

GLIMMER-MG Release Notes

Version 0.1

David R. Kelley, Art L. Delcher

17 May 2011

1 Introduction

This document describes Version 0.1 of the GLIMMER-MG gene-finding software for metagenomic sequences. Users discovering problems or errors are encouraged to report them to dakelley@umiacs.umd.edu.

GLIMMER is a collection of programs for identifying genes in microbial DNA sequences. The system works by creating a variable-length Markov model from a training set of genes and then using that model to attempt to identify all genes in a given DNA sequence. The three versions of GLIMMER are described in [SDKW98], [DHK⁺99], and [DBPS07].

GLIMMER-MG is released as OSI Certified Open Source Software under the Artistic License. The license is contained in the file, `LICENSE`, in the distribution.

2 Installation

GLIMMER-MG software was written for the Linux software environment. The following instructions assume a Linux system. They also work under Mac OSX.

To install GLIMMER-MG, download the compressed tarfile `glimmer-mg-1.0.1.tar.gz` from the website. Then uncompress the file by typing

```
tar xzf glimmer-mg.1.0.1.tar.gz
```

A directory named `glimmer-mg` should result. In that directory is a script called `install_glimmer.py` which will download and install the prerequisite software and compile the GLIMMER-MG code. More specifically, it will

1. Download, compile, and install `Phymm`

`Phymm` is used to compute the phylogenetic classifications of sequences used by GLIMMER-MG to parameterize gene prediction models. `Phymm` will install a database of interpolated Markov models (IMMs) to be used for classification that requires ~ 50 Gb of disk space and takes about ~ 24 hours to build. If you already have `Phymm` installed, you need only set the variable `prior_phymm_dir` in `install_glimmer.py` to its installation directory.

2. Download, compile, and install `Scimm`

`Scimm` is used to cluster the sequences to enable an unsupervised retraining step followed by a second iteration of gene predictions.

3. Download, compile, and install `ELPH`

`ELPH` is used to compute a motif position weight matrix for the ribosomal binding sites in the promoters of the training gene sets.

4. Compile and install GLIMMER-MG

This will compile the main gene prediction code.

5. Prepare GLIMMER-MG training data

There are three steps here. First, the script `train_all.py` will train the models for all features on every reference genome in the `Phymm` database. Second, the script `informative_genomes.py` will decide which reference genomes will be useful for gene prediction and which (like small plasmids) will not be useful. Finally, the script `double_icms.py` will determine which ICMs trained on two reference genomes are worth building and build them. These steps require some time.

3 Running Glimmer-MG

GLIMMER-MG can be run in a few different modes, depending on the characteristics of the sequences provided, e.g., whether they represent accurate contigs or reads from the Illumina or 454 technologies. Then within each mode, GLIMMER-MG can be run with varying amounts of classification/clustering preprocessing.

3.1 Preprocessing options

The GLIMMER-MG pipeline can be run a number of different ways, including a Python script to run the full pipeline and options to classify/cluster the reads separately and predict genes using those results.

3.1.1 Running on a single genome

The Python script `g3-iterated.py` runs the unsupervised single genome pipeline.

```
g3-iterated.py genome.fasta out
```

This script will extract initial ORFs for training using the `long-orfs` program and then run multiple iterations of Glimmer and retraining.

3.1.2 Running from scratch

The Python script `glimmer-mg.py` runs the full GLIMMER-MG pipeline.

```
glimmer-mg.py seqs.fasta
```

This script will classify the sequences with PHYMM [BS09], make initial predictions for all sequences, cluster the sequences with SCIMM [KS10], and make final predictions within each cluster.

3.1.3 Running from scratch with no retraining

By passing the option `--iter 0` to `glimmer-mg.py`, the clustering and re-training steps will be skipped. That is, the sequences will be classified with PHYMM and predictions made.

```
glimmer-mg.py --iter 0 seqs.fasta
```

3.1.4 Classification separate

Some users may prefer to run the computationally intensive classification of sequences separately (e.g. on a computer cluster). `glimmer-mg.py` can be made to expect this by using the `--raw` option, which specifies that the raw PHYMM output file already exists in the current working directory.

```
glimmer-mg.py --raw seqs.fasta
```

3.1.5 Clustering separate

In a similar vein, clustering can be performed separately before making gene predictions by specifying the `--clust` option, which specifies that SCIMM or PHYSCIMM output is in the current working directory.

```
glimmer-mg.py --clust seqs.fasta
```

3.2 Sequence modes

Depending on the error characteristics of the input sequences, GLIMMER-MG has three modes that best handle certain types of data. In each case, the options described can be passed to the C++ binary `glimmer-mg` or the Python script to manage the entire pipeline `glimmer-mg.py`.

3.2.1 Accurate contigs

If you believe that the sequences on which you would like to find genes are very accurate, than run GLIMMER-MG in the default mode with no additional options.

3.2.2 454 indels

If the input sequences are reads or contigs from the 454 or Ion Torrent technologies and indel errors are expected to exist, GLIMMER-MG can be made to predict such errors and account for them in its gene predictions by using *indel* mode.

```
glimmer-mg.py --indels 454.fasta
```

3.2.3 Illumina substitution errors

If the input sequences are reads or contigs from the Illumina technology and substitution errors are expected to exist, GLIMMER-MG can be made to predict such errors and account for them in its gene predictions by using *substitution* mode.

```
glimmer-mg.py --sub illumina.fasta
```

4 Sample Run Directory

The directory `sample_run` contains a sample run of GLIMMER-MG.

4.1 Single Genome

The subdirectory `glimmer3` contains the genome sequence of a *Helicobacter Pylori* strain. The files in the directory `results` are the result of running the script

```
g3-iterated.py NC_000915.fna NC_000915
```

4.2 Metagenomics

The subdirectory `glimmer-mg` contains a simulated metagenome in the fasta file `seqs.fa` with the sequence origins in `map.txt`. The files in the directory `results` are the result of running the script

```
glimmer-mg.py seqs.fa
```

Don't be alarmed if your results are slightly different. One explanation is that a slightly different set of GenBank reference genomes were available to PHYMM. Another explanation is that SCIMM has a stochastic component and may have produced a different set of clusters used for retraining.

5 Notes on Glimmer-MG programs

5.1 glimmer-mg

This is the main program that makes gene predictions.

5.1.1 glimmer-mg Parameters & Options

The invocation for `glimmer-mg` is:

```
glimmer-mg [options] sequence tag
```

where *sequence* is the name of the file containing the DNA sequence(s) to be analyzed and *tag* is a prefix used to name the output files.

options can be the following:

`-b filename` or `--rbs_pwm filename`

Read a position weight matrix (PWM) from *filename* to identify the ribosome binding site to help choose start sites. The format of this file is indicated by the following example:

6						
a	212	309	43	36	452	138
c	55	58	0	19	48	26
g	247	141	501	523	5	365
t	64	70	34	0	73	49

The first line is the number of positions in the pattern, *i.e.*, the number of columns in the matrix (not counting the first column of labels). The column values are the relative frequencies of nucleotides at each position.

-c *filename* or **--class** *filename*

Read sequence classifications from *filename*. Each line of *filename* should specify a sequence fasta header followed by a series of GenBank reference genome classifications. The classifications should name the reference genome's Phymm directory (corresponding to the GenBank strain name), followed by the character '|', followed by the genomic sequence's unique identifier. It is our expectation that users will not create these files, but they generate them with **glimmer-mg.py**. For example:

```
read1      Mycobacterium_leprae_Br4923|NC_011896 Corynebacterium_glutamicum_ATCC_13032|NC_003450
read2      Rhodopseudomonas_palustris_BisA53|NC_008435 Rhodopseudomonas_palustris_HaA2|NC_007778
...
```

-f *filename* or **--features** *filename*

filename specifies counts for features such as gene length, start codon usage, adjacent gene orientations, and adjacent gene distances for coding and noncoding ORFs. GLIMMER-MG will convert these counts into probability models for use in log-likelihood computations for the features. It is our expectation that users will not create these files, but generate them with **train_features.py** (via **glimmer-mg.py**). For an example of the input format, see the **sample_run** directory.

-g *n* or **--gene_len** *n*

Set the minimum gene length to *n* nucleotides. This does not include the bases in the stop codon.

-h or **--help**

Print the usage message.

-i or **--indel**

Predict genes in "indel-mode" where gene predictions may shift the coding frame, implicitly predicting an insertion or deletion in the sequence. If quality values are provided with the **-q** option, positions at which frame shifts should be considered will be identified by their low quality. If no quality values are provided, frame shifts will be considered at long homopolymer runs, assuming that the sequences were generated by the 454 or Ion torrent technologies.

-m *filename* or **--icm** *filename*

filename contains an ICM trained on coding sequence to be used for ORF scoring rather than ICM's obtained via classification results.

-o *n* or --max_olap *n*

Set the maximum overlap length to *n*. Overlaps of this many or fewer bases are allowed between genes. The new dynamic programming algorithm should *never* output genes that overlap by more than this many bases.

-q *filename* or --quality *filename*

filename contains quality values in fasta format matching up with the sequences fasta file. The quality values are used in “indel-mode” and “substitution-mode” to identify low quality positions in the sequences where errors are most likely.

-r or --circular

Assume a circular rather than linear genome, *i.e.*, there may be genes that “wraparound” between the beginning and end of the sequence.

-s or --sub

Predict genes in “substitution-mode” where gene predictions may pass through stop codons, implicitly predicting a sequencing error that mutated a standard codon to a stop codon. If quality values are provided with the **-q** option, they will be used to compute a log-likelihood penalty for passing through a given stop codon. Otherwise, we assume all quality values are Phred 30.

-u *n* or --fudge *n*

n specifies a “fudge factor” to be added to the log-likelihood score for every ORF. The “fudge factor” acts as a means to tune the sensitivity versus specificity of the predictions.

-z *n* or --trans_table *n*

Use Genbank translation table number *n* to specify stop codons.

-Z *codon-list* or --stop_codons *codon-list*

Specify stop codons as a comma-separated list. Sample format: **-Z tag,tga,taa**. The default stop codons are **tag**, **tga** and **taa**.

5.1.2 glimmer-mg Output Formats

.predict File

This file has the final gene predictions. It’s format is the fasta-header line of the sequence followed by one line per gene. Here is a sample of the beginning of such a file:

```
>SRR029690.117009 E4LJNJL02B7ZT9 length=531
orf00045      386      -1  -2    52.14 I:272 D: S:
orf00057      532      450  -3    3.86 I: D:507 S:
```

The columns are:

Column 1 The identifier of the predicted gene.

- Column 2 The start position of the gene.
- Column 3 The end position of the gene. This is the last base of the stop codon, *i.e.*, it includes the stop codon.
- Column 4 The reading frame.
- Column 5 The log-likelihood ratio score of the gene.
- Column 6 The positions of insertion, deletion, and substitution predictions within the ORF.

5.2 `extract_aa.py`

`extract_aa.py` can be used to extract a multi-fasta file of amino acid sequences from the gene predictions made by `Glimmer`.

5.2.1 `extract_aa.py` Parameters & Options

The invocation for `extract_aa.py` is:

```
extract_aa.py [options]
```

options can be the following:

`-s filename`

filename specifies the fasta file of sequences on which gene predictions were made.

`-p filename`

filename specifies the Glimmer output `.predict` file containing gene predictions.

`-o filename`

filename specifies the output file in which to print amino acid sequences in fasta format.

5.3 `informative_genomes.py`

Make a list of the GenBank reference genomes which have sufficient non-hypothetical gene annotations to be useful for training. Only Phymm classifications to these genomes will be used for training.

5.4 `train_features.py`

Generate data files to be directly used for gene prediction training.

5.4.1 train_features.py Parameters & Options

The invocation for `train_features.py` is:

```
train_features.py [options]
```

options can be the following:

-f

Print a “feature” file as expected by the **-f** option of the `glimmer-mg`

--gbk *filename*

filename specifies the GenBank RefSeq annotation file from which to generate data files. Either this option or the combination of **--predict** and **--seq** must be specified.

--icm

Only produce the gene composition ICM model.

--indels

Note that the gene predictions may contain indels, which must be carefully considered when processing the predictions.

-l *n*

n specifies the minimum length of an ORF to be considered as a gene.

--predict *filename*

filename specifies `Glimmer` gene prediction output to be used for training. **--seq** must also be specified. Either this option or **--gbk** must be specified.

-o *n* or **--max_overlap** *n*

--rbs

Only produce the ribosomal binding site motif model.

--seq *filename*

-z

5.4.2 train_features.py Output Formats

.gicm File

Interpolated context model (ICM) trained on coding sequence from the genome given.

.gc.txt File

GC% of the genome given.

.lengths.<genes/non>.txt File

Counts of gene or noncoding ORF lengths from the genome given.

.starts.<genes/non>.txt File

Counts of start codon usage for the genes and noncoding ORFs from the genome given.

.motif File

Ribosomal binding site (RBS) position weight matrix motif model as generated by ELPH from the promoters of genes in the genome given.

.adj_orients.<genes/non>.txt File

Counts of gene or noncoding adjacent gene orientations. 1 specifies an ORF in the forward direction, and -1 specifies an ORF in the reverse direction. Thus, 1, -1 specifies the number of times a forward gene is followed by a reverse gene.

.adj_dist.<1/-1>.<1/-1>.<genes/non>.txt File

Counts of distances between adjacent genes or noncoding ORFs. 1 specifies an ORF in the forward direction, and -1 specifies an ORF in the reverse direction. Thus, 1, -1 specifies the distance counts when a forward gene is followed by a reverse gene. In the noncoding case, the distance counts refer to each pair of a noncoding ORF and the genes adjacent to it.

5.5 train_all.py

Run `train_features.py` on all GenBank reference genomes in Phymm's database.

5.5.1 train_all.py Parameters & Options

The invocation for `train_all.py` is:

```
train_all.py [options]
```

options can be the following:

-p *n*

Spread the processes launched over *n* processes.

-u

Only train on GenBank reference genomes for which no data files have yet been produced.

5.6 double_icms.py

Generate ICMs trained on pairs of GenBank reference genome annotations. Only pairs of reference genomes with similar composition are generated. The similarity function is defined in the GLIMMER-MG manuscript.

6 Notes on Glimmer programs

6.1 build-icm Program

This program constructs an interpolated context model (ICM) from an input set of sequences.

6.1.1 build-icm Parameters & Options

The format for invoking `build-icm` is:

```
build-icm [options] output-file < input-file
```

Sequences are reads from standard input, the ICM is built and written to *output-file*. If *output-file* is “-”, then the output will be sent to standard output. Since input comes from standard input, one also can “pipe” the input into this program, *e.g.*,

```
cat abc.in | build-icm xyz.icm
```

or even type in the input directly.

Possible *options* are:

-d num or **--depth num**

Set the depth of the ICM to *num*. The depth is the maximum number of positions in the context window that will be used to determine the probability of the predicted position. The default value is 7.

-F or **--no_stops**

Do not use any input strings with in-frame stop codons. Stop codons are determined by either the **-z** or **-Z** option.

-h or **--help**

Print the usage message.

-p num or **--period num**

Set the period of the ICM to *num*. The period is the number of different submodels for different positions in the text in a cyclic pattern. *E.g.*, if the period is 3, the first submodel will determine positions 1, 4, 7, ...; the second submodel will determine positions 2, 5, 8, ...; and the third submodel will determine positions 3, 6, 9, For a non-periodic model, use a value of 1. The default value is 3.

-r or **--reverse**

Use the reverse of the input strings to build the ICM. Note that this is merely the reverse and NOT the reverse-complement. In other words, the model is built in the backwards direction.

-t or **--text**

Output the model in a text format. This is for informational/debugging purposes only—the **glimmer3** program cannot read models in this form.

The format of the output is a header line containing the parameters of the model, followed by individual probability lines. The entries on each probability line are:

Column	Description
1	ID number
2	Context pattern
3	Mutual information
4	Probability of A
5	Probability of C
6	Probability of G
7	Probability of T

The context pattern is divided into codons by the vertical lines (this option assumes the default 3-periodic model). The “?” represents the position being predicted. Letters represent specific values in their respective positions in the context window. The asterisk indicates the position that has maximum mutual information with the predicted position.

-v *num* or **--verbose** *num*

Set the verbose level to *num*. This controls extra debugging output—the higher the value the more output.

-w *num* or **--width** *num*

Set the width of the ICM to *num*. The width includes the predicted position. The default value is 12.

-z *n* or **--trans_table** *n*

Use Genbank translation table number *n* to specify stop codons.

-Z *codon-list* or **--stop_codons** *codon-list*

Specify stop codons as a comma-separated list. Sample format: **-Z** tag,tga,taa. The default stop codons are tag, tga and taa.

6.2 long-orfs Program

This program identifies long, non-overlapping open reading frames (orfs) in a DNA sequence file. These orfs are very likely to contain genes, and can be used as a set of training sequences for the **build-icm** program. More specifically, among all orfs longer than a minimum length ℓ , those that do not overlap any others are output. The start codon used for each orf is the first possible one. The program, by default, automatically determines the value ℓ that maximizes the number of orfs that are output. With the **-t** option, the initial set of candidate orfs also can be filtered using entropy distance, which generally produces a larger, more accurate training set, particularly for high-GC-content genomes. Entropy distance is described in [OZWS04].

6.2.1 long-orfs Parameters & Options

The format for invoking `long-orfs` is:

```
long-orfs [options] sequence output
```

where *sequence* is the name of the file containing the DNA sequence to be analyzed and *output* is the name of the output file of coordinates. *sequence* may contain only one sequence. If *output* is “-”, then the output is directed to standard output.

Possible *options* are:

`-A codon-list` or `--start_codons codon-list`

Specify allowable start codons as a comma-separated list. Sample format:

`-A atg,gtg`. The default start codons are `atg`, `gtg` and `ttg`.

`-E filename` or `--entropy filename`

Read entropy profiles from *filename*. The format is one header line, then 20 lines of 3 columns each, which is the format produced by the program `entropy-profile` with the `-b` option. The columns are amino acid, positive entropy, and negative entropy, respectively. Rows must be in alphabetical order by amino acid code letter.

The entropy profiles are used only if the `-t` option is specified.

`-f` or `--fixed`

Do *NOT* automatically calculate the minimum gene length that maximizes the number or length of output regions, but instead use either the value specified by the `-g` option or else the default, which is 90.

`-g n` or `--min_len n`

Set the minimum gene length to *n* nucleotides. This does not include the bases in the stop codon.

`-h` or `--help`

Print the usage message.

`-i filename` or `--ignore filename`

filename specifies regions of bases that are off limits, so that no bases within that area will be examined. The format for entries in this file is described above for the same option in the `glimmer3` program.

`-l` or `--linear`

Assume a linear rather than circular genome, *i.e.*, there will be no “wraparound” genes with part at the beginning of the sequence and the rest at the end of the sequence.

-L or **--length_opt**

Find and use as the minimum gene length the value that maximizes the total *length* of non-overlapping genes, instead of the default behaviour, which is to maximize the total *number* of non-overlapping genes.

-n or **--no_header**

Do not include the program-settings header information in the output file. With this option, the output file will contain only the coordinates of the selected orfs.

-o *n* or **--max_olap** *n*

Set the maximum overlap length to *n*. Overlaps of this many or fewer bases between genes are not regarded as overlaps.

-t *x* or **--cutoff** *x*

Only genes with an entropy distance score less than *x* will be considered. This cutoff is made before any subsequent steps in the algorithm.

-w or **--without_stops**

Do *NOT* include the stop codon in the region described by the output coordinates. By default it is included.

-z *n* or **--trans_table** *n*

Use Genbank translation table number *n* to specify stop codons.

-Z *codon-list* or **--stop_codons** *codon-list*

Specify allowable stop codons as a comma-separated list. Sample format:
-Z tag,tga. The default stop codons are tag, tga and taa.

7 Versions

7.1 Version 0.10

- Initial release.

References

- [BS09] A. Brady and S.L. Salzberg. Phymm and phymmbl: metagenomic phylogenetic classification with interpolated markov models. *Nature Methods*, 6(9):673–676, 2009.
- [DBPS07] A.L. Delcher, K.A. Bratke, E.C. Powers, and S.L. Salzberg. Identifying bacterial genes and endosymbiont dna with glimmer. *Bioinformatics*, 23(6):673, 2007.

- [DHK⁺99] A. Delcher, D. Harmon, S. Kasif, O. White, and S. Salzberg. Improved microbial gene identification with GLIMMER. *Nucl. Acids Res.*, 27(23):4636–4641, 1999.
- [KS10] D.R. Kelley and S.L. Salzberg. Clustering metagenomic sequences with interpolated markov models. *BMC Bioinformatics*, 11(1):544, 2010.
- [OZWS04] Z. Ouyang, H. Zhu, J. Wang, and S.Z. She. Multivariate entropy distance method for prokaryotic gene identification. *J. Bioinformatics & Comp. Biol.*, 2(2):353–373, 2004.
- [SDKW98] S. Salzberg, A. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucl. Acids Res.*, 26(2):544–548, 1998.