HMMER User's Guide

Biological sequence analysis using profile hidden Markov models

http://hmmer.org/
Version 3.0rc1; February 2010

Sean R. Eddy
for the HMMER Development Team
Janelia Farm Research Campus
19700 Helix Drive
Ashburn VA 20147 USA
http://eddylab.org/

Contents

1 Introduction

HMMER is used for searching sequence databases for homologs of protein sequences, and for making protein sequence alignments. HMMER can be used to search sequence databases with single query sequences but it becomes particularly powerful when the query is an multiple sequence alignment of a sequence family. HMMER makes a *profile* of the query, a probabilistic consensus with a position-specific scoring system for substitutions, insertions, and deletions. HMMER profiles are probabilistic models called "profile hidden Markov models" (profile HMMs) (**???**).

Compared to BLAST, FASTA, and other sequence alignment and database search tools based on older scoring methodology, HMMER aims to be significantly more accurate and more able to detect remote homologs, because of the strength of its underlying mathematical models. In the past, this strength came at a significant computational cost, with profile HMM implementations running about 100x slower than comparable BLAST searches. With the release of HMMER3 in 2010, HMMER is now essentially as fast as BLAST.

How to avoid reading this manual

I hate reading documentation. If you're like me, you're sitting here thinking, 35 pages of documentation, you must be joking! I just want to know that the software compiles, runs, and gives apparently useful results, before I read some 35 exhausting pages of someone's documentation. For cynics that have seen one too many software packages that don't work:

- Follow the quick installation instructions on page 8. An automated test suite is included, so you will know immediately if something went wrong.
- Go to the tutorial, section on page 11, which walks you through some examples of using HMMER on real data.

Everything else, you can come back and read later.

How to avoid using this software (links to similar software)

Several implementations of profile HMM methods and related position-specific scoring matrix methods are available.

Software	URL
HMMER	`http://hmmer.org/`
SAM	`http://www.cse.ucsc.edu/research/compbio/sam.html`
PSI-BLAST	`http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/psi1.htm`
PFTOOLS	`http://www.isrec.isb-sib.ch/profile/profile.html`

The UC Santa Cruz SAM software is the closest relative of HMMER.

What profile HMMs are

Profile HMMs are statistical models of multiple sequence alignments, or even of single sequences. They capture position-specific information about how conserved each column of the alignment is, and which residues are likely. Anders Krogh, David Haussler, and co-workers at UC Santa Cruz introduced profile HMMs to computational biology (**?**), adopting HMM techniques which have been used for years in speech recognition. HMMs had been used in biology before the Krogh/Haussler work, notably by Gary Churchill (**?**), but the Krogh paper had a dramatic impact because HMM technology was so well-suited to the popular "profile" methods for searching databases using multiple sequence alignments instead of single query sequences.

"Profiles" had been introduced by Gribskov and colleagues (**??**), and several other groups introduced similar approaches at about the same time, such as "flexible patterns" (**?**), and "templates"(**??**). The term "profile" has stuck.[1] All profile methods (including PSI-BLAST (**?**)) are more or less statistical descriptions of the consensus of a multiple sequence alignment. They use *position-specific* scores for amino acids (or nucleotides) and position specific penalties for opening and extending an insertion or deletion. Traditional pairwise alignment (for example, BLAST (**?**), FASTA (**?**), or the Smith/Waterman algorithm (**?**)) uses position-*independent* scoring parameters. This property of profiles captures important information about the degree of conservation at various positions in the multiple alignment, and the varying degree to which gaps and insertions are permitted.

The advantage of using HMMs is that HMMs have a formal probabilistic basis. We use probability theory to guide how all the scoring parameters should be set. Though this might sound like a purely academic issue, this probabilistic basis lets us do things that more heuristic methods cannot do easily. One of the most important is that HMMs have a consistent theory for setting position-specific gap and insertion scores. The methods are consistent and therefore highly automatable, allowing us to make libraries of hundreds of profile HMMs and apply them on a very large scale to whole genome analysis. One such database of protein domain models is Pfam (**??**), which is a significant part of the Interpro protein domain annotation system (**?**). The construction and use of Pfam is tightly tied to the HMMER software package.

Profile HMMs do have important limitations. One is that HMMs do not capture any higher-order correlations. An HMM assumes that the identity of a particular position is independent of the identity of all other positions.[2] Profile HMMs are often not good models of structural RNAs, for instance, because an HMM cannot describe base pairs.

Applications of profile HMMs

HMMER can be used to replace BLASTP and PSI-BLAST for searching databases with single query sequences. With HMMER3, we now include two programs for searching databases with single query sequences: `phmmer` and `jackhmmer`, where `jackhmmer` is an iterative search akin to PSI-BLAST.

Another application of HMMER is when you are working on a protein sequence family, and you have carefully constructed a multiple sequence alignment. Your family, like most protein families, has a number of strongly (but not absolutely) conserved key residues, separated by characteristic spacing. You wonder if there are more members of your family in the sequence databases, but the family is so evolutionarily diverse, a BLAST search with any individual sequence doesn't even find the rest of the sequences you already know about. You're sure there are some distantly related sequences in the noise. You spend many pleasant evenings scanning weak BLAST alignments by eye to find ones with the right key residues are in the right places, but you wish there was a computer program that did this a little better.

Another application is the automated annotation of the domain structure of proteins. Large databases of curated alignments and HMMER models of known domains are available, including Pfam (**?**) and SMART (**?**) in the Interpro database consortium (**?**). (Many "top ten protein domains" lists, a standard table in genome analysis papers, rely heavily on HMMER annotation.) Say you have a new sequence that, according to a BLAST analysis, shows a slew of hits to receptor tyrosine kinases. Before you decide to call your sequence an RTK homologue, you suspiciously recall that RTK's are, like many proteins, composed of multiple functional domains, and these domains are often found promiscuously in proteins with a wide variety of functions. Is your sequence really an RTK? Or is it a novel sequence that just happens to have a protein kinase catalytic domain or fibronectin type III domain?

Another application is the automated construction and maintenance of large multiple alignment databases. It is useful to organize sequences into evolutionarily related families. But there are thousands of protein sequence families, some of which comprise tens of thousands of sequences – and the primary sequence

[1] There has been agitation in some quarters to call all such models "position specific scoring matrices", PSSMs, but certain small nocturnal North American marsupials have a prior claim on the name.

[2] This is not strictly true. There is a subtle difference between an HMM's state path (a first order Markov chain) and the sequence described by an HMM (generated from the state path by independent emissions of symbols at each state).

databases continue to double every year or two. This is a hopeless task for manual curation; but on the other hand, manual curation is really necessary for high-quality, biologically relevant multiple alignments. Databases like Pfam (**?**) are constructed by distinguishing between a stable curated "seed" alignment of a small number of representative sequences, and "full" alignments of all detectable homologs; HMMER is used to make a model of the seed, search the database for homologs, and automatically produce the full alignment by aligning every sequence to the seed consensus.

You may find other applications as well. Using hidden Markov models to make a linear consensus model of a bunch of related strings is a pretty generic problem, and not just in biological sequence analysis. HMMER3 has already been used to model mouse song [Elena Rivas, personal communication] and in the past HMMER has reportedly been used to model music, speech, and even automobile engine telemetry. If you use it for something particularly strange, I'd be curious to hear about it (but I never, ever want to see my error messages showing up on the console of my Saab).

Design goals of HMMER3

In the past, profile HMM methods were considered to be too computationally expensive, and BLAST has remained the main workhorse of sequence similarity searching. The main objective of the HMMER3 project (begun in 2004) is to combine the power of probabilistic inference with high computational speed. We aim to upgrade some of molecular biology's most important sequence analysis applications to use more powerful statistical inference engines, without 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; they are 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/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 (**?**).

And as far as speed goes, there's really nothing new about HMMER3's speed either. Besides Karlin/Altschul statistics, the main 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. It certainly benchmarks that way in practice, at least in our hands. 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's still missing in HMMER3

Even though this is a stable public release that we consider suitable for large-scale production work (genome annotation, Pfam analysis, whatever), at the same time, HMMER3 remains a work in progress. We think the codebase is a suitable foundation for development of a number of significantly improved approaches to classically important problems in sequence analysis. Some of the more important "holes" for us are:

DNA sequence comparison. HMMER3'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.

Profile/profile comparison. A number of pioneering papers and software packages have demonstrated the power of profile/profile comparison for even more sensitive remote homology detection. We will aim to develop profile HMM methods in HMMER with improved detection power, and at HMMER3 speed.

We also have some technical and usability issues we will be addressing Real Soon Now:

More sequence input formats. HMMER3 will work fine with FASTA files for unaligned sequences, and Stockholm and "aligned FASTA" 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 may require a reformat to Stockholm or aligned FASTA 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.

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 we 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 we 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 you may 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.

More processor support. One of the attractive features of the HMMER3 "MSV" acceleration 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 significantly more speed out of HMMER in the future.

How to learn more about profile HMMs

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.

Reviews of the profile HMM literature have been written by myself (**??**) and by Anders Krogh (**?**).

For details on how profile HMMs and probabilistic models are used in computational biology, see the pioneering 1994 paper from Krogh et al. (**?**) or our book *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* (**?**).

To learn more about HMMs from the perspective of the speech recognition community, an excellent tutorial introduction has been written by Rabiner (**?**).

> ▷ ***How do I cite HMMER?*** *There is still no "real" paper on HMMER. If you're writing for an enlightened (url-friendly) journal, the best reference is* `http://hmmer.org/`*. If you must use a paper reference, the best one to use is my 1998 profile HMM review (**?**).*

2 Installation

Quick installation instructions

Download `hmmer-3.0rc1.tar.gz` from `http://hmmer.org/`, or directly from `ftp://selab.janelia.org/pub/software/hmmer3/hmmer-3.0rc1.tar.gz`; untar; and change into the newly created directory `hmmer-3.0rc1`:

```
> wget ftp://selab.janelia.org/pub/software/hmmer3/hmmer-3.0rc1.tar.gz
> tar xf hmmer-3.0rc1.tar.gz
> cd hmmer-3.0rc1
```

The beta 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`.

That's it. You can keep reading if you want to know more about customizing a HMMER3 installation, or you can skip ahead to the next chapter, the tutorial.

System requirements

Operating system: HMMER is designed to run on POSIX-compatible platforms, including UNIX, Linux and MacOS/X. The POSIX standard essentially includes all operating systems except Microsoft Windows.[1]

Our distribution tarball 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 on MacOS/X, because these are the machines we develop on. We test on a variety of other POSIX-compliant systems in our compile farm. We aim to be portable to all POSIX platforms. We currently do not develop or test on Windows.

Processor: HMMER3 depends on vector parallelization methods that are supported on most modern processors. H3 requires either an x86-compatible (IA32/IA64) processor that supports the SSE2 vector instruction set, or a PowerPC processor that supports the Altivec/VMX instruction set. SSE2 is supported on Intel processors from Pentium 4 on, and AMD processors from K8 (Athlon 64) on; we believe this includes almost all Intel processors since 2000 and AMD processors since 2003. Altivec/VMX is supported on Motorola G4, IBM G5, and IBM Power6 processors, which we believe includes almost all PowerPC-based desktop systems since 1999 and servers since 2007.

[1] There are add-on products available for making Windows more POSIX-compliant and more compatible with GNU-ish configures and builds. One such product is Cygwin, `http:www.cygwin.com`, which is freely available. Although we do not test on Windows platforms, we understand HMMER builds fine in a Cygwin environment on Windows.

If your platform does not support one of these vector instruction sets, the configure script will revert to an unoptimized implementation called the "dummy" implementation. This implementation is two orders of magnitude slower. It will enable you to see H3's features on a much wider range of processors, but is not suited for real production work.

We do aim to be portable to all modern processors. The acceleration algorithms are designed to be portable despite their use of specialized SIMD vector instructions. We hope to add support for the Sun SPARC VIS instruction set, for example. 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 test the code using both the `gcc` and `icc` compilers. We find that `icc` produces somewhat faster code at present.

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. Bundling Easel instead of making it a separate installation requirement is a deliberate design decision to simplify the installation process.[2]

Multithreaded parallelization for multicores is the default

The four search programs and `hmmbuild` support multicore parallelization using POSIX threads. By default, the configure script will identify whether your platform supports POSIX threads (almost all platforms do), and will automatically compile in multithreading support.

If you want to disable multithreading at compile time, recompile from source after giving the `--disable-threads` flag to `./configure`.

By default, our multithreaded programs will use all available cores on your machine. You can control the number of cores each HMMER process will use for computation with the `--cpu <x>` command line option or the `HMMER_NCPU` environment variable. Even with a single processing thread (`--cpu 1`), HMMER will devote a second execution thread to database input, resulting in significant speedup over serial execution.

If you specify `--cpu 0`, the program will run in serial-only mode, with no threads. This might be useful if you suspect something is awry with the threaded parallel implementation.

MPI parallelization for clusters is optional

The four search programs and hmmbuild now also support MPI (Message Passing Interface) parallelization on clusters. To use MPI, you first need to have an MPI library installed, such as OpenMPI (`www.open-mpi.org`). We use Intel MPI at Janelia.

MPI support is not enabled by default, and it is not compiled into the precompiled binaries that we supply with HMMER. To enable MPI support at compile time, give the `--enable-mpi` option to the `./configure` command.

To use MPI parallelization, each program that has an MPI-parallel mode has an `--mpi` command line option. This option activates a master/worker parallelization mode. (Without the `--mpi` option, if you run a program under `mpirun` on N nodes, you'll be running N independent duplicate commands, not a single MPI-enabled command. Don't do that.)

The MPI implementation for `hmmbuild` scales well up to hundreds of processors, and `hmmsearch` scales all right. The other search programs (`hmmscan`, `phmmer`, and `jackhmmer`) scale poorly, and probably shouldn't be used on more than tens of processors at most. Improving MPI scaling is one of our goals.

[2] If you install more than one package that uses the Easel library, it may become an annoyance; you'll have multiple instantiations of Easel lying around. The Easel API is not yet stable enough to decouple it from the applications that use it, like HMMER and Infernal.

Using build directories

The installation process from source supports using separate build directories, using the GNU-standard VPATH mechanism. This allows you to maintain separate builds for different processors or with different configuration/compilation options. All you have to do is run the configure script from the directory you want to be the root of the build directory. For example:

```
> mkdir my-hmmer-build
> cd my-hmmer-build
> /path/to/hmmer/configure
> make
```

You'd probably only use this feature if you're a developer, maintaining several different builds for testing purposes.

Makefile targets

`all` Builds everything. Same as just saying `make`.

`check` Runs automated test suites in both HMMER and the Easel library.

`clean` Removes all files generated by compilation (by `make`). Configuration (files generated by `./configure`) is preserved.

`distclean` Removes all files generated by configuration (by `./configure`) and by compilation (by `make`).

Note that if you want to make a new configuration (for example, to try an MPI version by `./configure --enable-mpi; make`) you should do a `make distclean` (rather than a `make clean`), to be sure old configuration files aren't used accidentally.

3 Tutorial

Here's a tutorial walk-through of some small projects with HMMER3. This should suffice to get you started on work of your own, and you can (at least temporarily) skip the rest of the Guide, such as all the nitty-gritty details of available command line options.

The programs in HMMER

Single sequence queries: new to HMMER3

phmmer	Search a sequence against a sequence database. (BLASTP-like)
jackhmmer	Iteratively search a sequence against a sequence database. (PSIBLAST-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 the core functionality for protein domain analysis and annotation pipelines, for instance using profile databases like Pfam or SMART.

The `phmmer` and `jackhmmer` programs are new to HMMER3. They searches a single sequence against a sequence database, akin to BLASTP and PSIBLAST, respectively. (Internally, they just produce a profile HMM from the query sequence, then run HMM searches.)

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

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 (**?**).

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

`globins4.out` An example `hmmsearch` output file that results from searching the `globins4.hmm` against Uniprot 15.7.

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` and `jackhmmer`.

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 can read Stockholm or aligned FASTA alignments.[1] The file `tutorial/globins4.sto` is an example of a simple Stockholm file. It looks like this:

```
# 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:

[1] It expects Stockholm by default. To read aligned FASTA files, which HMMER calls "afa" format, specify `--informat afa` on the command line of any program that reads an input alignment.

```
# hmmbuild :: profile HMM construction from multiple sequence alignments
# HMMER 3.0rc1 (February 2010); http://hmmer.org/
# Copyright (C) 2010 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 eff_nseq re/pos description
#---- -------------------- ----- ----- ----- -------- ------ -----------
1     globins4                 4   171   149     0.96  0.589

# CPU time: 0.23u 0.00s 00:00:00.23 Elapsed: 00:00:00.25
```

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 149 consensus positions, which means it defined 22 gap-containing alignment columns to be insertions relative to consensus. The 4 sequences were only counted as an "effective" total sequence number (`eff_nseq`) of 0.96. The model ended up with a relative entropy per position ((`re/pos`; information content) of 0.589 bits.

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/b [3.0rc1 | February 2010]
NAME  globins4
LENG  149
ALPH  amino
RF    no
CS    no
MAP   yes
DATE  Wed Feb 10 07:19:09 2010
NSEQ  4
EFFN  0.964844
CKSUM 2027839109
STATS LOCAL MSV       -9.9014  0.70957
STATS LOCAL VITERBI  -10.7224  0.70957
STATS LOCAL FORWARD   -4.1637  0.70957
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   2.36553  4.52577  2.96709  2.70473  3.20818  3.02239  3.41069  2.90041 ... 4.55393  3.62921
          2.68640  4.42247  2.77497  2.73145  3.46376  2.40504  3.72516  3.29302 ... 4.58499  3.61525
          0.57544  1.78073  1.31293  1.75577  0.18968  0.00000        *
      1   1.70038  4.17733  3.76164  3.36686  3.72281  3.29583  4.27570  2.40482 ... 5.32720  4.10031      9 - -
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354 ... 4.58477  3.61503
          0.03156  3.86736  4.58970  0.61958  0.77255  0.34406  1.23405
...
    149   2.92198  5.11574  3.28049  2.65489  4.47826  3.59727  2.51142  3.88373 ... 5.42147  4.18835    165 - -
          2.68634  4.42241  2.77536  2.73098  3.46370  2.40469  3.72511  3.29370 ... 4.58493  3.61418
          0.22163  1.61553        *  1.50361  0.25145  0.00000        *
//
```

The HMMER3 ASCII save file format is defined in Section 5.

If you're used to HMMER2, you may now 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 (**?**). The determination of the statistical parameters is part of `hmmbuild`. These are the parameter values on the three 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. And 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 15.7 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(s) against a sequence database
# HMMER 3.0rc1 (February 2010); http://hmmer.org/
# Copyright (C) 2010 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=149]
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
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
      6e-65  222.7   3.2    6.7e-65  222.6   2.2    1.0  1  sp|P02185|MYG_PHYCA   Myoglobin OS=Physeter catodon GN=MB PE
    3.1e-63  217.2   0.1    3.4e-63  217.0   0.0    1.0  1  sp|P02024|HBB_GORGO   Hemoglobin subunit beta OS=Gorilla gor
    4.5e-63  216.6   0.0      5e-63  216.5   0.0    1.0  1  sp|P68871|HBB_HUMAN   Hemoglobin subunit beta OS=Homo sapien
    4.5e-63  216.6   0.0      5e-63  216.5   0.0    1.0  1  sp|P68872|HBB_PANPA   Hemoglobin subunit beta OS=Pan paniscu
    4.5e-63  216.6   0.0      5e-63  216.5   0.0    1.0  1  sp|P68873|HBB_PANTR   Hemoglobin subunit beta OS=Pan troglod
    6.4e-63  216.1   3.0    7.1e-63  216.0   2.0    1.0  1  sp|P02177|MYG_ESCGI   Myoglobin OS=Eschrichtius gibbosus GN=
```

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.[2] 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

[2]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.

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
 ------- ------ -----    ------- ------ -----   ---- --  --------    -----------
 1.9e-57  178.0   0.4    1.2e-16   47.2   0.7    9.4  9  7LESS_DROME RecName: Full=Protein sevenless;          EC=2.7
```

OK, now let's pick up the explanation where we left off. The total sequence score of 178.0 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.

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, `MYG_PHYCA` (sperm whale myoglobin, if you're curious) has a full sequence score of 222.7 and a single best domain score of 222.6. 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 `MYG_PHYCA` 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 annotation for each sequence (and alignments):
```

Now for each sequence in the top hits list, there will be a section 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;
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.3   0.0      0.17      0.17      61      74 ..     396     409 ..     395     411 .. 0.85
   2 !   40.7   0.0   1.3e-14   1.3e-14       2      84 ..     439     520 ..     437     521 .. 0.95
   3 !   14.4   0.0     2e-06     2e-06      13      85 ..     836     913 ..     826     914 .. 0.73
   4 !    5.1   0.0    0.0016    0.0016      10      36 ..    1209    1235 ..    1203    1259 .. 0.82
   5 !   24.3   0.0   1.7e-09   1.7e-09      14      80 ..    1313    1380 ..    1304    1386 .. 0.82
   6 ?    0.0   0.0     0.063     0.063      58      72 ..    1754    1768 ..    1739    1769 .. 0.89
   7 !   47.2   0.7   1.2e-16   1.2e-16       1      85 [.    1799    1890 ..    1799    1891 .. 0.91
   8 !   17.8   0.0   1.8e-07   1.8e-07       6      74 ..    1904    1966 ..    1901    1976 .. 0.90
   9 !   12.8   0.0   6.6e-06   6.6e-06       1      86 []    1993    2107 ..    1993    2107 .. 0.89
```

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

The `!` or `?` symbol indicates whether this domain does or does not satisfy both per-sequence and per-domain inclusion thresholds. Inclusion thresholds are used to determine what matches should be considered to be "true", as opposed to reporting thresholds that determine what matches will be reported (often including the top of the noise, so you can see what interesting sequences might be getting tickled by your search). By default, inclusion thresholds usually require a per-sequence E value of 0.01 or less and a per-domain conditional E-value of 0.01 or less (except jackhmmer, which requires a more stringent 0.001 for both), and reporting E-value thresholds are set to 10.0.

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 (15.7 contains 497293 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:

```
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 !   40.7   0.0   9.1e-12   6.4e-09       2      84 ..     439     520 ..     437     521 .. 0.95
```

```
  2 !   14.4   0.0    0.0014          1      13      85 ..     836     913 ..     826     914 .. 0.73
  3 ?    5.1   0.0       1.1   7.9e+02      10      36 ..    1209    1235 ..    1203    1259 .. 0.82
  4 !   24.3   0.0   1.2e-06   0.00084      14      80 ..    1313    1380 ..    1304    1386 .. 0.82
  5 !   47.2   0.7   8.3e-14   5.8e-11       1      85 [.    1799    1890 ..    1799    1891 .. 0.91
  6 !   17.8   0.0   0.00013     0.091       6      74 ..    1904    1966 ..    1901    1976 .. 0.90
  7 !   12.8   0.0    0.0047       3.3       1      86 []    1993    2107 ..    1993    2107 .. 0.89
```

Notice that *almost* everything's the same (it's the same target sequence, after all) *except* for what happens with E-values. The independent E-value is calculated assuming a search space of all 497293 sequences. For example, look at the highest scoring domain (domain 5 here; domain 7 above). When we only looked at a single sequence, its score of 47.2 bits has an E-value of 5.8e-11. When we search a database of 497293 sequences, a hit scoring 47.2 bits would be expected to happen 497293 times as often: 1.2e-16 $\times$ 497293 $=$ 5.97e-11 (it's showing 5.8e-11 because of roundoff issues; the 1.2e-16 in fact isn't exactly 1.2e-16 inside HMMER). In this Uniprot search, 711 sequences were reported in the top hits list (with E-values ≤ 10). If we were to assume that all 711 are true homologs, x out the domain(s) that made us think that, and then went looking for *additional* domains in those 711 sequences, we'd be searching a smaller database of 711 sequences: the expected number of times we'd see a hit of 47.2 bits or better is now 1.2e-16 $\times$ 711 $=$ 8.3e-14. That's where the conditional E-value (c-Evalue) is coming from.

Notice that a couple of domains disappeared in the Uniprot search, because now, in this larger search space size, they're not significant. Domain 1 (the one with the score of -1.3 bits) got a conditional E-value of 0.17 $\times$ 711 = 121, and domain 6 (with a bit score of 0.0) got a c-Evalue of 0.063 $\times$ 711 = 45. These fail the default reporting threshold of 10.0. The domain with a score of 5.1 bits also shifted from being above to below the default inclusion thresholds, so now it's marked with a `?` instead of a `!`.

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 the Uniprot output, for example, I'd be pretty sure of four of the domains (1, 4, 5, and maybe 6), each of which has a strong enough independent E-value to declare `7LESS_DROME` to be an fnIII-domain-containing protein. Domains 2 and 7 wouldn't be significant if they were all I saw in the sequence, but once I decide that `7LESS_DROME` contains fn3 domains on the basis of the other hits, their conditional E-values indicate that they are probably also fn3 domains too. Domain 3 is too weak to be sure of, from this search alone, but would be something to pay attention to.

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[3]).

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.
```

These domains are a pretty tough case to call, actually. HMMER fails to see anything significant overlapping two of these domains (311-431 and 1680-1794) in the Uniprot search, though it sees a smidgen of them when `7LESS_DROME` alone is the target. HMMER3 sees two new domains (1205-1235 and 1993-2098) that Uniprot currently doesn't annotate, but these are pretty plausible domains (given that the extracellular domain of Sevenless is pretty much just a big array of ∼100aa fibronectin repeats.

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

```
== domain 2    score: 40.7 bits;  conditional E-value: 1.3e-14
                 ---CEEEEEECTTEEEEEEE--S..SS--SEEEEEEEETTTCCGCEEEEEETTTSEEEES--TT-EEEEEEEEETTEE.E CS
         fn3   2 saPenlsvsevtstsltlsWsppkdgggpitgYeveyqekgegeewqevtvprtttsvtltgLepgteYefrVqavngagegp 84
                 saP   ++ +  ++ l ++W p +  +gpi+gY++++++++++  + e+ vp+   s+ +++L++gt+Y++ +  +n++gegp
  7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                 78999999999*****************************9998.*********************************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 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

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

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.[4]

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 pipeline statistics summary:
-------------------------------------
Query model(s):                            1  (149 nodes)
Target sequences:                     497293  (175274722 residues)
Passed MSV filter:                     19416  (0.0390434); expected 9945.9 (0.02)
Passed bias filter:                    15923  (0.0320194); expected 9945.9 (0.02)
Passed Vit filter:                      2207  (0.00443803); expected 497.3 (0.001)
Passed Fwd filter:                      1076  (0.00216371); expected 5.0 (1e-05)
Initial search space (Z):              497293  [actual number of targets]
Domain search space  (domZ):             1075  [number of targets reported over threshold]
# CPU time: 6.02u 0.07s 00:00:06.09 Elapsed: 00:00:02.43
# Mc/sec: 10747.30
//
```

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 497,293 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 "Multi ungapped 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 ≤ 0.02 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 4% of the database got through the MSV filter.

A quick check is then done to see if the target sequence is "obviously" so biased in its composition that it's unlikely to be a true homolog. This is called the "bias filter". If you don't like it (it can occasionally be overaggressive) you can shut it off with the `--nobias` option. Here, 15923 sequences pass through the bias 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 four-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 - 2207 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 $\leq 10^{-5}$% through; 1076 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), the CPU time, and the elapsed time. This search took about 2.43 seconds of elapsed (wall clock time) (running with `--cpu 2` on two cores). That's in the same ballpark as BLAST, depending on which BLAST you compare to. On the same machine, also running dual-core, NCBI BLAST with one of these globin sequences took 2.3 seconds, and WU-BLAST took 4.8 seconds.

[4] 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 two or three 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 (Intel MPI), 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 impi 128
'mpirun -np 128 ./hmmbuild --mpi Pfam.hmm Pfam-A.seed > hmmbuild.out'
```

This reduces the time to build all of Pfam to about 40 seconds.

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 that means both "I haven't found time to document them yet" and "I still might decide to change them arbitrarily without telling you".

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 database
# HMMER 3.0rc1 (February 2010); http://hmmer.org/
# Copyright (C) 2010 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query sequence file:             7LESS_DROME
# target HMM database:             minifam
# per-seq hits tabular output:     7LESS.tbl
# per-dom hits tabular output:     7LESS.domtbl
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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
    -------  ------ -----    ------- ------ -----   ---- --  -------- -----------
    5.6e-57  178.0   0.4    3.5e-16   47.2   0.7    9.4  9  fn3      Fibronectin type III domain
    1.1e-43  137.2   0.0    1.7e-43  136.5   0.0    1.3  1  Pkinase  Protein kinase domain
```

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:

```
Domain and alignment annotation for each model:
>> fn3  Fibronectin type III domain
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.3   0.0      0.33       0.5      61      74 ..     396     409 ..     395     411 .. 0.85
   2 !   40.7   0.0   2.6e-14   3.8e-14       2      84 ..     439     520 ..     437     521 .. 0.95
   3 !   14.4   0.0   4.1e-06   6.1e-06      13      85 ..     836     913 ..     826     914 .. 0.73
   4 !    5.1   0.0    0.0032    0.0048      10      36 ..    1209    1235 ..    1203    1259 .. 0.82
   5 !   24.3   0.0   3.4e-09     5e-09      14      80 ..    1313    1380 ..    1304    1386 .. 0.82
   6 ?    0.0   0.0      0.13      0.19      58      72 ..    1754    1768 ..    1739    1769 .. 0.89
   7 !   47.2   0.7   2.3e-16   3.5e-16       1      85 [.    1799    1890 ..    1799    1891 .. 0.91
   8 !   17.8   0.0   3.7e-07   5.5e-07       6      74 ..    1904    1966 ..    1901    1976 .. 0.90
   9 !   12.8   0.0   1.3e-05     2e-05       1      86 []    1993    2107 ..    1993    2107 .. 0.89
```

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

```
== domain 2    score: 40.7 bits;  conditional E-value: 2.6e-14
                ---CEEEEEEECTTEEEEEEE--S..SS--SEEEEEEEETTTCCGCEEEEEETTTSEEEEES--TT-EEEEEEEEETTEE.E CS
        fn3   2 saPenlsvsevtstsltlsWsppkdgggpitgYeveyqekgegeewqevtvprtttsvtltgLepgteYefrVqavngagegp 84
                saP   ++ +  ++ l ++W p +  +gpi+gY++++++++++  + e+ vp+   s+ +++L++gt+Y++ +  +n++gegp
7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                78999999999**************************9998.*********************************9997 PP
```

You'd think 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 this is actually the case – but in fact, it's actually not so obvious this should be so, and HMMER is going out of its way to make it so. HMMER uses stochastic sampling algorithms to infer some parameters, and also to infer the exact domain number and domain boundaries in certain difficult cases. If HMMER ran its stochastic samples "properly", it would see different samples every time you ran a program, and all of you would complain to me that HMMER was weird and buggy because it gave different answers on the same problem. To suppress run-to-run variation, HMMER seeds its random number generator(s) identically every time you do a sequence comparison. If you're an expert, and you really want to see the proper stochastic variation that results from any sampling algorithms that got run, you can pass a command-line argument of `--seed 0` to programs that have this property (hmmbuild and the four search programs).

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          .-VLSDAEWQLVLNIWAKVEADVAGHGQDILIRLFKGHPETLEKFDKFKH
#=GR MYG_ESCGI  PP ..69*************************************************
MYG_HORSE          g--LSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKFKH
#=GR MYG_HORSE  PP 8..89*************************************************
MYG_PROGU          g--LSDGEWQLVLNVWGKVEGDLSGHGQEVLIRLFKGHPETLEKFDKFKH
#=GR MYG_PROGU  PP 8..89*************************************************
MYG_SAISC          g--LSDGEWQLVLNIWGKVEADIPSHGQEVLISLFKGHPETLEKFDKFKH
#=GR MYG_SAISC  PP 8..89*************************************************
MYG_LYCPI          g--LSDGEWQIVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFKH
#=GR MYG_LYCPI  PP 8..89*************************************************
MYG_MOUSE          g--LSDGEWQLVLNVWGKVEADLAGHGQEVLIGLFKTHPETLDKFDKFKN
#=GR MYG_MOUSE  PP 8..89*************************************************
MYG_MUSAN          v------DWEKVNSVWSAVESDLTAIGQNILLRLFEQYPESQNHFPKFKN
...
```

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 a "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 probably 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 80% 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         v-HWTAEEKAVINSVWQKV--DVEQDGHEALTRLFIVYPWTQRYFSTFGD
#=GR HBBL_RANCA PP 6.6799*************..****************************
```

```
HBB2_TRICR          .VHLTAEDRKEIAAILGKV--NVDSLGGQCLARLIVVNPWSRRYFHDFGD
#=GR HBB2_TRICR PP .69****************..****************************
#=GC PP_cons        .679*******************************************
#=GC RF             .xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

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.

The `#=GC RF` line is Stockholm-format *reference coordinate annotation*, with an x marking each column that the profile considered to be consensus.

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`.

Iterative searches using jackhmmer

The `jackhmmer` program is for searching a single sequence query iteratively against a sequence database, much as `PSI-BLAST` would do.

The first round is identical to a `phmmer` search. All the matches that pass the inclusion thresholds are put in a multiple alignment. In the second (and subsequent) rounds, a profile is made from these results, and the database is searched again with the profile.

Iterations continue either until no new sequences are detected or the maximum number of iterations is reached. By default, the maximum number of iterations is 5; you can change this with the `-N` option.

Your original query sequence is always included in the multiple alignments, whether or not it appears in the database.[5] The "consensus" columns assigned to each multiple alignment always correspond exactly to the residues of your query, so the coordinate system of every profile is always the same as the numbering of residues in your query sequence, 1..L for a sequence of length L.

Assuming you have Uniprot or something like it handy, here's an example command line for a jackhmmer search:

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

One difference from `phmmer` output you'll notice is that `jackhmmer` marks "new" sequences with a `+` and "lost" sequences with a `-`. New sequences are sequences that pass the inclusion threshold(s) in this round, but didn't in the round before. Lost sequences are the opposite: sequences that passed the

[5] If it *is* in the database, it will almost certainly be included in the internal multiple alignment twice, once because it's the query and once because it's a significant database match to itself. This redundancy won't screw up the alignment, because sequences are downweighted for redundancy anyway.

inclusion threshold(s) in the previous round, but have now fallen beneath (yet are still in the reported hits – it's possible, though rare, to lose sequences utterly, if they no longer even pass the reporting threshold(s)). In the first round, everything above the inclusion thresholds is marked with a `+`, and nothing is marked with a `-`. For example, the top of this output looks like:

```
# jackhmmer :: iteratively search a protein sequence against a protein database
# HMMER 3.0rc1 (February 2010); http://hmmer.org/
# Copyright (C) 2010 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query sequence file:             HBB_HUMAN
# target sequence database:        uniprot_sprot.fasta
# per-seq hits tabular output:     hbb-jack.tbl
# per-dom hits tabular output:     hbb-jack.domtbl
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       HBB_HUMAN  [L=146]
Description: Human beta hemoglobin.

Scores for complete sequences (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
+   2.3e-98  331.4   0.0    2.5e-98  331.2   0.0    1.0  1  sp|P68871|HBB_HUMAN   Hemoglobin subunit beta OS=Homo sapien
+   2.3e-98  331.4   0.0    2.5e-98  331.2   0.0    1.0  1  sp|P68872|HBB_PANPA   Hemoglobin subunit beta OS=Pan paniscu
+   2.3e-98  331.4   0.0    2.5e-98  331.2   0.0    1.0  1  sp|P68873|HBB_PANTR   Hemoglobin subunit beta OS=Pan troglod
+   9.1e-98  329.4   0.0      1e-97  329.3   0.0    1.0  1  sp|P02024|HBB_GORGO   Hemoglobin subunit beta OS=Gorilla gor
+     2e-96  325.1   0.0    2.2e-96  324.9   0.0    1.0  1  sp|P02025|HBB_HYLLA   Hemoglobin subunit beta OS=Hylobates l
+     2e-95  321.8   0.0    2.2e-95  321.7   0.0    1.0  1  sp|P02032|HBB_SEMEN   Hemoglobin subunit beta OS=Semnopithec
...
```

That continues until the inclusion threshold is reached, at which point you see a tagline "inclusion threshold" indicating where the threshold was set:

```
+   0.00076   24.1   0.0    0.00083   24.0   0.0    1.0  1  sp|P02180|MYG_BALPH   Myoglobin OS=Balaenoptera physalus GN=
+   0.00087   23.9   0.0     0.0009   23.9   0.0    1.0  1  sp|P02148|MYG_PONPY   Myoglobin OS=Pongo pygmaeus GN=MB PE=1
  ------ inclusion threshold ------
     0.0013   23.3   0.3      0.021   19.4   0.2    2.1  1  sp|P81044|HBAZ_MACEU  Hemoglobin subunit zeta (Fragments) OS
     0.0021   22.7   0.0     0.0022   22.6   0.0    1.0  1  sp|P02182|MYG_ZIPCA   Myoglobin OS=Ziphius cavirostris GN=MB
```

The domain output and search statistics are then shown just as in `phmmer`. At the end of this first iteration, you'll see some output that starts with `@@` (this is a simple tag that lets you search through the file to find the end of one iteration and the beginning of another):

```
@@ New targets included:   894
@@ New alignment includes: 895 subseqs (was 1), including original query
@@ Continuing to next round.

@@
@@ Round:                  2
@@ Included in MSA:        895 subsequences (query + 894 subseqs from 894 targets)
@@ Model size:             146 positions
@@
```

This (obviously) is telling you that the new alignment contains 895 sequences, your query plus 894 significant matches. For round two, it's built a new model from this alignment. Now for round two, it fires off what's essentially an `hmmsearch` of the target database with this new model:

```
Scores for complete sequences (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
    1.5e-67  231.0   0.2    1.7e-67  230.8   0.1    1.0  1  sp|P02055|HBB_MELME   Hemoglobin subunit beta OS=Meles meles
    2.3e-67  230.4   0.4    2.6e-67  230.2   0.3    1.0  1  sp|P81042|HBE_MACEU   Hemoglobin subunit epsilon OS=Macropus
    2.7e-67  230.2   0.3    2.9e-67  230.0   0.2    1.0  1  sp|P15449|HBB_MELCA   Hemoglobin subunit beta OS=Mellivora c
    3.3e-67  229.9   0.2    3.7e-67  229.7   0.2    1.0  1  sp|P68046|HBB_ODORO   Hemoglobin subunit beta OS=Odobenus ro
...
```

If you skim down through this output, you'll start seeing newly included sequences marked with `+`'s, such as:

```
...
    9.8e-30  108.3   0.0    1.1e-29  108.2   0.0    1.0  1  sp|P87497|MYG_CHIRA   Myoglobin OS=Chionodraco rastrospinosu
+   1.4e-29  107.8   0.3    1.5e-29  107.7   0.2    1.0  1  sp|P14399|MYG_MUSAN   Myoglobin OS=Mustelus antarcticus GN=m
    1.5e-29  107.7   0.3      2e-29  107.3   0.2    1.0  1  sp|P80017|GLBD_CAUAR  Globin D, coelomic OS=Caudina arenicol
      3e-29  106.8   0.0    3.3e-29  106.6   0.0    1.0  1  sp|P02022|HBAM_RANCA  Hemoglobin heart muscle subunit alpha-
      4e-29  106.3   0.0    4.4e-29  106.2   0.0    1.0  1  sp|Q9DEP0|MYG_CRYAN   Myoglobin OS=Cryodraco antarcticus GN=
+   9.3e-29  105.2   0.2      1e-28  105.0   0.1    1.0  1  sp|P14397|MYG_GALGA   Myoglobin OS=Galeorhinus galeus GN=mb
    1.3e-28  104.7   0.0    1.6e-28  104.4   0.0    1.0  1  sp|P0C227|GLB_NERAL   Globin OS=Nerita albicilla PE=1 SV=1
      2e-28  104.1   0.0    2.4e-28  103.8   0.0    1.0  1  sp|P09106|HBAT_PAPAN  Hemoglobin subunit theta-1 OS=Papio an
+   2.8e-28  103.6   0.1    3.1e-28  103.5   0.1    1.0  1  sp|P14398|MYG_GALJA   Myoglobin OS=Galeorhinus japonicus GN=
    7.9e-26   95.7   0.0    8.8e-26   95.5   0.0    1.0  1  sp|P23216|GLBP1_GLYDI Globin, major polymeric component P1 O
...
```

It's unusual to see sequences get lost (and marked with `-`), but it can happen; it doesn't happen in this globin example.

After round 2, many more globin sequences have been found:

```
@@ New targets included:   167
@@ New alignment includes: 1064 subseqs (was 895), including original query
@@ Continuing to next round.

@@
@@ Round:                  3
@@ Included in MSA:        1064 subsequences (query + 1063 subseqs from 1061 targets)
@@ Model size:             146 positions
@@
```

Because new sequences were included, it keeps going to round three, and then again to round four, then again to round five. After round five (where this example has found 1113 included hits in the database), the search ends quietly because there's a default maximum of five iterations, and you get:

```
@@ New targets included:   1
@@ New alignment includes: 1114 subseqs (was 1113), including original query
//
```

That `//` marks the end of the results for one query.

4 Manual pages

5 File formats

HMMER profile HMM files

The file `tutorial/fn3.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/b [3.0b2 | June 2009]
NAME  fn3
ACC   PF00041.12
DESC  Fibronectin type III domain
LENG  86
ALPH  amino
RF    no
CS    yes
MAP   yes
DATE  Mon Jun 15 09:39:41 2009
NSEQ  106
EFFN  11.415833
CKSUM 72638555
GA    8.00 7.20
TC    8.00 7.20
NC    7.90 7.90
STATS LOCAL MSV       -9.4043  0.71847
STATS LOCAL VITERBI   -9.7737  0.71847
STATS LOCAL FORWARD   -3.8341  0.71847
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   2.70271  4.89246  3.02314  2.64362  3.59817  2.82566  3.74147  3.08574  (...) 3.22607
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00338  6.08833  6.81068  0.61958  0.77255  0.00000        *
      1   3.16986  5.21447  4.52134  3.29953  4.34285  4.18764  4.30886  3.35801  (...) 3.93889      1 - -
          2.68629  4.42236  2.77530  2.73088  3.46365  2.40512  3.72505  3.29365  (...) 3.61514
          0.09796  2.38361  6.81068  0.10064  2.34607  0.48576  0.95510
      2   2.70230  5.97353  2.24744  2.62947  5.31433  2.60356  4.43584  4.79731  (...) 4.25623      3 - -
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00338  6.08833  6.81068  0.61958  0.77255  0.48576  0.95510
(...)
     85   2.48488  5.72055  3.87501  1.97538  3.04853  3.48010  4.51877  3.51898  (...) 3.43366    120 - B
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00338  6.08833  6.81068  0.61958  0.77255  0.48576  0.95510
     86   3.03720  5.94099  3.75455  2.96917  5.26587  2.91682  3.66571  4.11840  (...) 4.99111    121 - E
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  (...) 3.61503
          0.00227  6.08723        *  0.61958  0.77255  0.00000        *
//
```

An HMM file consists of one or more HMMs. Each HMM starts with the identifier `HMMER3/b` 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 b. (The 3/a format was used in alpha test versions of HMMER3.) 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/b` Unique identifier for the save file format version; the `/b` means that this is HMMER3 HMM file format version b. When HMMER3 changes its save file format, the revision code advances. This way, parsers may easily remain backwards compatible. The remainder of the line after the `HMMER3/b` tag is free text that is ignored by the parser. HMMER currently writes its version number and release date in brackets here, e.g. `[3.0b2 | June 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.[1] **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.**

[1] 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 unsigned 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> <f1> <f2> 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 score distribution: currently MSV, VITERBI, and FORWARD are recognized. <f1> and <f2> are two real-valued parameters controlling location and slope of each distribution, respectively; μ and λ for Gumbel distributions for MSV and Viterbi scores, and τ and λ for exponential tails for Forward scores. λ values must be positive. 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.[2] 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:

[2] That is, the first two lines after the optional COMPO line. Don't be confused by the presence of an optional COMPO line here. The COMPO line is placed in the model section, below the residue column headers, because it's an array of numbers much like residue scores, but it's not really part of the model.

$B \rightarrow M_1$, $B \rightarrow I_0$, $B \rightarrow D_1$; $I_0 \rightarrow M_1$, $I_0 \rightarrow I_0$; then a 0.0 and a '*', because 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.

PP Posterior probability for an aligned residue. This represents the probability that this residue is assigned to the HMM state corresponding to this alignment column, as opposed to some other state. (Note that a residue can be confidently *unaligned*: a residue in an insert state or flanking N or C state may have high posterior probability.) The posterior probability is encoded as 11 possible characters $0-9\star$: $0.0 \leq p < 0.05$ is coded as 0, $0.05 \leq p < 0.15$ is coded as 1, (... and so on ...), $0.85 \leq p < 0.95$ is coded as 9, and $0.95 \leq p \leq 1.0$ is coded as '*'. Gap characters appear in the PP line where no residue has been assigned.

6 Acknowledgements and history

HMMER 1 was developed on slow weekends in the lab at the MRC Laboratory of Molecular Biology, Cambridge UK, while I was a postdoc with Richard Durbin and John Sulston. I thank the Human Frontier Science Program and the National Institutes of Health for their remarkably enlightened support at a time when I was really supposed to be working on the genetics of neural development in *C. elegans*.

HMMER 1.8, the first public release of HMMER, came in April 1995, shortly after I moved to Washington University in St. Louis. A few bugfix releases followed. A number of more serious modifications and improvements went into HMMER 1.9 code, but 1.9 was never released. Some versions of HMMER 1.9 inadvertently escaped St. Louis and make it to some genome centers, but 1.9 was never documented or supported. HMMER 1.9 collapsed under its own weight in 1996.

HMMER 2 was a nearly complete rewrite, based on the new Plan 7 model architecture. Implementation was begun in November 1996. I thank the Washington University Dept. of Genetics, the NIH National Human Genome Research Institute, and Monsanto for their support during this time. Also, I thank the Biochemistry Academic Contacts Committee at Eli Lilly & Co. for a gift that paid for the trusty Linux laptop on which much of HMMER 2 was written. The laptop was indispensable. Far too much of HMMER was written in coffee shops, airport lounges, transoceanic flights, and Graeme Mitchison's kitchen. The source code still contains a disjointed record of where and when various bits were written.

HMMER then settled into a comfortable middle age, like its primary author – still actively maintained, though dramatic changes seemed increasingly unlikely. HMMER 2.1.1 was the stable release for three years, from 1998-2001. HMMER 2.2g was intended to be a beta release, but became the *de facto* stable release for two more years, 2001-2003. The final release of the HMMER2 series, 2.3, was assembled in spring 2003. The last bugfix release, 2.3.2, came out in October 2003.

If the world worked as I hoped and expected, the combination of the 1998 Durbin/Eddy/Krogh/Mitchison book *Biological Sequence Analysis* and the existence of HMMER2 as a widely-used proof of principle *should* have motivated the widespread adoption of probabilistic modeling methods for sequence analysis, particularly database search. We would declare Victory and move on. Richard Durbin moved on to human genomics; Anders Krogh moved on to pioneer a number of other probabilistic approaches for other biological sequence analysis problems; Graeme Mitchison moved on to quantum computing; I moved on to noncoding RNAs.

Yet BLAST continued to be the most widely used search program. HMMs seemed to be widely considered to be a mysterious and orthogonal black box, rather than a natural theoretical basis for important programs like BLAST. The NCBI, in particular, seemed to be slow to adopt or even understand HMM methods. This nagged at me; the revolution was unfinished!

When we moved the lab to Janelia Farm in 2006, I had to make a decision about what we should spend our time on. It had to be something "Janelian": something that I would work on with my own hands; something that would be difficult to accomplish under the usual reward structures of academic science; and something that would make the largest possible impact on science. I decided that we should aim to replace BLAST with an entirely new generation of software. The result is the HMMER3 project.

Thanks

HMMER is increasingly not just my own work, but the work of great people in my lab, including Steve Johnson, Alex Coventry, Dawn Brooks, Sergi Castellano, Michael Farrar, Travis Wheeler, and Elena Rivas. The current HMMER development team at Janelia Farm includes Sergi, Michael, and Travis as well as myself.

I would call the Janelia computing environment world-class except that it's even better than that. That's entirely due to Goran Ceric. HMMER3 testing now spins up thousands of processors at a time, an unearthly amount of computing power.

Over the years, the MRC-LMB computational molecular biology discussion group contributed many ideas to HMMER. In particular, I thank Richard Durbin, Graeme Mitchison, Erik Sonnhammer, Alex Bate-

man, Ewan Birney, Gos Micklem, Tim Hubbard, Roger Sewall, David MacKay, and Cyrus Chothia.

The UC Santa Cruz HMM group, led by David Haussler and including Richard Hughey, Kevin Karplus, Anders Krogh (now back in Copenhagen) and Kimmen Sjölander, has been a source of knowledge, friendly competition, and occasional collaboration. All scientific competitors should be so gracious. The Santa Cruz folks have never complained (at least in my earshot) that HMMER started as simply a re-implementation of their original ideas, just to teach myself what HMMs were.

In many places, I've reimplemented algorithms described in the literature. These are too numerous to credit and thank here. The original references are given in the comments of the code. However, I've borrowed more than once from the following folks that I'd like to be sure to thank: Steve Altschul, Pierre Baldi, Phillip Bucher, Warren Gish, Steve and Jorja Henikoff, Anders Krogh, and Bill Pearson.

HMMER is primarily developed on GNU/Linux and Apple Macintosh machines, but is tested on a variety of hardware. Over the years, Compaq, IBM, Intel, Sun Microsystems, Silicon Graphics, Hewlett-Packard, Paracel, and nVidia have provided generous hardware support that makes this possible. I owe a large debt to the free software community for the development tools I use: an incomplete list includes GNU gcc, gdb, emacs, and autoconf; the amazing valgrind; the indispensable Subversion; the ineffable perl; LaTeX and TeX; PolyglotMan; and the UNIX and Linux operating systems.

Finally, I will cryptically thank Dave "Mr. Frog" Pare and Tom "Chainsaw" Ruschak for a totally unrelated free software product that was historically instrumental in HMMER's development – for reasons that are best not discussed while sober.

References