FlaGs uses the assembly identifer to retrieve the Genomic Feature Format (gff3) file for each protein query and uses this to identify annotated flanking genes from upstream and downstream regions. Then it retrieves the sequences for all flanking genes and searches for homologues within the set using *Jackhmmer* from the Hmmer package (hmmer.org, (Eddy 2011)). Flanking genes are then clustered and each cluster is assigned by a

specific number. Sets of homologues found with the *Jackhmmer* searches are joined together to make one cluster (the union) if there is one or more common protein among the Jackhmmer search results (Figure 1). The numbering of clusters begins with 1. The lower the cluster number the more conserved the protein is, i.e the more frequently it is found among all the flanking genes. Finally, FlaGs generates a visual representation of the output using the tkinter Python module, and this is saved as a postscript file (Figure 2). The black block arrow represents the query proteins and the rest represent the flanking genes. The figure is proportional to actual gene lengths and intergenic space. If the query protein is encoded on the negative strand, the entire context is then flipped to make it easily comparable. The number and colour represent the cluster, and the "…_outdesc.txt" output file provides the legend for interpretation. Unconserved proteins are uncoloured and unnumbered, and non-protein coding genes are outlined in green.

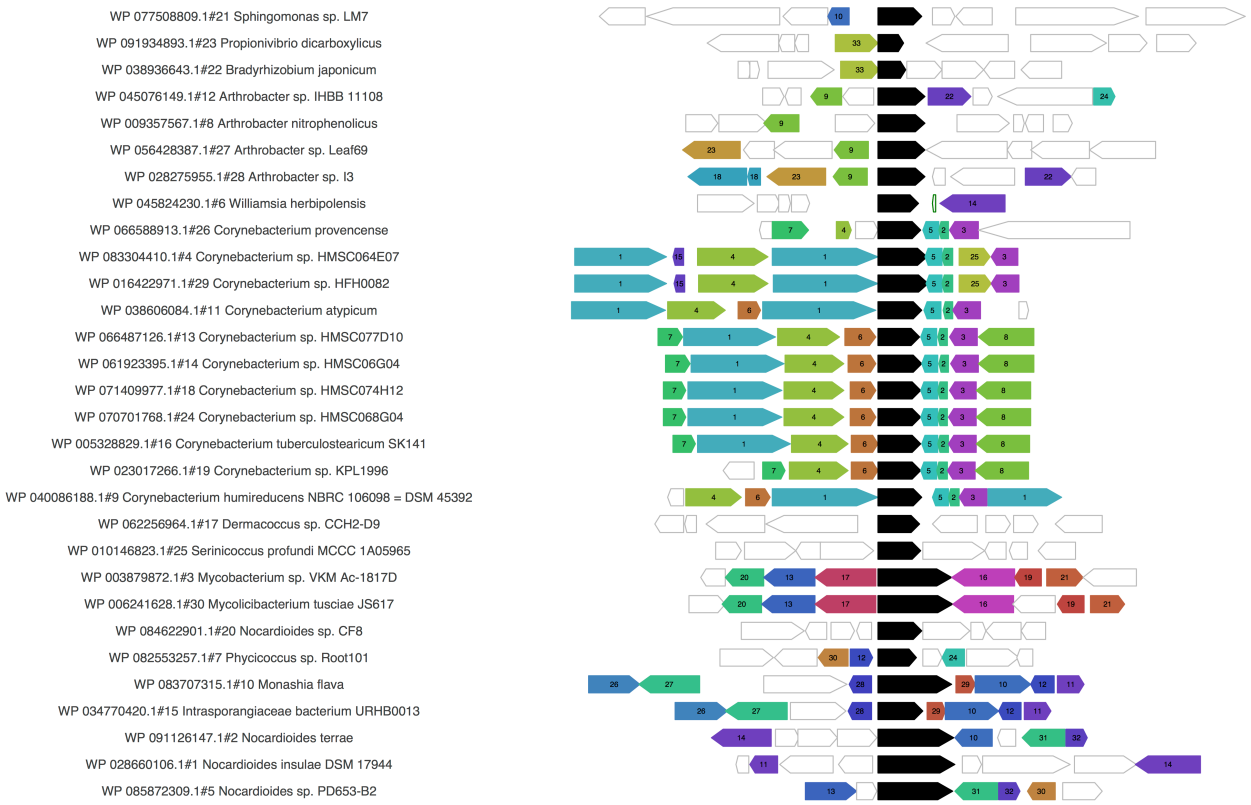

Figure 2. The to-scale postscript output. The black gene is the query, and otherwise colours and numbers represent the clusters. The "…_outdesc.txt" output file provides the legend for interpretation. Genes and intergenic spaces are to scale.

An optional feature of FlaGs is the generation of a phylogenetic tree of the query sequences, on which the flanking genes are annotated (Figure 3). This is acheived using the ETE-toolkit (etetoolkit.org, (Huerta-Cepas, et al. 2016)). The ETE "mafft_default-trimal01-none-fasttree_full" workflow is used to generate the tree (see ETE user manual, http://etetoolkit.org/cookbook/). The flanking genes and the queries are represented as triangular pennant flag-like shapes, with colours and numbers representing clusters. The "…_outdesc.txt" output file again provides the legend for interpretation.

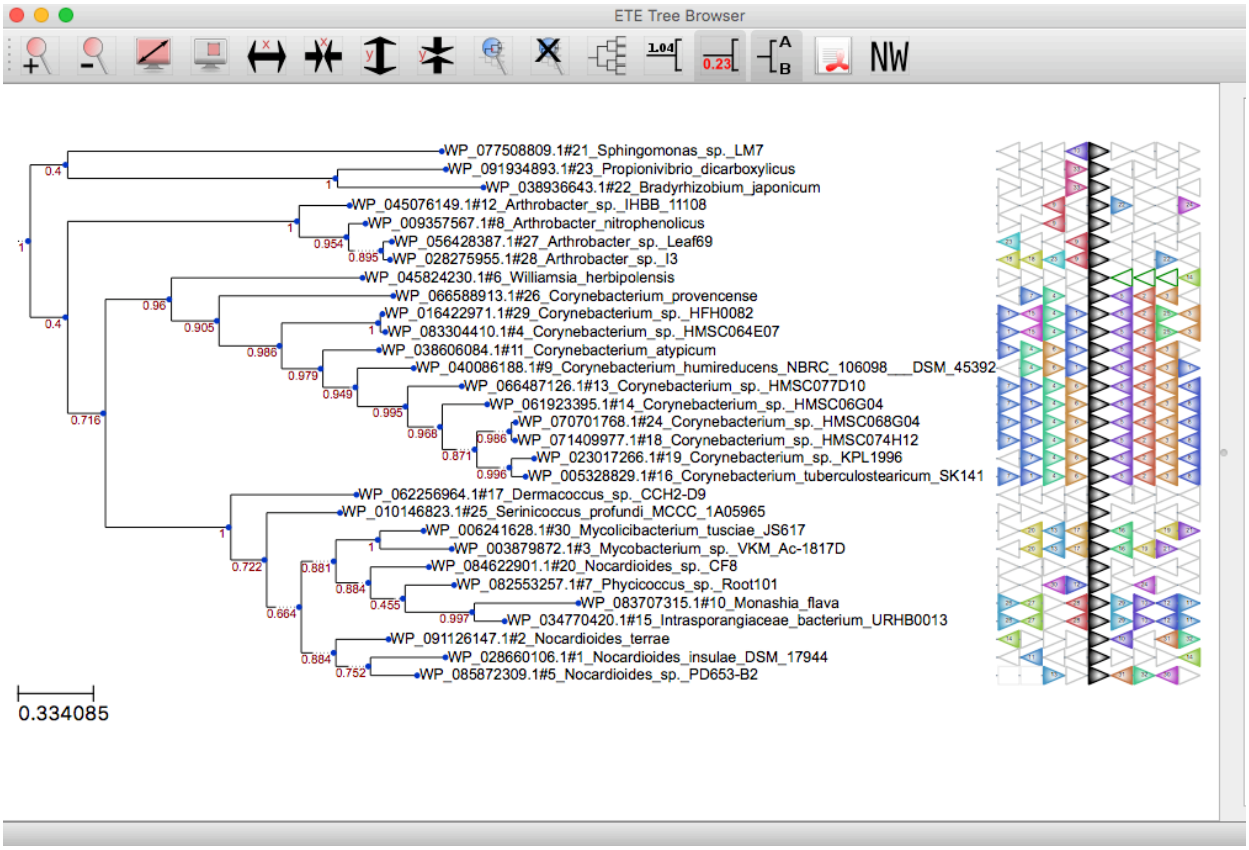

Figure 3. The annotated phylogenetic tree output. The tree is made with ETE3 using the mafft_default-trimal01-none-fasttree_full method and is saved as an SVG vector image file. If the –t option is invoked without –to, an interactive ETE3 window opens (as in this figure) for manipulation of how the tree is displayed.

2. Prerequisites

FlaGs.py is available for download here: https://github.com/GCA-VH-lab/FlaGs/

FlaGs has been primarily tested in Mac and Linux environments. Windows is possible with some workarounds (updates in this direction are planned).

1. Python 3

We recommend the use of Anaconda Python to simplify installlation of other required and optional dependencies. It can be downloaded from https://www.anaconda.com/download/

2. Biopython

Using Anaconda, Biopython can be installed using the following command:

```
conda install -c bioconda biopython
```

3. Jackhmmer

```
conda install -c biocore hmmer
```

Hmmer can alternatively be downloaded from http://hmmer.org/ and installed according to the instructions.

4. ETE-toolkit (only required if making a phylogenetic tree; –t option)

```
# Install ETE and external tools
conda install -c etetoolkit ete3 ete_toolchain

# Check installation
ete3 build check

# Activate the environment (you can add the line to your .bashrc to make it permanent)
export PATH=~/anaconda_ete/bin:$PATH
```

(these instructions come from http://etetoolkit.org/download/)

3. Usage:

```
./FlaGs.py  <options>
```

or `Python3 FlaGs.py <options>`

or (if Python 3 is the default python) `Python FlaGs.py <options>`

4. Running the example files:

We have two different FlaGs python scripts for different operating system.

1. FlaGs.py, which works in MAC OS.
2. FlaGs_linux.py which works in Linux OS.

Without tree (only Biopython neccessary):

```
python FlaGs.py -a GCF_accession_input.txt -o GCF_accession_output -u example@gmail.com -vb
```

or

```
python FlaGs.py -p accession_input.txt -o accession_output -u example@gmail.com -vb
```

With tree (Biopython and ETE necessary):

```
python FlaGs.py -a GCF_accession_input.txt -t -to -o GCF_accession_output -u example@gmail.com -vb
```

or

```
python FlaGs.py -p accession_input.txt -t -to -o accession_output -u example@gmail.com -vb
```

With tree and interactive ETE tree editing window:

```
python FlaGs.py -a GCF_accession_input.txt -t -o GCF_accession_output -u example@gmail.com -vb
```

or

```
python FlaGs.py -p accession_input.txt -t -o accession_output -u example@gmail.com -vb
```

5. Arguments:

1. "-a", "--assemblyList"

Protein Accession with Genome Assembly Identifier eg. GCF_000001765.3 in a text input file separated by tab.

Input File Example:

GCF_000001765.3 WP_047256880.1 #tab separated
GCF_000002753.1 WP_012725678.1
…….

2. "-p", "--proteinList"

Protein Accession eg. XP_ or WP_047256880.1 in a text Input file separated by newline.

Input File Example:

WP_047256880.1

WP_012725678.1
…….

3. “-l”, “--localGenomeList”

FlaGs v1.0.5 can use local genomes too. For that, user needs to provide the following files:

i. Compressed (gzip) FASTA format of the predicted protein products annotated on the genome assembly (eg. Assembly_1.faa.gz, file should contain “.faa.gz” suffix)

   Recommended format can be downloaded from this link. Or use the following command from terminal:

```
wget
ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/428/725/GCF_000428
725.1_ASM42872v1/*ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/
428/725/GCF_000428725.1_ASM42872v1/GCF_000428725.1_ASM42872v1_pro
tein.faa.gz
```

ii. Compressed (gzip) Annotation of the genomic sequence(s) in Generic Feature Format Version 3 (GFF3) (eg. Assembly_1.gff.gz, file should contain “.gff.gz” suffix)

   Recommended format can be downloaded from this link. Or use the following command from terminal:

```
wget
ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/428/725/GCF_000428
725.1_ASM42872v1/GCF_000428725.1_ASM42872v1_genomic.gff.gz
```

So, with “-l” flag user can feed the text file with Assembly name and the query protein accession.

Input File Example:

GCF_000001765.3 WP_047256880.1 #tab separated
GCF_000002753.1 WP_012725678.1

In this case, both of the fasta and gff file should contain the Assembly Identifier as prefix.
Like, for assembly identifier GCF_000001765.3, the predicted protein products containing compressed Fasta file should be named as GCF_000001765.3.faa.gz and GFF file should be named as GCF_000001765.3.gff.gz

Or Input File Example:

Assembly_1 WP_047256880.1 #tab separated

Assembly_2 WP_012725678.1

For assembly identifier Assembly_1, the predicted protein products containing compressed Fasta file should be named as Assembly_1.faa.gz and GFF file should be named as Assembly_1.gff.gz

4. “-ld”, “--localGenomeDirectory”

This flag allows the user to specify the path or directory where *.faa.gz* and *.gff.gz* are stored. By default, FlaGs script will look for the *.faa.gz* and *.gff.gz* files in the same directory where the script is located or running from.

User can only use this flag when ‘-l’ flag is being used.

For example: If the files are stored in ~/FlaGs/local directory user can specify by following:

```
-ld ~/FlaGs/local
```

5. "-r", "--redundant”

Search flanking genes in all available GCFs for each query. It’s time consuming.

6. "-e", "--ethreshold"

This e-value is used by Jackhmmer as a cutoff parameter to detect homology among all flanking genes. By default it is 1e-10.

7. "-n", "--number"

This number represents number of Jackhmmer iterations that allows to find more distant homolog, by default it is set as 3.

8. "-g", "--gene"

Using this parameter in FlaGs user can define number of upstream and downstream genes to look for each query in input list. By default, it is 4, which means for each query it will try to find 4 upstream and 4 downstream flanking genes and then process further.

9. "-t", "--tree"

This requires an ETE3 installation. The option enables showing flanking genes along with a phylogenetic tree.

10. "-ts", "--tshape"

This requires an ETE3 installation and thus this option only works when -t is used. This parameter can increase or decrease the size of triangle shapes that represent flanking genes, by default it is 12.

11. "-tf", "--tfontsize"

This also requires an ETE3 installation and thus this option only works when -t is used. This parameter can increase or decrease the size of font inside triangles that represent flanking genes, by default it is 4.

12. "-to", "--tree_order"

In combination with –t, it will first generate the tree output, and then use the tree order to generate the other (without tree) output file.

13. “-u”, “--user_email”

Since we are retrieving information from NCBI, it requires valid email address to retrieve information for only 3 queries per seconds.

14. “-api”, “--api_key”

Valid API-key allows 10 queries per seconds, which makes the tool to run faster. And it is necessary if user wants to run multiple FlaGs in parallel. For details please check:

https://ncbiinsights.ncbi.nlm.nih.gov/2017/11/02/new-api-keys-for-the-e-utilities/

15. "-o", "--out_prefix"

Any Keyword to define your output eg. MyQuery.

16. "-k", "--keep"

If user wants to keep the intermediate files eg. gff3, this option is useful then. By default, it will remove.

17. "-v", "--version"
Version number of the program.

18. "-vb", "--verbose"

User can use this option to see the work progress for each query as STDOUT.

6. Output files

In addition to the one or two figure files described above, FlaGs generates the following output files:

1. Discarded protein ids with improper accession are listed in a text file with "_NameError.txt" suffix.
2. Discarded protein ids lacking proper information in RefSeq DB are listed in a text file with "_Insufficient_Info_In_DB.txt" suffix
3. Information about the flank genes for each queries retrieved from Databases are stored in a tab delimited file named with "_flankgene.tsv" suffix
4. A general summary report is also generated for each query if flanking gene information is found or not in a tab delimited file with "_flankgene_Report.log" suffix
5. File with suffix "_jackhits.tsv" contains *Jackhmmer* output
6. After the clustering steps a file named with suffix "_clusters.tsv" is generated with detailed information of the cluster. For example;

   1 4 WP_001229260.1;WP_001229255.1
   2 4 WP_001164213.1;WP_001164217.1;WP_001164219.1;WP_055027268.1

   Column 1 represents the cluster number; the lower the number the more frequent it is in the figure. The second column shows the frequency of proteins in the cluster. In first row (Cluster 1), 4 means that two protein accessions WP_001229260.1 and WP_001229255.1 are 4 times frequent and in cluster 2 you can see 4 proteins accessions are frequent for 4 times.

7. A description file is generated after clustering with suffix "_outdesc.txt". For example;

   1(2) WP_001229260.1 MULTISPECIES: integration host factor subunit alpha
   1(2) WP_001229255.1 MULTISPECIES: integration host factor subunit alpha

   2(1) WP_001164213.1 MULTISPECIES: phenylalanine--tRNA ligase subunit alpha
   2(1) WP_001164217.1 MULTISPECIES: phenylalanine--tRNA ligase subunit alpha
   2(1) WP_001164219.1 phenylalanine--tRNA ligase subunit alpha
   2(1) WP_055027268.1 phenylalanine--tRNA ligase subunit alpha

   Here you can see in column 1, it starts with cluster number 1 and "(2)" means the protein WP_001229260.1 has been found twice in the cluster and the last column is the protein description retrieved from the database. The protein description gives information about the proteins in a cluster (accessions and title from NCBI).

   This file can be used as an input for the additional python script descriptionCloud.py to get wordcloud visualization.

8. If the –t option used to make a phylogenetic tree, ETE3 saves additional results files to a folder ending with "[output name]_tree_". This includes a Newick format version of the tree.

FlaGs also generates some intermediate files to proceed the overall process but only the files mentioned above are useful for interpretation.

7. Recommendations:

FlaGs can handle relatively large input datasets, although of course the smaller, the faster. Since FlaGs retrieves information from NCBI (multiple times for each input accession), it is dependent on both an internet connection and the stability of the NCBI server. Our example input file "GCF_accession_input.txt" with 30 accessions takes around 7 minutes to run with the –a, -t, -to options on a 2015 MacBook Pro with an internet connection of around ~75 mbps. Without the -t option it takes around 6 minutes. We have also tested input files containing around 1000 protein accessions (four flanking genes each side) and they worked successfully over the course of a few hours.

8. References

Eddy SR. 2011. Accelerated Profile HMM Searches. PLoS Comput Biol 7:e1002195.
Huerta-Cepas J, Serra F, Bork P. 2016. ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. Mol Biol Evol 33:1635-1638.