HMMER3 alpha test: User's Guide

Biological sequence analysis using profile hidden Markov models

http://hmmer.org/
Version 3.0a1; January 2009

Sean R. Eddy
Howard Hughes Medical Institute
Janelia Farm Research Campus
19700 Helix Drive
Ashburn VA 20147 USA
http://eddylab.org/

Contents

1 Introduction

This is a user's guide to the HMMER3 alpha test distribution.

It really isn't meant for a new user. I will assume you are already familiar with profile hidden Markov models (profile HMMs) (Krogh et al., 1994; Eddy, 1998; Durbin et al., 1998); with the previous version of HMMER [HMMER2, `http://hmmer.org`]; with other popular biological sequence comparison tools, such as BLAST (Altschul et al., 1997); and with running sequence analysis tools on a UNIX or UNIX-like command line. If this isn't true of you, you should probably not be using the HMMER3 code yet. Instead, you should wait for a later and more stable version, when the user documentation will take less for granted.

Design goals of HMMER3

HMMER3's objective is to combine the power of probabilistic inference with BLAST's speed. We aim to upgrade some of molecular biology's most important sequence analysis applications to use more powerful statistical inference engines, while not sacrificing computational performance.

Specifically, HMMER3 has three main design features that, in combination, distinguish it from previous tools:

Explicit representation of alignment uncertainty. Most sequence alignment analysis tools report only a single best-scoring alignment. However, sequence alignments are uncertain, and the more distantly related sequences are, the more uncertain alignments become. HMMER3 calculates complete posterior alignment ensembles rather than single optimal alignments. Posterior ensembles get used for a variety of useful inferences involving alignment uncertainty. For example, any HMMER3 sequence alignment is accompanied by posterior probability annotation, representing the degree of confidence in each individual aligned residue.

Sequence scores, not alignment scores. Alignment uncertainty has an important impact on the power of sequence database searches. It's precisely the most remote homologs – the most difficult to identify and potentially most interesting sequences – where alignment uncertainty is greatest, and where the statistical approximation inherent in scoring just a single best alignment breaks down the most. To maximize power to discriminate true homologs from nonhomologs in a database search, statistical inference theory says you ought to be scoring sequences by integrating over alignment uncertainty, not just scoring the single best alignment. HMMER3's log-odds scores are sequence scores, not just optimal alignment scores, integrated over the posterior alignment ensemble.

Speed. A major limitation of previous profile HMM implementations (including HMMER2) was their slow performance. HMMER3 implements a new heuristic acceleration algorithm. For most queries, it's about as fast as BLAST.

Individually, none of these points is new. As far as alignment ensembles and sequence scores go, pretty much the whole reason why hidden Markov models are so theoretically attractive for sequence analysis is that they are good probabilistic models for explicitly dealing with alignment uncertainty. The SAM profile HMM software from UC Santa Cruz has always used full probabilistic inference (the HMM "Forward" and "Backward" algorithms) as opposed to optimal alignment scores (the HMM "Viterbi" algorithm). HMMER2 had the full HMM inference algorithms available as command-line options, but used Viterbi alignment by default, in part for speed reasons. Calculating alignment ensembles is even more computationally intensive than calculating single optimal alignments.

One reason why it's been hard to deploy sequence scores for practical large-scale use is that we haven't known how to accurately calculate the statistical significance of a log-odds score that's been integrated over alignment uncertainty. Accurate statistical significance estimates are essential when one is trying to discriminate homologs from millions of unrelated sequences in a large sequence database search. The statistical significance of optimal alignment scores can be calculated by Karlin/Altschul statistics (Karlin and

Altschul, 1990, 1993). Karlin/Altschul statistics are one of the most important and fundamental advances introduced by BLAST. However, this theory doesn't apply to integrated log-odds sequence scores (HMM "Forward scores"). The statistical significance (expectation values, or E-values) of HMMER3 sequence scores is determined by using recent theoretical conjectures about the statistical properties of integrated log-odds scores which have been supported by numerical simulation experiments (Eddy, 2008).

And as far as speed goes, there's really nothing new about HMMER3's speed either. Besides Karlin/Altschul statistics, the other big reason BLAST has been so useful is that it's so fast. Our design goal in HMMER3 was to achieve rough speed parity between BLAST and more formal and powerful HMM-based methods. The acceleration algorithm in HMMER3 is a new heuristic. It seems likely to be more sensitive than BLAST's heuristics on theoretical grounds, and it certainly benchmarks that way in practice (Eddy, 2009, manuscript in preparation). Additionally, it's very well suited to modern hardware architectures. We expect to be able to take good advantage of GPUs and other parallel processing environments in the near future.

What made it into the HMMER3.0 alpha test code

Single sequence queries: new to HMMER3

phmmer	Search a sequence against a sequence database. (BLASTP-like)

Replacements for HMMER2's functionality

hmmbuild	Build a profile HMM from an input multiple alignment.
hmmsearch	Search a profile HMM against a sequence database.
hmmscan	Search a sequence against a profile HMM database.
hmmalign	Make a multiple alignment of many sequences to a common profile HMM.

Other utilities

hmmconvert	Convert profile formats to/from HMMER3 format.
hmmemit	Generate (sample) sequences from a profile HMM.
hmmfetch	Get a profile HMM by name or accession from an HMM database.
hmmpress	Format an HMM database into a binary format for `hmmscan`.
hmmstat	Show summary statistics for each profile in an HMM database.

The quadrumvirate of `hmmbuild/hmmsearch/hmmscan/hmmalign` is supposed to suffice to replace HMMER2's core functionality of `hmmbuild/hmmsearch/hmmpfam/hmmalign` in people's domain analysis and annotation pipelines, for instance using profile databases like Pfam or SMART. These four programs have already been subjected to some serious independent testing by Rob Finn of the Pfam Consortium, who is visiting at Janelia for several months, preparing to adopt HMMER3 at Pfam, apparently by beating the living tar out of it. I haven't yet fixed all the issues the hated Rob has identified, but I have fixed the showstopping bugs.[1]

The `phmmer` program is new to HMMER3. It searches a single sequence against a sequence database, akin to BLASTP. (Internally, it just produces a profile HMM from the query sequence, then runs an HMM search.) An additional program, `jackhmmer`, has narrowly missed being supported as part of the initial alpha test release; there's still some final work to do there. `jackhmmer` is an iterative sequence search program, essentially like `PSI-BLAST`.

In the Tutorial section, I'll show examples of running each of these programs, using examples in the `tutorial/` subdirectory of the distribution.

[1] I think.

What's still missing

Oh, lots. The most egregious lacunae include:

More parallelization. HMMER3 is designed to be parallelized at three levels: SIMD vector instructions within the CPU, multithreaded (probably using OpenMP) across processor cores, and MPI across individual nodes of a cluster. Relatively little of this potential has been realized. The SIMD parallelization is part of what's scientifically new and perhaps most interesting in HMMER3, because SIMD parallelization is at the heart of the new acceleration algorithm. However, for now HMMER's new SIMD "MSV" algorithm is implemented only for Intel x86 compatible architectures. We will port it to other SIMD vector architectures soon. No multithreading is implemented yet; and MPI parallelization is only implemented in `hmmbuild`.

More processor support. One of the attractive features of the MSV algorithm is that it is a very tight and efficient piece of code, which ought to be able to take advantage of recent advances in using massively parallel GPUs (graphics processing units), and other specialized processors such as the Cell processor, or FPGAs. We have prototype work going on in a variety of processors, but none of this is far along as yet. But this work (combined with the parallelization) is partly why we expect to wring significant more speed out of HMMER in the future.

More speed. Even on x86 platforms, HMMER3's acceleration algorithms are still on a nicely sloping bit of their asymptotic optimization curve. I still think I can accelerate the code by another two-fold or so. Additionally, for a small number of HMMs ($< 1\%$ of Pfam models), the acceleration core is performing relatively poorly, for reasons I pretty much understand (having to do with biased composition; most of these pesky models are hydrophobic membrane proteins), but which are nontrivial to work around. This'll produce an annoying behavior that some testers are sure to notice: if you look systematically, sometimes you'll see a model that runs at something more like HMMER2 speed, 100x or so slower than an average query. This, needless to say, Will Be Fixed.

DNA sequence comparison. HMMER's search pipeline is somewhat specialized to protein/protein comparison: specifically, the pipeline works by filtering individual sequences, winnowing down to a subset of the sequences in a database that need close attention from the full heavy artillery of Bayesian inference. This strategy doesn't work for long DNA sequences; it doesn't filter the human genome much to say "there's a hit on chromosome 1". The algorithms need to be adapted to identify high-scoring regions of a target sequence, rather than filtering by whole sequence scores. (You can chop a DNA sequence into overlapping windows and HMMER3 would work fine on such a chopped-up database, but that's a disgusting kludge and I don't want to know about it.)

Translated comparisons. We'd of course love to have the HMM equivalents of BLASTX, TBLASTN, and TBLASTX. They'll come.

More sequence input formats. HMMER3 will work fine with FASTA files for unaligned sequences, and Stockholm files for multiple sequence alignments. It has parsers for a handful of other formats (Genbank, EMBL, and Uniprot flatfiles; SELEX format alignments) that we've tested somewhat. It's particularly missing parsers for some widely used alignment formats such as Clustal format, so using HMMER3 on the MSAs produced by many popular multiple alignment programs (MUSCLE or MAFFT for example) is harder than it should be, because it requires a reformat to Stockholm format.

More alignment modes. HMMER3 *only* does local alignment. HMMER2 also could do glocal alignment (align a complete model to a subsequence of the target) and global alignment (align a complete model to a complete target sequence). The E-value statistics of glocal and global alignment remain poorly understood. HMMER3 relies on accurate significance statistics, far more so than HMMER2 did, because HMMER3's acceleration pipeline works by filtering out sequences with poor P-values.

Part of the reason for the alpha test is to confirm that these points are just as annoying to you as they are to me, and therefore important to fix asap. Feel free to tell me you want these things even though I already know about them. I also want to find out what glaring problems you find that I'm *not* already losing sleep over. (Really.)

What I hope to accomplish in alpha test

The core of HMMER3's functionality seems stable to me, but all the stuff wrapped around it – the stuff *you* see, like the applications, command line options, i/o formats – is prototypical and still fluid. The main objective of the alpha test period is for a small number of savvy power users to have the opportunity to give feedback while the user-oriented layers of HMMER3 are still under development – in particular, before its basic feature set, command line options, and input and output formats get frozen. You might want it to spit out XML, or you like tab-delimited format, or you want this number or that number on such-and-such a line to make it really fit in your analysis pipeline, or you really really need a command line option for slowing the search programs back down to HMMER2 speed so you have more time for coffee[2]. This is the kind of stuff I'd most like to hear now while the code is still fluid.

An obvious corollary of this responsiveness to your feedback is, **don't write any heavy duty output parsers around HMMER3 just yet.** You should expect all the output formats to change, at least slightly, before a public release is finalized.

Of course, since it's alpha test code, I'd like to also hear about bugs: how you manage to break it, or when it produces inconsistent, wrong, or confusing results, or when it doesn't compile or run at all. Some bugs, I already know about, but I'd still like to hear about them just to know you care.

Cryptogenomicon (`http://cryptogenomicon.org/`) is a blog where I'll be talking about issues as they arise in HMMER3, and where you can comment or follow the discussion.

[2] Actually, this option already exists: `--max`.

2 Installation

Quick installation instructions

Download `hmmer-3.0a1.tar.gz` from `http://hmmer.org/`, or directly from `ftp://selab.janelia.org/pub/software/hmmer3/hmmer-3.0a1.tar.gz`; untar; and change into the newly created directory `hmmer-3.0a1`:

```
> wget ftp://selab.janelia.org/pub/software/hmmer3/hmmer-3.0a1.tar.gz
> tar xf hmmer-3.0a1.tar.gz
> cd hmmer-3.0a1
```

The alpha test code includes precompiled binaries for x86/Linux platforms. These are in the `binaries` directory. You can just stop here if you like, if you're on a x86/Linux machine and you want to try the programs out without installing them.

To compile new binaries from source, do a standard GNUish build:

```
> ./configure
> make
```

To compile and run a test suite to make sure all is well, you can optionally do:

```
> make check
```

All these tests should pass.

You don't have to install HMMER programs to run them. The newly compiled binaries are now in the `src` directory; you can run them from there. To install the programs and man pages somewhere on your system, do:

```
> make install
```

By default, programs are installed in `/usr/local/bin` and man pages in `/usr/local/man/man1/`. You can change `/usr/local` to any directory you want using the `./configure --prefix` option, as in `./configure --prefix /the/directory/you/want`.

If you have the Intel C compiler `icc`, we strongly recommend that you use it (instead of `gcc`, for example), for performance reasons, by specifying CC=icc either in your environment or on the `> ./configure` command line.

For example, on our systems, we would do:

```
> ./configure CC=icc LDFLAGS=-static --prefix=/usr/local/hmmer-3.0/
> make
> make check
> make install
```

System requirements

Operating system: HMMER is designed to run on UNIX platforms, including Linux and MacOS/X. The code is POSIX-compliant, meaning it should run on any POSIX-compliant operating system. This essentially includes all operating systems except Microsoft Windows.

The alpha test code includes precompiled binaries for Linux. These were compiled with the Intel C compiler (`icc`) on an x86_64 Intel platform running Red Hat Enterprise Linux AS4. We believe they should be widely portable to different Linux systems.

We have tested most extensively on Linux, and to a lesser extent on MacOS/X. We aim to be portable to all other POSIX platforms. We currently do not plan to develop for Windows.[3]

Processor: HMMER3 currently runs effectively only on x86 (IA32/IA64) processors that support the SSE2 vector instruction set. This includes all Intel processors from the Pentium 4 on, and all AMD processors from the K8 (Athlon) on.

[3] Though I'd expect it should compile and run fine on Windows, especially using add-on products that are available for making Windows more POSIX-compliant.

We aim to be portable to all modern processors. The acceleration algorithms are designed to be portable despite their use of specialized SIMD vector instructions, and to take advantage of multiple levels of parallelization available on modern hardware. HMMER3 has prototype code for processors that use VMX/Altivec vector instructions, including PowerPC processors and the Cell processor. We expect that this processor support will appear in the 3.0 public release. We expect to add support for the Sun SPARC VIS instruction set at some point. We believe that the code will be able to take advantage of GP-GPUs and FPGAs in the future.

Compiler: The source code is C, conforming to POSIX and ANSI C99 standards. It should compile with any ANSI C99 compliant compiler, including the GNU C compiler `gcc`. We have tested the code using both the `gcc` and `icc` compilers.

If you compile HMMER from source, we strongly recommend using the Intel C compiler `icc` rather than `gcc`. `icc` is free for noncommercial use and heavily discounted for academic use. HMMER3 makes extensive use of SIMD vector instructions (SSE, Intel's Streaming SIMD Extensions). `gcc` is not as effective at compiling SSE code. It produces HMMER programs that are significantly slower than what `icc` produces.

Libraries and other installation requirements: HMMER includes a software library called Easel, which it will automatically compile during its installation process. By default, HMMER3 does not require any additional libraries to be installed by you, other than standard ANSI C99 libraries that should already be present on a system that can compile C code.

One of the objectives of the alpha test is to identify portability issues. If HMMER3 fails to compile and/or run under a POSIX-compliant OS, on an x86 processor, using an ANSI C99-compliant compiler, please report the problem.

Experimental: MPI support

Optionally, a couple of programs in HMMER3 include support for MPI (Message Passing Interface) parallelization. To use MPI, you first need to have MPI installed, such as OpenMPI (`www.open-mpi.org`), and you add `--enable-mpi` to the `./configure` command line. HMMER3's MPI support is not completely written at this time, and has not been extensively tested yet.

One place it's useful is in parallelizing `hmmbuild`.

It's highly advantageous to get `mpicc` to use the Intel C compiler rather than its default, which is often `gcc`. Different MPI distributions may have different ways of selecting the C compiler and its options. OpenMPI can be controlled by environment variables. For example, in our environment, we currently build HMMER3 for MPI using:

```
> setenv OMPI_MPICC "icc"
> setenv OMPI_MPICC_CFLAGS "-O3"
> setenv OMPI_MPICC_LDFLAGS "-static"
> ./configure --enable-mpi --prefix=/usr/local/hmmer3
> make
```

3 Tutorial

Here's a tutorial walk-through of some small projects with HMMER3. This should suffice to get you oriented to a "safe path" through HMMER3 alpha test code that should work as advertised – before you start boldly exploring later-to-be-documented command line options that might or might not be doing anything sensible yet.

Files used in the tutorial

The subdirectory `/tutorial` in the HMMER distribution contains the files used in the tutorial, as well as a number of examples of various file formats that HMMER reads. The important files for the tutorial are:

`globins4.sto` An example alignment of four globin sequences, in Stockholm format. This alignment is a subset of a famous old published structural alignment from Don Bashford (Bashford et al., 1987).

`globins4.hmm` An example profile HMM file, built from `globins4.out`, in HMMER3 ASCII text format.

`globins4.out` An example `hmmsearch` output file that results from searching the `globins4.hmm` against Uniprot 7.0.

`fn3.sto` An example alignment of 106 fibronectin type III domains. This is the Pfam 22.0 `fn3` seed alignment. It provides an example of a Stockholm format with more complex annotation. We'll also use it for an example of `hmmsearch` analyzing sequences containing multiple domains.

`fn3.hmm` A profile HMM created from `fn3.sto` by `hmmbuild`.

`7LESS_DROME` A FASTA file containing the sequence of the *Drosophila* Sevenless protein, a receptor tyrosine kinase whose extracellular region is thought to contain seven fibronectin type III domains.

`fn3.out` Output of `hmmsearch fn3.hmm 7LESS_DROME`.

`Pkinase.sto` The Pfam 22.0 Pkinase seed alignment of protein kinase domains.

`minifam` An example HMM flatfile database, containing three models: `globins4`, `fn3`, and `Pkinase`.

`minifam.h3{m,i,f,p}` Binary compressed files corresponding to `minifam`, produced by `hmmpress`.

`HBB_HUMAN` A FASTA file containing the sequence of human $\beta-$hemoglobin, used as an example query for `phmmer`.

Searching a sequence database with a single profile HMM

Step 1: build a profile HMM with hmmbuild

HMMER starts with a multiple sequence alignment file that you provide. Currently HMMER3 only reads Stockholm alignments.[4] The file `tutorial/globins4.sto` is an example of a simple Stockholm file. It looks like this:

[4] I'm lying. It can read more. I just don't trust its other parsers yet.

```
# STOCKHOLM 1.0

HBB_HUMAN   ........VHLTPEEKSAVTALWGKV....NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVL
HBA_HUMAN   .........VLSPADKTNVKAAWGKVGA..HAGEYGAEALERMFLSFPTTKTYFPHF.DLS.....HGSAQVKGHGKKVA
MYG_PHYCA   .........VLSEGEWQLVLHVWAKVEA..DVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKHGVTVL
GLB5_PETMA  PIVDTGSVAPLSAAEKTKIRSAWAPVYS..TYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERII

HBB_HUMAN   GAFSDGLAHL...D..NLKGTFATLSELHCDKL..HVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANAL
HBA_HUMAN   DALTNAVAHV...D..DMPNALSALSDLHAHKL..RVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVL
MYG_PHYCA   TALGAILKK....K.GHHEAELKPLAQSHATKH..KIPIKYLEFISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDI
GLB5_PETMA  NAVNDAVASM..DDTEKMSMKLRDLSGKHAKSF..QVDPQYFKVLAAVIADTVAAG.........DAGFEKLMSMICILL

HBB_HUMAN   AHKYH......
HBA_HUMAN   TSKYR......
MYG_PHYCA   AAKYKELGYQG
GLB5_PETMA  RSAY.......
//
```

Most popular alignment formats are similar block-based formats, and can be turned into Stockholm format with a little editing or scripting. Don't forget the `# STOCKHOLM 1.0` line at the start of the alignment, nor the `//` at the end. Stockholm alignments can be concatenated to create an alignment database flatfile containing many alignments.

The `hmmbuild` command builds a profile HMM from an alignment (or HMMs for each of many alignments in a Stockholm file), and saves the HMM(s) in a file. For example, type:

```
> hmmbuild globins4.hmm tutorial/globins4.sto
```

and you'll see some output that looks like:

```
# hmmbuild :: profile HMM construction from a multiple sequence alignment
# HMMER 3.0a1 (January 2009); http://hmmer.org/
# Copyright (C) 2008 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# input alignment file:             globins4.sto
# output HMM file:                  globins4.hmm
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# idx name                  nseq  alen  mlen  description
#---- -------------------- ----- ----- -----  -----------
1     globins4                 4   171   148

# CPU time: 0.37u 0.00s 00:00:00.37 Elapsed: 00:00:01
```

If your input file had contained more than one alignment, you'd get one line of output for each model. For instance, a single `hmmbuild` command suffices to turn a Pfam seed alignment flatfile (such as `Pfam-A.seed`) into a profile HMM flatfile (such as `Pfam.hmm`).

The information on these lines is almost self-explanatory. The `globins4` alignment consisted of 4 sequences with 171 aligned columns. HMMER turned it into a model of 148 consensus positions, which means it defined 23 gap-containing alignment columns to be insertions relative to consensus.

This output format is rudimentary. HMMER3 knows quite a bit more information about what it's done to build this HMM. Some of this information is likely to be useful to you, the user. As H3 testing and development proceeds, we're likely to expand the amount of data that `hmmbuild` reports.

The new HMM was saved to `globins4.hmm`. If you were to look at this file (and you don't have to – it's intended for HMMER's consumption, not yours), you'd see something like:

```
HMMER3/a [3.0a1 | January 2009]
NAME  globins4
LENG  148
ALPH  amino
RF    no
CS    no
MAP   yes
DATE  Fri Jan  9 11:58:38 2009
NSEQ  4
EFFN  0.962402
CKSUM 247350336
STATS LOCAL     VLAMBDA 0.709641
STATS LOCAL         VMU -9.735310
```

```
STATS LOCAL        FTAU -4.078249
HMM          A        C        D        E        F        G        H        I    ...   W        Y
           m->m     m->i     m->d     i->m     i->i     d->m     d->d
  COMPO   2.37156  4.52373  2.96553  2.70511  3.20656  3.01795  3.40301  2.91175 ... 4.55553  3.62859
          2.68606  4.42250  2.77504  2.73148  3.46379  2.40510  3.72519  3.29311 ... 4.58502  3.61528
          0.34002  1.27034  4.89152  1.48427  0.25705  0.00000        *
      1   2.62751  4.46690  3.32467  2.83143  3.63123  3.49811  2.73769  3.02526 ... 5.05801  3.76691     10 - -
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354 ... 4.58477  3.61503
          0.02324  4.16917  4.89152  0.61958  0.77255  0.48576  0.95510
...
    148   2.93123  5.12550  3.29461  2.66392  4.49241  3.60571  2.49799  3.89555 ... 5.42846  4.19770    165 - -
          2.68633  4.42241  2.77535  2.73099  3.46369  2.40470  3.72510  3.29370 ... 4.58492  3.61421
          0.20946  1.66612        *  1.49476  0.25399  0.00000        *
//
```

The HMMER3 ASCII save file format is defined in Section 4.

If you're used to HMMER2, now you may be expecting to calibrate the model with H2's `hmmcalibrate` program. HMMER3 models no longer need a separate calibration step. We've figured out how to calculate the necessary parameters rapidly, bypassing the need for costly simulation (Eddy, 2008). The determination of the statistical parameters is part of `hmmbuild`. These are the parameter values λ, μ, and τ on the lines marked `STATS`.

You also may be expecting to need to configure the model's alignment mode, as in HMMER2's `hmmbuild -f` option for building local "fragment search" alignment models, for example. HMMER3's `hmmbuild` does not have these options. `hmmbuild` builds a "core profile", which the search and alignment programs configure as they need to. At least for the moment, they always configure for local alignment.

Step 2: search the sequence database with hmmsearch

Presumably you have a sequence database to search. Here I'll use the Uniprot 7.0 Swissprot FASTA format flatfile (not provided in the tutorial, because of its large size), `uniprot_sprot.fasta`. If you don't have a sequence database handy, run your example search against `tutorial/globins45.fa` instead, which is a FASTA format file containing 45 globin sequences.

`hmmsearch` accepts any FASTA file as input. It also accepts EMBL/Uniprot text format. It will automatically determine what format your file is in; you don't have to say. An example of searching a sequene database with our `globins4.hmm` model would look like:

```
> hmmsearch globins4.hmm uniprot_sprot.fasta > globins4.out
```

Depending on the database you search, the output file `globins4.out` should look more or less like the example of a Uniprot search output provided in `tutorial/globins4.out`.

The first section is the *header* that tells you what program you ran, on what, and with what options:

```
# hmmsearch :: search profile HMM(s) against a sequence database
# HMMER 3.0a1 (January 2009); http://hmmer.org/
# Copyright (C) 2008 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query HMM file:                  globins4.hmm
# target sequence database:        uniprot_sprot.fasta
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       globins4  [M=148]
Scores for complete sequences (score includes all domains):
```

The second section is the *sequence top hits* list. It is a list of ranked top hits (sorted by E-value, most significant hit first), formatted in a BLAST-like style:

```
 --- full sequence ---   --- best 1 domain ---    -#dom-
  E-value  score  bias    E-value  score  bias    exp  N  Sequence    Description
  ------- ------ -----    ------- ------ -----   ---- --  --------    -----------
  2.1e-64  219.4   0.1    2.3e-64  219.2   0.0    1.0  1  HBB_GORGO   (P02024) Hemoglobin beta subunit
  2.8e-64  219.0   3.2    3.1e-64  218.8   2.2    1.0  1  MYG_PHYCA   (P02185) Myoglobin
    3e-64  218.9   0.0    3.4e-64  218.7   0.0    1.0  1  HBB_HUMAN   (P68871) Hemoglobin beta subunit
    3e-64  218.9   0.0    3.4e-64  218.7   0.0    1.0  1  HBB_PANPA   (P68872) Hemoglobin beta subunit
    3e-64  218.9   0.0    3.4e-64  218.7   0.0    1.0  1  HBB_PANTR   (P68873) Hemoglobin beta subunit
    5e-64  218.2   0.1    5.5e-64  218.0   0.1    1.0  1  HBB_HYLLA   (P02025) Hemoglobin beta subunit
  7.7e-64  217.5   0.2    8.6e-64  217.4   0.1    1.0  1  HBB_COLBA   (P02033) Hemoglobin beta subunit
```

The last two columns, obviously, are the name of each target sequence and optional description.

The most important number here is the first one, the *sequence E-value*. This is the statistical significance of the match to this sequence: the number of hits we'd expect to score this highly in a database of this size if the database contained only nonhomologous random sequences. The lower the E-value, the more significant the hit.

The E-value is based on the *sequence bit score*, which is the second number. This is the log-odds score for the complete sequence. Some people like to see a bit score instead of an E-value, because the bit score doesn't depend on the size of the sequence database, only on the profile HMM and the target sequence.

The next number, the *bias*, is a correction term for biased sequence composition that's been applied to the sequence bit score.[5] The only time you really need to pay attention to this value is when it's large, and on the same order of magnitude as the sequence bit score. This might be a sign that the target sequence isn't really a homolog, but merely shares a similar strong biased composition with the query model. The biased composition correction usually works well, but occasionally will not knock down a falsely "significant" nonhomologous hit as far as it should.

The next three numbers are again an E-value, score, and bias, but only for the single best-scoring domain in the sequence, rather than the sum of all its identified domains. The rationale for this isn't apparent in the globin example, because all the globins in this example consist of only a single globin domain. So let's set up a second example, using a model of a single domain that's commonly found in multiple domains in a single sequence. Build a fibronectin type III domain model using the `tutorial/fn3.sto` alignment (this happens to be a Pfam seed alignment; it's a good example of an alignment with complex Stockholm annotation). Then use that model to analyze the sequence `tutorial/7LESS_DROME`, the *Drosophila* Sevenless receptor tyrosine kinase:

```
> hmmbuild fn3.hmm tutorial/fn3.sto
> hmmsearch fn3.hmm tutorial/7LESS_DROME > fn3.out
```

An example of what that output file will look like is provided in `tutorial/fn3.out`. The sequence top hits list says:

```
--- full sequence ---   --- best 1 domain ---    -#dom-
 E-value  score  bias    E-value  score  bias    exp  N  Sequence    Description
 ------- ------ -----    ------- ------ -----   ---- --  --------    -----------
 2.5e-56  174.0   0.1    2.1e-15   43.3   0.4    9.6  9  7LESS_DROME RecName: Full=Protein sevenless;
```

OK, now let's pick up the explanation where we left off. The total sequence score of 174.0 sums up *all* the fibronectin III domains that were found in the `7LESS_DROME` sequence. The "single best dom" score and E-value are the bit score and E-value as if the target sequence only contained the single best-scoring domain, without this summation.

The idea is that we might be able to detect that a sequence is a member of a multidomain family because it contains multiple weakly-scoring domains, even if no single domain is solidly significant on its own. On the other hand, if the target sequence happened to be a piece of junk consisting of a set of identical internal repeats, and one of those repeats accidentially gives a weak hit to the query model, all the repeats will sum up and the sequence score might look "significant" (which mathematically, alas, is the correct answer: the null hypothesis we're testing against is that the sequence is a *random* sequence of some base composition, and a repetitive sequence isn't random).

So operationally:

- if both E-values are significant ($<< 1$), the sequence is likely to be homologous to your query.
- if the full sequence E-value is significant but the single best domain E-value is not, the target sequence is probably a multidomain remote homolog; but be wary, and watch out for the case where it's just a repetitive sequence.

[5] The method that HMMER3 uses to compensate for biased composition is unpublished, and different from HMMER2. We will write it up when there's a chance.

OK, the sharp eyed reader asks, if that's so, then why in the globin4 output (all of which have only a single domain) do the full sequence bit scores and best single domain bit scores not exactly agree? For example, the top ranked hit, HBB_GORGO (gorilla β-hemoglobin) has a full sequence score of 219.4 and a single best domain score of 219.2. What's going on? What's going on is that the position and alignment of that domain is uncertain – in this case, only very slightly so, but nonetheless uncertain. The full sequence score is summed over all possible alignments of the globin model to the HBB_GORGO sequence. When HMMER3 identifies domains, it identifies what it calls an **envelope** bounding where the domain's alignment most probably lies. (More on this later, when we discuss the reported coordinates of domains and alignments in the next section of the output.) The "single best dom" score is calculated after the domain envelope has been defined, and the summation is restricted only to the ensemble of possible alignments that lie within the envelope. The fact that the two scores are slightly different is therefore telling you that there's a small amount of probability (uncertainty) that the domain lies somewhat outside the envelope bounds that HMMER has selected.

The two columns headed `#doms` are two different estimates of the number of distinct domains that the target sequence contains. The first, the column marked `exp`, is the *expected* number of domains according to HMMER's statistical model. It's an average, calculated as a weighted marginal sum over all possible alignments. Because it's an average, it isn't necessarily a round integer. The second, the column marked `N`, is the number of domains that HMMER3's domain postprocessing and annotation pipeline finally decided to identify, annotate, and align in the target sequence. This is the number of alignments that will show up in the domain report later in the output file.

These two numbers should be about the same. Rarely, you might see that they're wildly different, and this would usually be a sign that the target sequence is so highly repetitive that it's confused the H3 domain postprocessors. Such sequences aren't likely to show up as significant homologs to any sensible query in the first place.

The sequence top hits output continues until it runs out of sequences to report. By default, the report includes all sequences with an E-value of 10.0 or less.

Then comes the third output section, which starts with

```
Domain and alignment annotation for each sequence:
```

Now for each sequence in the top hits list, there will be a section of containing a table of where HMMER3 thinks all the domains are, followed by the alignment inferred for each domain. Let's use the `fn3` vs. 7LESS_DROME example, because it contains lots of domains, and is more interesting in this respect than the globin4 output. The domain table for 7LESS_DROME looks like:

```
>> 7LESS_DROME  RecName: Full=Protein sevenless;          EC=2.7.10.1;
     # bit score    bias    E-value ind Evalue hmm from   hmm to    ali from   ali to    env from   env to    ali-acc
   --- --------- ------- ---------- ---------- -------- --------    -------- --------    -------- --------    -------
     1      -2.3     0.0       0.37       0.37       60       72 ..      396      408 ..      395      410 ..    0.86
     2      42.0     0.0    5.2e-15    5.2e-15        2       83 ..      439      520 ..      437      521 ..    0.95
     3      15.1     0.0    1.3e-06    1.3e-06       14       82 ..      837      911 ..      827      914 ..    0.82
     4       3.6     0.0     0.0055     0.0055        6       35 ..     1205     1235 ..     1202     1258 ..    0.81
     5      22.9     0.0    4.8e-09    4.8e-09       13       79 ..     1312     1380 ..     1304     1385 ..    0.81
     6      -0.6     0.0       0.11       0.11       57       72 ..     1754     1769 ..     1747     1769 ..    0.87
     7      43.3     0.4    2.1e-15    2.1e-15        1       82 [.     1799     1888 ..     1799     1891 ..    0.89
     8      19.2     0.0    7.2e-08    7.2e-08        6       73 ..     1904     1966 ..     1900     1976 ..    0.90
     9      12.0     0.0    1.2e-05    1.2e-05        1       77 [.     1993     2098 ..     1993     2107 ..    0.74
```

Domains are reported in the order they appear in the sequence, not in order of their significance.

The bit score and bias values are as described above for sequence scores, but are the score of just one domain's envelope.

The first of the two E-values is the **conditional E-value**. This is an odd number, and it's not even clear we're going to keep it. Pay attention to what it means! It is an attempt to measure the statistical significance of each domain, *given that we've already decided that the target sequence is a true homolog*. It is the expected number of *additional* domains we'd find with a domain score this big in the set of sequences reported in the top hits list, if those sequences consisted only of random nonhomologous sequence outside the region that sufficed to define them as homologs.

The second number is the **independent E-value**: the significance of the sequence in the *whole* database search, if this were the only domain we had identified. It's exactly the same as the "best 1 domain" E-value in the sequence top hits list.

The different between the two E-values is not apparent in the 7LESS_DROME example because in both cases, the size of the search space as 1 sequence. There's a single sequence in the target sequence database (that's the size of the search space that the independent/best single domain E-value depends on). There's one sequence reported as a putative homolog in the sequence top hits list (that's the size of the search space that the conditional E-value depends on). A better example is to see what happens when we search Uniprot (7.0 contains 207132 sequences) with the fn3 model:

```
> hmmsearch fn3.hmm uniprot_sprot.fasta
```

(If you don't have Uniprot and can't run a command like this, don't worry about it - I'll show the relevant bits here.) Now the domain report for 7LESS_DROME looks like:

```
>> 7LESS_DROME  (P13368) Sevenless protein (EC 2.7.1.112)
     # bit score    bias    E-value ind Evalue hmm from   hmm to    ali from   ali to    env from   env to    ali-acc
   --- --------- ------- ---------- ---------- -------- --------    -------- --------    -------- --------    -------
     1      -2.3     0.0      2e+02    7.7e+04       60       72 ..      396      408 ..      395      410 ..    0.86
     2      42.0     0.0    2.7e-12    1.1e-09        2       83 ..      439      520 ..      437      521 ..    0.95
     3      15.1     0.0    0.00071       0.28       14       82 ..      837      911 ..      827      914 ..    0.82
     4       3.6     0.0        2.9    1.1e+03        6       35 ..     1205     1235 ..     1202     1258 ..    0.81
     5      22.9     0.0    2.6e-06      0.001       13       79 ..     1312     1380 ..     1304     1385 ..    0.81
     6      -0.6     0.0         59    2.3e+04       57       72 ..     1754     1769 ..     1747     1769 ..    0.87
     7      43.3     0.4    1.1e-12    4.3e-10        1       82 [.     1799     1888 ..     1799     1891 ..    0.89
     8      19.2     0.0    3.8e-05      0.015        6       73 ..     1904     1966 ..     1900     1976 ..    0.90
     9      12.0     0.0     0.0065        2.6        1       77 [.     1993     2098 ..     1993     2107 ..    0.74
```

Notice that everything's the same (it's the same target sequence, after all) *except* those E-values. The independent E-value is calculated assuming a search space of all 207132 sequences. For example, look at domain number 7, the highest scoring domain. When we only looked at a single sequence, its score of 43.3 bits has an E-value of 2.1e-15. When we search a database of 207132 sequences, a hit scoring 43.3 bits would be expected to happen 207132 times as often: 2.1e-15 $\times$ 207132 $=$ 4.3e-10. In this Uniprot search, 530 sequences were reported in the top hits list (with E-values ≤ 10). If we were to assume that all 530 are true homologs, x out the domain(s) that made us think that, and then went looking for *additional* domains in those 530 sequences, we'd be searching a smaller database of 530 sequences: the expected number of times we'd see a hit of 43.3 bits or better is now 2.6e-15 $\times$ 530 $=$ 1.1e-12.

So, operationally:

- If the independent E-value is significant ($<< 1$), that means that even this single domain *by itself* is such a strong hit that it suffices to identify the sequence as a significant homolog with respect to the size of the entire original database search. You can be confident that this is a homologous domain.
- Once there's one or more high-scoring domains in the sequence already, sufficient to decide that the sequence contains homologs of your query, you can look (with some caution) at the conditional E-value to decide the statistical significance of additional weak-scoring domains.

In this case, for example, I'd be pretty sure of four of the domains (2, 5, 7, and 8), each of which has a strong enough independent E-value to declare *7LESS_DROME* to be an fnIII-domain-containing protein. Domains 3 and 9 wouldn't be significant if they were all I saw in the sequence, but in a small search space, their conditional E-values indicate that they are probably also fn3 domains. Domains 1, 4, and 6 are too weak to be sure of, from this search alone, but would be something to pay attention to and follow up on.

The next four columns give the endpoints of the reported local alignment with respect to both the query model ("hmm from" and "hmm to") and the target sequence ("ali from" and "ali to").

It's not immediately easy to tell from the "to" coordinate whether the alignment ended internally in the query or target, versus ran all the way (as in a full-length global alignment) to the end(s). To make this more readily apparent, with each pair of query and target endpoint coordinates, there's also a little symbology. `..` meaning both ends of the alignment ended internally, and `[]` means both ends of the alignment were

full-length flush to the ends of the query or target, and `[.` and `.]` mean only the left or right end was flush/full length.

The next two columns ("env from" and "env to") define the *envelope* of the domain's location on the target sequence. The envelope is almost always a little wider than what HMMER chooses to show as a reasonably confident alignment. As mentioned earlier, the envelope represents a subsequence that encompasses most of the posterior probability for a given homologous domain, even if precise endpoints are only fuzzily inferrable. You'll notice that for higher scoring domains, the coordinates of the envelope and the inferred alignment will tend to be in tighter agreement, corresponding to sharper posterior probability defining the location of the homologous region.

Operationally, I would use the envelope coordinates to annotate domain locations on target sequences, not the alignment coordinates. However, be aware that when two weaker-scoring domains are close to each other, envelope coordinates can and will overlap, corresponding to the overlapping uncertainty of where one domain ends and another begins. In contrast, alignment coordinates generally do not overlap (though there are cases where even they will overlap[6]).

The last column is the average posterior probability of the aligned target sequence residues; effectively, the expected accuracy per residue of the alignment.

For comparison, current Uniprot consensus annotation of Sevenless shows seven domains:

```
FT   DOMAIN      311    431       Fibronectin type-III 1.
FT   DOMAIN      436    528       Fibronectin type-III 2.
FT   DOMAIN      822    921       Fibronectin type-III 3.
FT   DOMAIN     1298   1392       Fibronectin type-III 4.
FT   DOMAIN     1680   1794       Fibronectin type-III 5.
FT   DOMAIN     1797   1897       Fibronectin type-III 6.
FT   DOMAIN     1898   1988       Fibronectin type-III 7.
```

Each of these overlaps with one of the domains identified by HMMER, plus HMMER calls an additional two domains (numbered 4 and 9 in its output).

Under the domain table, an "optimal posterior accuracy" alignment (Holmes, 1998) is computed within each domain's envelope, and displayed. For example, (skipping domain 1 because it's weak and unconvincing), fibronectin III domain 2 of `7LESS_DROME` is shown as:

```
== domain 2    score: 42.0 bits;  conditional E-value: 5.2e-15
                  ---CEEEEEEECTTEEEEEEE--S..SS--SEEEEEEEETTTCCGCEEEEEETTTSEEEEES--TT-EEEEEEEEEETTEE.E CS
          fn3   2 saptnlsvtevtstsltlsWsppe.gngpitgYeveyreknegeeekeltvpgtttsvtltgLkpgteYevrVqavngagegp 83
                  sap++++++++++l+++W+p++++ngpi+gY++++++++++ ++e++vp++++s+++++L++gt+Y+++++++n++gegp
 7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRfTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                  78888999999**************************9997.**********************************9997 PP
```

The initial header line starts with a `==` as a little handle for a parsing script to grab hold of. The rest of that line, we'll probably put more information on eventually.

If the model had any consensus structure or reference line annotation that it inherited from your multiple alignment (`#=GC SS_cons`, `#=GC RF` annotation in Stockholm files), that information is simply regurgitated as `CS` or `RF` annotation lines here. The `fn3` model had a consensus structure annotation line.

The line starting with `fn3` is the consensus of the query model. Capital letters represent the most conserved (high information content) positions. Dots (`.`) in this line indicate insertions in the target sequence with respect to the model.

The midline indicates matches between the query model and target sequence. A `+` indicates positive score, which can be interpreted as "conservative substitution", with respect to what the model expects at that position.

The line starting with `7LESS_DROME` is the target sequence. Dashes (`-`) in this line indicate deletions in the target sequence with respect to the model.

The bottom line is new to HMMER3. This represents the posterior probability (essentially the expected accuracy) of each aligned residue. A 0 means 0-5%, 1 means 5-15%, and so on; 9 means 85-95%, and a `*` means 95-100% posterior probability. You can use these posterior probabilities to decide which parts of the

[6]Not to mention one (mercifully rare) bug/artifact that I'm betting alpha testers don't even see an example of – but we'll see.

alignment are well-determined or not. You'll often observe, for example, that expected alignment accuracy degrades around locations of insertion and deletion, which you'd intuitively expect.

You'll also see expected alignment accuracy degrade at the ends of an alignment – this is because "alignment accuracy" posterior probabilities currently not only includes whether the residue is aligned to one model position versus others, but also confounded with whether a residue should be considered to be homologous (aligned to the model somewhere) versus not homologous at all.[7]

These domain table and per-domain alignment reports for each sequence then continue, for each sequence that was in the per-sequence top hits list.

Finally, at the bottom of the file, you'll see some summary statistics. For example, at the bottom of the globins search output, you'll find something like:

```
//
Internal statistics summary:
----------------------------
Query HMM(s):                            1  (148 nodes)
Target sequences:                   207132  (75438310 residues)
Passed MSV filter:                    7444  (0.0359384); expected 4142.6 (0.02)
Passed Vit filter:                    1602  (0.0077342); expected 207.1 (0.001)
Passed Fwd filter:                    1189  (0.0057403); expected 2.1 (1e-05)
Initial search space (seqZ):        207132  [actual number of target seqs]
Domain search space  (domZ):          1049  [number of seqs reported over threshold]
Mc/sec:                            1555.00
# CPU time: 7.18u 0.11s 00:00:07.29 Elapsed: 00:00:07
```

This gives you some idea of what's going on in HMMER3's acceleration pipeline. You've got one query HMM, and the database has 207,132 target sequences. Each sequence goes through a gauntlet of three scoring algorithms called MSV, Viterbi, and Forward, in order of increasing sensitivity and increasing computational requirement.

MSV (the "ungapped Multi Segment Viterbi" algorithm) is the new algorithm in HMMER3. It essentially calculates the HMM equivalent of BLAST's sum score – an optimal sum of ungapped high-scoring alignment segments. Unlike BLAST, it does this calculation directly, without BLAST's word hit or hit extension step, using a SIMD vector-parallel algorithm. By default, HMMER3 is configured to allow sequences with a P-value of $\leq 2\%$ through the MSV score filter (thus, if the database contained no homologs and P-values were accurately calculated, the highest scoring 2% of the sequences will pass the filter). Here, about 3.6% of the database got through the MSV filter.

The Viterbi filter then calculates a gapped optimal alignment score. This is a bit more sensitive than the MSV score, but the Viterbi filter is about three-fold slower than MSV. By default, HMMER3 lets sequences with a P-value of ≤ 0.001 through this stage. Here (because there's a little over a thousand true globin homologs in this database), much more than that gets through - 1602 sequences.

Then the full Forward score is calculated, which sums over all possible alignments of the profile to the target sequence. The default allows sequences with a P-value of $\leq 0.001\%$ through; 1189 sequences passed.

All sequences that make it through the three filters are then subjected to a full probabilistic analysis using the HMM Forward/Backward algorithms, first to identify domains and assign domain envelopes; then within each individual domain envelope, Forward/Backward calculations are done to determine posterior probabilities for each aligned residue, followed by optimal accuracy alignment. The results of this step are what you finally see on the output.

Recall the difference between conditional and independent E-values, with their two different search space sizes; these search space sizes are reported in the statistics summary.

Finally, it reports the speed of the search in units of Mc/sec (million dynamic programming cells per second), and the CPU time. This search took 7.2 seconds. That's in the same ballpark as BLAST, depending on which BLAST you compare to. On the same machine, an NCBI BLAST with one of these globin sequences took 3.7 seconds, and WU-BLAST took 8.5 seconds.

[7] It may make more sense to condition the posterior probabilities on the assumption that the residue is indeed homologous: given that, how likely is it that I've got it correctly aligned.

Searching a profile HMM database with a query sequence

The `hmmscan` program is for annotating all the different known/detectable domains in a given sequence. It takes a single query sequence and an HMM database as input. The HMM database might be Pfam, SMART, or TIGRFams, for example, or another collection of your choice.

Step 1: create an HMM database flatfile

An HMM "database" flatfile is simply a concatenation of individual HMM files. To create a database flatfile, you can either build individual HMM files and concatenate them, or you can concatenate Stockholm alignments and use `hmmbuild` to build an HMM database of all of them in one command.

Let's create a tiny database called `minifam` containing models of globin, fn3, and Pkinase (protein kinase) domains by concatenating model files:

```
> hmmbuild globins4.hmm tutorial/globins4.sto
> hmmbuild fn3.hmm tutorial/fn3.sto
> hmmbuild Pkinase.hmm tutorial/Pkinase.sto
> cat globins4.hmm fn3.hmm Pkinase.hmm > minifam
```

We'll use `minifam` for our examples in just a bit, but first a few words on other ways to build HMM databases, especially big ones. The file `tutorials/minifam` is the same thing, if you want to just use that.

Alternatively, you can concatenate Stockholm alignment files together (as Pfam does in its big `Pfam-A.seed` and `Pfam-A.full` flatfiles) and use `hmmbuild` to build HMMs for all the alignments at once. This won't work properly for our tutorial alignments, because the `globins4.sto` alignment doesn't have an `#=GF ID` annotation line giving a name to the globins4 alignment, so `hmmbuild` wouldn't know how to name it correctly. To build a multi-model database from a multi-MSA flatfile, the alignments have to be in Stockholm format (no other MSA format that I'm aware of supports having more than one alignment per file), and each alignment must have a name on a `#=GF ID` line.

But if you happen to have a Pfam seed alignment flatfile `Pfam-A.seed` around, an example command would be:

```
> hmmbuild Pfam-A.hmm Pfam-A.seed
```

This would take about four hours to build all 10,000 models or so in Pfam. To speed the database construction process up, `hmmbuild` supports MPI parallelization,

As far as HMMER's concerned, all you have to do is add `--mpi` to the command line for `hmmbuild`, assuming you've compiled support for MPI into it; see the installation instructions.

You'll also need to know how to invoke an MPI job in your particular environment, with your job scheduler and MPI distribution. We can't really help you with this – different sites have different cluster environments.

With our scheduler (SGE, the Sun Grid Engine) and our MPI distro (OpenMPI), an example incantation for building `Pfam.hmm` from `Pfam-A.seed` is:

```
> qsub -N hmmbuild -j y -o errors.out -b y -cwd -V -pe openmpi 200
'mpirun -mca btl self,tcp --prefix /usr/local/openmpi -np 200 ./hmmbuild --mpi Pfam.hmm Pfam-A.seed >
hmmbuild.out'
```

This reduces the time to build Pfam from about four hours to about a minute.

Step 2: compress and index the flatfile with hmmpress

The `hmmscan` program has to read a lot of profile HMMs in a hurry, and HMMER's ASCII flatfiles are bulky. To accelerate this, `hmmscan` uses binary compression and indexing of the flatfiles. To use `hmmscan`, you must first compress and index your HMM database with the `hmmpress` program:

```
> hmmpress minifam
```

This will quickly produce:

```
Working...    done.
Pressed and indexed 3 HMMs (3 names and 2 accessions).
Models pressed into binary file:   minifam.h3m
```

```
SSI index for binary model file:   minifam.h3i
Profiles (MSV part) pressed into:  minifam.h3f
Profiles (remainder) pressed into: minifam.h3p
```

and you'll see these four new binary files in the directory.

The `tutorial/minifam` example has already been pressed, so there are example binary files `tutorial/minifam.h3{m,i,f,p}` included in the tutorial.

Their format is "proprietary", which is an open source term of art meaning "haven't found time to document them yet".

Step 3: search the HMM database with hmmscan

Now we can analyze sequences using our HMM database and `hmmscan`.

For example, the receptor tyrosine kinase `7LESS_DROME` not only has all those fibronectin type III domains on its extracellular face, it's got a protein kinase domain on its intracellular face. Our `minifam` database has models of both `fn3` and `Pkinase`, as well as the unrelated `globins4` model. So what happens when we scan the `7LESS_DROME` sequence:

```
> hmmscan minifam tutorial/7LESS_DROME
```

The header and the first section of the output will look like:

```
# hmmscan :: search sequence(s) against a profile HMM database
# HMMER 3.0a1 (January 2009); http://hmmer.org/
# Copyright (C) 2009 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query sequence file:             tutorial/7LESS_DROME
# target HMM database:             tutorial/minifam
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       7LESS_DROME  [L=2554]
Accession:   P13368
Description: RecName: Full=Protein sevenless;          EC=2.7.10.1;
Scores for complete sequence (score includes all domains):
 --- full sequence ---   --- best 1 domain ---    -#dom-
  E-value  score  bias    E-value  score  bias    exp  N  Model    Description
  ------- ------ -----    ------- ------ -----   ---- --  -------- -----------
  7.4e-56  174.0   0.0    6.2e-15   43.3   0.4    9.6  9  fn3      Fibronectin type III domain
  5.9e-43  135.0   0.0    9.2e-43  134.4   0.0    1.3  1  Pkinase  Protein kinase domain
     0.75   -2.5   0.0          0    0.0   0.0    1.2  0  globins4
```

The output fields are in the same order and have the same meaning as in `hmmsearch`'s output.

The size of the search space for `hmmscan` is the number of models in the HMM database (here, 3; for a Pfam search, on the order of 10000). In `hmmsearch`, the size of the search space is the number of sequences in the sequence database. This means that E-values may differ even for the same individual profile vs. sequence comparison, depending on how you do the search.

For domain, there then follows a domain table and alignment output, just as in `hmmsearch`. The `fn3` annotation, for example, looks like:

```
>> fn3  Fibronectin type III domain
     # bit score    bias    E-value ind Evalue hmm from   hmm to    ali from   ali to    env from   env to    ali-acc
   --- --------- ------- ---------- ---------- -------- --------    -------- --------    -------- --------    -------
     1      -2.3     0.0        1.1        1.1       60       72 ..      396      408 ..      395      410 ..    0.86
     2      42.0     0.0    1.5e-14    1.5e-14        2       83 ..      439      520 ..      437      521 ..    0.95
     3      15.1     0.0      4e-06      4e-06       14       82 ..      837      911 ..      827      914 ..    0.82
     4       3.6     0.0      0.017      0.017        6       35 ..     1205     1235 ..     1202     1258 ..    0.81
     5      22.9     0.0    1.5e-08    1.5e-08       13       79 ..     1312     1380 ..     1304     1385 ..    0.81
     6      -0.6     0.0       0.34       0.34       57       72 ..     1754     1769 ..     1747     1769 ..    0.87
     7      43.3     0.4    6.2e-15    6.2e-15        1       82 [.     1799     1888 ..     1799     1891 ..    0.89
     8      19.2     0.0    2.2e-07    2.2e-07        6       73 ..     1904     1966 ..     1900     1976 ..    0.90
     9      12.0     0.0    3.7e-05    3.7e-05        1       77 [.     1993     2098 ..     1993     2107 ..    0.74
```

and an example alignment (of that second domain again):

```
== domain 2    score: 42.0 bits;  conditional E-value: 1.5e-14
                  ---CEEEEEEECTTEEEEEEE--S..SS--SEEEEEEEETTTCCGCEEEEEETTTSEEEEES--TT-EEEEEEEEETTEE.E CS
        fn3   2 saptnlsvtevtstsltlsWsppe.gngpitgYeveyreknegeeekeltvpgtttsvtltgLkpgteYevrVqavngagegp 83
                  sap++++++++++++l+++W+p++++ngpi+gY++++++++++ ++e++vp++++s+++++L++gt+Y+++++++n++gegp
7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRfTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                  78888999999**************************9997.*********************************9997 PP
```

You'd think (and you'd usually be right) that except for the E-values (which depend on database search space sizes), you should get exactly the same scores, domain number, domain coordinates, and alignment every time you do a search of the same HMM against the same sequence. And in this case, with the comparison of the `fn3` model and `7LESS_DROME`, that's what's happened. But it isn't guaranteed to happen. In some cases, particularly cases where exact domain number and boundaries are difficult to infer, HMMER3 uses stochastic sampling algorithms. Therefore you may see differences from run to run. We know this isn't what many people expect or like – even though probabilistically it's just another way that HMMER's more or less correctly reflecting true uncertainty of its inferences. We plan to implement some ways to make results more reproducible by default, by more precisely controlling the state of the pseudorandom number generators.

Creating multiple alignments with hmmalign

The file `tutorial/globins45.fa` is a FASTA file containing 45 unaligned globin sequences. To align all of these to the `globins4` model and make a multiple sequence alignment:

```
> hmmalign globins4.hmm tutorial/globins45.fa
```

The output of this is a Stockholm format multiple alignment file. The first few lines of it look like:

```
# STOCKHOLM 1.0

MYG_ESCGI          ...VLSDAEWQLVLNIWAKVEADVAGHGQDILIRLFKGHPETLEKFDKFK
#=GR MYG_ESCGI  PP ...69**********************************************
MYG_HORSE          ..g-LSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKFK
#=GR MYG_HORSE  PP ..7.89**********************************************
MYG_PROGU          ..g-LSDGEWQLVLNVWGKVEGDLSGHGQEVLIRLFKGHPETLEKFDKFK
#=GR MYG_PROGU  PP ..7.89**********************************************
MYG_SAISC          ..g-LSDGEWQLVLNIWGKVEADIPSHGQEVLISLFKGHPETLEKFDKFK
#=GR MYG_SAISC  PP ..7.89**********************************************
MYG_LYCPI          ..g-LSDGEWQIVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFK
#=GR MYG_LYCPI  PP ..7.89**********************************************
MYG_MOUSE          ..g-LSDGEWQLVLNVWGKVEADLAGHGQEVLIGLFKTHPETLDKFDKFK
#=GR MYG_MOUSE  PP ..7.89**********************************************
MYG_MUSAN          ..v-----DWEKVNSVWSAVESDLTAIGQNILLRLFEQYPESQNHFPKFK
#=GR MYG_MUSAN  PP ..7.....89**************************************9
...
```

and so on.

Notice those `PP` annotation lines. That's posterior probability annotation, as in the single sequence alignments that `hmmscan` and `hmmsearch` showed. This essentially represents the confidence that each residue is aligned where it should be.

`hmmalign` currently has an undesirable/problematic "feature" that we're aware of. Recall that HMMER3 only does local alignments. Here, we know that we've provided full length globin sequences, and `globins4` is a full length globin model. We'd really like `hmmalign` to produce a global alignment. It can't currently do that. If it doesn't quite manage to extend its local alignment to the full length of a target globin sequence, you'll get a weird-looking effect, as the nonmatching termini are pulled out to the left or right. For example, look at the N-terminal `g` in `MYG_HORSE` above. H3 is about 70% confident that this residue is nonhomologous, though any sensible person would align it into the first globin consensus column.

Look at the end of that first block of Stockholm alignment, where you'll see:

```
...
HBBL_RANCA         vhw--TAEEKAVINSVWQKV--DVEQDGHEALTRLFIVYPWTQRYFSTFG
#=GR HBBL_RANCA PP *65..899***********..****************************
HBB2_TRICR         ..vHLTAEDRKEIAAILGKV--NVDSLGGQCLARLIVVNPWSRRYFHDFG
#=GR HBB2_TRICR PP ..*79***************..****************************
#=GC PP_cons       ...799*********************************************
```

The `#=GC PP_cons` line is Stockholm-format *consensus posterior probability* annotation for the entire column. It's calculated simply as the arithmetic mean of the per-residue posterior probabilities in that column. This should prove useful in phylogenetic inference applications, for example, where it's common to mask away nonconfidently aligned columns of a multiple alignment. The `PP_cons` line provides an objective measure of the confidence assigned to each column.

Single sequence queries using phmmer

The `phmmer` program is for searching a single sequence query against a sequence database, much as `BLASTP` or `FASTA` would do. `phmmer` works essentially just like `hmmsearch` does, except you provide a query sequence instead of a query profile HMM.

Internally, HMMER builds a profile HMM from your single query sequence, using a simple position-independent scoring system (BLOSUM62 scores converted to probabilities, plus a gap-open and gap-extend probability).

The file `tutorial/HBB_HUMAN` is a FASTA file containing the human $\beta-$globin sequence as an example query. If you have a sequence database such as `uniprot_sprot.fasta`, make that your target database; otherwise, use `tutorial/globins45.fa` as a small example:

```
> phmmer tutorial/HBB_HUMAN uniprot_sprot.fa
```

or

```
> phmmer tutorial/HBB_HUMAN tutorial/globins45.fa
```

Everything about the output is essentially as previously described for `hmmsearch`.

4 File formats

HMMER profile HMM files

The file `tutorial/rrm.hmm` gives an example of a HMMER3 ASCII save file. An abridged version is shown here, where (. . .) mark deletions made for clarity and space:

```
HMMER3/a [3.0a1 | January 2009]
NAME  RRM_1
ACC   PF00076.13
DESC  RNA recognition motif. (a.k.a. RRM, RBD, or RNP domain)
LENG  72
ALPH  amino
RF    no
CS    yes
MAP   yes
DATE  Fri Jan  9 13:51:17 2009
NSEQ  79
EFFN  7.722980
CKSUM 512533816
STATS LOCAL     VLAMBDA 0.721565
STATS LOCAL         VMU -8.724843
STATS LOCAL        FTAU -4.034479
HMM          A        C        D        E        F        G        H        I     (...)    Y
           m->m     m->i     m->d     i->m     i->i     d->m     d->d
  COMPO   2.59889  4.44747  2.92534  2.57970  2.91651  2.81030  3.90624  2.78237  (...) 3.49702
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00484  5.72961  6.45195  0.61958  0.77255  0.00000        *
      1   3.33305  4.36957  6.20460  5.67416  4.57399  5.57414  6.09195  1.31189  (...) 5.28530      1 - E
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00484  5.72961  6.45195  0.61958  0.77255  0.48576  0.95510
      2   3.33570  4.71399  5.27635  4.66262  1.20335  3.96868  4.83458  2.71778  (...) 1.66170      2 - E
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00484  5.72961  6.45195  0.61958  0.77255  0.48576  0.95510
(...)
     71   3.08639  5.73256  2.87655  2.29760  4.27257  3.66073  3.11090  3.98349  (...) 3.32445     79 - E
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00484  5.72961  6.45195  0.61958  0.77255  0.48576  0.95510
     72   2.98106  3.47896  4.24834  4.59436  3.48125  4.49311  4.82169  1.78353  (...) 4.16621     80 - E
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00326  5.72803        *  0.61958  0.77255  0.00000        *
//
```

An HMM file consists of one or more HMMs. Each HMM starts with the identifier `HMMER3/a` and ends with `//` on a line by itself. The identifier allows backward compatibility as the HMMER software evolves: it tells the parser this file is from HMMER3's save file format version a. The closing `//` allows multiple HMMs to be concatenated.

The format is divided into two regions. The first region contains textual information and miscalleneous parameters in a roughly tag-value scheme. This section ends with a line beginning with the keyword `HMM`. The second region is a tabular, whitespace-limited format for the main model parameters.

All probability parameters are all stored as negative natural log probabilities with five digits of precision to the right of the decimal point, rounded. For example, a probability of 0.25 is stored as $-\log 0.25 = 1.38629$. The special case of a zero probability is stored as '*'.

Spacing is arranged for human readability, but the parser only cares that fields are separated by at least one space character.

A more detailed description of the format follows.

header section

The header section is parsed line by line in a tag/value format. Each line type is either **mandatory** or **optional** as indicated.

`HMMER3/a` Unique identifier for the save file format version; the `/a` means that this is HMMER3 HMM file format version a. If HMMER3 ever changes its save file format, the revision code will change from a to b. This way, parsers may easily remain backwards compatible. The remainder of the line after the `HMMER3/a` tag is free text that is ignored by the parser. HMMER currently writes its version number and release date in brackets here, e.g. `[3.0a1 | January 2009]` in this example. **Mandatory.**

`NAME <s>` Model name; `<s>` is a single word containing no spaces or tabs. The name is normally picked up from the `#=GF ID` line from a Stockholm alignment file. If this is not present, the name is created from the name of the alignment file by removing any file type suffix. For example, an otherwise nameless HMM built from the alignment file `rrm.slx` would be named `rrm`. **Mandatory.**

`ACC <s>` Accession number; `<s>` is a one-word accession number. This is picked up from the `#=GF AC` line in a Stockholm format alignment. **Optional.**

`DESC <s>` Description line; `<s>` is a one-line free text description. This is picked up from the `#=GF DE` line in a Stockholm alignment file. **Optional.**

`LENG <d>` Model length; `<d>`, a positive nonzero integer, is the number of match states in the model. **Mandatory.**

`ALPH <s>` Symbol alphabet type. For biosequence analysis models, `<s>` is `amino`, `DNA`, or `RNA` (case insensitive). There are also other accepted alphabets for purposes beyond biosequence analysis, including `coins`, `dice`, and `custom`. This determines the symbol alphabet and the size of the symbol emission probability distributions. If `amino`, the alphabet size K is set to 20 and the symbol alphabet to "ACDEFGHIKLMNPQRSTVWY" (alphabetic order); if `DNA`, the alphabet size K is set to 4 and the symbol alphabet to "ACGT"; if `RNA`, the alphabet size K is set to 4 and the symbol alphabet to "ACGU". **Mandatory.**

`RF <s>` Reference annotation flag; `<s>` is either `no` or `yes` (case insensitive). If `yes`, the reference annotation character field for each match state in the main model (see below) is valid; if `no`, these characters are ignored. Reference column annotation is picked up from a Stockholm alignment file's `#=GC RF` line. It is propagated to alignment outputs, and also may optionally be used to define consensus match columns in profile HMM construction. **Optional**; assumed to be no if not present.

`CS <s>` Consensus structure annotation flag; `<s>` is either `no` or `yes` (case insensitive). If `yes`, the consensus structure character field for each match state in the main model (see below) is valid; if `no` these characters are ignored. Consensus structure annotation is picked up from a Stockholm file's `#=GC SS_cons` line, and propagated to alignment displays. **Optional**; assumed to be no if not present.

`MAP <s>` Map annotation flag; `<s>` is either `no` or `yes` (case insensitive). If set to `yes`, the map annotation field in the main model (see below) is valid; if `no`, that field will be ignored. The HMM/alignment map annotates each match state with the index of the alignment column from which it came. It can be used for quickly mapping any subsequent HMM alignment back to the original multiple alignment, via the model. **Optional**; assumed to be no if not present.

`DATE <s>` Date the model was constructed; `<s>` is a free text date string. This field is only used for logging purposes.[8] **Optional.**

`COM [<n>] <s>` Command line log; `<n>` counts command line numbers, and `<s>` is a one-line command. There may be more than one `COM` line per save file, each numbered starting from $n = 1$. These lines record every HMMER command that modified the save file. This helps us reproducibly and automatically log how Pfam models have been constructed, for example. **Optional.**

[8] HMMER does not use dates for any purpose other than human-readable annotation, so it is no more prone than you are to Y2K, Y2038, or any other date-related eschatology.

`NSEQ <d>` Sequence number; `<d>` is a nonzero positive integer, the number of sequences that the HMM was trained on. This field is only used for logging purposes. **Optional.**

`EFFN <f>` Effective sequence number; `<f>` is a nonzero positive real, the effective total number of sequences determined by `hmmbuild` during sequence weighting, for combining observed counts with Dirichlet prior information in parameterizing the model. This field is only used for logging purposes. **Optional.**

`CKSUM <d>` Training alignment checksum; `<d>` is a nonnegative 32-bit integer. This number is calculated from the training sequence data, and used in conjunction with the alignment map information to verify that a given alignment is indeed the alignment that the map is for. **Optional.**

`GA <f> <f>` Pfam gathering thresholds GA1 and GA2. See Pfam documentation of GA lines. **Optional.**

`TC <f> <f>` Pfam trusted cutoffs TC1 and TC2. See Pfam documentation of TC lines. **Optional.**

`NC <f> <f>` Pfam noise cutoffs NC1 and NC2. See Pfam documentation of NC lines. **Optional.**

`STATS <s1> <s2> <f>` Statistical parameters needed for E-value calculations. `<s1>` is the model's alignment mode configuration: currently only `LOCAL` is recognized. `<s2>` is the name of the parameter: currently `VLAMBDA`, `VMU`, and `FTAU` are recognized, representing the Viterbi score slope parameter λ, Viterbi score location parameter μ, and Forward score location parameter τ (Eddy, 2008). Each parameter is a real number `f`; λ must be positive. Either all three lines or none of them must be present: when all three are present, the model is considered to be calibrated for E-value statistics. **Optional.**

`HMM` Flags the start of the main model section. Solely for human readability of the tabular model data, the symbol alphabet is shown on the `HMM` line, aligned to the fields of the match and insert symbol emission distributions in the main model below. The next line is also for human readability, providing column headers for the state transition probability fields in the main model section that follows. Though unparsed after the `HMM` tag, the presence of two header lines is **mandatory:** the parser always skips the line after the `HMM` tag line.

`COMPO <f>*K` The first line in the main model section may be an optional line starting with `COMPO`: these are the model's overall average match state emission probabilities, which are used as a background residue composition in the "filter null" model. The K fields on this line are log probabilities for each residue in the appropriate biosequence alphabet's order. **Optional.**

main model section

All the remaining fields are **mandatory**.

The first two lines in the main model section are atypical. They contain information for the core model's BEGIN node. This is stored as model node 0, and match state 0 is treated as the BEGIN state. The begin state is mute, so there are no match emission probabilities. The first line is the insert 0 emissions. The second line contains the transitions from the begin state and insert state 0. These seven numbers are: $B \rightarrow M_1$, $B \rightarrow I_0$, $B \rightarrow D_1$; $I_0 \rightarrow M_1$, $I_0 \rightarrow I_0$; and by convention, nonexistent transitions from the nonexistent delete state 0 are set to $\log 1 = 0$ and $\log 0 = -\infty =$ '*'.

The remainder of the model has three lines per node, for M nodes (where M is the number of match states, as given by the `LENG` line). These three lines are (K is the alphabet size in residues):

Match emission line The first field is the node number ($1 \ldots M$). The parser verifies this number as a consistency check (it expects the nodes to come in order). The next K numbers for match emissions, one per symbol, in alphabetic order.

The next field is the `MAP` annotation for this node. If `MAP` was `yes` in the header, then this is an integer, representing the alignment column index for this match state (1..alen); otherwise, this field is '-'.

The next field is the `RF` annotation for this node. If `RF` was `yes` in the header, then this is a single character, representing the reference annotation for this match state; otherwise, this field is '-'.

The next field is the `CS` annotation for this node. If `CS` was `yes`, then this is a single character, representing the consensus structure at this match state; otherwise this field is '-'.

Insert emission line The K fields on this line are the insert emission scores, one per symbol, in alphabetic order.

State transition line The seven fields on this line are the transitions for node k, in the order shown by the transition header line: $M_k \rightarrow M_{k+1}, I_k, D_{k+1}$; $I_k \rightarrow M_{k+1}, I_k$; $D_k \rightarrow M_{k+1}, D_{k+1}$.

For transitions from the final node M, match state $M+1$ is interpreted as the END state E, and there is no delete state $M+1$; therefore the final $M_k \rightarrow D_{k+1}$ and $D_k \rightarrow D_{k+1}$ transitions are always * (zero probability), and the final $D_k \rightarrow M_{k+1}$ transition is always 0.0 (probability 1.0).

Finally, the last line of the format is the "//" record separator.

Stockholm, the recommended multiple sequence alignment format

The Pfam and Rfam Consortiums have developed a multiple sequence alignment format called "Stockholm format" that allows rich and extensible annotation.

Most popular multiple alignment file formats can be changed into a minimal Stockholm format file just by adding a Stockholm header line and a trailing `//` terminator:

```
# STOCKHOLM 1.0

seq1  ACDEF...GHIKL
seq2  ACDEF...GHIKL
seq3  ...EFMNRGHIKL

seq1  MNPQTVWY
seq2  MNPQTVWY
seq3  MNPQT...
//
```

The first line in the file must be `# STOCKHOLM 1.x`, where `x` is a minor version number for the format specification (and which currently has no effect on my parsers). This line allows a parser to instantly identify the file format.

In the alignment, each line contains a name, followed by the aligned sequence. A dash, period, underscore, or tilde (but not whitespace) denotes a gap. If the alignment is too long to fit on one line, the alignment may be split into multiple blocks, with blocks separated by blank lines. The number of sequences, their order, and their names must be the same in every block. Within a given block, each (sub)sequence (and any associated `#=GR` and `#=GC` markup, see below) is of equal length, called the *block length*. Block lengths may differ from block to block. The block length must be at least one residue, and there is no maximum.

Other blank lines are ignored. You can add comments anywhere to the file (even within a block) on lines starting with a `#`.

All other annotation is added using a tag/value comment style. The tag/value format is inherently extensible, and readily made backwards-compatible; unrecognized tags will simply be ignored. Extra annotation includes consensus and individual RNA or protein secondary structure, sequence weights, a reference coordinate system for the columns, and database source information including name, accession number, and coordinates (for subsequences extracted from a longer source sequence) See below for details.

syntax of Stockholm markup

There are four types of Stockholm markup annotation, for per-file, per-sequence, per-column, and per-residue annotation:

`#=GF <tag> <s>` Per-file annotation. `<s>` is a free format text line of annotation type `<tag>`. For example, `#=GF DATE April 1, 2000`. Can occur anywhere in the file, but usually all the `#=GF` markups occur in a header.

`#=GS <seqname> <tag> <s>` Per-sequence annotation. `<s>` is a free format text line of annotation type `tag` associated with the sequence named `<seqname>`. For example, `#=GS seq1 SPECIES_SOURCE Caenorhabditis elegans`. Can occur anywhere in the file, but in single-block formats (e.g. the Pfam distribution) will typically follow on the line after the sequence itself, and in multi-block formats (e.g. HMMER output), will typically occur in the header preceding the alignment but following the `#=GF` annotation.

`#=GC <tag> <...s...>` Per-column annotation. `<...s...>` is an aligned text line of annotation type `<tag>`. `#=GC` lines are associated with a sequence alignment block; `<...s...>` is aligned to the residues in the alignment block, and has the same length as the rest of the block. Typically `#=GC` lines are placed at the end of each block.

`#=GR <seqname> <tag> <.....s.....>` Per-residue annotation. `<...s...>` is an aligned text line of annotation type `<tag>`, associated with the sequence named `<seqname>`. `#=GR` lines are associated with one sequence in a sequence alignment block; `<...s...>` is aligned to the residues in that sequence, and has the same length as the rest of the block. Typically `#=GR` lines are placed immediately following the aligned sequence they annotate.

semantics of Stockholm markup

Any Stockholm parser will accept syntactically correct files, but is not obligated to do anything with the markup lines. It is up to the application whether it will attempt to interpret the meaning (the semantics) of the markup in a useful way. At the two extremes are the Belvu alignment viewer and the HMMER profile hidden Markov model software package.

Belvu simply reads Stockholm markup and displays it, without trying to interpret it at all. The tag types (`#=GF`, etc.) are sufficient to tell Belvu how to display the markup: whether it is attached to the whole file, sequences, columns, or residues.

HMMER uses Stockholm markup to pick up a variety of information from the Pfam multiple alignment database. The Pfam consortium therefore agrees on additional syntax for certain tag types, so HMMER can parse some markups for useful information. This additional syntax is imposed by Pfam, HMMER, and other software of mine, not by Stockholm format per se. You can think of Stockholm as akin to XML, and what my software reads as akin to an XML DTD, if you're into that sort of structured data format lingo.

The Stockholm markup tags that are parsed semantically by my software are as follows:

recognized #=GF annotations

`ID <s>` Identifier. `<s>` is a name for the alignment; e.g. “rrm”. One word. Unique in file.

`AC <s>` Accession. `<s>` is a unique accession number for the alignment; e.g. “PF00001”. Used by the Pfam database, for instance. Often a alphabetical prefix indicating the database (e.g. “PF”) followed by a unique numerical accession. One word. Unique in file.

`DE <s>` Description. `<s>` is a free format line giving a description of the alignment; e.g. “RNA recognition motif proteins”. One line. Unique in file.

`AU <s>` Author. `<s>` is a free format line listing the authors responsible for an alignment; e.g. “Bateman A”. One line. Unique in file.

`GA <f> <f>` Gathering thresholds. Two real numbers giving HMMER bit score per-sequence and per-domain cutoffs used in gathering the members of Pfam full alignments.

`NC <f> <f>` Noise cutoffs. Two real numbers giving HMMER bit score per-sequence and per-domain cutoffs, set according to the highest scores seen for unrelated sequences when gathering members of Pfam full alignments.

`TC <f> <f>` Trusted cutoffs. Two real numbers giving HMMER bit score per-sequence and per-domain cutoffs, set according to the lowest scores seen for true homologous sequences that were above the GA gathering thresholds, when gathering members of Pfam full alignments.

recognized #=GS annotations

`WT <f>` Weight. `<f>` is a positive real number giving the relative weight for a sequence, usually used to compensate for biased representation by downweighting similar sequences. Usually the weights average 1.0 (e.g. the weights sum to the number of sequences in the alignment) but this is not required. Either every sequence must have a weight annotated, or none of them can.

`AC <s>` Accession. `<s>` is a database accession number for this sequence. (Compare the `#=GF AC` markup, which gives an accession for the whole alignment.) One word.

`DE <s>` Description. `<s>` is one line giving a description for this sequence. (Compare the `#=GF DE` markup, which gives a description for the whole alignment.)

recognized #=GC annotations

`RF` Reference line. Any character is accepted as a markup for a column. The intent is to allow labeling the columns with some sort of mark.

`SS_cons` Secondary structure consensus. For protein alignments, DSSP codes or gaps are accepted as markup: [HGIEBTSCX.-_], where H is alpha helix, G is 3/10-helix, I is p-helix, E is extended strand, B is a residue in an isolated b-bridge, T is a turn, S is a bend, C is a random coil or loop, and X is unknown (for instance, a residue that was not resolved in a crystal structure).

`SA_cons` Surface accessibility consensus. 0-9, gap symbols, or X are accepted as markup. 0 means ¡10% accessible residue surface area, 1 means ¡20%, 9 means ¡100%, etc. X means unknown structure.

recognized #=GR annotations

`SS` Secondary structure consensus. See `#=GC SS_cons` above.

`SA` Surface accessibility consensus. See `#=GC SA_cons` above.

References

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucl. Acids Res.*, 25:3389–3402.

Bashford, D., Chothia, C., and Lesk, A. M. (1987). Determinants of a protein fold: Unique features of the globin amino acid sequences. *J. Mol. Biol.*, 196:199–216.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. J. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge UK.

Eddy, S. R. (1998). Profile hidden Markov models. *Bioinformatics*, 14:755–763.

Eddy, S. R. (2008). A probabilistic model of local sequence alignment that simplifies statistical significance estimation. *PLoS Comput. Biol.*, 4:e1000069.

Holmes, I. (1998). *Studies in Probabilistic Sequence Alignment and Evolution*. PhD thesis, University of Cambridge.

Karlin, S. and Altschul, S. F. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. USA*, 87:2264–2268.

Karlin, S. and Altschul, S. F. (1993). Applications and statistics for multiple high-scoring segments in molecular sequences. *Proc. Natl. Acad. Sci. USA*, 90:5873–5877.

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., and Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.*, 235:1501–1531.