GLIMMER Release Notes
Version 3.02

Arthur L. Delcher

9 May 2006

1 Introduction

This document describes Version 3.02 of the GLIMMER gene-finding software. This version incorporates a nearly complete rewrite of the code, resulting in improvements in both sensitivity and specificity of the predictions.

This is a complete version of the software with all features implemented. Users discovering problems or errors are encouraged to report them to `adelcher@umiacs.umd.edu`.

2 About Glimmer

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. Version 1 of GLIMMER was described in [SDKW98] and Version 2 was described in [DHK+99]. An article describing GLIMMER3 is in preparation.

GLIMMER3 is released as OSI Certified Open Source Software under the Artistic License. The license is contained in the file, `LICENSE`, in the distribution.

3 What's Changed from Version 2 to Version 3

Changes have been made in the algorithms to score and select genes in GLIMMER3, and also in the options and output formats:

1. In both GLIMMER2 and GLIMMER3, orfs are scored, and those scoring above the threshold value form the candidate set.

   In GLIMMER2, pairwise overlaps between these candidates are examined, and using a series of rules, orfs are eliminated or start sites adjusted. This continues in an iterative fashion until no further changes occur. In many cases, the rules cannot resolve an overlap between two orfs, and both are output in the final list of predictions, which have comment tags indicating this.

   In GLIMMER3, a single dynamic-programming, HMM-like algorithm is used to select the highest-scoring orfs and their start sites. This algorithm guarantees that the predictions have no overlaps longer than the specified length (which can be set by the `-o` option). Thus, there are no longer any comments with the GLIMMER3 predictions, and in general there are fewer predictions, reducing the false-positive rate. Out tests indicate that there is no corresponding increase in false negatives for GLIMMER3 compared to GLIMMER2.

2. GLIMMER3 scores orfs in the reverse direction, *i.e.*, 3' to 5'. This improves the accuracy of scores near the start codon of genes because the trailing context of the ICM is in the coding region of the gene (on which it has been trained).

3. The `long-orfs` program now uses an amino-acid distribution model to filter the set of candidate orfs before a subset of sufficiently long, non-overlapping orfs is selected.

4. The `make` system and directory structure has been revised so that the source, object and executable files are now in separate directories.

5. There have been some changes in program parameters, including:

   (a) Program options are now specified *before* required parameters, rather than after. Most options now have a long form in addition to the single letter form.

   (b) `build-icm` uses a parameter to specify the output file for the ICM, instead of sending it to standard output like GLIMMER2. This parameter can be "-" to direct output to standard out, if desired.

   (c) `glimmer3` requires a third parameter, which is used as a prefix for its output files.

6. There have been some changes in the format and/or meaning of output values. Specifically:

   (a) GLIMMER3 produces two output files: a file with detailed information about all orfs (similar to the first part of GLIMMER2 output), and a file containing just the final predictions (like the second part of GLIMMER2 output).

   (b) The prediction coordinates in GLIMMER3 now include the stop codon. Thus the end coordinates will differ from GLIMMER2 values by 3.

   (c) Orfs are now printed with a score, which is 100 times the log odds per base of the in-frame coding score versus the score of the independent, non-coding model. These scores provide a consistent scale to compare scores of different orfs.

   (d) The `-X` option will now report genes extending past the end of a sequence with a coordinate that is either less than or equal to zero, or greater than the sequence length.

7. GLIMMER3 can now process multiple-sequence input files. The outputs for each sequence are preceded by the fasta-header line of the sequence in both the `.detail` and the `.predict` files.

8. Two GLIMMER2 options have been eliminated:

   `-p` Was used to specify acceptable overlaps of genes as a percentage of their lengths. This is problematic since the choice of start site affects gene length.

   `-w` Specified the minimum length of an orf that might be considered a gene based on scores of intersecting orfs. Setting a suitably low score threshold (with the `-t` option) effectively includes these orfs.

4 Installing and Running Glimmer3

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

4.1 Installation

To install GLIMMER3, download the compressed tarfile `glimmer302.tar.gz` from the website. Then uncompress the file by typing

```
tar xzf glimmer302.tar.gz
```

A directory named `glimmer3.02` should result. In that directory, is a subdirectory named `src`. Within the `src` subdirectory type

```
make
```

(or alternately `gmake`). This will compile the GLIMMER3 programs and put the executable files in the directory `glimmer3.02/bin`. These files can be copied or moved to whatever directory is convenient to the user.

4.1.1 Troubleshooting

If the make fails, one possibility is that long options are not installed on your system. To compile without long options, edit file `delcher.hh` in directory `src/Common` to change line

```
#define  ALLOW_LONG_OPTIONS  1
```

near the top of the file to

```
#define  ALLOW_LONG_OPTIONS  0
```

and then retry make. It also may be necessary to comment-out or delete the line

```
#include  <getopt.h>
```

in this file. If you turn off long options then only the single-letter form of program options will work.

Another reason the make may fail is if your version of make does not support all the features of GNU make. If this is the case, you can try an alternative, simplified version of the make system by going to directory

```
glimmer3.02/SimpleMake
```

and type `make` there.

4.2 Running Glimmer

Running GLIMMER is a two-step process. First, a probability model of coding sequences, called an interpolated context model or ICM, must be built. This is done by the program `build-icm` from a set of training sequences. These sequences can be obtained in several ways:

1. From known genes in the genome, *e.g.*, genes identified by homology searches
2. From long, non-overlapping orfs in the genome as produced by the program `long-orfs`.
3. From genes in a highly similar species/strain.

Once the probability model is built, the `glimmer3` program itself is run to analyze the sequences and make gene predictions. `glimmer3` has a number of different options that affect its predictions. One of these (`-b`) provides the program with a position weight matrix (PWM) representing the ribosome binding site for genes and is used to improve the accuracy of start site predictions.

To obtain the best results with GLIMMER, the largest possible training set of genes should be used from the same genome on which predictions are to be made. If genes are known from homology searches, they can be used. If only a few such genes are available, they can be combined with the training set produced by the `long-orfs` program (but do not include duplicate genes in the training set). If you are running GLIMMER on small genome fragments, the genome of the nearest available evolutionary relative of the target organism can be used to provide a training set of genes.

4.2.1 Speed & Memory Usage

The speed and memory usage of GLIMMER3 programs will depend on the system speed and the size and nature of the data files. The `build-icm` program takes time roughly proportional to the size of its input file. On a 3.0GHz Intel Xeon Linux system, using default parameters, it takes roughly 10 seconds per megabyte of input. Its memory requirement is less than 50 Mb for bacterial-size genomes. The run-time of the `glimmer3` program depends both on the size of the input genome and the number of potential genes in it. High-GC genomes, which have more long open reading frames, take longer to process than low-GC genomes. Again, using a 3.0GHz Intel Xeon Linux system as a benchmark, `glimmer3` on *Campylobacter jejuni RM1221* (1.77Mb, 30.3takes about 15 seconds and uses less than 8Mb of memorey; *Pseudomonas fluorescens Pf-5* (7.07Mb, 63takes less than 2 minutes and uses about 27Mb of memory.

4.3 Useful Scripts

In the `scripts` subdirectory are several C-shell scripts that are useful for running GLIMMER3. At the top of each script are specified the directory paths to the GLIMMER executables and Awk scripts (the lines beginning with `set glimmerpath` and `set awkpath`). The user will need to change these entries to the directories where these files were installed on his/her system. The first lines of these files may also need to be modified if the user's `csh` and `awk` programs are in a directory other than `/bin`.

`g3-from-scratch.csh` is a sample shell script that first uses program `long-orfs` to find a training set of (putative) genes and then runs `glimmer3` on the result. It may be desirable to change the `glimmer3` options on the `set glimmeropts` line.

To run the script, say, on the genome sequence in file `genom.seq` and prefix the output files with the tag `run1`, simply type:

```
g3-from-scratch.csh genom.seq run1
```

The script would then run the commands:

```
long-orfs -n -t 1.15 genom.seq run1.longorfs
extract -t genom.seq run1.longorfs > run1.train
build-icm -r run1.icm < run1.train
glimmer3 -o50 -g110 -t30 genom.seq run1.icm run1
```

`g3-from-training.csh` is a sample shell script that uses a given set of gene coordinates to extract a training set and then run `glimmer3`. This script uses the program `elph` (available from TIGR at `www.tigr.org/software/ELPH`) to create a PWM from the region upstream of the start sites in the specified coordinate sets. It also uses the first codons in the training set to estimate the start-codon distribution for the genome.

To run the script on the genome sequence in file `genom.seq`, with file `train.coords` containing the positions of the training sequences in `genom.seq`, and using tag `run2` to prefix the output files, type:

```
g3-from-training.csh genom.seq train.coords run2
```

The script would then run the commands:

```
extract -t genom.seq train.coords > run2.train
build-icm -r run2.icm < run2.train
upstream-coords.awk 25 0 train.coords | extract genom.seq - > run2.upstream
elph run2.upstream LEN=6 | get-motif-counts.awk > run2.motif
set startuse = `start-codon-distrib -3 genom.seq train.coords`
glimmer3 -o50 -g110 -t30 -b run2.motif -P $startuse genom.seq run2.icm run2
```

`g3-iterated.csh` is a shell script that combines the two preceding scripts. It uses the predictions from the scratch run to create a training set for the second run. The reason for a second run is that the output from the first run will have a more accurate set of start sites than the output from the `long-orfs` program, which automatically uses the most upstream start site. These start sites allow the creation of a PWM for the ribosome binding site and the estimation of start-codon usage in the genome.

To run the script on the genome sequence in file `genom.seq` and prefix the output files with the tag `run3`, type:

```
g3-iterated.csh genom.seq run3
```

The script would then run the commands:

```
long-orfs -n -t 1.15 genom.seq run3.longorfs
extract -t genom.seq run3.longorfs > run3.train
build-icm -r run3.icm < run3.train
glimmer3 -o50 -g110 -t30 genom.seq run3.icm run3.run1
```

```
tail +2 run3.run1.predict > run3.coords
upstream-coords.awk 25 0 run3.coords | extract genom.seq - > run3.upstream
elph run3.upstream LEN=6 | get-motif-counts.awk > run3.motif
set startuse = `start-codon-distrib -3 genom.seq run3.coords`
glimmer3 -o50 -g110 -t30 -b run3.motif -P $startuse genom.seq run3.icm run3
```

Several Awk scripts, including those called by the above scripts, are in the same directory, `scripts`, as these C-shell scripts. Each script has a comment at the beginning describing what it does.

5 Sample Run Directory

A directory containing a sample run of GLIMMER3 is provided. This directory, named `sample-run` contains the genome sequence for *Treponema pallidum* (file `tpall.fna`) and a list of annotated genes for it (file `tpall.nh`), both downloaded from GenBank. The files whose names begin `from-scratch` are the result of running the script

```
g3-from-scratch.csh tpall.fna from-scratch
```

The files whose names begin `from-training` are the result of running the script

```
g3-from-training.csh tpall.fna tpall.nh from-training
```

The files whose names begin `iterated` are the result of running the script

```
g3-iterated.csh tpall.fna iterated
```

Users will need to modify the path directories at the top of these scripts to be able to run them (see Section 4.3 above).

6 Notes on the 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 $1, 4, 7, \ldots$; the second submodel will determine positions $2, 5, 8, \ldots$; and the third submodel will determine positions $3, 6, 9, \ldots$. 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 `glimmer3` Program

This is the main program that makes gene preditions.

6.2.1 `glimmer3` Parameters & Options

The invocation for `glimmer3` is:

`glimmer3` [*options*] *sequence icm tag*

where *sequence* is the name of the file containing the DNA sequence(s) to be analyzed and *icm* is the name of the file containing the ICM model produced by `build-icm`. *tag* is a prefix used to name the two output files: *tag* `.detail` and *tag* `.predict`.

options can be the following:

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

Specify start codons as a comma-separated list. Sample format: `-A atg,gtg`. The default start codons are `atg`, `gtg` and `ttg`. Use the `-P` option to specify the relative proportions of use. If `-P` is not used, then the proportions will be equal.

`-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` *p* or `--gc_percent` *p*

Use *p* as the GC percentage of the independent model, *i.e.*, the model of intergenic sequence. Note: *p* should be a percentage, *e.g.*, `-C 45.2`

If this option is not specified, the GC percentage will be counted from the input file.

`-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. This currently does not affect GLIMMER3 predictions, but is used in the `long-orfs` program. If the option is specified, the entropy-distance ratio for each potential gene is printed as the last column of the `.detail` file. If *filename* is "`#`", then a set of default entropy profiles, constructed from a wide range of species, is used.

`-f` or `--first_codon`

Use the first possible codon in an orf as the start codon for initial scoring purposes. Otherwise, the highest-scoring codon will be used. This only affects the start positions in the `.detail` file. The final start predictions in the `.predict` file are always based on the scoring functions.

`-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` *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 one line per region, with the start and end positions of the region specified as the first two fields on the line. The rest of the line is regarded as comments. Additionally, any line beginning with a `#` is regarded as a comment. *E.g.*, the following file:

```
   1001     1600   Comment here
# The region can be specified high-low as well as low-high
   5600     5001
```

would ignore bases 1001 . . . 1600 and 5001 . . . 5600 in the input sequence. This option should not be used with multi-sequence input files.

`-l` or `--linear`

Assume a linear rather than circular genome, *i.e.*, there will be no genes that "wraparound" between the beginning and end of the sequence.

`-L` *filename* or `--orf_coords` *filename*

filename specifies a list of orfs that should be scored separately, with no attempt to resolve overlaps or determine start codons. The format of the list is one orf per line, with entries separated by white space. The first entry is an identifier for the orf. It can be an arbitrary string without spaces. The next two entries are the start and end positions of the orf, respectively, (coordinates counting from 1), including the stop codon. The fourth entry is the reading frame. This is used only to determine the direction of the orf in cases of circular genomes where the orf might "wrap around" the end of the input sequence. If positive the orf is presumed to be on the positive DNA strand; otherwise, on the negative strand. Any further entries on the line are ignored.

The output with this option goes both to the `.predict` file and to the `.detail` file.

`-M` or `--separate_genes`

sequence-file is a multifasta file of separate genes to be scored separately, with no overlap rules. Each sequence is assumed to be in $5'$ to $3'$ order and to include the stop codon.

`-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 <u>*never*</u> output genes that overlap by more than this many bases.

`-P` *number-list* or `--start_probs` *number-list*

Specify the probability of different start codons (same number and order as in `-A` option). If no `-A` option is given, then there should be 3 values: for `atg`, `gtg` and `ttg`, in that order. Sample format: `-P 0.6,0.35,0.05`. If `-A` is specified without `-P`, then each start codon is equally likely (which is very unusual).

`-q` n or `--ignore_score_len` n

Consider any gene n or more bases long as a potential gene, regardless of its in-frame score. Without this option, this value is calculated automatically to be the length such that the expected number of orfs this long or longer in a random sequence of a million bases is one.

`-r` or `--no_indep`

Don't use the independent probability score column at all. Using this option will produce more short gene predictions.

`-t` *n* or `--threshold` *n*

Set the threshold score for consideration as a gene to *n*. If the in-frame score $\geq n$, then the region is given a number and considered a potential gene. Note this is the integer score in the column labelled "InFrm" in the `.detail` file, not the decimal score in the column labelled "Raw".

`-X` or `--extend`

Also score orfs that extend off the end of the sequence(s). This option presumes that the sequence(s) is linear and not circular. Reported positions off the end of the sequence are the nearest positions in the correct reading frame. Note that this ignores any partial codons at the ends of a sequence. Suppose, for example, that a sequence is 998bp long and an orf in reading frame +1 starts at position 601 and extends off the end of the sequence. Then the end of that gene/orf will be reported at position 999, as if the stop codon were in positions 997...999. This is true even if the last two characters of the sequence are, say, `cc` and cannot possibly be part of a stop codon.

Any scores associated with orfs that extend past the end of a sequence are computed using only complete codons contained in the sequence.

`-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`.

6.2.2 `glimmer3` Output Formats

`.detail` File

The `.detail` file begins with the command that invoked the program and a list of the parameters used by the program. Here is a sample:

```
Command:  /fs/szgenefinding/Glimmer3/bin/glimmer3 -o 50 -g 110 -t 30 -b iterated.motif -P
0.603,0.338,0.059 tpall.fna iterated.icm iterated

Sequence file = tpall.fna
Number of sequences = 1
ICM model file = iterated.icm
Excluded regions file = none
List of orfs file = none
Input is NOT separate orfs
Independent (non-coding) scores are used
Circular genome = true
Truncated orfs = false
Minimum gene length = 110 bp
Maximum overlap bases = 50
Threshold score = 30
Use first start codon = false
Start codons = atg,gtg,ttg
```

```
Start probs = 0.603,0.338,0.059
Stop codons = taa,tag,tga
GC percentage = 52.8%
Ignore score on orfs longer than 799
```

Following that, for each sequence in the input file the fasta-header line is echoed and followed by a list of orfs that were long enough for `glimmer3` to score. Here is a sample of the beginning of such a section:

```
>gi|15638995|ref|NC_000919.1| Treponema pallidum subsp. pallidum str. Nichols, complete ge
nome
Sequence length = 1138011
```

		Start			Length		Scores							
ID	Frame	of Orf	of Gene	Stop	of Orf	of Gene	Raw	InFrm	F1	F2	F3	R1	R2	R3	NC
	+2	17	20	139	120	117	-4.94	0	99	0	-	0	-	-	0
	+2	140	242	361	219	117	0.99	0	87	0	-	12	-	-	0
	-1	435	417	148	285	267	5.48	2	97	-	-	2	-	-	0
	+2	668	668	790	120	120	2.89	0	99	0	-	-	-	-	0
	-3	899	839	717	180	120	-0.86	1	95	-	-	-	-	1	3
	-1	936	933	808	126	123	0.38	13	78	-	-	13	-	-	8
	-3	1124	1109	918	204	189	-1.32	0	99	-	-	-	-	0	0
0001	+1	4	4	1398	1392	1392	6.61	99	99	-	-	-	-	-	0
	-2	1750	1720	1457	291	261	-0.92	8	-	-	-	-	8	-	91
	-2	1957	1945	1751	204	192	-1.47	1	-	-	70	-	1	-	27
	-3	2078	2063	1908	168	153	-1.88	4	-	-	20	-	-	4	75
	-2	2308	2293	2174	132	117	-0.38	5	-	-	85	-	5	-	9
0002	+3	1542	1641	2756	1212	1113	3.20	99	-	-	99	-	-	-	0
	-3	2807	2774	2616	189	156	-2.08	3	0	-	-	-	-	3	96

Below is a description of the columns. All positions are counted from the beginning of the sequence with the first base being position 1.

`ID` An identification number for a potential gene. Only orfs whose in-frame (`InFrm`) score is above the threshold score (set by the `-t` option) or are longer than the ignore-score length have an entry in this column.

`Frame` The reading frame of the orf—positive for forward strand, negative for reverse strand. It is determined by the position of the leftmost base of the stop codon:

frame $+1$ if the stop begins in position $1, 4, 7, \ldots$;

frame $+2$ if the stop begins in position $2, 5, 8, \ldots$;

frame $+3$ if the stop begins in position $3, 5, 9, \ldots$;

frame -1 if the stop begins in position $3, 5, 9, \ldots$ (so the leftmost base is position $1, 4, 7, \ldots$);

frame -2 if the stop begins in position $4, 7, 10, \ldots$ (left base position $2, 5, 8, \ldots$);

frame -3 if the stop begins in position $5, 8, 11, \ldots$ (left base position $3, 6, 9 \ldots$).

Note that if the genome length is not a multiple of 3, for genes that wrap around the end of the sequence the same rules applied to the start codon position will not yield the same reading frame.

`Start` The positions of the first base of the orf and the first base of the start codon of the gene. Note that the gene start may be different for the same orf in the `.predict` file.

`Stop` Position of the last base of the stop codon.

`Length` Number of bases in the orf and in the gene. It does *NOT* include the bases of the stop codon.

`Raw` Score This is 100 times the per-base log-odds ratio of the in-frame coding ICM score to the independent (*i.e.*, non-coding) model score. It gives a rough quantification to how well an orf scores that can be compared between any two orfs.

`InFrm` Score The normalized (to the range 0...99) score of the gene in its reading frame. This is just the appropriate-frame value among the next six scores.

Frame Scores The normalized (to the range 0...99) score of the gene in each reading frame. A "`-`" indicates the presence of a stop codon in that reading frame. The normalization compares only scores without stop codons and the independent (non-coding) `NC` score. If the orf is sufficiently long, *i.e.*, longer than the value stated in "`Ignore score on orfs longer than...`", the score is not used.

`NC` Score The normalized independent (*i.e.*, non-coding or intergenic) model score. This model is adjusted for the fact that the orf, by definition, has no in-frame stop codons.

`EDR` Score An additional column of scores is produced if the `-E` option is specified. This is the entropy-distance ratio, *i.e.*, the ratio of the distance of the amino-acid distribution from a positive model to the distance from a negative model. Scores below 1.0 are more likely to be genes; scores above 1.0 less likely to be genes. It is not currently used in the scoring process.

`.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:

```
>gms:3447|cmr:632 chromosome 1 {Mycobacterium smegmatis MC2}
orf00001      499     1692  +1    13.14
orf00004     1721     2614  +2    14.20
orf00006     2624     3778  +2    10.35
orf00009     3775     4359  +1     9.34
```

The columns are:

Column 1 The identifier of the predicted gene. The numeric portion matches the number in the `ID` column of the `.detail` file.

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 per-base "raw" score of the gene. This is slightly different from the value in the `.detail` file, because it includes adjustments for the PWM and start-codon frequency.

6.3 `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].

6.3.1 `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`.

6.4 Other Programs

A number of other utility programs are included in the GLIMMER3 package. For all of these programs, running the program with the "`-h`" option, will give a brief description of the program usage and options.

6.4.1 `anomaly` Program

This program reads a genome sequence and list of gene coordinates for it and reports genes with bad start codons, bad stop codons, in-frame stop codons, or frame shifts.

`anomaly` [*options*] *sequence coords*

6.4.2 `build-fixed` Program

This program builds a fixed-length interpolated context model from a set of sequences. The sequences must all be the same length. The model is actually an array of separate ICM's, one for each position in the fixed-length sequences.

`build-fixed` [*options*] `<` *sequence* `>` *output-model*

6.4.3 `entropy-profile` Program

This program builds a multi-fasta list of gene sequences and determines the natural and entropy distributions of all amino acid residues contained in them.

`entropy-profile` [*options*] `<` *sequences*

6.4.4 `entropy-score` Program

This program reads a genome sequence and a list of gene coordinates for it and computes the entropy distance ratio for each gene. Output goes to standard output and is the same as the coordinate input with the entropy ratio appended to each line.

`entropy-score` [*options*] *sequence coords*

6.4.5 `extract` Program

This program reads a genome sequence and a list of coordinates for it and outputs a multi-fasta file of the regions specified by the coordinates. Output goes to standard output.

`extract` [*options*] *sequence coords*

6.4.6 `multi-extract` Program

This program is a multi-fasta version of the preceding program. The only difference is that the input sequence file can be a multi-fasta file, and accordingly, the coordinate file must have an extra field (at the beginning) that specifies to which sequence the coordinates refer.

`multi-extract` [*options*] *sequences coords*

6.4.7 `score-fixed` Program

This program scores a set of fixed-length input sequences using two fixed-length interpolated context models. Output goes to standard output.

`score-fixed` [*options*] *pos-model neg-model* `<` *sequences*

6.4.8 `start-codon-distrib` Program

This program reads a genome sequence and list of coordinates for it and frequencies of the start codons of the genes. Output goes to standard output.

`start-codon-distrib` [*options*] *sequence coords*

6.4.9 `uncovered` Program

This program reads a genome sequence and list of coordinates for it and outputs a multi-fasta file contained the regions of the sequences that are *NOT* contained in any of the regions specified in the coordinates file. Output goes to standard output.

`uncovered` [*options*] *sequence coords*

6.4.10 `window-acgt` Program

This program finds the distribution of nucleotides in each of a series of windows across a DNA sequence. The command-line parameters specify the width of the window and the distance between successive windows. The input sequence comes from standard input and the output goes to standard output.

`window-acgt` [*options*] *window-len window-skip* `<` *input-file*

7 Versions

7.1 Version 3.01

- Eliminated unused functions.
- Eliminated `-p` and `-w` options.

- Implemented the `-X` option allowing orfs extending off the end (of a non-circular) sequence to be scored.
- Changed the width of the PWM in the scripts from 5 to 6.
- Added the `g3-iterated` script to combine running GLIMMER3 from scratch and using the output as a training set for a second run.
- Lowered default threshold score (`-t` option) in scripts.

7.2 Version 3.02

- Correct error in handling ORFs that wrap around the start/end of circular sequences.
- Change the make system to work on Mac OSX.
- Implement the `-L` and `-M` options.
- Change the orf scoring not to score the start codon with the ICM or with the independent score model.

References

[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.

[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.